Ergodic Hidden Markov Models for Visual-Only Isolated Digit Recognition

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Electrical Engineering

By

Louis Terry

EVANSTON, ILLINOIS

June 2007
ABSTRACT

Accurate, robust, automatic speech recognition (ASR) systems represent the cornerstone of advanced human-computer interaction. An inherent asymmetry exists in current human-computer interfaces (HCIs); the human communicates through physical interaction via a keyboard and/or mouse while the computer communicates through visual and auditory channels. This asymmetry prevents human-computer interaction in the familiar, natural style of human-human interaction. With systems able to mimic human-human interaction, technology becomes more natural, free-flowing, and seamlessly integrated into modern life. This broad-reaching goal serves as motivation for this work.

Upon examination of modern speech recognition systems, one finds that no system actually models the speech production phenomenon. These state of the art systems instead model the symbols produced by speech in the form of words or sub-word units. With this realization in mind, this work proposes a system architecture which directly models the speech production phenomenon. As a direct consequence, this approach separates articulation modeling and articulation interpretation.

This work formulates this framework, dubbed the ”Articulation Model” framework, and then compares it against the state of the art speech recognizer using only video cues to recognize isolated digits.
Acknowledgments

I would like to thank my advisor and committee chair, Dr. Aggelos Katsaggelos, for his inspiration, kindness, and guidance. His efforts have greatly helped in every respect of my academic and professional development. I would also like to thank my committee members, Dr. Janet Pierrehumbert and Dr. Ying Wu, for their enthusiasm and guidance as well.

I would like to thank all my labmates for their support and for making each day a fun day to go to work. Without all of them, this work would not be possible: specifically, and in no particular order, Derek Shiell, Petar Aleksic, Peshala Pahalawatta, Xiaonan Zhao, Xiaoming Yin, Sotos Tsaftaris, Dejan Depalov, Andrei Makhanov, Dev Ghosh, Derin Babacan, Fan Jiang, Ehsan Maani, Jana Zujovic, Martin Luessi, and Serhan Uslubas. In particular, I would like to thank Derek Shiell for his many insights and for always being there to bounce ideas off of.

Without the support, guidance, and love of my family, I would not be where I am today. I wish to profoundly thank my mother and father for their boundless love, understanding, and guidance. My father’s perseverance and good humor through his many medical challenges, and my mother’s unwavering support and perseverance have shaped my life. I would also not be who I am today without the love of my sister, which has withstood many, many childhood “pranks”.

This is dedicated to the memory of my father, Alan D. Terry.
Table of Contents

ABSTRACT 3

Acknowledgments 4

List of Figures 7

Chapter 1. Introduction 10

Chapter 2. Background 13

2.1. General automatic speech recognition system overview 13
2.2. Recognizer architecture 17
2.3. Hidden Markov models 18
2.4. Left-to-right hidden Markov models in ASR 26
2.5. Ergodic hidden Markov models in ASR 26

Chapter 3. Recognizer Description 28

3.1. Design motivation 28
3.2. System overview 30
3.3. Articulation model 31
3.4. Path distances 36
3.5. Word Recognition 39

Chapter 4. Experiments and Comparative Results 43
4.1. Database description
4.2. Experimental paradigm
4.3. Baseline system
4.4. Proposed system
4.5. Comparison

Chapter 5. Conclusions
5.1. Future model refinements
5.2. Future model extensions

References

Appendix A. Basic implementation
Appendix B. Voting implementation
Appendix C. Vector implementation
Appendix D. Vector implementation with voting
List of Figures

2.1 Block diagram of general ASR system 13
2.2 Block diagram of traditional word model based system 18
2.3 Four state, three mixture fully connected HMM 19
2.4 Three state, three mixture left-to-right HMM 20
3.1 Proposed system architecture 30
3.2 Traditional system architecture 30
3.3 Six state articulation model 31
3.4 Articulation unit identification and modeling 32
4.1 Sample frames from CMU database 44
4.2 Sample frame with preprocessing results superimposed 44
4.3 Baseline system speaker recognition rates for DCT_Z & DCT_D_A_Z 46
4.4 Baseline system speaker recognition rates for HW_Z & HW_D_A_Z 46
4.5 Basic system speaker performance for DCT_Z & DCT_D_A_Z 48
4.6 Basic system speaker performance for HW_Z & HW_D_A_Z 49
4.7 Voting system speaker performance for DCT_Z & DCT_D_A_Z 49
4.8 Voting system speaker performance for HW_Z & HW_D_A_Z 50
4.9 Vector metric speaker performance rates for DCT_Z & DCT_D_A_Z 50
4.10 Vector metric speaker performance for $HW_Z \& HW_{D_A Z}$

4.11 Vector voting speaker performance rates for $DCT_Z \& DCT_{D_A Z}$

4.12 Vector voting speaker performance for $HW_Z \& HW_{D_A Z}$

4.13 Peak system performances for $DCT_Z$

4.14 Peak system performances for $DCT_{D_A Z}$

4.15 Peak system performances for $HW_Z$

4.16 Peak system performances for $HW_{D_A Z}$

A.1 Basic system performance for $DCT_Z$ with $M = 1$

A.2 Basic system performance for $DCT_Z$ with $M = 2$

A.3 Basic system performance for $DCT_{D_A Z}$ with $M = 1$

A.4 Basic system performance for $DCT_{D_A Z}$ with $M = 2$

A.5 Basic system performance for $HW_Z$ with $M = 1$

A.6 Basic system performance for $HW_Z$ with $M = 2$

A.7 Basic system performance for $HW_{D_A Z}$ with $M = 1$

A.8 Basic system performance for $HW_{D_A Z}$ with $M = 2$

B.1 Voting system performance for $DCT_Z$ with $M = 1$

B.2 Voting system performance for $DCT_Z$ with $M = 2$

B.3 Voting system performance for $DCT_{D_A Z}$ with $M = 1$

B.4 Voting system performance for $DCT_{D_A Z}$ with $M = 2$

B.5 Voting system performance for $HW_Z$ with $M = 1$

B.6 Voting system performance for $HW_Z$ with $M = 2$
B.7 Voting system performance for $HW_{DAZ}$ with $M = 1$

B.8 Voting system performance for $HW_{DAZ}$ with $M = 2$

C.1 Vector metric performance for $DCT_Z$ with $M = 1$, varying $\alpha$

C.2 Vector metric performance for $DCT_Z$ with $M = 2$, varying $\alpha$

C.3 Vector metric performance for $DCT_{DAZ}$ with $M = 1$, varying $\alpha$

C.4 Vector metric performance for $DCT_{DAZ}$ with $M = 2$, varying $\alpha$

C.5 Vector metric performance for $HW_Z$ with $M = 1$, varying $\alpha$

C.6 Vector metric performance for $HW_Z$ with $M = 2$, varying $\alpha$

C.7 Vector metric performance for $HW_{DAZ}$ with $M = 1$, varying $\alpha$

C.8 Vector metric performance for $HW_{DAZ}$ with $M = 2$, varying $\alpha$

D.1 Vector voting performance for $DCT_Z$ with $M = 1$, varying $\alpha$

D.2 Vector voting performance for $DCT_Z$ with $M = 2$, varying $\alpha$

D.3 Vector voting performance for $DCT_{DAZ}$ with $M = 1$, varying $\alpha$

D.4 Vector voting performance for $DCT_{DAZ}$ with $M = 2$, varying $\alpha$

D.5 Vector voting performance for $HW_Z$ with $M = 1$, varying $\alpha$

D.6 Vector voting performance for $HW_Z$ with $M = 2$, varying $\alpha$

D.7 Vector voting performance for $HW_{DAZ}$ with $M = 1$, varying $\alpha$

D.8 Vector voting performance for $HW_{DAZ}$ with $M = 2$, varying $\alpha$
CHAPTER 1

Introduction

Speech recognition technology continues to spread through modern life to a near ubiquituous role. One finds automatic speech recognition in automated telephone systems, automobiles, computer interfaces, and numerous applications in between. As this expansion unfolds, however, the limits of current speech recognition become apparent; modern systems’ performances degrade rapidly in the presence of audio noise [1], and even in clean acoustic environments performance can lag human speech perception by up to an order of magnitude [2]. Only so much signal processing can be done on the incoming audio stream before the returns finally do not outweigh the cost; the need for alternative approaches has become apparent. In recent years visual information has been successfully incorporated into the speech recognition task (for recent reviews, see [3, 4]). While this bimodal approach has seen success, performance still lags behind human perception. An accurate, robust automatic speech recognition system remains elusive.

Such accurate, robust, automatic speech recognition (ASR) systems represent the cornerstone of advanced human-computer interaction. Current human-computer interfaces (HCIs) are inherently unbalanced; the human communicates through physical interaction via a keyboard and/or mouse while the computer communicates through visual and auditory channels. This inherent issue precludes human-computer interaction in the familiar, natural style of human-human interaction. By building systems able to mimic human-human interaction,
technology becomes more natural, free-flowing, and seamlessly integrated into modern life. It is this broad-reaching goal which motivates the development of ASR systems.

Automatic speech recognition systems generally fall into three categories: audio-only ASR (A-ASR), video-only ASR (V-ASR), and bimodal audio-visual ASR (AV-ASR). The development of A-ASR systems began in the 1950s throughout the United States and in England. By the 1960s, A-ASR development spread to Japan and during the 1970s the first limited vocabulary, isolated word recognitions systems became viable. Through the 1980s A-ASR research evolved into limited vocabulary, continuous word recognition (ie., words spoken in a conversational manner without distinct pauses between words) and, by the end of the decade, large vocabulary continuous speech recognition (LVCSR), defined by vocabularies numbering in the thousands of words rather than tens or hundreds [5].

The first foray into V-ASR and AV-ASR occurred in 1984 with Petajan’s doctoral thesis [6]. His system extracted the height, width, perimeter, and area of a speaker’s mouth and used these features as inputs to a speech recognition system for V-ASR. His AV-ASR system utilized the results of the V-ASR system to refine and ultimately improve the results of an A-ASR system.

While the fields of audio-visual and visual-only ASR are over twenty years old, these areas offer abundant research opportunities. Much investigation has been done to determine the best visual features to use, how to extract them, and if, and how, to preprocess those features [3, 4, 7]. In parallel, significant research efforts have gone into how best to fuse the audio and visual information. The commonality throughout nearly all the research in AV and V-only ASR has been the basic formulation of the speech recognizer itself. The recognizer’s architecture comes directly from the A-only ASR development cycle. The best architecture from A-ASR became the de facto architecture for AV and V-only ASR. Audio and video,
however, have vastly different characteristics. By nature, audio is a continuous signal while video is discrete. A single video frame observation can completely capture the state of the speaker’s visible articulators (mouth, lips, jaw, etc) whereas a single audio sample or even audio frame (traditionally 10ms in length) observation does not directly describe the speech articulators. This fundamental difference leads one to question if the best architecture for A-ASR indeed should be the best architecture for AV and V-only ASR.

This thesis explores this architecture question, introduces a previously unused system architecture for V-ASR, and compares it to the traditional system architecture for V-ASR. This thesis focuses on comparisons related to isolated digit recognition using only visual cues. The isolated digit recognition problem represents the basic test of a system and is thus natural for use in comparing architecture performance.

Chapter 2 provides a survey of background information related to the general ASR problem as well as specific issues related to V-ASR. Chapter 3 presents the system architecture and discusses its implementation. Chapter 4 compares the performance of this work to that of a traditional system. Chapter 5 presents conclusions based on this work as well as future applications and extensions.
CHAPTER 2

Background

2.1. General automatic speech recognition system overview

Single modality automatic speech recognition systems all contain the same basic building blocks (shown in Fig. 2.1):

(1) **Input Stream:** This may be audio data, video data, or some other relevant observation data.

(2) **Feature Extraction:** Signal/Video processing takes place to extract salient features from the input stream.

(3) **Recognizer:** A pattern matching mechanism takes the input features and outputs the recognized word.

Multi-modal systems have this same general structure, however the multiple input streams must be fused at some point. Multi-modal fusion represents an extremely active research field in its own right and, as it is not the focus of this thesis, will not be elaborated upon further.

A more thorough examination of the input stream, feature extraction, and recognizer blocks follow in sections 2.1.1, 2.1.2, and 2.1.3 respectively.

![Figure 2.1. Block diagram of general ASR system](image-url)
2.1.1. Block 1: Input stream

The input stream of a unimodal speech recognition system consists of unprocessed sensor measurements. In the case of audio input, the stream consists of microphone measurements sampled in time. For video input, the stream consists of video frames at some frames per second rate. The input stream is a general component and could include of a variety of other relevant information sources such as physical sensor inputs, range finding information, etc.

2.1.2. Block 2: Feature extraction

Feature extraction involves the computation of salient features from the input stream. Depending on the type of input, this process could consist of major input processing or relatively little. Audio and visual feature extraction involves significant processing while physical sensor inputs may require little or none. Sections 2.1.2.1 and 2.1.2.2 explore audio and visual feature extraction in more depth.

2.1.2.1. Audio feature extraction. While the focus of this thesis is on visual-only automatic speech recognition, it would be remiss to not to at least mention audio-only automatic speech recognition. The selection of audio features has been a focal point of speech recognition research since the inception of the field. Most modern audio-only speech recognition systems utilize features derived from experiments in human perception. These audio features, Mel Frequency Cepstral Coefficients (MFCCs), mimic the non-linear nature of human frequency resolution and perception. Naturally, the selection of perceptual features has directed the development of the actual recognizers towards more perceptual implementations.

2.1.2.2. Video feature extraction. Unlike in audio-only automatic speech recognition, there is no accepted ”standard” visual feature. The question of feature selection remains an
open problem in the community. Consequently, the framework presented in this thesis has been formulated independent of the input features.

The analysis of the visual stream typically consists of the detection, tracking, and extraction of the important visual articulators. The extraction of the visual features requires robust face detection and tracking, as well as accurate estimation of a speaker’s mouth location. The detection and tracking algorithms should be able to detect the effects of a possibly non-ideal environment on the visual data, adapt, and reliably locate the visual features of interest. Naturally, the improvement of the ASR performance depends strongly on the accuracy of the visual feature extraction algorithms.

Typical algorithms for face detection and tracking include neural networks [8], template matching, Active Shape/Appearance Models (ASMs / AAMs) [9], and AdaBoosting [10]. Once the face has been located various techniques may be used for localizing and/or tracking the mouth using any of the above frameworks in addition to other methods such as Gaussian mixture models or deformable contours (snakes) [11]. Given the mouth localization, which is primarily a computer vision problem, features must now be extracted, primarily a speech recognition problem.

Features used include the outer and inner lip contours [11], the teeth and tongue location, and the pixel intensities (texture) of an image of the mouth area. Choosing the visual features that contain the most useful information about the speech is of great importance, but still an open problem. There are three main approaches for visual feature extraction from image sequences; namely, image-based, geometry-based, and a combination of image and geometry-based methods.

**Image-Based Approach:** The two-dimensional mouth image is transformed into a feature vector by directly operating on the pixel intensities of the image. Examples
of such approaches include Principal Component Analysis (PCA), Discrete Wavelet Transform (DWT), and Discrete Cosine Transform (DCT). These approaches, however, lead to feature spaces of high-dimensionality. To remedy this, the energy compacting nature of these transforms is exploited and only the transformed coefficients corresponding to the top 85 – 95% of the energy is used. This normally results in feature vectors of manageable dimensionality.

**Geometry-Based Approach:** The two-dimensional mouth image is analyzed and parameters describing the geometry or shape of the mouth are extracted. The face features important for visual speech perception include lip contours, tongue, and teeth positions. Any or all of these features, along with others not mentioned, may be modeled and the model parameters concatenated into a feature vector. Generally these vectors are of relatively low dimensionality, however, dimensionality reduction via PCA or linear discriminant analysis (LDA) may be used to further lower the feature vector dimensionality.

In addition to these so-called static features, so-called dynamic features are used to create more information laden feature vectors. Dynamic features are formed through time differences of the static features. Delta and delta-delta (or acceleration) coefficients denote the first and second differences respectively. One may then concatenate the static and dynamic features to form the final feature vector. Additionally, one may create zero-mean input vectors by calculating the mean of each individual feature over a sequence and subtracting that mean from the respective feature coefficients. This procedure generally creates feature vectors more robust to inter-speaker variations [12].
2.1.3. Block 3: Recognizer

Following feature extraction, the actual speech recognition occurs. This vital block in the system is, in many senses, just a glorified pattern matching algorithm. As such, a variety of algorithms have been used for speech recognition including many flavors of hidden Markov models (HMMs) and support-vector machines (SVMs). In recent years, HMMs have been the most prevalent method of speech recognition and, accordingly, much research has gone into their continued development. Hidden Markov models are discussed at length in Sec. 2.3.

2.2. Recognizer architecture

While the pattern matching algorithm plays a large role in the recognition performance, the recognizer’s architecture represents an equally important, yet independent, variable affecting recognition performance. The recognizer architecture used today results from many years of continued development. This development, however, has only been in the context of A-ASR, and the V-ASR architecture has simply been carried over from A-ASR.

The standard, and almost always used, recognizer architecture for isolated word recognition is shown in Fig. 2.2. In this paradigm, an observation is passed from the feature extractor into the recognizer (pattern matcher) where it is then fed into "word models". Each word model has been independently trained to produce a likelihood that the observation came from that model and, thus, the model outputting the highest likelihood is deemed recognized word.

Viewing the speech recognition problem in the context of a general information theoretic communications systems leads to an interesting analysis. On a low level, the words being uttered represent the messages sent through the communications system and the verbal
articulations denote the encoded transmissions. The object of the speech recognizer is then to detect these articulations in the presence of noise and subsequently decode them into words. In this light, the standard recognizer architecture models the encoded symbols, not the phenomenon producing those symbols. Decoding is then accomplished by finding the word model that most closely models the observation without regard to the underlying symbol production. This ignorance of the underlying production phenomenon provides major motivation to the work presented in this thesis.

2.3. Hidden Markov models

Hidden Markov models (HMMs) form the backbone of modern speech recognition but also find great use in a variety of disciplines including normality/abnormality detection [13], DNA sequencing [14], detection of ECG events [15], economics [16], and many more (see [17] and references therein). Their versatility has let to wide-spread popularity and, thus, a plethora of practical and theoretical results. Hidden Markov models represent a doubly stochastic process in which one process, the "hidden" or unobservable process, progresses through a discrete state space while a second observable process takes on distinct stochastic
properties dependent upon the hidden state. In this context, unobservable implies that the process itself does not emit any information that one may directly gather, therefore it is hidden from the observer. One, however, may infer information about this hidden process by gathering information produced by the directly observable process due to its dependance on the hidden process. This inference lies at the heart of HMMs. Section 2.3.1 presents a high-level formulation of HMMs while Sec. 2.3.2 delves into the mathematical formulation.

2.3.1. HMMs: High-level overview

A classic, illustrative example of an HMM [18] now follows. There are $M$ urns in a room, each containing balls of $N$ different colors. An urn is randomly chosen (the hidden state) according to some Markovian process (the hidden process), and then a ball drawn from that urn according to the distribution of colored balls in that urn (the observable process). The color of the ball is recorded (the observation), and then the ball redeposited. Now assume that as an observer one only has access to the sequence of observations but wants to infer the which urn was used for each observation. This contrived example illustrates the inference problem associated with using hidden Markov models.

Figure 2.3 shows an example HMM setup involving four hidden states, $S_1$ through $S_4$. Each hidden state emits observable data modeled by a probability density function (PDF) equal to the linear combination (mixture) of three separate PDFs. It is important to note that this figure depicts states with emissions in a continuous range rather than a discrete set of possible emissions as in the preceding example.

Naturally, the question arises of how to best choose a model. If one knows the structure of the underlying phenomenon, for instance the number of urns and the probability mass functions (PMFs) of the colored balls in each urn, one can specify the model topology and
Figure 2.3. *Four state, three mixture fully connected HMM*

characteristics (in this case three states emitting according to the known PMFs). The model selection problem can be broken up into two sub-problems detailed on a high-level in Secs. 2.3.1.1 and 2.3.1.2. A more technical treatment of HMMs follows in Sec. 2.3.2.

**2.3.1.1. Problem 1: Choosing the model topology.** To fully define the topology of the HMM, one must first select the number of states, the allowed state transitions, and the probability density/mass function(s) to use in modeling the hidden states’ emissions.

Traditionally the number of states is determined empirically, as the structure of the phenomenon to be modeled is usually unknown. For some state \( S_i \), the states that are allowed to directly follow it (sequentially in time) denote the allowed state transitions. One may force a specific set of allowed transitions or let it be data-driven. Two standard topologies are the "left-to-right" and "ergodic" models \([18]\). In the so-called left-to-right topology, shown in Fig. 2.4, all states except the last one may transition only to itself and its right neighbor while the last state may only transition to itself. This topology, thus, models the sequential
evolution of a process. Unlike the left-to-right topology, an ergodic topology does not impose such strict conditions. Technically speaking, for a model to be ergodic all states must be recurrent and aperiodic [17]. Practically speaking, this means one must be able to travel from any state to any other state in finite time and that over time states are not visited in a periodic manner. For most ergodic HMM implementations this constraint is relaxed to just allowing that any state may transition to any other state as shown in Fig 2.3. The concept of left-to-right and ergodic topologies will be at the heart of this work.

Lastly, one must specify what type and how many density/mass functions to use in modeling. Whether to use density or mass functions depends directly on the type of data observed. The type of functions are typically Gaussian for continuous observations and binomial for the discrete case due to their ease of use, but any arbitrary function may in theory be used. Using multiple functions per state (ie. mixtures of functions), may indeed
lead to more accurate modeling but these more complex models require significantly more training data and may be prone to over-fitting.

2.3.1.2. Problem 2: Determining the properties of the stochastic processes. The goal of modeling a phenomenon as a hidden Markov model is to infer the hidden state based upon observable data. To achieve this goal, one must fully specify the parameters of the underlying stochastic processes, and then, using the model, observe real data and complete the inference. A rigorous treatment of this topic will follow in Sec. 2.3.2.

To determine the necessary parameters of an HMM, one must have either a priori knowledge or sample observation data from which to infer the parameters. As a priori knowledge is rarely available, one normally infers from sample data. The question now becomes how does one actually do this inference. This is identified by Rabiner as the third of three basic problems for HMMs [18] and also as the hardest of the three.

2.3.2. HMMs: Mathematical formulation

The notation used in the formulation and analysis of hidden Markov models in this thesis mirrors that in [18] and is as follows:

- Let $O = \{o_1, o_2, ..., o_i\}$ be a sequence of $i$ data observations.
- Let $N$ be the number of states $S_1$ through $S_N$ and $q_t$ be the state occupied at time $t$.
- Define:

$$a_{i,j} = P\{q_{t+1} = S_j | q_t = S_i\}$$

$$A = \{a_{i,j}\}$$
with the standard stochastic constraints

\[ a_{i,j} \geq 0, \forall i, j \]

\[ \sum_{j=1}^{N} a_{i,j} = 1, \forall i \]

- Let \( B = \{ b_j(k) \} \) be the probability of observing symbol \( k \) while in state \( j \). For discrete alphabets \( \{ b_j(k) \} \) is a probability mass function and a probability density function for continuous alphabets. Additionally for continuous alphabets, \( \{ b_j(k) \} \) is normally represented by a mixture of mixture of log-concave or elliptically symmetric density functions (ie. Gaussians) [19]. For multidimensional observations, these density functions take on a multi-variate form. For this work multi-variate Gaussian densities were used unless otherwise stated. Thus, \( \{ b_j(k) \} \) becomes \( \{ b_j(o) \} \) where \( o \) denotes a data observation leading to,

\[ b_j(o) = \sum_{k=1}^{M} c_{jk} N(o; \mu_{jk}, U_{jk}), \quad 1 \leq j \leq N \]

where \( \mu_{jk} \) and \( U_{jk} \) denote the Gaussian density’s mean vector and covariance matrix and \( c_{jk} \) is the \( k^{th} \) mixture in state \( j \).

- Let \( \pi = \{ \pi_i \} \) where,

\[ \pi_i = P\{ q_i = S_i \}, \quad 1 \leq i \leq N \]

Thus, an HMM is fully specified by \( N, M, A, B, \) and \( \pi \). Given these parameters, an HMM may be used in a generative fashion to stochastically generate an observation sequence. This generative nature will prove useful for determining the likelihood that some observed
sequence emanated from a given HMM. Since $N$ and $M$ are design choices, a compact notation for the parameters is introduced:

$$\lambda = (A, B, \pi) \quad (2.1)$$

The three major questions, posed in [18] and paraphrased here for completeness, that now arise are enumerated below and discussed in depth in Secs. 2.3.2.1, 2.3.2.2, and 2.3.2.3.

1. **Evaluation**: How does one evaluate the probability of an observed sequence given $\lambda$?
2. **Hidden state recovery**: How can the hidden state sequence be determined from an observation sequence given $\lambda$?
3. **Model updating**: How can one determine the parameters of an HMM from multiple observations?

**2.3.2.1. Evaluation: The Forward Procedure.** Given an observed sequence $\mathcal{O}$ and an HMM $\lambda$, how does one compute $P\{\mathcal{O}|\lambda\}$? A natural first, but naïve approach, is to enumerate all possible state sequences. If one takes a specific sequence $Q = q_1, q_2, \ldots q_T$ where $T$ is the number of observations (length) in $\mathcal{O}$ and assumes statistical independence of the observation sequence then,

$$P\{\mathcal{O}|Q, \lambda\} = \prod_{t=1}^{T} P\{O_t|q_t, \lambda\}. \quad (2.2)$$

This may be expressed terms of the state density functions $\{b_j(o)\}$ as

$$P\{\mathcal{O}|Q, \lambda\} = \prod_{t=1}^{T} b_{q_t}(O_t). \quad (2.3)$$

The probability of $Q$ may be written
**Algorithm 1: Forward Procedure**

Define the forward variable $\alpha_t(i)$, the probability of the partial observation sequence up to time $t$ and state $S_i$ at time $t$ given $\lambda$, as

\[
\alpha_t(i) = P\{O_1, O_2, ..., O_t, q_t = S_i | \lambda\}
\]

1. **Initialization:**
   \[
   \alpha_1(j) = \pi_j b_j(O_1), \quad 1 \leq i \leq N
   \]

2. **Induction:**
   \[
   \alpha_{t+1}(j) = \sum_{i=1}^{N} \alpha_t(i) a_{ij} b_{j}(O_{t+1}), \quad 1 \leq j \leq N, \quad 1 \leq t \leq T - 1
   \]

3. **Termination:**
   \[
   P\{\theta | \lambda\} = \sum_{i=1}^{N} \alpha_T(i)
   \]

\[
P\{Q|\lambda\} = \pi_{q_1} a_{q_1,q_2} a_{q_2,q_3}...a_{q_{T-1},q_T},
\]

leading to the joint probability of $\theta$ and $Q$

\[
P\{\theta, Q|\lambda\} = P\{\theta|Q, \lambda\} P\{Q|\lambda\}.
\]

By summing the above equation over all $Q$, one may compute $P\{\theta|\lambda\}$. This brute force approach leads to an extraordinary amount of computation and a more clever method of computation must be employed. The method used here is known as the Forward Procedure (Algorithm 1). For $N = 5$ and $T = 100$, this approach saves computation of about 69 orders of magnitude.

**2.3.2.2. Hidden state recovery: The Viterbi Algorithm.** In many cases, the probability of an observation is not the only piece information needed. Many times, one needs to know the sequence of hidden states that produced the observation. The solution to this problem, unlike the evaluation problem, has no closed-form, analytical solution. The heart
Algorithm 2: Viterbi Algorithm

Define the highest probability along a single path at time $t$ accounting for the first $t$ observations and ending at state $S_i$ as

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P\{q_1, q_2, \ldots, q_t = i, O_1, O_2, \ldots, O_t | \lambda\}$$

By induction,

$$\delta_{t+1}(j) = \left[ \max_i \delta_t(i) a_{ij} \right] \cdot b_j(O_{t+1})$$

(1) Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N$$

$$\psi_1(i) = 0$$

(2) Recursion:

$$\delta_t(j) = \max_{1 \leq i \leq N} \left[ \delta_{t-1}(i) a_{ij} \right] \cdot b_j(O_t), \quad 2 \leq t \leq T, \quad 1 \leq j \leq N$$

$$\psi_t(j) = \arg\max_{1 \leq i \leq N} \left[ \delta_{t-1}(i) a_{ij} \right], \quad 2 \leq t \leq T, \quad 1 \leq j \leq N$$

(3) Termination:

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$$

$$q^*_T = \arg\max_{1 \leq i \leq N} [\delta_T(i)]$$

(4) State sequence backtracking:

$$q^*_t = \psi_{t+1}(q^*_{t+1}), \quad t = T - 1, T - 2, \ldots, 1$$

of this issue lies in the fact that many state sequences may produce the same observations sequence and how does one differential between these state sequences to choose the "best" one. Upon defining an optimality criterion, the hidden state recovery problem may be solved. Traditionally, a maximum likelihood (ML) criterion is used and this problem can now be solved using the well known Viterbi Algorithm (Algorithm 2).

2.3.2.3. Model updating: The Baum-Welch Algorithm. The final, and by far most difficult, problem associated with HMMs is that of adjusting the model parameters $\lambda = (A, B, \pi)$ to maximize the probability of the observation sequence given the model. Currently, no known method exists to analytically solve this problem, however, iterative approaches allow one to find $\lambda$ such that $P\{O|\lambda\}$ is locally maximized. Approaches such a
expectation-maximization (EM) may be used, however it is more common to use the Baum-Welch Algorithm. As the work in this thesis does not make use of model adjustments, a full algorithmic description of the Baum-Welch Algorithm will not be presented.

2.4. Left-to-right hidden Markov models in ASR

Given the tools of hidden Markov models, how may they be used in automatic speech recognition systems? As shown in Fig. 2.2, data observations flow into word models which then output likelihoods. Hidden Markov models take such data observations and, in the evaluation context, can output the likelihood that the given model produced the observed data. This traditional method of using HMMs in ASR results in the use of left-to-right HMMs for a variety of reasons. Firstly, left-to-right models naturally depict sequences with distinct causal, time-evolutions. Secondly, the limited design parameters of the model add to its appeal. Lastly, the limited model parameters \( \lambda = (A, B, \pi) \) allow for more stochastically stable modeling.

While much research has taken place in the field of HMMs for ASR, almost all approaches use left-to-right models and attempt to model the verbal units (in the case of this thesis whole words) independent of each other and without regard to the underlying production phenomenon: the time varying verbal articulations.

2.5. Ergodic hidden Markov models in ASR

If one wishes to directly model the sequence of verbal units (phonemes, visemes, words, etc.) articulated, rather than the verbal units themselves, a left-to-right model no longer makes sense. The sequence of these verbal units does not obey the simple sequential rules of a left-to-right model and much more closely fits in to the ergodic model framework. During
articulation, only certain verbal units may follow each other, but one may progress from one verbal unit to any other unit if enough intermediate states are allowed. In the terms of an HMM, if one defines each verbal unit as a state, then one may visit any state from any other state in finite time, the practical definition of an ergodic HMM.

Levinson first implemented this approach for A-ASR with phonemes as verbal units in 1986 [20]. Also introduced was a revised HMM model with explicit state duration densities rather than the geometric densities inherent when self-transitions are allowed. Later this implementation was dubbed the acoustic/phonetic model [21]. In this model, each state represented a phoneme with a delta distributed duration and each state was modeled as a mixture of Gaussian densities. The features used were linear prediction coefficients (LPCs) rather than MFCCs. Using this paradigm allows for simple phonetic transcription of speech as just the state sequence of the ergodic HMM. Results produced from this method proved very promising [22], however dissemination of further results abruptly stopped soon after the results were reported. Additionally, very little to almost no work seems to have continued on this work thereafter.

This work takes much inspiration from the works of Levinson and Ljolje and expands upon this model. It is important to note at this point, however, some of this approach’s issues that will be revisited and addressed later in this work. One striking consequence of this model is that the individual verbal units are still modeled independently. Additionally, the verbal units are modeled as a whole in a static sense rather than as a dynamic sequence. Because of the nature of the auditory articulators and the continuous signal nature of speech, one may not infer much information from a single observation (traditionally 10ms in length). The acoustic/phonetic model, however, requires that significant information be inferred from each observation.
CHAPTER 3

Recognizer Description

This chapter presents the main body of this work: a new architectural approach to the speech recognition problem. Section 3.1 motivates the design of the proposed architecture. A high-level overview of the complete system is provided by Sec. 3.2, while Secs. 3.3 through 3.5 detail the system components in depth.

3.1. Design motivation

Upon examination of modern speech recognition systems, one finds that no system actually models the speech production phenomenon. These state of the art systems instead model the symbols produced by speech in the form of words or sub-word verbal units. Additionally, the features generally extracted from the input stream, whether it be audio or video, try to describe how humans perceive speech rather than how humans generate speech. For audio-only automatic speech recognition (A-ASR), these features tend to be Mel Frequency Cepstral Coefficients (MFCCs) which attempt to filter audio signals in a manner similar to human perception. In video-only automatic speech recognition (V-ASR), features tend to describe physical attributes of the visual articulators in the form of parameterized shape models or some decomposition of the articulators’ appearance (Discrete Cosine Transform, Principle Component Analysis, etc.). As a consequence of the symbol modeling approach in
In contrast to the variable temporal granularity and coarse resolution of the traditional systems, the proposed system utilizes fine resolution at a fixed temporal scale. By doing so, the temporal dynamics of speech can be explicitly modeled in parallel with the static speech production phenomenon. As a direct consequence of this approach, the verbal units are not modeled completely independently of each other, although recognition takes place independently. In fact, this architecture can jointly model verbal units that may even be fully contained within other verbal units. For example, if the verbal units are digits, then "oh", an alternate pronunciation of "zero", is actually contained within the unit "zero". With the proposed architecture, the verbal articulation of the "oh" sound is modeled using data from both verbal units "zero" and "oh", rather than each digit model separately modeling the "oh" sound. In effect, this approach separates articulation modeling and articulation interpretation.

1When state-tying is used in hidden Markov models, some verbal units may share training data, but the vast majority of the units are trained independently of each other.
modeling. Additionally, this leads to state modeling in a very loose contextual manner. The contextual information, however, may be incorporated during the interpretation stage of the proposed system. Finally, these differing approaches lead to a vastly smaller state space.

### 3.2. System overview

Akin to the traditional recognizer architecture, the proposed system takes as input some data observation sequence. This observation sequence then drives the ”articulation model”, an ergodic hidden Markov model detailed in Sec. 3.3. The input sequence traces out a state sequence, or path, through the articulation model and is the model’s output. This path represents the time evolution of the speaker’s verbal articulators, hence the name of the model. The interpretation stage of this approach now begins. Each verbal unit being modeled, in this case digits, has a variety of articulation paths representing different pronunciations. At this point, the observed articulation must be compared to known articulations, detailed in Sec. 3.4. This comparison provides a ”distance” between the articulations. The known articulations are compiled for each verbal unit and the distances pooled over each unit resulting in an overall verbal unit distance, detailed in Sec. 3.5. The verbal unit with the shortest distance from the observed articulation is declared as the recognized unit. Figure 3.1 diagrams this proposed architecture, while Fig. 3.2 shows the traditional architecture for comparison (reproduced from earlier in this thesis).

### 3.3. Articulation model

As the name implies, the articulation model seeks to model how humans articulate speech. As stated previously in Sec. 2.5, an ergodic HMM naturally fits this task. Contrary to [20, 21, 22], however, the states of the articulation model will not represent the verbal units
themselves, but instead will represent sub-verbal “articulation units”. Additionally, the state durations are not explicitly modeled, and, thus, are left to their inherent geometrically distributed form. Also, these articulation units are far more static than the dynamic verbal units which should yield better inference and modeling. Because such articulation units are not explicitly defined, they shall be derived from training data leading to a data-driven model. Section 3.3.1 addresses how to train such an articulation model, and Sec. 3.3.2 details its evaluation during the recognition process. Figure 3.3 depicts a sample six-state
articulation model. As will be shown in Chapter 4, the best results were achieved with twenty states.

Figure 3.3. Six state articulation model

3.3.1. Training

Given a set of training utterances (observation sequences) \( \mathcal{X} = \{ \Theta_1, \Theta_2, ..., \Theta_{D_t} \} \) where \( D_t \) is the number of training utterances, the goal is to construct (train) the articulation model. Training the model breaks down into two distinct phases. Phase I consists of determining the articulation units from the data and modeling them using Gaussian mixture models. This determines the \( B \) matrix parameter of the hidden Markov model. The \( A \) and \( \pi \) parameters, representing the state transition matrix and the initial state probabilities respectively, are then extracted during Phase II of training. Section 3.3.1.1 describes training Phase I, while Sec. 3.3.1.2 details Phase II.

3.3.1.1. Phase I: Articulation unit identification and modeling. Phase I of training focuses on extracting and modeling the articulation units. For this step, the observations contained in the training sequences are not viewed as sequentially related and, instead, are viewed as independent observations. These ”independent” observations each represent
a different realization of an articulation. When grouped together into clusters of similar realizations, these clusters may be interpreted as states of articulation. These states can then be modeled as Gaussian mixture models using the observations belonging to that state. Algorithm 3 and Fig. 3.4 detail this general procedure.

It is important at this time to remark about various details regarding this work’s implementation of this approach. Firstly, the observations must reside in some vector-space in which unsupervised clustering makes logical sense. In this work, all data observations reside in orthonormalized vector-spaces conducive to simple K-Means (nearest neighbor) unsupervised clustering. Secondly, the vector-spaces must be continuous in $\mathbb{R}^n$, thus, allowing for the use of Gaussian mixture modeling. If the incoming data observations do not follow these guidelines, only simple adjustments are needed. If the vector-spaces are not orthonormal, either the clustering algorithm must take that into account or the data must be orthonormalized. If the vector-spaces are not continuous in $\mathbb{R}^n$, mixture modeling must utilize a discrete mass function. Thus, these two problems can be easily remedied if necessary.
Algorithm 3: General articulation unit identification and modeling algorithm

1. Separate input sequences into individual observations
2. Cluster observations using an unsupervised clustering technique
3. Train models for each cluster using the observations belonging to the cluster
4. Define each trained model as an articulation unit (state) in the ergodic HMM

While this work utilizes K-Means for clustering and Gaussian mixture modeling for cluster modeling, in theory any other algorithms may be used. The future model refinements section of this thesis, Sec. 5.1, addresses theses issues.

3.3.1.2. Phase II: State transition probability extraction. Phase II of the articulation model training seeks to determine the $A$ and $\pi$ parameters of the ergodic HMM, representing the state transition matrix and the initial state probabilities respectively.

Unlike in training phase I, phase II treats the training sequences as complete sequences, not collections of independent observations. An initial estimate of $A$ and $\pi$ is constructed by directly assigning each observation in each sequence the state into which it was clustered in phase I. The transition matrix is then populated directly from this data. The initial state probabilities may also be initialized in this way, however, this work assumes that all states are equally likely to be the initial state\(^2\). A second pass through the training data aims to refine the initial estimates. Each observation sequence will now be fed as input to the articulation model and the state transitions recorded. This procedure will be addressed in depth in 3.3.2 and Algorithm 5. This new transition matrix will be the final $A$ matrix used and $\pi$ will again be assumed as uniformly distributed. In theory, multiple refinement passes may be made but these extra refinements showed no significant improvements during testing. Algorithm 4 details this procedure.

\(^2\)This assumption come from the observation that visual articulators frequently do to not return to or start from a "resting state" like auditory articulators (ie. silence).
At this point, one may also use a method such as the Baum-Welch algorithm to better refine the model. This work chooses not to take this step, however. Although the Baum-Welch algorithm adjusts the model parameters to maximize the probability of the observation given the model, it may significantly change the model parameters to the point that the states no longer represent distinct articulation states. It is for this reason that this method is not employed.

### 3.3.2. Evaluation

During the recognition process, as well as during the second phase of training, the articulation model must take observations as inputs and output the sequence of articulation states. As described in Sec. 2.3.2.2, recovering the hidden state sequence utilizes the Viterbi algorithm (Algorithm 2). The traditional implementation of the Viterbi algorithm, however, is a non-causal process due to the need to backtrack. While the work in this thesis is not explicitly for real-time applications, the non-causality of the Viterbi algorithm may prove to be an unwanted nuisance. A causal approximation to the Viterbi algorithm, known as the Token Passing algorithm [23] may be used as a substitute. In addition to being causal, the Token Passing algorithm results in less computation than the Viterbi algorithm.

The Token Passing algorithm works by propagating "tokens" through the hidden Markov model based on the observed sequence and the model parameters. Each of these so-called tokens carry with them an accumulated likelihood as well as a record of its path through the model. Because of the information carried by each token, the most likely state sequence at time $t$ can be identified by looking at the path taken by the token with highest likelihood.

The Token Passing algorithm has two parameters, $N_T$ and $N_K$, that dictate the number of tokens propagated per time step and the number of tokens that survive pruning at each
Algorithm 4: State transition probability extraction

\( N \) = number of states in articulation model
\( D_t \) = number of training sequences
\( a (i, j) \) = transition probability from state \( i \) to state \( j \)
\( \pi (i) \) = probability of beginning in state \( i \)

Denote \( \mathcal{O}_{i,j} \) = \( j^{th} \) observation in \( i^{th} \) training sequence
Define \( C (\mathcal{O}_{i,j}) \) = the cluster into which \( \mathcal{O}_{i,j} \) was placed during training phase I
Define \( n (i) \) = the number of transitions out of state \( i \)
Define \( \tilde{a} (i, j) \) = refined transition probability from state \( i \) to state \( j \)
Define \( \tilde{n} (i) \) = refined number of transitions out of state \( i \)
Define \( q_{i,j} \) = the state occupied during \( j^{th} \) observation of the \( i^{th} \) sequence

Pass 1: Initial Estimation
Initialize \( a (i, j) = 0 \), \( \forall i, j \)
Initialize \( n (i) = 0 \), \( \forall i \)
for all \( i, i \in [1, D_t] \) do
  for all \( j, j \in [1, \text{length} (\mathcal{O}_i)] \) do
    Increment \( a (C (\mathcal{O}_{i,j-1}), C (\mathcal{O}_{i,j})) \)
    Increment \( n (C (\mathcal{O}_{i,j-1})) \)
  end for
end for
for all \( i, i \in [1, N] \) do
  for all \( j, j \in [1, N] \) do
    \( a (i, j) = \frac{a(i,j)}{n(i)} \)
  end for
\( \pi (i) = 1/N \)
end for

Pass 2: Estimation Refinement
Initialize \( \tilde{a} (i, j) = 0 \), \( \forall i, j \)
Initialize \( \tilde{n} (i) = 0 \), \( \forall i \)
for all \( i, i \in [1, D_t] \) do
  Traverse HMM using \( \mathcal{O}_i \) as input (see Sec. 3.3.2)
  for all \( j, j \in [1, \text{length} (\mathcal{O}_i)] \) do
    Increment \( \tilde{a} (q_{i,j-1}, q_{i,j}) \)
    Increment \( \tilde{n} (q_{i,j-1}) \)
  end for
end for
for all \( i, i \in [1, N] \) do
  for all \( j, j \in [1, N] \) do
    \( \tilde{a} (i, j) = \frac{\tilde{a}(i,j)}{\tilde{n}(i)} \)
  end for
end for
time step, respectively. At the first time step, when the first observation is received, \( N_K \) tokens are initialized at the \( N_K \) states most likely to have produced that observation in isolation. Upon receipt of the next observation, each of the \( N_K \) tokens propagate \( N_T \) tokens to the \( N_T \) states most likely to have produced the new observation, taking into account the state transition probabilities. The \( N_K \times N_T \) tokens are then sorted and all but the top \( N_K \) tokens pruned. This process repeats until all observations have been processed. Algorithm 5 presents a detailed description of Token Passing.

### 3.4. Path distances

Once the articulation path has been determined by the articulation model, the observed path must now be classified. An essential component to the classification task is a means of comparing one path to another. Section 3.5 details the actual classification process, while an algorithm for comparing paths is presented here.

This system component seeks to define some "distance" between two paths. Two major problems, however, come about due to the nature of these paths. Firstly, the paths consist of discrete nodes which do not reside in a Euclidean vector-space, thus, complicating the ability to define distances between nodes. Secondly, the number of nodes in each path may be different complicating the path comparisons as a whole. Section 3.4.1 investigates the first problem, while Sec. 3.4.2 investigates the second.

#### 3.4.1. State distance computation

Each node in the articulation path represents a different articulation unit. Conveniently, each of these articulation units have been modeled using mixtures of Gaussian density functions allowing the use of some results from information theory. The Kullback-Leibler (KL)
Algorithm 5: Token Passing approximation to the Viterbi algorithm

\begin{itemize}
\item $N$ = number of states in the model
\item $N_T$ = number of tokens propagated per time step
\item $N_K$ = number of tokens kept during pruning
\item $LL(i)$ = log-likelihood of token $i$
\item $LL_j(i)$ = log-likelihood of token $i$ propagated from token $j$
\item $q_i(j)$ = $j^{th}$ state in sequence traversed by token $i$
\item $O_{i}$ = $i^{th}$ observation in sequence
\item $a(i, j)$ = transition probability from state $i$ to state $j$
\item $P\{\mathcal{O}_t|q_j(t) = n, \lambda\}$ evaluated directly from the Gaussian mixture model of state $n$
\end{itemize}

**time $t = 0$: Initialization**

\begin{algorithmic}
\ForAll{$n \in [1, N]$}
\State $LL(n) = P\{\mathcal{O}_0|q_n(0) = n, \lambda\}$
\EndFor
\State Sort $LL(n)$
\State Keep $N_K$ tokens with highest $LL(n)$
\end{algorithmic}

**time $t \geq 0$: Propagation**

\begin{algorithmic}
\ForAll{$n \in [1, N_K]$}
\ForAll{$k \in [1, N]$}
\State $LL_n(k) = P\{\mathcal{O}_t|q_k(t) = k, q_n(t-1), \lambda\}$
\State $= a(q_n(t-1), k) \cdot P\{\mathcal{O}_t|q_k(t) = k, \lambda\}$
\EndFor
\State Sort $LL_n(k)$
\State Keep $N_K$ tokens with highest $LL_n(k)$
\EndFor
\State Sort $LL_j(i), i \in [1, N_T], j \in [1, N_K]$
\State Keep $N_K$ tokens with highest $LL_j(i)$
\end{algorithmic}

divergence, one such result, allows the calculation a non-symmetric "distance" between two Gaussian density functions. However, there exists no closed form solution for computing the KL-divergence between mixtures of Gaussians. Many approximations to the KL-divergence exist for Gaussian mixtures, and this work utilizes an approximation based on the Unscented Transformation \cite{24} presented in \cite{25}. Using this approximation, state distances may now be computed for use in the path distance computation.
3.4.2. Variable-length path distance computation

With the state distance problem solved, the focus now turns to computing the distance between paths with differing numbers of nodes. This problem, similar to the variable-length sequence modeling problem, has seen much research in recent years. A variety of approaches exist including the use of left-to-right HMMs [26] and the Smith-Waterman [27] algorithm. Both of these methods, however, view the inputs as sequences of values rather than paths through some abstract space. As paths through some space, not only are the discrete nodes important to the distance calculation, but so is what happens between the nodes. The "Improved Double Time Dimension Mapping Function" (IDTDMF) derived by Felner, et al. [28] considers both the discrete nodes and the between node areas when computing the path distance. To use the IDTDMF for the purposes of this work, however, a few interpretational changes must be made.

At its foundation, the IDTDMF assumes nodes occur at various times along some path and that each node has some distance relationship to all the other nodes. In this work, "time" will be interpreted as the "distance" traveled between nodes since all observations take place at the same discrete times. Using this interpretation, each sequence of nodes has some total distance traveled equal to the sum of the distances between each node. When comparing paths, the paths may now be normalized to a constant "length" and correspondences established by mapping nodes between paths. These mappings must be done in a monotonic fashion (ie. nodes may only be mapped equal to or forward in "distance"). One may define the total distance from path A to path B as the sum of the distances between nodes associated with each other during the mapping process. Unfortunately, this mapping process may vary depending on whether one calculates the distance from A to B or from B
to $A$ leading to an asymmetry. One may reconcile this by defining the "double" distance from $A$ to $B$ as the average of the distances from $A$ to $B$ and from $B$ to $A$. For a full algorithmic description, see [28].

3.5. Word Recognition

Given the ability to determine a distance between paths, path classification may now take place. This work takes a case-based reasoning [29] style approach to the classification problem. With this approach, the system obtains training "cases": in the case of this work, an articulation path and the spoken word associated with that path. These training cases are then directly used in evaluation. One first compares unknown cases to known training cases. Classification takes place by using the classification of the known case declared most similar to the unknown case. Essentially, the system classifies new cases based on similarity with old cases. Section 3.5.1 explains the training method and Sec. 3.5.2 details the specifics of the case-based reasoning style approach used in this work. Section 3.5.3 describes an $N$-best voting scheme used to combine the results of the $N$-best articulation paths output by the model for a given observation sequence.

3.5.1. Training

Training this case-based reasoning style system simply consists of gathering known cases and grouping them by classification. This means utilizing the articulation paths output from the model during phase II of model training (Sec. 3.3.1.2). All training paths are then grouped by verbal unit for use in the evaluation process.
3.5.2. Closest word evaluation

With the training data and path comparison metrics in place, the classification of unknown paths may begin. A first attempt at a case-based reasoning approach may be to find the closest training path to the unknown path and use the classification of the training path. In an ideal world where pronunciations and articulations do not vary, this approach may work quite well. However, pronunciations and articulations do vary and the closest path may not be the best choice. As an alternative, this work defines the "Brute Force Numerical Average - $K^3$" metric.

The "Brute Force Numerical Average - $K$" ($BFNA_K$) metric takes into account more than just closest training path. The $BFNA_K$ metric operates over each classification label producing a single distance score from the unknown path to that classification label. The label (word) with the shortest distance is then declared the recognized word. To compute the $BFNA_K$ metric for a word, the distances between all training paths associated with that word and the unknown path are calculated. The $K$ shortest distances are then averaged and used as the metric for that word. Setting $K = \infty$ implies averaging over all paths in the word. For convenience, the $BFNA_\infty$ case may also referred to as $BFA$. Algorithm 6 details the implementation of the $BFNA_K$ metric.

While the $BFNA_K$ metric takes into account multiple training paths per word label, it only focuses on the top $K$ paths. By using a linear combination of multiple $BFNA_K$ metrics per word label, one may effectively weight different types of similarities. For instance, for $BFNA_K$ metrics with small $K$ characterize how close a path is to the best possible matches in a word model, while $BFNA_K$ metrics with large $K$ characterize how close a path is to the best possible matches.

---

$^3$Brute Force comes from the idea that all training data is used in evaluation, thus, this is a brute force approach.
Algorithm 6: Brute Force Numerical Average - $K$ ($BFNA_{K}$) metric

\begin{algorithm}
\begin{algorithmic}
\State $P_n = n^{th}$ training path
\State $L(P_n) =$ word label of $n^{th}$ training path
\State $D_t =$ number of training sequences
\State $N_L =$ number of word labels
\State $\mathcal{U} =$ path to be classified
\State $d(P_i, P_j) =$ IDTDMF distance between $P_i$ and $P_j$
\State $M_x = BFNA_{K}$ metric for label $x$
\ForAll{$x \in [1, N_L]$}
\ForAll{$i \in [1, D_t], \text{s.t.} L(P_i) = x$}
\State $m(i)_x = d(P_i, \mathcal{U})$
\EndFor
\State $\bar{m}(i)_x = m(i)_x$ sorted in ascending order
\State $M_x = \sum_{i=1}^{K} \frac{\bar{m}(i)_x}{K}$
\EndFor
\State $L(\mathcal{U}) = \arg\min_x M_x, x \in [1, N_L]$
\end{algorithmic}
\end{algorithm}

is to the overall space of possible articulation and pronunciations. By leveraging $BFNA_{K}$ metrics with both small and large $K$, one may balance the importance of specificity versus generality. The vector $\beta = [BFNA_i, BFNA_j, ... BFNA_m]$ represents the metrics to be used, and $\alpha = [\alpha_i, \alpha_j, ... \alpha_m], \sum \alpha = 1$ denotes their respective weights.

3.5.3. $N$-best voting

Voting methods are common throughout pattern recognition implementations. Up to this point in this work, a single path through the articulation model was used for recognition. From the token passing algorithm, the $N \leq T_K$-best paths can be extracted. Each path carries with it an accumulated likelihood. By performing recognition on each path and then attributing a voting weight equal to the paths' likelihoods, the paths may "vote" for the
recognition result. The word label receiving the largest number of votes is then declared the recognition result.
CHAPTER 4

Experiments and Comparative Results

This chapter describes and presents the video-only automatic speech recognition (V-ASR) experimental results of the articulation model architecture. As architecture performance is the highlight of this work, a database recorded in ideal conditions, and a simplistic visual front-end are used. Additionally, this architecture is directly compared, using the same database and front-end, with a baseline, state of the art, left-to-right hidden Markov model based V-ASR system. Section 4.1 examines the database used, and Sec. 4.2 describes the visual front-end, along with other parameters constant between architectures. Section 4.3 details the implementations and results of the baseline system, while Sec. 4.4 does so for the proposed system. Section 4.5 compares and discusses the results obtained from each architecture.

4.1. Database description

The database used for these experiments, the CMU Audio-Visual Speech Recognition database [30], consists of ten speakers speaking in a controlled, studio environment. The vocabulary of the database consists of isolated numbers, months, days, and other common words. In this context, isolated means that distinct pauses exist between all word utterances. Additionally, each speaker took part in ten recording sessions, thus giving one hundred instances of each word in the vocabulary. This work looks only at the digits "one" through "ten", inclusive. Figure 4.1 shows sample frames from each of the ten speakers.
The CMU database also includes some handy preprocessing results shown in Fig. 4.2. These results contain the mouth corners, \((x_1, y_1)\) and \((x_2, y_2)\), and the distances of the upper and lower lips from the center line, \(h_1\) and \(h_2\). This information will be used in the feature extraction step of the V-ASR system.

Figure 4.1. *Sample frames from CMU database*

Figure 4.2. *Sample frame with preprocessing results superimposed*
4.2. Experimental paradigm

In order to have comparable results between the baseline and proposed systems, many factors are held constant between architecture experiments. All experiments were speaker independent and cross-validated. Speaker independent experimentation means that all data from a speaker is held out of the training set and then used for testing. Thus, the testing data has never been presented to the system before. Cross-validation was conducted over all ten speakers and the recognition results averaged to compute final recognition rate for the experiment.

The visual front-end also remains the same for both systems. Utilizing the supplied preprocessing data, the front-end extracts\(^1\) two sets of static features briefly explained below. For a thorough description of the extraction process, see [7].

**Feature set 1: Discrete Cosine Transform coefficients**

Using the mouth corner points from preprocessing, the front-end extracts 65 pixel by 65 pixel region of interest (ROI) around the mouth. A two-dimensional Discrete Cosine Transform (DCT) is then performed on the image. The seventeen "most significant" coefficients, residing in a hyperbolic lattice as suggested in [31], are then extracted to form a singly-dimensional feature vector of length seventeen.

**Feature set 2: Geometric coefficients**

A simple geometric parameterization of the mouth involves directly using the preprocessing data as features. By concatenating the height of the upper lip, height of the lower lip, and the distance between the mouth corners, one may create a feature vector to use for recognition.

\(^1\)These static features were extracted by R. Biffiger in his M.S. thesis [7] and used in this work.
These static features, denoted DCT and HW, were then zero-mean normalized for each individual word utterance resulting in the $\text{DCT}_Z$ and $\text{HW}_Z$ feature sets. Additionally, experiments were performed using the dynamic delta and delta-delta features concatenated with the static features. Such feature sets are denoted by $\text{DCT}_{D\text{A}Z}$ and $\text{HW}_{D\text{A}Z}$.

4.3. Baseline system

The baseline system used mimics that used in [7]. The system consists of a standard speech recognition architecture using word models. Each model consists of ten states, eight of which are emitting. Each state is modeled using four Gaussian mixture models. Experimentation was performed using the Hidden Markov Model Toolkit (HTK) [32].

Figure 4.3 shows the per-speaker recognition rates for the $\text{DCT}_Z$ and $\text{DCT}_{D\text{A}Z}$. Figure 4.4 shows the per-speaker recognition rates for the $\text{HW}_Z$ and $\text{HW}_{D\text{A}Z}$. Table 4.3 presents the average recognition rates for each case.

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>Recognition Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{DCT}_Z$</td>
<td>68.52%</td>
</tr>
<tr>
<td>$\text{DCT}_{D\text{A}Z}$</td>
<td>76.70%</td>
</tr>
<tr>
<td>$\text{HW}_Z$</td>
<td>57.40%</td>
</tr>
<tr>
<td>$\text{HW}_{D\text{A}Z}$</td>
<td>62.96%</td>
</tr>
</tbody>
</table>

Table 4.1. Baseline system average recognition rates per feature set

As expected, the DCT based systems vastly outperform the HW based systems. The reasons for this are beyond the scope of this thesis, but the answer lies in the amount of information contained in each feature set. Also, recognition rates vary significantly across users. While part of this has been attributed to tracking errors in the preprocessing stage, inter and intra speaker variability explains the bulk of the variance. Because this thesis focuses on architectural comparisons, the tracking errors make no difference since they are present in all experiments.
4.4. Proposed system

For the proposed architecture, a variety of experiments were conducted. These experiments seek to not only determine the effects of the number states, the number of mixtures,
and the type of evaluation metric on system performance, but to find the best overall set of
design parameters. Akin to the baseline tests, experiments showed significant cross speaker
performance variance and, in most cases, the variances are very similar. For all tests, the
number of states varied from five to twenty while either one or two mixtures were used.

Section 4.4.1 examines the results from the training phase and looks into what the data-
derived states ended up representing. Section 4.4.2 presents results from the base imple-
mentation, while Secs. 4.4.3, 4.4.5, and 4.4.5 present selected results from adding $N$-best
voting, adding vector metrics, and then adding both. As a reminder, the vector metric
implementation refers to using a linear combination of base metrics.

Complete results can be found in Appendices A through D.

4.4.1. State representation examination

The question arises as to what the states of the articulation model actually represent. By
examining where states appear in a word-to-state transcription, inferences can be made as
to the concrete state representations. Table 4.2 shows the beginning of sample transcriptions
from the words "two" through "five" with an $N = 20$ state model.

<table>
<thead>
<tr>
<th>Word</th>
<th>State Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Two&quot;</td>
<td>19 19 19 19 3 3 8 8 2 ...</td>
</tr>
<tr>
<td>&quot;Three&quot;</td>
<td>19 19 19 19 3 16 16 8 ...</td>
</tr>
<tr>
<td>&quot;Four&quot;</td>
<td>18 18 18 18 18 7 6 6 [2] [2] [2] ...</td>
</tr>
<tr>
<td>&quot;Five&quot;</td>
<td>18 18 7 6 6 4 13 10 ...</td>
</tr>
</tbody>
</table>

Table 4.2. Partial state sequences for samples of the words "two" through "five"

Upon examination of the samples from words "two" and "three", one finds that both
begin with the same state sequence. Samples of "four" and "five" also begin with the same
sequence but this sequence is distinct from the beginning of "two" and "three". Multiple
instances of state 19 followed by state 3 seems to represent the "t" sound present at the beginning of both "two" and "three". Likewise, the state sequence 18 → 7 → 6 seems to represent the "f" sound present in both "four" and "five". This confirms that using the clustering method to define states generates states with distinct speech related representations. Additionally, this provides confirmation that varying paths through the articulation model represent different pronunciations.

4.4.2. Proposed system: basic implementation

For the base implementation tests, the metrics used included $BFNA_\infty$(a.k.a $BFA$), $BFNA_1$, $BFNA_3$, $BFNA_5$, $BFNA_10$, and $BFNA_{20}$.

The peak performance for the DCT$Z$ feature set occurred with $N = 20$, $M = 2$, and $BFNA_\infty$. For the DCT$D_A Z$ case, performance peaked with the same parameters but metric $BFNA_5$. Figure 4.5 depicts the per-speaker results for these scenarios. Figures A.1 and A.2 show how different metrics compared for the DCT$Z$ feature set with $M = 1$ and $M = 2$, respectively. Table 4.3 reports the peak performances averaged over the speakers.

<table>
<thead>
<tr>
<th>Recognition Rate</th>
<th>DCT$Z$</th>
<th>DCT$D_A Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>57.71%</td>
<td>59.29%</td>
</tr>
</tbody>
</table>

Table 4.3. Basic system performance rates for DCT$Z$ & DCT$D_A Z$

For the HW$Z$ feature set, the best results were obtained with $N = 20$, $M = 1$, and $BFNA_{10}$. Using the HW$D_A Z$ feature set, performance peaked using the same metric but with $N = 15$ and $M = 2$. Figure 4.6 depicts the per-speaker results for these scenarios. Figures A.5 and A.6 show how different metrics compared for the HW$Z$ feature set with $M = 1$ and $M = 2$, respectively. Table 4.4 reports the peak performances averaged over the speakers.
4.4.3. Proposed system: basic implementation with voting

The experimental setup used for the base implementation with voting mirrored that of the base implementation without voting.

Using $N = 20$, $M = 2$, and $BFNA_3$ achieved peak performance for the $DCT_Z$ feature set. For the $DCT_D-A_Z$ case, performance peaked with $N = 15$, $M = 2$, and $BFNA_{20}$. Figure 4.7 depicts the per-speaker results for these scenarios. Figures B.1 and B.2 show how different metrics compared for the $DCT_Z$ feature set with $M = 1$ and $M = 2$, respectively. Table 4.5 reports the peak performances averaged over the speakers.

The peak performance for the $HW_Z$ feature set was achieved with $N = 20$, $M = 1$, and $BFNA_{10}$, while for the $HW_D-A_Z$ case, performance peaked with $N = 15$, $M = 2$, and
Figure 4.6. Basic system speaker performance for HW\textsubscript{Z} \& HW\textsubscript{DAZ}

<table>
<thead>
<tr>
<th>Recognition Rate</th>
<th>DCT\textsubscript{Z}</th>
<th>DCT\textsubscript{DAZ}</th>
</tr>
</thead>
<tbody>
<tr>
<td>58.76%</td>
<td>59.81%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5. Voting system performance rates for DCT\textsubscript{Z} \& DCT\textsubscript{DAZ}

BFNA\textsubscript{10}. Figure 4.8 depicts the per-speaker results for these scenarios. Figures B.5 and B.6 show how different metrics compared for the HW\textsubscript{Z} feature set with $M = 1$ and $M = 2$, respectively. Table 4.6 reports the peak performances averaged over the speakers.

<table>
<thead>
<tr>
<th>Recognition Rate</th>
<th>HW\textsubscript{Z}</th>
<th>HW\textsubscript{DAZ}</th>
</tr>
</thead>
<tbody>
<tr>
<td>54.56%</td>
<td>55.40%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6. Voting system performance rates for HW\textsubscript{Z} \& HW\textsubscript{DAZ}

4.4.4. Proposed system: vector implementation

To limit the enormous space of possible metric/weight combinations, the number of metrics in use was limited to two and the weights limited to steps of 0.1\textsuperscript{2}.

\textsuperscript{2}Because the weights must sum to unity, this reduces the space significantly.
The best results attained for the DCT\_Z case arrose from $\beta = [BFNA_\infty, BFNA_3]$ with $\alpha = [0.2, 0.8]$, $N = 20$ states, and $M = 2$ mixtures. Using $\beta = [BFNA_20, BFNA_5]$ with $\alpha = [0.3, 0.7]$, $N = 20$ states, and $M = 2$ mixtures yielded the best results for the
DCT\_{DAZ} features. Figure 4.9 shows the results of this setup, while the average results are presented in Table 4.7. Figures C.1 and C.2 depict how varying $\alpha$ affects average recognition rates for the DCT\_Z case with $M = 1$ and $M = 2$, respectively. Figures C.3 and C.4 show these same results for the DCT\_{DAZ} feature set.

<table>
<thead>
<tr>
<th></th>
<th>DCT_Z</th>
<th>DCT_{DAZ}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognition Rate</td>
<td>58.56%</td>
<td>59.81%</td>
</tr>
</tbody>
</table>

Table 4.7. Vector metric performance for DCT\_Z & DCT\_{DAZ}

The best results for the HW\_Z case were attained with $\beta = [BFNA_{10}, BFNA_{3}]$ with $\alpha = [0.8, 0.2]$, $N = 20$ states, and $M = 1$ mixtures, the HW\_{DAZ} case required $\beta = [BFNA_{10}, BFNA_{1}]$ with $\alpha = [0.6, 0.4]$, $N = 15$ states, and $M = 2$. Figure 4.10 shows the results of this setup, while the average results are presented in Table 4.8. Figures C.5 and C.6 depict how varying $\alpha$ affects average recognition rates for the HW\_Z case with $M = 1$. 

Figure 4.9. Vector metric speaker performance rates for DCT\_Z & DCT\_{DAZ}
and $M = 2$, respectively. Figures C.7 and C.8 show these same results for the $HW_D A Z$ feature set.

Interestingly, not much is gained in either HW case by adding more states. This may be a result of the HW feature set "overly quantizing" the deformations of the mouth. For instance, using only the lip heights and mouth width, one may not be able to discriminate between a sufficient variety different mouth deformations. Hence, adding more states to the articulation model may result in an over-fitting type scenario. The discriminating power of the DCT feature sets avoid this issue up to 20 states, however returns begin to diminish quickly thereafter.

<table>
<thead>
<tr>
<th></th>
<th>HW_Z</th>
<th>HW_D A Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognition Rate</td>
<td>54.77%</td>
<td>55.51%</td>
</tr>
</tbody>
</table>

Table 4.8. *Vector metric performance for HW_Z & HW_D A Z*

![Proposed System: Vector Implementation](image)

Figure 4.10. *Vector metric speaker performance for HW_Z & HW_D A Z*
For almost all cases, the linear combination of two metrics, given the correct weights, outperformed either metric alone. This validates the idea of the vector metric in that different metrics inherently favor different types of relative closeness (i.e., the closeness of specific word paths to the unknown path or the closeness of the general word to the unknown path). Additionally, these differences may then be exploited through combination.

4.4.5. Proposed system: vector implementation with voting

For these tests, the vector metric space was restricted similarly as before.

For the DCT\textsubscript{Z} case, peak performance arose from $\beta = [BFNA_{\infty}, BFNA_{3}]$ with $\alpha = [0.2, 0.8]$, $N = 20$ states, and $M = 2$ mixtures. With the DCT\textsubscript{D\textsubscript{A}Z} feature set, $\beta = [BFNA_{20}, BFNA_{5}]$ with $\alpha = [0.9, 0.1]$, $N = 15$ states, and $M = 2$ mixtures were needed for top performance. Figure 4.11 shows the results of this setup, while the average results are presented in Table 4.9. Figures D.1 and D.2 depict how varying $\alpha$ affects average recognition rates for the DCT\textsubscript{Z} case with $M = 1$ and $M = 2$, respectively. Figures D.3 and D.4 show these same results for the DCT\textsubscript{D\textsubscript{A}Z} feature set.

<table>
<thead>
<tr>
<th>Recognition Rate</th>
<th>DCT\textsubscript{Z}</th>
<th>DCT\textsubscript{D\textsubscript{A}Z}</th>
</tr>
</thead>
<tbody>
<tr>
<td>59.60%</td>
<td>60.13%</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.9. Vector voting performance for DCT\textsubscript{Z} & DCT\textsubscript{D\textsubscript{A}Z}

The best results for the HW\textsubscript{Z} case were attained with $\beta = [BFNA_{10}, BFNA_{1}]$ with $\alpha = [0.7, 0.3]$, $N = 20$ states, and $M = 1$ mixtures, the HW\textsubscript{D\textsubscript{A}Z} case required $\beta = [BFNA_{10}, BFNA_{1}]$ with $\alpha = [0.6, 0.4]$, $N = 15$ states, and $M = 1$. Figure 4.12 shows the results of this setup, while the average results are presented in Table 4.10. Figures D.5 and D.6 depict how varying $\alpha$ affects average recognition rates for the HW\textsubscript{Z} case with $M = 1$.
and $M = 2$, respectively. Figures D.7 and D.8 show these same results for the HW_D_A_Z feature set. All figures are shown for the $N = 15$ and $N = 20$ state cases only.

<table>
<thead>
<tr>
<th>Recognition Rate</th>
<th>HW_Z</th>
<th>HW_D_A_Z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>55.19%</td>
<td>56.98%</td>
</tr>
</tbody>
</table>

Table 4.10. Vector voting performance for HW_Z & HW_D_A_Z

4.5. Comparison

This section compares the baseline system with all variations of the proposed system in a variety of contexts. Although the proposed system competes well with the baseline system when using the HW feature set, it does not compete favorably when the DCT feature sets are used. The baseline system, however, is state of the art with over twenty years of research behind it. The proposed system represents an infant groundwork and, when viewed in this
Figure 4.12. *Vector voting speaker performance for HW\_Z & HW\_D\_A\_Z*

light, performs rather well. Table 4.11 shows the best recognition rates for each system with each feature set, and Table 4.15 gives the metrics used to attain such results.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>AM: Base</th>
<th>AM: Voting</th>
<th>AM: Vector</th>
<th>AM: Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT_Z</td>
<td>68.52%</td>
<td>57.71%</td>
<td>58.76%</td>
<td>58.55%</td>
<td>59.60%</td>
</tr>
<tr>
<td>DCT_D_A_Z</td>
<td>76.70%</td>
<td>59.29%</td>
<td>59.81%</td>
<td>59.81%</td>
<td>60.13%</td>
</tr>
<tr>
<td>HW_Z</td>
<td>57.40%</td>
<td>54.67%</td>
<td>54.56%</td>
<td>54.77%</td>
<td>55.19%</td>
</tr>
<tr>
<td>HW_D_A_Z</td>
<td>62.96%</td>
<td>54.67%</td>
<td>55.40%</td>
<td>55.51%</td>
<td>56.98%</td>
</tr>
</tbody>
</table>

Table 4.11. *Summary of peak performance results*

As one progresses from the basic articulation model implementation to the vector implementation with voting, performance generally increases, albeit incrementally. This implies that room for performance increases exists in this framework, although a significant amount of investigation may be required. As mentioned, the articulation model approach competes closely with the baseline approach for the HW feature set. This leads to the conclusion that
the articulation model approach may be more suited to feature sets which directly represent the visual articulators configuration in some form.

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>AM: Base</th>
<th>AM: Voting</th>
<th>AM: Vector</th>
<th>AM: Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT_Z</td>
<td>100</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>DCT_D_A_Z</td>
<td>100</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>HW_Z</td>
<td>100</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>HW_D_A_Z</td>
<td>100</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 4.12. Number of states for peak performance

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>AM: Base</th>
<th>AM: Voting</th>
<th>AM: Vector</th>
<th>AM: Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT_Z</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DCT_D_A_Z</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>HW_Z</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HW_D_A_Z</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.13. Number of mixtures per state for peak performance

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>AM: Base</th>
<th>AM: Voting</th>
<th>AM: Vector</th>
<th>AM: Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCT_Z</td>
<td>400</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>DCT_D_A_Z</td>
<td>400</td>
<td>40</td>
<td>30</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>HW_Z</td>
<td>400</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>HW_D_A_Z</td>
<td>400</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 4.14. Total number of mixtures for peak performance

Tables 4.12 through 4.14 present comparable parameters of the systems. Table 4.12 shows how many more states are needed in the baseline system versus any of the proposed models. Table 4.14 depicts how the number of mixtures contained in the whole model compare across systems. The more mixtures, the more training data needed to sufficiently train each mixture at the same "level". Table 4.14 thus implies that the articulation model may also be more suited to situations with limited training data than the baseline approach. Additionally, the significantly lower number of states implies a significant computational savings. When trying
to find the best path through the model, a Gaussian log-likelihood must be evaluated at each state. With more states and more mixtures per state, the traditional approach requires more computation.

<table>
<thead>
<tr>
<th></th>
<th>AM: Base</th>
<th>AM: Voting</th>
<th>AM: Vector</th>
<th>AM: Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>(DCT_{Z} )</td>
<td>(BFNA_{\infty} )</td>
<td>(BFNA_{3} )</td>
<td>([BFNA_{\infty}, BFNA_{3}] )</td>
<td>([BFNA_{\infty}, BFNA_{3}] )</td>
</tr>
<tr>
<td>(DCT_{D_A_Z} )</td>
<td>(BFNA_{5} )</td>
<td>(BFNA_{20} )</td>
<td>([BFNA_{20}, BFNA_{5}] )</td>
<td>([BFNA_{20}, BFNA_{5}] )</td>
</tr>
<tr>
<td>(HW_{Z} )</td>
<td>(BFNA_{10} )</td>
<td>(BFNA_{10} )</td>
<td>([BFNA_{10}, BFNA_{3}] )</td>
<td>([BFNA_{10}, BFNA_{1}] )</td>
</tr>
<tr>
<td>(HW_{D_A_Z} )</td>
<td>(BFNA_{10} )</td>
<td>(BFNA_{10} )</td>
<td>([BFNA_{10}, BFNA_{1}] )</td>
<td>([BFNA_{10}, BFNA_{1}] )</td>
</tr>
</tbody>
</table>

Table 4.15. Metrics for peak performance

Also interesting to note is the difference in performance increases provided by the delta and acceleration coefficients for the different systems. For the DCT feature set and the baseline system, the addition of the dynamic features provides a performance boost of almost 6%, while for the articulation model approaches the increases were much more modest. This can be attributed to what information is provided by these extra coefficients. These coefficients help specify the dynamics of the signal and are especially helpful for the left-to-right HMMs because no articulation dynamics are explicitly or implicitly modeled. The articulation model paradigm, however, models these dynamics and, thus, the extra information proved less useful.

As shown in Figures 4.13 through 4.16, the across speaker performance varies similarly for all approaches. Some notable exceptions occur for "Jon" in both HW feature set cases. In both instances, all the articulation models outperform the baseline, and, for the \(HW_{D\_A\_Z} \) case, the performance gap is drastic. Again, this provides encouraging results for the viability of this approach.
Figure 4.13. *Peak system performances for DCT_Z*

Figure 4.14. *Peak system performances for DCT_D_A_Z*
Figure 4.15. Peak system performances for HW_Z

Figure 4.16. Peak system performances for HW_D_A_Z
CHAPTER 5

Conclusions

This thesis presents a novel approach to the isolated video-only automatic speech recognition (V-ASR) problem based on an ergodic hidden Markov model (HMM). While applied here to V-ASR, this approach was formulated independent of the input features, thus enabling its future application in audio-only automatic speech recognition (A-ASR) and audio-visual automatic speech recognition (AV-ASR). By modeling the speech production phenomenon, rather than the spoken symbols, this approach also leads to a more natural attempt at speech recognition than the traditional methods.

By implementing a more compact and natural modeling approach, many advantages over traditional methods are realized. Far fewer states implies far fewer Gaussian mixture models (GMMs) which in turn leads to less computation, but, more importantly, reduces the amount of training data needed to sufficiently train each mixture. With no more than one tenth the number of GMMs used in the traditional methods, the articulation model approach has vastly more data per mixture for the same training set.

Articulation and pronunciation variations between speakers (inter-speaker) and within speakers (intra-speaker) always prove difficult to model using the traditional method. To model these variations, one usually adds more mixtures envisioning that the inter- and intra-speaker variations correspond to extra Gaussian modes. In this way, these variations drastically affect the actual state and mixture models in a traditional system. The articulation model approach, however, separates articulation from interpretation. By doing so,
articulations may be modeled independently of their interpretations, thus allowing for inter- and intra-speaker variations to be addressed in the articulation paths, not in the states of the articulation model itself. This separation should enable a more effective handling of such inter- and intra-speaker variations, although this thesis did not specifically perform experiments related to this task. As a direct consequence of the separation of interpretation and articulation, verbal units with overlapping articulations get modeled concurrently rather than separately. This allows for better representation of each articulation, as opposed to the traditional approach which models each verbal unit separately.

The ergodic approach to speech modeling also leads to speech analysis with fixed temporal granularity but fine resolution as opposed to variable granularity with coarse resolution. This higher resolution approach directly allows for the explicit modeling of speech dynamics, while the traditional approach requires extra features (delta and delta-delta coefficients).

The foundation of the articulation model approach, while still just a foundation, compares admirably with modern state of the art speech recognition systems. While not surpassing these modern systems, this new approach certainly lies within striking distance. The results presented here also show how one may go about improving these results. Although the improvements proved incremental, they serve as a proof of concept that there exists a high performance ceiling for this method. Section 5.1 explores possible refinements to the current model, and Sec. 5.2 presents some future extensions to this modeling approach.

5.1. Future model refinements

As this thesis lays the groundwork for this articulation model approach, many possible refinements to the approaches and algorithms presented may be envisioned.
• **State clustering** The first training step, and possibly the most important, requires the clustering of observations in a rather high dimensional space. Currently, K-Means clustering is used. K-Means relies on a Euclidean distance metric, but perhaps other distance metrics may prove more suited for clustering in the observation space. Additionally, other clustering methods, such as agglomerative techniques which automatically choose the number of clusters, may improve performance while also removing a design parameter.

• **State modeling improvements** In a similar vein as the agglomerative clustering refinement, one may envision an agglomerative-type Gaussian mixture modeling algorithm which detects the optimal number of mixtures. This could drastically improved modeling by allowing states with multi-modal distributions to have enough mixtures while, at the same time, preventing uni-modal distributions from being modeled with multiple mixtures.

• **HMM training refinements** This work presented an iterative method for refining the ergodic HMM at the heart of this approach. However, no ”stopping metric” was developed, and, at present, only one iteration takes place. By determining an appropriate stopping metric, these iterations could improve the state transition estimates as well as the state models themselves by feeding the results back into the GMM training.

• **Path distance metrics** Determining the distance between variable-length sequences (VLS) of discrete states represents a wide open research field. In theory, any VLS algorithm could be used to determine the path distances which lie at the base of interpreting the articulation path and could drastically affect performance.
• **Word distance metrics** In the same way that modifying the path distance metric could improve recognition, so could modifying the word distance metric. Additionally, one may compact the word representation into a model or representative paths and move away from the case-based reasoning approach presented as a foundation in this work.

### 5.2. Future model extensions

While this work has focused on the use of the articulation model in isolated word recognition, a couple future extensions naturally come to mind.

• **Continuous Speech Recognition** The first step towards a completely natural speech recognizer is the extension to continuous speech applications. The biggest hurdle in such a step lies in determining when words start and end. One possible approach may be to use a sliding path window over which word metrics are calculated. Once the word metrics begin to decline, the peak may be marked as the end of the word. A new sliding window may be instantiated from that point forward looking for the next word\(^1\). Research into this extension could prove to be a fruitful research area.

• **Biometric applications** Since this approach models articulations per word, one may just exchange the words for speakers and begin modeling articulations per speaker. Using the exact same methods described in this thesis, the system has now been transformed from speech recognition to a speaker identification system.

---

\(^1\)This idea was suggested by Derek Shiell, and the author thanks him for this insight
References


APPENDIX A

Basic implementation

Figure A.1. Basic system performance for $DCT_Z$ with $M = 1$
Figure A.2. Basic system performance for DCT\_Z with $M = 2$

Figure A.3. Basic system performance for DCT\_D\_A\_Z with $M = 1$
Figure A.4. Basic system performance for $DCT_{D\cdot A\cdot Z}$ with $M = 2$

Figure A.5. Basic system performance for $HW_{Z}$ with $M = 1$
Figure A.6. Basic system performance for $HW_{-Z}$ with $M = 2$

Figure A.7. Basic system performance for $HW_{-D-A-Z}$ with $M = 1$
Figure A.8. *Basic system performance for* $HW_{D\_A\_Z}$ *with* $M = 2$
APPENDIX B

Voting implementation

Figure B.1. Voting system performance for DCT_Z with M = 1
Figure B.2. Voting system performance for $DCT_Z$ with $M = 2$

Figure B.3. Voting system performance for $DCT_{DA}Z$ with $M = 1$
Figure B.4. Voting system performance for $DCT_{D_A}Z$ with $M = 2$

Figure B.5. Voting system performance for $HW_Z$ with $M = 1$
Figure B.6. Voting system performance for $HW_Z$ with $M = 2$

Figure B.7. Voting system performance for $HW_{D_A} Z$ with $M = 1$
Figure B.8. Voting system performance for $HW_{D,A,Z}$ with $M = 2$
APPENDIX C

Vector implementation

Figure C.1. Vector metric performance for DCT\(Z\) with \(M = 1\), varying \(\alpha\)
Figure C.2. Vector metric performance for DCT\textsubscript{Z} with $M = 2$, varying $\alpha$.

Figure C.3. Vector metric performance for DCT\textsubscript{D\_A\_Z} with $M = 1$, varying $\alpha$. 
Figure C.4. Vector metric performance for $DCT_{D\_A\_Z}$ with $M = 2$, varying $\alpha$

Figure C.5. Vector metric performance for $HW_{\_Z}$ with $M = 1$, varying $\alpha$
Figure C.6. Vector metric performance for HW_Z with $M = 2$, varying $\alpha$

Figure C.7. Vector metric performance for HW_D_A_Z with $M = 1$, varying $\alpha$
Figure C.8. Vector metric performance for $HW_{D\_A\_Z}$ with $M = 2$, varying $\alpha$
APPENDIX D

Vector implementation with voting

Figure D.1. Vector voting performance for DCT_ω with $M = 1$, varying $\alpha$
Figure D.2. Vector voting performance for $DCT_Z$ with $M = 2$, varying $\alpha$

Figure D.3. Vector voting performance for $DCT_{D\_A\_Z}$ with $M = 1$, varying $\alpha$
Figure D.4. Vector voting performance for $DCT_{D_AZ}$ with $M = 2$, varying $\alpha$.

Figure D.5. Vector voting performance for $HW_{Z}$ with $M = 1$, varying $\alpha$. 
Figure D.6. Vector voting performance for $HW_Z$ with $M = 2$, varying $\alpha$

Figure D.7. Vector voting performance for $HW_{D_A Z}$ with $M = 1$, varying $\alpha$
Figure D.8. Vector voting performance for $HW_{D\_A\_Z}$ with $M = 2$, varying $\alpha$