A Speech Recognition Module for Speech-to-Text Language Translation

by

Sadiki Pili Mwanyoha

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Master of Engineering and Bachelor of Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

© Massachusetts Institute of Technology 1998. All rights reserved.

Author

Department of Electrical Engineering and Computer Science

May 18, 1998

Certified by...

Clifford J. Weinstein

Group Leader, Information Systems Technology Group, MIT Lincoln Laboratory Thesis Supervisor

Accepted by

Arthur C. Smith

Chairman, Department Committee on Graduate Theses
A Speech Recognition Module for Speech-to-Text Language Translation

by

Sadiki Pili Mwanyoha

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 1998, in partial fulfillment of the requirements for the degrees of Master of Engineering and Bachelor of Science

Abstract

This thesis involved the design and implementation of a speech recognition module to be used in a speech-to-text translation system. The module accepts continuous (English) speech input from a task-specific grammar. The two main areas of system development were: 1) speech modeling and 2) interfacing a recognition kernel to the GUI of an existing automated language translation system. The performance of several candidate speech models are explored. A detailed discussion of the kernel interface is also presented. Future applications center on facilitating communication in multinational environments.

Thesis Supervisor: Clifford J. Weinstein
Title: Group Leader, Information Systems Technology Group, MIT Lincoln Laboratory
Acknowledgments

First, I want to thank my thesis advisor for helping me to set realistic goals, helping me maintain my self-esteem, and looking for the benefits behind every apparent mishap.

Next, I would like to thank Young-Suk for putting up with my stifling appetite for disk space and always showing an interest in my work, using kind, encouraging words.

Lastly, I want to thank Linda for always lending an ear, helping me to see the humor in everyday academia, and giving me sound advice on a wide range of topics.
# Contents

1 Introduction ........................................................................... 8
  1.1 Problem Statement ..................................................... 9

2 Background and Related Work ........................................... 11
  2.1 Background .................................................................. 11
  2.1.1 TINA-GENESIS .................................................... 11
  2.1.2 Speech Recognition Primer ........................................ 12
  2.2 Related Work ............................................................. 21
    2.2.1 Parameter Interpolation ........................................... 21
    2.2.2 Previous Live Input System ...................................... 25

3 Design ................................................................................. 27
  3.1 HTK™ Development Environment ................................... 27
    3.1.1 Data Preparation .................................................. 27
    3.1.2 Acoustic Model Training ........................................ 28
    3.1.3 Language Model Training ....................................... 31
  3.2 Enabling Live Input ....................................................... 33

4 Results ................................................................................ 39
  4.1 Methods of Evaluation .................................................. 39
  4.2 Results Presented ........................................................ 40
  4.3 Discussion .................................................................... 41

5 Future Work ........................................................................ 43
5.1 Speech Modelling ............................................. 43
5.2 Interface to HTK ............................................. 44

A Concise hvite-interface Manual 45
List of Figures

2-1  Process flow for English-to-Korean text translation ............... 12
2-2  A 5-state HMM .................................................................. 15
2-3  A typical 3-word tree .......................................................... 16

3-1  Nonlinear mapping of probabilites when scale factor \( s \) is applied to (log) probabilities. .......................................................... 33
3-2  HVite state diagram .............................................................. 35
3-3  hvite-interface flow diagram ................................................. 38
List of Tables

2.1 Maximum likelihoods and \( \tilde{\lambda}_{opt} \) for different block sizes when performing DI on the OR text corpus. ........................................... 25

4.1 Model set results. Statistics shown are word-accuracy, sentence-accuracy, insertions (I), deletions (D), and substitutions (S). ......................... 41

4.2 Model set statistics. .................................................. 41
Chapter 1

Introduction

To date, researchers have invested considerable effort in automating the translation of human language. The ability to translate human language automatically has the potential to alleviate many of the barriers associated with global communication and has therefore captured the attention of various sectors. In particular, the demand for multilingual communication within the military community has grown steadily, and consequently, provided the motivation for several world-wide efforts in this area.

Traditionally, the US Military has relied solely on the services of human translators. But due to the inherent scarcity of human translators, the opportunities for effective multilingual communication are limited. For this reason, automation of the translation function promises to enhance multilingual communication at many levels by supplementing, and eventually replacing human translation. With this purpose in mind, the automatic translation problem can be stated as follows.

A translation system should receive a message in the source language as input and then produce this message in the target language as output. The input-output representations of the message to be translated can, in general, be any combination of speech and text (i.e. speech-speech, speech-text, etc.). For example, in a multilingual database query application the input could be either speech or text, while the output would be constrained to text (since databases are text-based). On the other hand, in a document translation situation both the input and output would, of course, be text.
A core translation function must convert text in the source language into equivalent (in meaning) text in the target language using state-of-the-art natural language analysis/synthesis techniques. In the case of document translation this core is all that is needed (aside from some preprocessing of the input). However, in speech-input and speech-output applications additional speech-recognition and speech-synthesis modules are needed respectively.

Set forth in this paper is a proposal to implement the speech-recognition module required in a English-to-Korean Speech-to-text translation system.

1.1 Problem Statement

The US Military has maintained an ongoing presence in Korea. To enable operation in a foreign territory it depends heavily on English-to-Korean translation. As mentioned above, there is much to be gained from the automation of the translation function, so an automatic speech-to-text translation system would be a highly desirable tool.

We will refer to the communication context, or task domain, in which this system will function as the Operation Reports (OR) domain. The OR corpus which contains the training and test data necessary for system development consists of 111 sentences represented in both English and Korean text. The sentences were collected from recordings of actual correspondence between officers during the course of military operations. Together, they define a domain-specific grammar around which a translation system can be designed. It is hoped that such a system could serve as a translator's aid during the course of multi-national operations similar to those which generated the OR corpus.

It should be noted at this point that this is by no means a first attempt at speech-to-text translation. Consequently, a mainstream approach to speech-recognition is taken which specializes to the particular grammars encountered in the OR domain. Furthermore, the core text translation system with which the speech module has been integrated, consists of the TINA and GENESIS modules developed at the Spoken Language Systems Group, MIT Laboratory of Computer Science; these subsystems
have demonstrated considerable success in similar projects already. As these modules will play a major role in the aggregate system, they will be discussed in the next chapter along with an introduction to speech recognition technology.
Chapter 2

Background and Related Work

2.1 Background

2.1.1 TINA-GENESIS

This project has been completed under the Information Systems Technology Group (IST), MIT Lincoln Laboratory. IST has been the first group to use TINA-GENESIS to solve language translation problems and the two modules continue to play an integral role in all of the group’s translation projects.[12]

The approach IST has taken involves the logical breakdown of the translation process into understanding and generation steps performed by TINA and GENESIS respectively. In the understanding step TINA takes source language text as input and using a source grammar and analysis lexicon produces a semantic frame. The semantic frame captures the meaning of the message in that it is a language-independent, or *interlingual*, representation.

In the generation step GENESIS uses a target grammar and synthesis lexicon to transform the semantic frame into the corresponding text representation in the target language. To make the process perceptually concrete the reader should note that grammars, lexicon, and messages are implemented as files while TINA and GENESIS are realized with text-processing programs. A pictorial representation of the process flow involved is presented in Figure 2-1.[12]
A fundamental advantage of this interlingual approach is that by decoupling the understanding and generation steps the influence of a particular source-target language pair on translation requirements is eliminated. Besides increasing performance, the absence of such an influence can greatly reduce system size and complexity.

2.1.2 Speech Recognition Primer

Today’s high-performance speech recognizers almost exclusively rely on a statistical model of speech production which relies heavily on Hidden Markov Models. What follows is a brief introduction to the theory on which the model is based.

Like all quantitative models of physical phenomena, a statistical model of speech can only be developed in light of a series of logical assumptions. The first assumption made is that we can define a statistical grammar \((S)\) as the set of all possible word sequences taken from a vocabulary, where each sequence has its own probability of occurrence. Next, we assume that each word sequence \((W)\) generates an acoustic observation sequence \((O)\) with probability \(P(W,O)\). Our estimate \((\hat{W})\) of the word sequence in \(S\) corresponding to a given observation sequence \((O)\) is that sequence which has the maximum a posteriori (MAP) probability. That is,

\[
\hat{W} = \underset{W \in S}{\operatorname{argmax}} P(W \mid O),
\]

which is of course optimal in the MAP sense. Using Bayes' Rule this can be rewritten as

\[
\hat{W} = \underset{W \in S}{\operatorname{argmax}} \frac{P(O \mid W)P(W)}{P(O)}.
\]
Since $P(O)$ does not depend on $W$ this reduces to

$$
\hat{W} = \arg \max_{W \in S} P(O \mid W) P(W).
$$

Before proceeding further, the reader should note that when the observations take on discrete values $P(\cdot)$ will be a probability. For the continuous case $P(\cdot)$ will be a likelihood. For the rest of the discussion, likelihoods and probabilities will be used interchangeably, so one should keep their respective senses in mind. The author's intention is not to confuse the reader, but rather to emphasize that both discrete and continuous realizations are viable.

$P(O \mid W)$ is termed the acoustic model, while $P(W)$ is called the language model. The acoustic model proposes a statistical framework for producing acoustic observations, whereas the language model tries to realize the probabilistic nature of the grammar $S$. We now discuss both models individually.

**The Acoustic Model**

To model the generation of spoken word sequences we must first decide on a fundamental speech unit from which to build these sequences. Our choice should negotiate the tradeoff between discrimination ability and trainability. That is, the set of speech units we choose should capture the inherent differences in speech sounds, implying a coarse (in time) partitioning; this results in a relatively large number of speech units. On the other hand, the Weak Law of Large Numbers suggests that training a model set on limited sample data requires a lot of data per unit; this implies a fine-grain (in time) partitioning, or relatively fewer speech units. For example, if we chose our speech units to be words, our model's ability to discriminate between words would be excellent, in theory. But the number of models relative to the amount of data available to train them would be prohibitive from an estimation standpoint.

In practice, a common choice for the fundamental speech unit is the phoneme. There are between 40 and 50 phonemes in the English language. This loose assessment is indicative of phoneme sets being defined *linguistically* rather than acoustically. Even
so, phonetic events are known to occur on the order of a few hundred milliseconds and do provide reasonable acoustic discrimination while being readily trainable.

The most basic choice of phoneme models are called monophones. Monophones are context-independent models of speech units. They do not attempt to model the effects that neighboring phonemes (defining a context) have on each other. Such effects are due to a phenomenon called coarticulation. Coarticulation can be described as the merging of adjacent phonemes that occurs during the course of natural speech. A phoneme can be accentuated or deemphasized by the presence of an adjacent phoneme depending on its neighbor. For example, Rabiner shows in [10] how phonemes that appear on word boundaries are accentuated as in the sentence *he eats several light tacos*. This demonstrates only one of numerous types of coarticulation.

A way to improve acoustic discrimination at the expense of trainability is to move to a much larger set of biphones or triphones, which are context-dependent phoneme models. Biphones model left or right contextual effects, while triphones model both left and right ones.

Contemporary approaches to modeling phonemes rely on the Hidden Markov Model (HMM). An HMM is a Finite State Network (FSN) possessing two defining characteristics: 1) FSN state transitions and occupations are probabilistic and 2) each FSN state is associated with an observable output which is also probabilistic. Figure 2-2 shows a five-state HMM that has continuous distributions. The $a_{ij}$ denote transition probabilities while the $p_i$ indicate output distributions. Note also the non-emitting entry and exit states. Their purpose is to enable the concatenation of multiple HMMs, which is a desirable feature, as we shall see.

We model each phoneme by assigning it its own HMM. That is, we assume that a set of acoustic observations can be generated by a corresponding sequence of HMM state occupations. Moreover, the dynamics of these acoustic observations are governed by HMM transition parameters.

The acoustic observations are usually chosen to be a speech parameterization derived from frequency measures, such as Mel-scale cepstra, or speech coding, such as linear prediction coefficients. Given the occurrence of a particular phone, the
observations form a vector of random variables obeying the state output distributions of the corresponding HMM; these random variables are often termed features. State output distributions are simple probability mass functions in the discrete case. In the continuous case it is customary to choose Gaussian mixture densities, which are entirely determined by means, covariances, and mixture weights.

Recall that the acoustic modeling problem arose from the need to compute \( P(O \mid W) \), or the probability that a specific word sequence \( (W) \) would generate an acoustic observation sequence \( (O) \). This quantity is readily computable once we have assumed a speech production model at the word sequence level. Logically, the desired model is no more than the concatenation of the HMMs constituting each of the words in the given sequence; in practice the necessary HMM substitutions are made using a pronouncing dictionary. The result is a single monolithic chain of HMMs. To compute \( P(O \mid W) \) we first note that there are many possible state sequences which could produce \( O \). For example, two consecutive observation samples could be produced by a double occupation of one state or two single occupations of adjacent states. The likelihood of a particular sequence is the product of the transition probabilities and output likelihoods associated with that sequence. That is, for a state sequence \( Q = q(1), q(2), ..., q(T) \),

\[
P(O, Q \mid W) = a_{q(0)q(1)} \prod_{t=1}^{T} b_{q(t)}(o_t)a_{q(t)q(t+1)}, \tag{2.4}
\]
assuming independence of observations; where $q(0)$ and $q(T + 1)$ are the entry and exit states of the model for $W$; where $a_{ij}$ is the probability of transitioning from state $i$ to $j$; and where $b_j(o_n)$ denotes the output distribution of state $j$ evaluated at the $n$-th observation sample. The total likelihood is then the sum of $P(O, Q | W)$ over all state sequences $Q$. A useful approximation to the total likelihood is the likelihood of the most likely state sequence. This quantity can be computed very efficiently using a dynamic programming technique called the Viterbi Algorithm. The Viterbi Algorithm assigns an observation sequence to a state sequence by considering one observation sample at a time and choosing the maximum likelihood path.[13, p. 48]

Now that the acoustic model has been formulated we can move onto the language model.

**The Language Model**

The language modeling problem involves the determination of $P(W)$, or the probability of an observed word sequence $W = w_1w_2...w_p$. We start by representing the sample space of all possible word sequences by an $M$-ary tree, where $M$ is the size of the vocabulary. An example is shown in Figure 2-3.

A word is represented by a tree node and it can be followed by any other word. The
root node marks the beginning of the sentence. The inter-word transitions are denoted
by arcs and, unlike HMM transitions, are dependent on the prior word sequence from
the root node to the active node. Our tree-based model is justified considering that
\( P(W) \) can be expressed as

\[
P(w_1w_2...w_P) = P(w_1)P(w_2 | w_1)P(w_3 | w_1w_2)...P(w_P | w_1w_2...w_{P-1}). \tag{2.5}
\]

Essentially, we have constructed a tree-shaped FSN. So, after assigning each factor to
the appropriate arc we can compute \( P(W) \) by multiplying the transition probabilities
associated with the path corresponding to \( W \).

Practically, this construction is not feasible because it implies the estimation of
conditional probabilities for a wide range of possible word combinations and sequence
lengths. Estimation of such a large number of probabilities would inevitably suffer
from lack of training data. One solution is to use the approximation

\[
P(w_i | w_1w_2...w_{i-1}) \approx P(w_i | w_{i-N+1}...w_{i-1}) \tag{2.6}
\]

In other words, an inter-word transition depends only on the prior \( N - 1 \) words.
This simplification leads to the so-called \( N \)-gram grammars. Two common choices
are the bigram (\( N=2 \)) and trigram (\( N=3 \)). If \( N = 1 \) there is no dependence on the
prior sequence, so this case corresponds to a no-grammar or unigram grammar.\[10, \ p. \ 447\]

Knowing how a particular language model compares to others is just as important
as being able to construct it. It is therefore appropriate at this point to introduce the
concept of \textit{perplexity}. Perplexity is an objective measure of language model complex-
ity used extensively in the development process. It represents the average number of
candidate words that the recognizer must consider solely on the basis of their acous-
tic likelihood. In other words, perplexity rates the ability of the language model to
eliminate unlikely words from consideration; one can think of it as a branching fac-
tor. Higher perplexities mean greater complexity and, in general, poorer recognition
performance. The perplexity, \( P_e \), is defined as,
\[ P_e = 2^{H_e}, \]

where
\[ H_e = \frac{\sum_k P(S_k)}{\sum_k |S_k|} \]  

(2.7)

\( H_e \) is the entropy, or average number of bits, needed to encode the branching factor. \( |S_k| \) is the length of the \( k \)-th sentence. \( P(S_k) \) is the probability that the language model will produce sentence \( S_k \). Perplexity is useful when developing a language model as we shall see in the next chapter.[14]

Thus far, we have modeled a statistical grammar and presented a measure of its complexity. The grammar uses a tree-shaped FSN in much the same way we modeled speech production using HMMs. As seen shortly, this similarity facilitates the merging of the two models during recognition.

The Recognition Process

So far, we have developed approximations to the acoustic and language models of Equation 2.3. We now address the problem of practically integrating the two models during recognition.

Equation 2.3 is an MAP decision rule which requires considering the acoustic and language likelihoods of the set \( S \) of all possible word sequences of finite (but arbitrary) length. This is much too expensive from a computation standpoint, so a recognition approach which only considered a small subset of \( S \) would be desirable. Such an approach should be able to disqualify unlikely sequences while completing likelihood calculation of more likely candidates.

An effective way to approximate the exhaustive search technique exploits the fact that the acoustic and language models both employ the FSN paradigm. To begin, the HMM substitution step used in the acoustic model is applied to the language model word-tree. The result is a monolithic tree of HMMs.

For reasons we will see in a moment, we would like to calculate the total likelihood of a partial word sequence given an observation. To do this we start at the root
node and take the product of the N-gram probabilities and HMM likelihoods (via the Viterbi Algorithm) encountered while charting the appropriate path through the word-tree.

The total likelihood of a partial word sequence gives us a criterion for separating good candidates from bad candidates at each level of the word-tree. So, at each level of the word-tree we consider all possible word transitions and reject those candidate sequences which are relatively unlikely. This streamlining of the word sequence search is called pruning. The pruned search is terminated when the last observation sample has been assigned to an HMM state. The chosen sequence is that survivor having the highest likelihood.

Parameter Estimation

We have assumed thus far that all transitions, output distributions, and N-grams were exactly known. In reality, however, they are not and must be estimated. The most common means for estimating transitions and distributions is the so-called Baum-Welch method named after its inventors.[2] Essentially, the algorithm iteratively computes maximum likelihood estimates which are based on sample means and variances appearing in a training data set (consisting of parameterized speech). It can be shown that this method is equivalent to maximizing the likelihood that a given model would produce the corresponding training data.[10, 7]

A widely used method of effecting a good discrimination/trainability trade-off is that of parameter tying (or clustering). Parameter tying involves the sharing of parameters between models. By forcing two models to share an output distribution, for example, this method effectively increases the amount of training data per parameter; this obviously boosts the reliability of the estimate. It is clear, however, that clustering cannot be performed indiscriminantly since the merging of two completely different monophones, say s and m, would result in a loss of model validity. For this reason, the decision whether to tie two parameters is usually based on some measure of their similarity. Typically, Euclidean distance measures are used for these purposes. An alternative criterion used to regulate the use of tying uses the so-called
Maximum Mutual Information (MMI) principle. This technique permits tying only in cases where the loss in entropy caused by tying is minimal.

N-gram estimates can be derived from the relative frequencies of N-length sequences appearing in a training (text) corpus. In instances where a particular N-gram is sufficiently scarce it can be helpful to transfer probability mass from the most frequent N-grams to the one in question to maintain robustness. This leads to the so-called back-off N-gram, where the name reflects the understating of frequent N-grams.

Another approach is to estimate specific N-grams by smoothing them with more general N-grams. The motivation is that detailed N-grams are statistically similar to their generalizations, which, in turn, have more reliable frequencies. An example might be the estimation of trigram probabilities based on trigram, bigram, and unigram frequencies.

\[
\hat{P}(w_3 \mid w_1, w_2) = p_1 \frac{F(w_1, w_2, w_3)}{F(w_1, w_2)} + p_2 \frac{F(w_1, w_2)}{F(w_1)} + p_3 \frac{F(w_1)}{\sum F(w_1)} \quad [10, p.448] \quad (2.8)
\]

Alternatively, the generalized N-gram sets used for averaging can also be based on grammatical rules as shown by Jelinek.[3] To illustrate, a detailed N-gram might be THE-HOUSE. One of its generalizations might be (ARTICLE)-HOUSE, where (ARTICLE) denotes any article. For the remainder of this paper we will refer to this as a POS, or part-of-speech, bigram.

This modeling technique tries to balance the discrimination/trainability trade-off. In the literature it is referred to as parameter smoothing and has been applied to several other problems in speech recognition.

An alternative which shares some elements of the first two approaches averages the detailed N-grams from the training corpus with the detailed N-grams of a larger, more reliable one.

This concludes our outline of current speech recognition techniques. At best, it is a cursory treatment of an intricate estimation theory and should be regarded only as suggestive of contemporary system requirements.
2.2 Related Work

In this section, we introduce two projects that are related to, but not part of, the principle work under discussion.

2.2.1 Parameter Interpolation

As discussed in the speech recognition primer, it is often useful to smooth bigram estimates by taking a linear combination of two or more different distributions. Such an approach is referred to as parameter interpolation.

We chose to consider using parameter interpolation to synthesize an OR language model because this method has been successful in prior work. Preliminary results, however, revealed that in actuality this technique was not appropriate for the OR task domain. Nevertheless, these results possess diagnostic value and are, therefore, interesting in their own right. We now present the theory of parameter interpolation, followed by a discussion of its application to an OR language model.

Parameter interpolation is a way to generate a hybrid probability distribution from an optimal combination of two or more component distributions; the combination is usually linear. The optimality criterion used is often based on the Maximum Likelihood (ML) concept. ML was discussed earlier in the context of Viterbi decoding, a classification problem. It has also been applied to the estimation of probability distribution parameters or parameter estimation. Parameter interpolation is one of several approaches to parameter estimation which relies on the ML concept. Since ML is so central to this technique, it is appropriate at this point to present its fundamental principles in the context of parameter estimation.

ML, popularized by R.A. Fisher, is a method of estimating probability distributions that optimally matches an assumed distribution to a sample of observed data. To use this method, one firsts assumes that the random variable under study, a bigram for instance, can be described by a distribution having an explicit functional form, within some parameter vector $\lambda$. Formally, we assert that a random variable $x$ has a probability density $P_x(x, \lambda)$. The idea is to use observed data to estimate $\lambda$. 

21
Assuming statistical independence of observations, the likelihood of an observed data set \( \tilde{y} = \{y_1, y_2, \ldots, y_n\} \) is simply a product of individual likelihoods. That is,

\[
\prod_{i=1}^{N} P_x(y_i, \tilde{\lambda}) \overset{\Delta}{=} L(\tilde{\lambda})
\]  

(2.9)

In the literature \( L(\tilde{\lambda}) \) is referred to as the likelihood function. The ML method prescribes that one choose \( \tilde{\lambda}_{opt} \) such that \( \tilde{\lambda}_{opt} = \underset{\lambda}{\text{argmax}} \ L(\tilde{\lambda}) \). From an intuitive standpoint, this is a logical choice for \( \tilde{\lambda}_{opt} \) since the likelihood of an observed data set should be greater than that of an unobserved one (whose likelihood is not available for comparison). In practice, the logarithm of the likelihood is used instead, which simplifies the maximization procedure. A classic example of ML is the estimation of the Gaussian distribution parameters \( \tilde{\lambda} = \{\mu, \sigma^2\} \), which are the mean and variance. Performing the above maximization results in choosing \( \mu \) and \( \sigma^2 \) to be the sample mean and variance respectively.

Interpolation is a scheme in which two or more distributions are combined to form a new, hybrid distribution. When the combination is linear, the weights of the component distributions are left as parameters to be estimated. That is, the hybrid distribution is defined as

\[
P_x(x, \tilde{\lambda}) = \sum_{k=1}^{M} \lambda_k P_{x,k}(x)
\]

(2.10)

where \( M \) is the number of components used. The \( \lambda_k \) are constrained so that \( \sum_{k=1}^{M} \lambda_k = 1 \) and \( \lambda_k > 0 \). In the discrete case this ensures that \( \sum_x P_x(x) = 1 \). In the special case where the \( P_{x,k}(x) \) are identical this constraint dictates that \( P_x(x) = P_{x,k}(x) \).

With this formulation one can preferentially weight component distributions according to how much they increase the likelihood of the data.

Combining relations 2.9 and 2.10 yields an expression for the likelihood of sample data \( \tilde{y} \), assuming an interpolated distribution for the random variable \( x \).

\[
\prod_{i=1}^{N} \sum_{k=1}^{M} \lambda_k P_{x,k}(y_i) \overset{\Delta}{=} L(\tilde{\lambda})
\]

(2.11)

Thus, an interpolated distribution yields a particular instance of the \( L(\tilde{y}) \) defined
in 2.9, if we assign the distribution weights to the parameter vector \( \tilde{\lambda} \). So, we may use the same technique used in general ML estimation to determine the optimal distribution weights.

It is often the case that the component distributions are derived from the same sample data that is used to calculate \( L(\tilde{y}) \). For example, when estimating bigram probabilities by interpolating relative frequencies of bigrams and unigrams appearing in sample text, the same text is then used to generate \( L(\tilde{\lambda}) \).

Deleted Interpolation (DI), popularized in the speech community by Jelinek[3], is a method of allowing the \( \tilde{\lambda} \) adjustment process to compensate for unseen data, while retaining the benefits of preferential weighting. DI is performed by first dividing the sample data \( \tilde{y} \) into \( B \) blocks. The likelihood of the deleted block \( q \) is defined as

\[
\prod_{i=1}^{N} \sum_{k=1}^{M} \lambda_k P_{x,k}^{(q)}(y_i) \triangleq L^{(q)}(\tilde{\lambda}),
\]

where \((q)\) indicates that all component distributions are derived from the data samples in the other \( B - 1 \) blocks collectively. So, in the case of interpolated bigrams, relative frequencies from the \( B - 1 \) other blocks are used to generate component distributions appearing in the likelihood of the \( q \)-th block; this block is termed deleted, since it is not used to derive component distributions.

The total likelihood is just the product of deleted likelihoods over all blocks \( L(\tilde{\lambda}) = \prod_{q=1}^{B} L^{(q)}(\tilde{\lambda}) \), where \( L^{(q)}(\tilde{\lambda}) \) is as defined in 2.12. This complementary treatment of blocks is based on cross-validation, a statistical concept. It aims to govern distribution weighting such that the ability of component distributions to characterize independent data is taken into account.

As before, the weighting vector \( \tilde{\lambda} \) is chosen by maximization of \( L(\tilde{\lambda}) \). We see that the prediction property is encapsulated in this choice of \( \tilde{\lambda} \), since each deleted likelihood can be viewed as the best prediction of the deleted block based on the other \( B - 1 \) blocks.

We now move on to the topic of using deleted interpolation to estimate bigrams. Let us denote a word pair as \( w = \{w_1, w_2\} \), where \( w_2 \) follows \( w_1 \). Recall that a
bigram grammar associates each word pair occurrence with a probability $Pr(w_2 \mid w_1)$. That is, given that the last word was $w_1$, $Pr(w_2 \mid w_1)$ is the probability that the next word will be $w_2$. This is a conditional probability. When we allow $w_2$ to be any word from a finite vocabulary, we are left with a conditional distribution, $P(w_2 \mid w_1)$. To model all possible word sequences with a bigram grammar such a distribution must be estimated for each prior $w_1$.

We decided to construct the estimated distributions from the interpolation of bigram, unigram, and POS bigram frequencies found in the text corpus. The interpolation weights were determined using DI. Applying DI to a text corpus is for the most part straightforward. There is, however, a practical issue that must be addressed when calculating likelihoods.

The key observation is that the total likelihood of the data is not simply the product of individual likelihoods pertaining to independent events. Word events appearing in the same sentence, for example, are highly correlated; this is, after all, what we try to exploit in using statistical grammars.

To simplify things, let us first consider the likelihood of a single sentence. Fortunately, we have already seen it in equation 2.5. It is the “telescoping” product of conditional probabilities that is used in Viterbi decoding. Remember that we approximated each factor by an N-gram probability. Jelinek has shown that such an approximation can also be used successfully in calculating the likelihood of a sentence observed in a text corpus.

In our case the N-grams are a linear combination of bigrams, unigrams, and POS bigrams, whose weights make up the same vector $\lambda$ that we are trying to estimate. So, the approximate likelihood of a single observed sentence is the product of interpolated 2-grams, which are parameterized by $\lambda$. We then assume that 2-grams found in different sentences are independent. This is a good assumption, since the grammar of one sentence should not, in general, be strongly related to the grammar of another. To construct the total likelihood of the data, then, we take the product of likelihoods for each observed sentence. We are left with a function of $\lambda$, $L(\lambda)$, which can be maximized to find $\lambda_{opt}$.
<table>
<thead>
<tr>
<th># Blocks</th>
<th>$\log L(\lambda_{opt})$</th>
<th>$\lambda_{ugm}$</th>
<th>$\lambda_{bgm}$</th>
<th>$\lambda_{pos}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-659.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>-1855.2</td>
<td>0.38</td>
<td>0.32</td>
<td>0.30</td>
</tr>
<tr>
<td>3</td>
<td>-1928.4</td>
<td>0.38</td>
<td>0.33</td>
<td>0.29</td>
</tr>
<tr>
<td>4</td>
<td>-1922.3</td>
<td>0.41</td>
<td>0.28</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Table 2.1: Maximum likelihoods and $\lambda_{opt}$ for different block sizes when performing DI on the OR text corpus.

We applied this algorithm to the OR corpus by tabulating the necessary data, determining the likelihood function, and maximizing it to find $\lambda_{opt}$. We considered four different data partitionings corresponding to dividing the data into one, two, three, and four blocks, where the number of blocks is the same $B$ mentioned above. Notice that the single block case reduces to classic ML estimation; there are no deleted blocks.

The results are shown in Table 2.1. As expected for the ML case ($B = 1$), the bigram relative frequency estimator received the maximum weight of unity. This agrees with the fact that relative frequencies are, in general, maximum likelihood estimates. For deleted interpolation ($B > 1$) we see that the unigram receives the most weight in all cases. This is consistent with the fact that unigrams appear in significantly higher numbers than bigrams and are, therefore, more reliable estimators. POS and standard bigrams seem to be slightly less significant. Based on these results, we concluded that the inclusion of POS bigrams in language model estimation would not result in any significant improvements over the back-off bigram language model, which incorporates both bigrams and unigrams.

### 2.2.2 Previous Live Input System

The original live input module used in the Translator’s Aid system used a modified version of HTK™ release 1.4A.¹ Specifically, modifications were made to an

---

¹HTK™ is a software environment for developing speech recognition systems; it will be introduced in the next chapter.
HTK® recognizing tool called HVite. The modifications allowed custom parameterization in the form of mel filter banks, which at the time, was an important feature to the original designers.

Control of the audio device was transferred from HVite to auxiliary modules. Requests to start/stop record were initiated via file creations. For example, the auxiliary modules directly controlling the audio device would periodically check for the existence of io.go and io.stop files. io.go indicated that the audio device should begin reading audio data into its buffers, while io.stop meant that it should stop reading data.

At a higher level, the files were created within the GUI in response to user requests to start/stop recording; requests were initiated by the user pressing the appropriate button on the GUI.

The buffered data was sent through a pipe to a mel filter bank module whose output was sent to the customized version of HVite which was altered to accept standard input.

The speech recognition development for the current project relied on the most recent version of HTK® (release 2.1). As such, it was necessary to use HVite release 2.1 to implement a live input recognition system. Enabling the Translator’s Aid System to accommodate the latest HTK® formats was only a minor benefit, however. The major advantage in upgrading to release 2.1 stems from the versatility possessed by HVite in controlling the audio device. We will revisit this recording issue in the next chapter.

Fortunately, custom parameterization was not an issue, since all development used native HTK® parameter types. For that reason, we were able to use HVite (release 2.1) without modification.
Chapter 3

Design

3.1 HTK\textsuperscript{TM} Development Environment

As mentioned earlier, the OR corpus is a set of text sentences which define a domain-specific vocabulary and grammar. In cases such as this, where the corpus includes no acoustic data, it is customary to train the acoustic models using a standard database and then use the text corpus to estimate parameters of the language model. This process is enabled by a software-based development tool called the Hidden Markov Model Toolkit\textsuperscript{TM}(HTK\textsuperscript{TM}). HTK\textsuperscript{TM} was developed at Cambridge University’s Speech Vision and Robotics Group and is commercially distributed by Entropic Research Laboratory, Inc. It provides all the computational features needed to synthesize continuous and discrete density recognition systems. For instance, HTK\textsuperscript{TM} offers ready-to-use programs which perform Baum-Welch estimation and Viterbi search. In addition, the toolkit can calculate bigram statistics needed in the language model.

3.1.1 Data Preparation

It should be emphasized that HTK\textsuperscript{TM} relies on externally supplied speech data to perform training and testing. For the development of acoustic models we have been using the Wall Street Journal (WSJ) corpus prepared by the Linguistic Data Consortium (LDC). The WSJ training data that we have used consists of about 30,000 utterances
covering 200 speakers along with word transcriptions. All utterances were recorded using Seinheiser noise-cancelling microphones at a 16 kHz sampling rate.

Recall that a pronouncing dictionary is required to expand words into their equivalent HMM chains. The pronouncing dictionaries we used were generated at LDC and Carnegie Mellon University. Neither dictionary could fully cover the entire vocabulary appearing in the WSJ corpus; this created the need for both. The resulting composite dictionary used a 39 phoneme set and covered nearly all the words appearing in the WSJ utterances we used. The residual entries were entered by hand.

The next step in the data preparation process involves parameterizing the recordings followed by compressing and storing the parameterized data using the appropriate HTK™ tools. The ASCII files containing the word transcriptions can also be converted to the HTK™ specific file format at this point; this is accomplished with shell scripts.

It should be noted that WSJ transcriptions are word-level and not phone-level; time-alignment information was not available. So, we were forced to use the so-called flat-start approach. Time-alignment is usually performed by a speech expert during corpus development. An attempt is made to identify the approximate time intervals corresponding to each phone appearing in the utterance. This interval information is then included in each transcription. Time-alignment must be done by hand and is very tedious. As such, it is common to omit them in large corpuses such as WSJ. In these cases all phones are assumed to have equal duration. This has the consequence of initially assigning equal shares of acoustic observations to each phone in an utterance. Such an approach is called a flat-start. Lee[4] has shown experimentally that the uniform intervals used in a flat-start will eventually converge to their true ones during Baum-Welch reestimation.

### 3.1.2 Acoustic Model Training

The HTK™ development philosophy is based on refining an evolving set of models incrementally. So, the development cycle is one of testing the current set, altering the architecture to balance the discrimination/trainability trade-off, and testing the
new set to gauge improvement. With this approach in mind, it is convenient to start with a set of monophone models which, after sufficient training, can be transformed into context-dependent triphones.

Monophone training begins by selecting a prototype HMM from which to generate models for the set of phonemes used. This is done by creating an HTK\textsuperscript{TM} format HMM file. HMM files contain specification of the parameterization expected by the model (cepstral coefficients, LPC coefficients, etc.), the number of states, the transition probability matrix, and initial statistics. We should note at this point that all of the model sets reported in this paper accept mel frequency cepstra as parameters, including difference and acceleration coefficients.

Once this has been done, the global statistics of the speech data replace the initial statistics resulting in a new prototype which is then replicated to create each initial monophone model. A silence model is also created at this point. Its purpose is to model the silences which appear at the start and finish of most utterances.

The word-level transcriptions are expanded using the pronouncing dictionary to generate phone-level transcriptions, which are then used in Baum-Welch reestimation; silence symbols are inserted in the beginning and end of each transcription. After several iterations of Baum-Welch a single-state short-pause model is created to model brief silences between words and short-pause symbols are inserted at the end of each word in every transcription.

After several more iterations of Baum-Welch the monophones models are ready to be tested and cloned to generate context-dependent triphones. To facilitate triphone training, monophone transcriptions are converted to triphone transcriptions. This is followed by another set of Baum-Welch reestimation iterations.

At this point a set of well-trained triphones has been generated. While these triphones cover all of the phonetic combinations appearing in the WSJ training data they do not in general comprise all of the triphones appearing in the application’s grammar. These unseen triphones are determined by expanding the word transcriptions of the application corpus to their triphone equivalent and recording all new triphones encountered.
The existence of unseen triphones is a common problem in training. The consequence is that parameters belonging to the unseen triphones must be pooled with those belonging to triphones that are seen in the training data. The standard method of accomplishing this pooling within HTK™ is by way of tree-based clustering.

Tree-based clustering is a method of merging model states according to phonetic similarity. Merging is controlled by using phonetic decision trees. Phonetic decision trees are a means of classifying a set of context-dependent models. Each node in the tree is assigned a phonetic question such as “is the left context a nasal?” or “is the right context a glottal stop?”. The questions always have yes/no answers, so the tree is binary in structure. To classify a state one starts at the root node and traverses the tree by applying each question encountered to the given model and taking the appropriate branch. When a terminal node is reached, the given state is merged with any other states sharing the same terminal node.

The trees are constructed in a self-organizing manner. The user presents a list of candidate questions to the clustering tool and it assigns questions to nodes such that the likelihood of the data is maximized. The states of models seen in the training corpus are used to calculate this likelihood. The tree is extended until the increase in likelihood achieved with a branch is less than a user specified value. Then, when states of unseen models are sent through the tree, they are assured to be clustered with those of seen ones. After several more iterations of Baum-Welch a set of baseline triphones are ready for testing.

In this project we considered three variations on the baseline set: 1) cross-word triphones, 2) pseudo-cross-word triphones, and 3) function-word triphones. Pseudo-cross-word triphones are generated by first determining the cross-word triphones appearing in the application corpus. Those triphones which also happen to appear in the WSJ word-internal triphone list can be modelled as such. The residual is modelled via tree-based clustering. So, they are, in effect, cross-words trained as word-internals. The advantage of using pseudo-cross-words is that unlike full-blown cross-words they do not substantially increase the number of models encountered in training, and therefore, maintain model robustness.
We used K.F. Lee's 42 function-word set that was used in CMU's SPHINX system. [6] The modifications necessary to accommodate function-words consisted of modifying the pronouncing dictionary and assigning an HMM to each function word. We considered three-state, six-state, and nine-state topologies. The longer topologies were considered based on the logical assumption that models representing multiple phonemes have longer duration, and therefore, consume more acoustic observations than monophones. An evaluation of all the model sets just described is included in the next chapter.

3.1.3 Language Model Training

As discussed earlier in the section on related work, parameter interpolation using POS bigrams was explored as a way to construct a reliable language model. Preliminary results indicated, however, that such an approach would not yield the advantages expected. Henceforth, we relied exclusively on the back-off bigram, whose parameters are readily computed using HTK\textsuperscript{TM} tools.

The formula HTK\textsuperscript{TM} uses to estimate bigram probabilities is as follows,

\[
P(j|i) = \begin{cases} 
(N(j|i) - D)/N(i), & \text{if } N(i,j) > t \\
 b(i)p(j), & \text{otherwise.}
\end{cases}
\]

where

\[
p(j) = \begin{cases} 
N(i)/N, & \text{if } N(i) > u \\
u/N, & \text{otherwise.}
\end{cases}
\]

and

\[N = \sum_{i=1}^{L} \max[N(i), u].[14]\]

Probability mass is transferred from the high occurring bigrams to the low occurring ones. Low occurring bigrams probabilities are backed-off to unigram frequencies. In some sense, this is a form of nonlinear smoothing. \(N(j|i)\) is the number of times word \(j\) follows word \(i\). \(N(i)\) is simply the number of times word \(i\) appears in the corpus. The \(b(i)\) factor insures that bigram probabilities sharing the same word history
i sum to unity; it can also be seen as a scaling operation that emphasizes infrequent bigrams. The other quantities in the relation, \( D, t, \) and \( u \), are the only free parameters. The discount parameter \( D \) specifies how much probability mass will be extracted from frequent bigrams. The threshold \( t \) differentiates between frequent and infrequent bigrams, while the threshold \( u \) sets a floor on the minimum unigram count to avoid zero probabilities. Both parameters are user specified; their optimum values are dependent on the application corpus and must be determined empirically. Values were chosen to minimize the perplexity of the resulting language model. The perplexity was calculated using the HTK\textsuperscript{TM} tool \texttt{HSGen}.

First, \texttt{HSGen} randomly generates a set of sentences according to the statistics of a given language model. The perplexity of this set is then computed. Since the sentences are generated randomly the perplexity will vary with each call to \texttt{HSGen}. The variance can be kept small, however, by configuring the tool to generate a large number of sentences; large sample sizes give low variances. We found that a variance under 3% of the sample mean was achieved when sampling 1,000 sentences. This yielded acceptable estimates of the true value.

For the OR corpus we found the best values to be \( D = 0, t = 0, \) and \( u = 1 \). The resulting perplexity is about 4. The conclusions one can draw from this are as follows: 1) the differences between the highest and lowest frequencies were not sufficient to warrant discounting, 2) if a bigram appeared at least once it is best to use its relative frequency rather than backing-off to a unigram frequency, and 3) the bigram probability floor should be no less than \( \frac{1}{V} \), the a priori probability of a given word, where \( V \) is the vocabulary size. Backing-off to unigram frequencies is only beneficial in cases where there are no bigram frequencies.

Often, minimizing the perplexity of a grammar results in a highly constrained language model which lacks the ability to predict new data. In cases where a less constrained grammar is desired, the HTK\textsuperscript{TM} recognizer can be invoked with an optional grammar scale factor \( s \). During the decoding process all language model probabilities will be raised to the power \( s \), allowing one to de-emphasize the language model to an arbitrary degree; the exponent operation non-linearly warps probabilities so that they
Figure 3-1: Nonlinear mapping of probabilities when scale factor $s$ is applied to (log) probabilities.

become closer in value.

When $s$ is greater than unity probabilities are *squeezed* toward zero, whereas when $s$ is less than unity they are *squeezed* toward one. When language model probabilities get closer in value their impact on decoding is lessened. The drawback of this technique, not surprisingly, is an increased perplexity. Figure 3-1 depicts this nonlinear mapping.

Having introduced the model development process, we are ready to discuss the integration of an HTK™ recognition kernel into the GUI to enable recognition of live speech input.

### 3.2 Enabling Live Input

So far, we have discussed the technique used to synthesize an HMM set for the OR task domain. We now move on to address the issue of integrating the set into a live audio input system.

As mentioned earlier, arriving at the desired model set for an application involves an iterative process of model adjustment followed by model evaluation. The later step is a means of deciding which combination of adjustments (e.g. the number of Gaussian mixtures to use and the degree of state clustering) yield the best performance. HTK™ provides a shell tool called HVite which performs Viterbi decoding
on a set of input speech files. This off-line decoding is useful in testing a sequence of model sets using a fixed set of speech files.

Alternatively, HVite can also control the audio device of its host to allow direct audio input. Thus, it serves as a useful building block for live-input applications. Via HTK\textsuperscript{TM}'s standard configuration files HVite can be configured to accept microphone or line input and speaker, line, or jack output. The main issue in controlling HVite is choosing the method of starting and stopping the record operation. HVite offers three ways to control recording which may be chosen via the configuration file. One method is key-press control, where the user can start and stop recording by pressing a specified key. Another employs a novel speech detector which intelligently records live speech. The last method relies on inter-process signalling to start and stop recording. For this project inter-process signalling was the most appropriate since it could be easily integrated into the existing graphical user interface.

The recognizer output is sent to standard output (usually a terminal) and can optionally be written to external files. For this project we chose to copy the standard output to a file since its format was compact and more importantly, indicated the state of the recognizer.

The simplest way to run HVite with inter-process signalling is to arrange for a parent process to spawn a child which, in turn, invokes HVite as a shell command. In our case, the GUI is the natural choice for the parent. The GUI can send signals to HVite when processing user requests to start/stop recording.

Signalling is allowed to be asynchronous, complying with a user’s general desire to record at will. There are distinct periods, however, when HVite does not expect, and cannot handle, start/stop signals. Usually, sending untimely signals to such a process will produce unpredictable, and often, undesirable results (e.g. killing the receiving process). This was a design decision imposed by the authors of HVite and warrants the attention of the application developer using the program in direct audio mode. We now describe HVite’s direct audio functionality in more detail.

Once invoked from a shell, HVite commences an initialization stage in which the recognizer loads model sets, pronouncing dictionaries, and word networks, as well as
setting several parameters. It should be noted that initialization is performed once only. During this time user signals are not handled explicitly and must be blocked by interface software. On completion, the recognizer waits for a signal to begin recording followed by another to cease recording. Next, HVite begins recognizing and no longer handles user signals explicitly. Again, user signals must be blocked at these times for the reasons mentioned above. At this point, HVite has completed the cycle and waits for a user signal to start recording again.

Because timing is critical when sending signals to HVite, it is convenient to treat the recognizer as a state machine; the recognizer responds to user signals in a manner that depends on its current state. A logical choice for recognizer states is 1) off, 2) initializing, 3) waiting, 4) recording, and 5) recognizing. Figure 3-2 shows the associated state diagram. We prohibit user signalling in states 1, 2, and 5 for reasons just mentioned.

Signalling is unidirectional. That means that HVite does not inform its parent
process as to its current state. The state must, therefore, be inferred from the standard output. It was mentioned above that HVite records its activity on standard output; it is a way of prompting the user for new audio input as well as displaying recognition results. This implies that standard output may be scanned to give not only recognition output, but also the current state of the recognizer. When the user elicits key actions (asynchronously), the state is updated automatically, based on standard output.

We decided to encapsulate all state bookkeeping and low-level signalling into an abstraction layer appropriately named hvite-interface. The interface consists of five functions which enable the caller to query recognition status and initiate record signals safely. Their functionality is expained briefly below. See synopsis and description in Appendix A.

startHVite takes the recognizer from the off to the initializing state. It is responsible for spawning a child process which, in turn, runs the HVite program. The child process invokes HVite in such a way that the program’s standard output is copied to a log file. This file is then used by the other interface functions to update the recognizer state so that subsequent user requests are processed appropriately; updating state during user calls means state is updated asynchronously.

cleanupHVite brings the recognizer to the off state. This function kills the child process running HVite.

signalHVite services a user request to signal HVite. If the recognizer is in the off, initializing, or recognizing states, this function takes no action and returns with a negative status. When in the waiting or recording states a signal is sent to HVite, which starts or stops record respectively. To ascertain the current state of the recognizer it checks the log file.

It should be noted that HVite cannot handle arbitrary length recordings; the record process relies on limited memory resources. This makes it necessary to limit recording duration. To deal with this issue an optional timeout capability can be added to hvite-interface during compilation. It allows recording duration to be limited to a specified number of seconds. This is accomplished by requesting the operating
system to send an alarm signal to the calling process after the desired period; this request is made whenever recording is started.

Alternatively, timeout capability can be implemented by the user. This is useful when hvite-interface is used within an event-driven environment, where some standard system calls can create conflicts. In this project, the GUI which accessed hvite-interface is an X Windows\textsuperscript{TM} application; so, timeout capability was implemented within the GUI.

\texttt{getRecogOut} If recognition output is ready, this function copies the current output to a user-specified array. If no output is available, then it returns with a negative status.

\texttt{clrRecogOut} effectively clears the last recognition output read from the log file. By calling this function after every successful call to \texttt{getRecogOut}, one ensures that future calls to \texttt{getRecogOut} will return new output only.

Figure 3-3 presents a flow diagram which demonstrates how these five functions are used in a live-input application.

HVite-interface is a modular interface to the recognizing function of HVite that is simple to use within a user interface environment. Although it accomplishes the necessary tasks, it is rather inefficient as far as CPU time, memory requirements, and process activity are concerned. In Chapter 5, we propose a better solution that should alleviate these inefficiencies.
Figure 3-3: hvite-interface flow diagram
Chapter 4

Results

This chapter is structured as follows. First, we discuss our evaluation methods. We then move onto the recognition performance of the monophone set, baseline triphone set, and three triphone variations; all five sets use single-mixture Gaussian densities.

4.1 Methods of Evaluation

The language model used in evaluation is the same back-off bigram model mentioned in the previous chapter. The test utterances were generated by reading sentences taken from the OR corpus. There were 62 sentences, covering 680 words, and each was read once by a single test speaker. The test speaker did not participate in the recording of the WSJ corpus which was used to train acoustic models. Therefore, the results for this test set give a good indication of the extent to which each system is speaker-independent.

Even so, all the text used in testing was also used to generate the language model. So, from a language model standpoint, this is not a fair test, in which test data is independent of training data. The decision to employ a subjective test arose from the fact that the 111 training sentences were not accompanied by a dedicated test set, making it the only alternative.

Dividing the corpus into a training and test set was considered as a solution to this problem. This approach only introduced a new problem, however; due to the
paucity of text data we could not find a training set which included examples of all
the words in the vocabulary. Unseen data is inevitable when trying to partition such
a small corpus into training and test sets; it also leads to poor language models.

The objective measures used for evaluating each model set is word accuracy. This
quantity is computed by comparing the recognized transcriptions to the true transcrip-
tions. Before making this comparison, however, it is necessary to align the true and
recognized transcriptions. This is accomplished by a dynamic programming string-
alignment algorithm which tries to line-up words appearing in both transcriptions.
The alignment makes it easier to distinguish between word-insertion, word-deletion
errors, and word-substitution errors.

Word accuracy is then, the ratio of the number of correctly recognized words, less
insertions, to the number of words in the test transcription. That is,

\[ A_w = \frac{R - I}{N}, \]  

(4.1)

where \( R \) is the number of recognized words, \( I \) the number of insertions, and \( N \)
the number of words in the test transcription. Sentence accuracy is also useful for
assessing performance. It is the ratio of the number of correct sentences to the total
number of sentences, where a sentence is considered correct only if all words in the
sentence are recognized correctly. There is an HTK\textsuperscript{TM} tool called \texttt{HResults}
which computes both of these statistics, along with insertion, deletion, and substitution
counts.

### 4.2 Results Presented

The results are shown in Table 4.1. The last three columns indicate the number of
insertions, deletions, and substitutions, respectively. First of all, we see that there is
considerable improvement when moving from monophones to word-internal triphones.
This was expected, since the triphones are better at acoustic discrimination.

There is little gain from switching to word-internal function-word triphones from
plain word-internal ones. The function-word set only saved two insertions, both of
<table>
<thead>
<tr>
<th>Model Set</th>
<th>Word Accuracy</th>
<th>Sentence Accuracy</th>
<th>I</th>
<th>D</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>monophone</td>
<td>94.4%</td>
<td>79.0%</td>
<td>19</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>baseline</td>
<td>96.3%</td>
<td>85.5%</td>
<td>12</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>X-word</td>
<td>92.8%</td>
<td>85.5%</td>
<td>36</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>pseudo X-word</td>
<td>95.2%</td>
<td>84.8%</td>
<td>13</td>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>function word</td>
<td>96.6%</td>
<td>85.5%</td>
<td>10</td>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 4.1: Model set results. Statistics shown are word-accuracy, sentence-accuracy, insertions (I), deletions (D), and substitutions (S).

<table>
<thead>
<tr>
<th>Model Set</th>
<th># OR models</th>
<th># WSJ models</th>
<th>% OR seen</th>
<th>% OR unseen</th>
</tr>
</thead>
<tbody>
<tr>
<td>monophone</td>
<td>41</td>
<td>41</td>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>baseline</td>
<td>1,273</td>
<td>7,894</td>
<td>86%</td>
<td>14%</td>
</tr>
<tr>
<td>X-word</td>
<td>1,210</td>
<td>17,289</td>
<td>91%</td>
<td>9%</td>
</tr>
<tr>
<td>pseudo X-word</td>
<td>1,210</td>
<td>7,894</td>
<td>58%</td>
<td>42%</td>
</tr>
<tr>
<td>fun. word</td>
<td>1,284</td>
<td>7,933</td>
<td>87%</td>
<td>13%</td>
</tr>
</tbody>
</table>

Table 4.2: Model set statistics.

which, as it turned out, were not even connected with errors associated with function-words. These results agree with the work of Lee[11, pp. 347-365], in which, word accuracy went from 95.1% to 95.2% with the addition of function-word models to a word-internal set.

The results for cross-word triphones, however, are unexpected. True cross-word triphones normally produce better performance than word-internal triphones due to their ability to model coarticulation effects. In a demonstration released with HTK™, for instance, cross-words showed a 2% increase in word accuracy over word-internals, when applied to the Naval Resource Management corpus.¹

### 4.3 Discussion

Interpreting these results is straightforward if we consider the key attributes of the relevant model set. Table 4.2 shows some statistics on model set size and training

¹The Naval Resource Management corpus is a standard database used by ARPA for benchmarking.
coverage which shed some light in this area. The second and third column show the number of triphones used in the OR transcriptions and WSJ training transcriptions. The fourth and fifth columns report the amount of OR triphones seen and unseen in the WSJ training transcriptions as a percentage of the total number of OR triphones.

Notice how the number of WSJ training models doubles when moving from word-internal triphones to cross-word triphones. This doubling implies less data per parameter trained, which in turn, indicates a loss in trainability. It seems as though this loss counteracted any gains in the ability to deal with coarticulation.

Pseudo cross-words clearly performed better than true cross-words. Let us see why. The training coverage is poorer for pseudo cross-words than for true cross-words (58% versus 91%). This implies that significantly fewer triphones could be trained explicitly (not through clustering) in the case of pseudo cross-words. Even so, this loss seems to be offset by the higher degree of trainability held by pseudo cross-words over true cross-words, which relies on twice the number of WSJ training models.

In conclusion, the results show that whether a particular set of models, in practice, exhibit the advantages expected, is highly dependent on the training coverage associated with that model set. Indeed, lack of training coverage results in a loss in model robustness. For this project, function-word and word-internal triphones yielded the best performance. In the final translation system, we chose to use the baseline word-internal set, since it requires less training time and, therefore, facilitates the development of future speech input systems covering new task-domains.
Chapter 5

Future Work

The current system can be improved in the areas of both speech modelling and interfacing to HΤK™. We explore these areas below.

5.1 Speech Modelling

The first aspect of speech modelling which can be improved greatly is evident from Table 4.2. It indicates that in the case of triphones there is a serious lack of training coverage. That is, a considerable percentage of OR triphones never occur in the WSJ training data and must, therefore, be trained implicitly via clustering. Clustering, however, is governed by human phonetic classifications, not by an optimality criterion. So, clustering is, in general, suboptimal and compromises performance; yet it is necessary when there are unseen triphones.

An obvious, though not always tractable, solution would be to acquire an alternative acoustic modelling corpus which offered full training coverage of the OR triphones. A more time-intensive alternative to this, would be to generate one's own acoustic training data. This way, one has complete control over triphone coverage in the training data.

A second aspect to be improved is the development test data. Recall that the test data used for final evaluation was also used for language model training. This data was used for development testing as well, which governed the model adjustment
process. For this reason, the choice of model architecture was profoundly influenced by the level of performance achieved with a training set, which is not necessarily representative of new data.

The need for a test set which is independent of the training set is unavoidable. It is only by using such a test set that model architecture can be chosen to enhance the performance on new data. An easy way of generating test data would be to simply concoct a set of test sentences by hand. These sentences should be produced by someone who is not directly involved in system development, so as to maintain an appropriate level of objectivity. An ideal candidate would be a military officer, or some other potential user of the system.

5.2 Interface to HTK

As mentioned in Chapter 3, hvite–interface is reliable and easy to use, but is also rather inefficient. A better way to access HTK's recognition capabilities within an application, is to use HAPI, HTK's application programmer's interface (API), which recently became available. It provides a complete, seamless interface to all HTK functionality. When recording live-input, there is no need to create auxiliary processes. Also, CPU time is not squandered on standard features, which are found in all of HTK's command-line tools, and are often not needed. An added advantage of using HAPI is that the HTK recognition modules used in a HAPI application can be upgraded without the need to change existing application code.[8] Thus, using HAPI could reduce maintenance requirements as well as improve program efficiency.
Appendix A

Concise hvite-interface Manual

NAME

startHVite, cleanupHVite, signalHVite, getRecogOut, clrRecogOut

SYNOPSIS

#include <hvite-interface.h>

int startHVite(void);
void cleanupHVite(void);
int signalHVite(void);
int getRecogOut(char *str);
void clrRecogOut(void);

DESCRIPTION

HVite-interface is an interface to the HTK™ command-line tool HVite (HTK™ Version 2.1.1 only), which performs Viterbi decoding for speech recognition applications. The purpose of the interface is to allow relatively easy access to HVite in its direct-audio input mode, which enables continuous speech recognition of live-input. In this mode, HVite provides complete audio recording capabilities, so separate recording processes are not necessary. Starting and stopping recording is accomplished by sending a pre-assigned signal to the process running HVite. The interface is responsible for gating user requests to start/stop recording so that signals are sent to HVite only when they will not endanger program integrity.

startHVite() invokes HVite via a fork, stores process ID information, and initializes state. The function returns 0 on success and -1 on failure. Possible reasons for failure are 1) unable to spawn child processes or 2) HVite was already started with a previous call to startHVite(). Also note that multiple instances of the interface are not
allowed to be accessed simultaneously.

cleanupHVite() kills all child processes started by startHVite() and resets state. If HVite is not in use, no actions are taken. This function has no return value.

signalHVite() can be used to request starting or stopping of record. Whether the request is interpreted as a start or stop request depends on the current state of HVite. This function returns 0 on success and -1 on failure. It fails when HVite is not ready to accept control signals or an attempt to signal the program failed.

getRecogOut() copies recognition output, if available, to str and returns with 0 status. If output is not available, this function returns with -1 status.

clrRecogOut() effectively clears the last recognition from interface memory. By calling this function after every successful call to getRecogOut(), one ensures that future calls to getRecogOut() will return new output only.

In normal operation of the interface, first, startHVite() is invoked to start running HVite. signalHVite() is called whenever a start/stop record is desired. getRecogOut() can be used for querying recognition status and/or fetching current recognition output. getRecogOut() will normally appear in a polling routine that periodically checks for new recognition output. In this case, each call to getRecogOut() should be followed by a call to clrRecogOut() to reset the state of the interface. The record/recognize cycle can be repeated indefinitely.
Bibliography


