

Speech Recognition with Hidden Markov Models in Visual Communication

Tomi Aarnio
Computer Science
Department of Mathematical Sciences
University of Turku, 1999
Master of Science Thesis

AARNIO, TOMI: HMM-pohjainen puheentunnistus visuaalisessa kommunikaatiossa

Pro Gradu -tutkielma, 73 s.

Tietojenkäsittelyoppi

Huhtikuu 1999

Puheella on kaksi eri ilmenemismuotoa: akustinen (ääni) ja visuaalinen (suun liikkeet). Koska muodostuva ääni riippuu huulten ja muiden äänielinten asennosta, vallitsee äänen ja suun liikkeiden välillä tietty yhteys. Tätä yhteyttä on perinteisesti käytetty huuliltalukemiseen.

Kehittynyt tietotekniikka avaa lisää mahdollisuuksia hyödyntää akustisen ja visuaalisen puheen välistä korrelaatiota. Eräs mahdollinen uusi sovellusalue on puheohjattu kasvoanimaatio, jossa akustinen puhe pyritään muuntamaan vastaaviksi suun liikkeiksi. Tätä voitaisiin hyödyntää mm. kielenopetuksessa, elokuvissa, tietokonepeleissä, sekä kuvapuhelimissa ja muissa visuaalisissa kommunikaatiosovelluksissa. Sellaista menetelmää, jolla tarvittava muunnos voitaisiin tarkasti ja luotettavasti suorittaa, ei ole kuitenkaan toistaiseksi löydetty.

Tässä tutkielmassa tarkastellaan erästä kirjallisuudessa esitettyä lähestymistapaa puheohjatun kasvoanimaation toteutukseen. Erityisesti tutkitaan tämän ns. foneettisen menetelmän soveltuvuutta reaaliaikaiseen kommunikaatioon. Menetelmä on kolmivaiheinen:

- (1) tunnistetaan akustisesta puheesta äänteet eli foneemit;
- (2) muunnetaan foneemit vastaaviksi suun asennoiksi eli viseemeiksi;
- (3) muodostetaan viseemien perusteella lopullinen animaatio.

Tutkielmassa tutustutaan lyhyesti kasvomallien muodostamiseen ja niiden animointiin viseemien perusteella (vaihe 3), sekä esitetään foneettisen aakkoston kuvaus viseemeiksi (vaihe 2). Pääosin paneudutaan kuitenkin ensimmäiseen vaiheeseen, eli puheentunnistukseen.

Yleisimmin käytetyt ja tehokkaimmat puheentunnistusmenetelmät perustuvat tilastolliseen mallinnukseen, ja erityisesti ns. kätkeytyihin Markovin malleihin (*hidden Markov model*, *HMM*). Tutkielmassa esitetään HMM:ien matemaattiset perusteet ja tarvittavat algoritmit, sekä sovelletaan niitä käytäntöön. Lisäksi käsitellään HMM-pohjaisen puheentunnistuksen keskeisiä ongelmia ja eräitä niihin esitettyjä ratkaisuja.

Lopuksi toteutetaan puheohjattu kasvoanimaatioprototyyppi ja tarkastellaan saatuja tuloksia. Tuloksista ilmenee, että foneettisen lähestymistavan soveltuvuus reaaliaikaiseen kommunikaatiokäyttöön on huono, koska sen vasteaika suhteessa puheen kulkuun on liian pitkä. Mikäli siis huulten liikkeet ja ääni halutaan synkronoida, on äänen toistoa viivästettävä. Tämä puolestaan on keskustelun sujuvuuden kannalta kiusallista. Tuloksista ilmenee myös, että nyky menetelmillä saavutettava foneemien tunnistustarkkuus (n. 60 %) on liian alhainen: tunnistusvirheiden suuri määrä aiheuttaa vääriä suun liikkeitä.

Asiasanat:

puheentunnistus, foneemien tunnistus, viseemit, tilastolliset menetelmät, Markovin mallit, puheohjattu kasvoanimaatio, visuaalinen kommunikaatio

UNIVERSITY OF TURKU
Department of Mathematical Sciences

AARNIO, TOMI: Speech Recognition with Hidden Markov Models in Visual
Communication

Master's Thesis, 73 pp.
Computer Science
April 1999

Speech is produced by the vibration of the vocal cords and the configuration of the articulators. Because some of these articulators are visible, there is an inherent relationship between the acoustic and the visual forms of speech. This relationship has been historically used in lip-reading.

Today's advanced computer technology opens up new possibilities to exploit the correlation between acoustic and visual speech. One such possibility is speech-driven facial animation, where the aim is to automatically convert acoustic speech into mouth movements. This would have applications in education, movies and computer games, as well as in video telephony and other kinds of visual communication. Unfortunately, a method capable of doing the required conversion accurately and reliably is yet to be found.

In this thesis, one approach will be examined that has been proposed for the acoustic-to-visual speech conversion task. The applicability of the method to real-time communication is studied in particular. This so-called phonetic approach consists of three phases:

- (1) recognizing the basic speech sounds, or phonemes;
- (2) converting the phonemes to corresponding mouth shapes, or visemes;
- (3) generating facial animation based on the visemes

The construction and animation of synthetic head models (phase 3) will be discussed briefly. A mapping from phonemes to visemes (phase 2) will be devised, as well. For the most part, however, we will concentrate on speech recognition.

The most common and successful speech recognition methods are based on statistical modeling, and especially on *hidden Markov models (HMMs)*. In this thesis, the theory of HMMs and the related algorithms will be presented, and then applied to speech recognition. Additionally, the major problems in HMM-based speech recognition will be reviewed.

Finally, a speech-driven facial animation prototype will be developed, and the results analyzed. The results indicate, that the phonetic approach is not very suitable for real-time communication, because it causes too much latency with respect to the progressing speech. In order to keep the animation and the sound synchronized to each other, audio playback must be deferred to compensate for the latency. This, in turn, causes annoying delays in the conversation. Furthermore, the accuracy attainable with current phoneme recognizers (about 60 %) is too low: the errors in recognition are instantly reflected as erroneous mouth movements.

Keywords:

speech recognition, phoneme recognition, visemes, statistical methods, hidden Markov models, speech-driven facial animation, visual communication

Contents

1	INTRODUCTION.....	1
2	FACIAL ANIMATION	3
2.1	HEAD MODEL CONSTRUCTION	3
2.2	SPEECH ANIMATION.....	6
2.2.1	<i>Phonetic foundations</i>	<i>7</i>
2.2.2	<i>Acoustic-to-visual speech conversion.....</i>	<i>10</i>
3	SPEECH RECOGNITION	12
3.1	THE CHALLENGE IN SPEECH RECOGNITION.....	12
3.1.1	<i>Dimensions of difficulty.....</i>	<i>13</i>
3.1.2	<i>Typical recognition tasks.....</i>	<i>14</i>
3.2	SPEECH RECOGNIZER DESIGN.....	15
3.3	SPEECH ANALYSIS	18
3.3.1	<i>Time domain analysis</i>	<i>19</i>
3.3.2	<i>Spectral analysis.....</i>	<i>21</i>
3.3.3	<i>A complete speech recognizer front end.....</i>	<i>23</i>
3.4	LANGUAGE MODELING	25
3.4.1	<i>Definition and requirements.....</i>	<i>26</i>
3.4.2	<i>Methods</i>	<i>27</i>
4	HIDDEN MARKOV MODELS.....	29
4.1	DEFINITIONS	29
4.1.1	<i>Markov chains</i>	<i>29</i>
4.1.2	<i>Hidden Markov models.....</i>	<i>31</i>
4.1.3	<i>Three basic problems.....</i>	<i>33</i>
4.2	PROBABILITY EVALUATION.....	33
4.3	DECODING: FINDING THE BEST PATH.....	36
4.4	TRAINING: ESTIMATING MODEL PARAMETERS	39
4.4.1	<i>Maximum likelihood recognition and training.....</i>	<i>39</i>
4.4.2	<i>Alternative criteria for parameter estimation</i>	<i>42</i>
5	HIDDEN MARKOV MODELS IN SPEECH RECOGNITION.....	44
5.1	ACOUSTIC MODELING	44
5.1.1	<i>Model topology.....</i>	<i>45</i>
5.1.2	<i>Subword modeling units</i>	<i>46</i>
5.1.3	<i>Observation densities</i>	<i>49</i>
5.2	LANGUAGE MODELING	51
5.3	INTEGRATED HYPOTHESIS SEARCH.....	53
5.4	LIMITATIONS OF HMMs IN SPEECH RECOGNITION	56
6	EXPERIMENTS	58
6.1	PROTOTYPE REQUIREMENTS	58
6.1.1	<i>Input speech characteristics</i>	<i>58</i>
6.1.2	<i>Response time constraints</i>	<i>59</i>

6.2	PROTOTYPE IMPLEMENTATION	60
6.2.1	<i>System overview</i>	60
6.2.2	<i>Phoneme recognition</i>	60
6.2.3	<i>Facial animation</i>	62
6.3	PERFORMANCE	62
6.4	DISCUSSION.....	66
6.4.1	<i>Problems in phoneme recognition</i>	66
6.4.2	<i>Problems in facial animation</i>	67
7	CONCLUSIONS	68
	REFERENCES	70

1 Introduction

Historically, audio and video have been treated as two independent streams of information, even though they are often reflections of the same physical event. When a person is speaking in front of a camera and microphone, for example, two forms of information are being captured – acoustic and visual. Although these information sources have vastly different characteristics, they are not completely independent. Otherwise, lip-reading would not be possible.

One application that may exploit this interaction between audio and video is known as *speech-driven facial animation*. Because there is significant correlation between the acoustic and visual forms of human speech, it should be possible to design algorithms capable of analyzing acoustic speech to predict the corresponding mouth shapes. This, if indeed possible, would open up interesting opportunities in various fields, including the following:

- **Education.** When learning foreign languages, for example, a student could see and hear a synthetic human (or a human-like character) on a computer screen to repeat his words. The character could then show how to pronounce those words more correctly. Moreover, computer software in general could use animated user interface agents to guide inexperienced users by spoken instructions.
- **Entertainment.** Synthetic actors in movies or computer games could be made to speak with the voice of a particular person, synchronizing their lip movements to that person's speech. Also, existing video footage could be used to create automatically new video of a person mouthing words that she did not speak in the original footage. This would be useful in movie dubbing, for instance.
- **Communication.** In videotelephony, the transmission of live video of a speaking person could be replaced by animating, at the receiving end, a synthetic look-alike of the speaker according to her speech. This would be useful if real video were not available (no camera at the transmitting terminal) or could not be sent (not enough bandwidth). In some situations, one might not even want to transmit live images, but would rather use a predefined synthetic model. Furthermore, the realism in virtual reality applications could be enhanced if the characters in the virtual worlds were able to speak with each other naturally. Finally, communication aids for the hearing-impaired – or for very noisy conditions – could be implemented by generating lip-readable animation from acoustic speech.

In this thesis, we will examine in detail one approach that has been proposed for synthesizing lip movements from speech. It involves first performing *speech recognition* to identify the string of basic speech sounds, or *phonemes*, that a given utterance consists of. The phonemes are then converted to corresponding visual units (*visemes*) to animate a three-dimensional face model.

To realize such a phonetically based speech-driven facial animation system, we need the following components: (1) a speech recognizer to convert acoustic speech into phonemes; (2) a means to convert phonemes to visemes; (3) a head model capable of being animated; and (4) a system for animating the head model with the visemes obtained from speech. The most crucial and difficult step in this approach is speech recognition: if phonemes can not be recognized accurately enough, the resulting animation is bound to look unsatisfactory.

This thesis is organized as follows. We will start, in chapter 2, with an introduction to three-dimensional head models: how they can be captured, represented and animated. For animation purposes, we adopt a set of visemes, and devise a mapping from a common phonetic alphabet to the chosen visemes. In chapter 3, we give an overview of speech recognition. Even though we do not actually need full-scale speech recognition here, it is useful to get an overall picture of the field. In fact, speech recognition is nothing but phoneme recognition with some postprocessing. Hence, the problems and potential solutions are roughly the same for both.

The standard paradigm in speech recognition is to use *hidden Markov models (HMMs)*. They are a robust mathematical tool for statistical modeling of various kinds of natural processes. In chapter 4, we present the definition of the hidden Markov model, as well as several related algorithms. In chapter 5, then, we apply hidden Markov models to speech recognition. The HMM framework allows convenient and computationally efficient integration of the different recognizer components.

In chapter 6, we present our experimental facial animation system. The prototype consists of an HMM-based phoneme recognizer and a viseme-based facial animation module. We also present the results achieved with the system, in terms of performance, accuracy and subjective quality. Finally, in chapter 7, we summarize the methods and results presented in this thesis, and conclude with directions for further research.

2 Facial Animation

2.1 Head model construction

Several methods of varying quality and complexity exist for capturing three-dimensional models of real-world objects, such as human beings. The most accurate of these methods are based on using lasers, either as range finders or illumination sources. In *laser beam scanning*, a beam of laser is used to measure the distance to a particular point on the surface of an object. Repeating this for thousands or millions of different surface points, possibly using an array of laser beams, the 3D geometry of the object can be accurately captured. A whole-body scanner based on laser range finders is shown in Figure 2-1.



Figure 2-1. A whole-body 3D scanner by *Cyberware* (www.cyberware.com).

In *laser stripe scanning*, a thin laser stripe is projected onto the surface of an object while a photograph is taken from a specific angle with respect to the object. Then, the stripe is moved a very small step forward and a new image is taken. This is repeated until the whole surface of the object has been covered (see Figure 2-2). In the resulting image sequence, the surface contour of the object is illuminated by the laser and can be used to construct a 3D model of the visible part of the object. Several stripe projectors and cameras can be combined so that all parts of the object are visible to at least one camera and one stripe projector.

Not surprisingly, the downside of laser scanning is the price of the required equipment. Fortunately, simpler and cheaper solutions exist that are based on digital cameras and traditional light sources, or no special light sources at all. Using just two digital cameras, placed a few centimeters apart from each other, a *stereo image* of an object can be acquired, similarly to the way our

eyes work. Again, only that part of the object that is visible can be modeled. Multiple cameras can be used to get wider coverage, but at the expense of increasing algorithmic complexity.

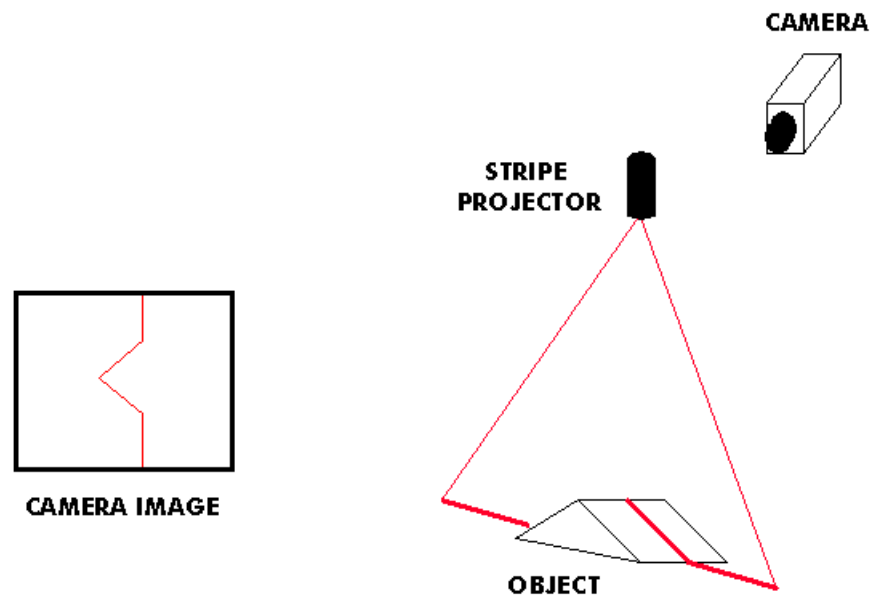


Figure 2-2. 3D scanning with the laser stripe technique. A thin laser stripe is moved along the surface of the object while a sequence of images is taken.

A 3D face model building scheme based on stereo imaging is outlined in Figure 2-3. Depth information is extracted by first identifying prominent regions in either of the images, then locating the same regions in the other image, and finally computing the distance between each pair of corresponding blocks. This knowledge can be used in geometric computations to get a *depth map* of the object. To make depth information extraction easier, *structured light* can be projected onto the surface of the object (see Figure 2-5). Stereo imaging, structured lighting and computer vision in general are discussed in, for example, chapter 24 of Russell & Norvig [27].

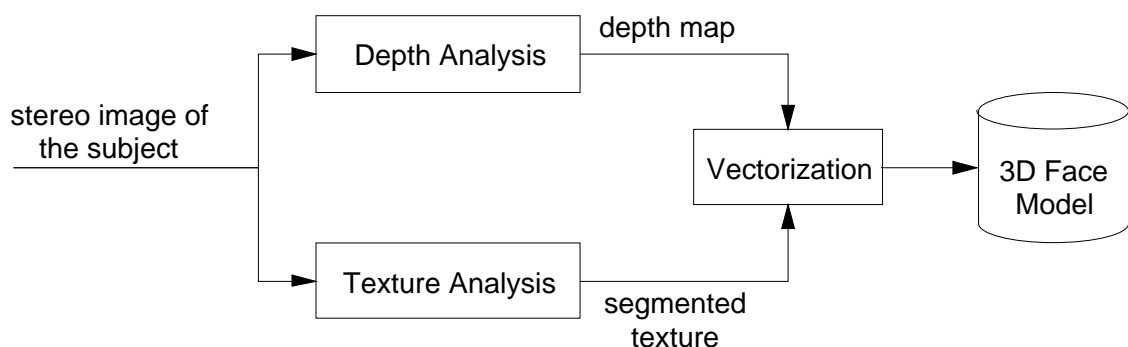


Figure 2-3. Constructing an artificial model of a human face by using stereo imaging. Vectorization is based on both depth and texture information.

Using depth information, whether obtained by stereo imaging or laser scanning, a *wireframe model* of the object can be formed (see the left half of Figure 2-4). The wireframe model consists of points in three-dimensional space, connected together to form a *polygon mesh*. The number of polygons is usually in the order of thousands for complex objects like human faces. The polygon mesh is by no means the only possible representation, although it is the simplest and also the *de facto* standard in 3D computer graphics. More elaborate schemes exist that are based on, for example, parametric surfaces (see, e.g., chapter 6 of Watt [36]).

To make the model look less synthetic, texture information is needed. Texture can be captured simultaneously with structure: all that is needed is a couple of images taken without any special lighting. By mapping the texture on the polygon mesh, we get a very realistic-looking "3D photograph" (see the right side of Figure 2-4).

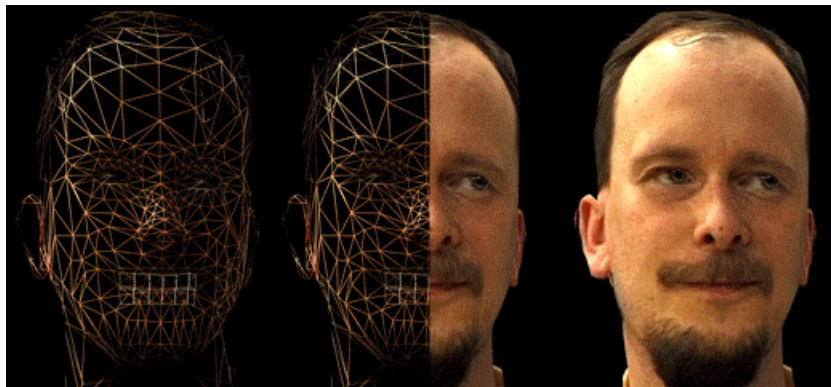


Figure 2-4. A three-dimensional artificial human head. The geometry of the head is defined with a mesh of polygons (*left*). Texture mapping and shading are applied on the polygons to yield a realistic appearance (*right*).



Figure 2-5. Structured lighting
(www.inspeck.com).



Figure 2-6. Facial feature extraction [2].

If we are to animate human faces, the most difficult part still remains: finding the facial features (eyes, eyebrows, lips, nose, chin, and so on; see Figure 2-6). Exact locations of the features must be known so that transformations can be applied on them according to the animation. Finding the lips is particularly important, because the attention of human viewers is naturally focused on their movement. Methods for facial feature extraction are discussed, for example, in Lanitis, *et al.* [16] and Beymer [2].

Facial feature extraction involves several tasks, each of which is rather difficult by itself: first segmenting the texture, then finding correspondences between features and segments, discarding segments that do not correspond to any important features, outlining the inner and outer boundaries of the lips, and so on. Differing lighting conditions and backgrounds further complicate the task. In fact, accurate and robust performance is not possible with current methods, unless some simplifications are enforced (by using a static, single-colored background, for instance).

2.2 Speech animation

Assuming that a three-dimensional model of a human face has been successfully obtained with the methods described in the preceding section (perhaps with some manual assistance, as well), we may now confront the problem of making the head "talk".

Talking heads come in three varieties: *synthesizer-driven*, *audio-driven* and *video-driven*. The first approach uses a speech synthesizer to produce both the speech sounds and the associated lip movements. The latter two, on the other hand, control the animation with information extracted from a real speaking person. In the audio-driven (or *speech-driven*) approach, the speech sounds are used to infer the mouth movements. In the video-driven approach, lip movements are tracked by a video camera and an image analysis system.

In this work, we will only consider facial animation driven by acoustic speech. Synthesizer-driven animation will not be discussed, because synthesized speech is not suitable for video-conferencing and similar applications. Video-driven animation, on the other hand, is not in the scope of this project.

In the following, we will first introduce several concepts of phonetics that are necessary to understand acoustic-to-visual speech conversion (and speech recognition). Then, we will present two different approaches for converting speech into lip movements. The other aims to find a

direct mapping from acoustic observations into mouth shapes, while the other uses an intermediate step of phoneme recognition.

2.2.1 Phonetic foundations

Human speech is produced by the vibration of the vocal cords and the configuration of the vocal tract that is composed of articulatory organs, including the nasal cavity, tongue, teeth, velum and the lips. Because some of these articulators are visible, there is an inherent relationship between acoustic and visual forms of speech.

The main assumption underlying speech-driven facial animation is that the visual component of speech is more or less redundant. That is, it is presumed that sufficient portions of the visual information can be inferred from the acoustic information. Whether this is in fact the case, depends on what is considered sufficient. For the purpose of merely creating an illusion of visual speech, the assumption most certainly holds. Recovering the necessary information from the audio, however, remains a major problem. On the other hand, if we require visual reproduction that is accurate enough to allow lip-reading, the assumption might not always hold anymore.

In this work, we are not aiming for entirely faithful reproduction of visual speech. Rather, an illusion of speech that is accurate enough to fool a casual observer from a distance would be considered sufficient. Even this relaxed goal is by no means an easy one to achieve, especially given that the acoustic-to-visual conversion should take place in real time to enable communication.

Phonemes are defined as the smallest acoustically distinguishable units of speech, or equivalently, the set of speech sounds that is sufficient to describe every possible word in a language. Despite the seemingly clear definition, phonemes are merely abstract units whose pronunciation depends on contextual effects and the speaker's characteristics. A unit called *phone* is defined as the acoustic realization of a phoneme, to differentiate between the linguistic units and the actual speech sounds. Minor variants of phones, in turn, are called *allophones*. As an example of allophonic variations, consider the words *top* and *put*. In *top*, the [t] is aspirated (due to the vowel that follows), but the [t] in *put* is not.

Should a phonetic alphabet differentiate between allophones, and to what extent? Unfortunately, there is no general agreement. Some authors prefer a restricted set of phonemes, while others advocate allophonic alphabets. In any case, about 40 phonemes are needed to describe

the English language. A phonetic alphabet for English that is often used in practice is the TIMIT¹ 48-phoneme set (Table 2-1). Another popular phoneme set for English is the ARPABET, which is approximately the same as the TIMIT set.

Finnish is much more coherent than English regarding the relationship between the written and spoken forms of the language. There are also less phonemes and allophones. As a result, the Finnish phonetic alphabet is easier to define: it is generally agreed to be roughly the same as the written alphabet. The main exception to this rule is that the phoneme [ng] does not have a counterpart in the written alphabet. The same holds for some non-native speech sounds that are fairly common in everyday speech, e.g. [sh] as in *shakki*. Conversely, there are several foreign letters (namely, *c*, *q*, *w*, *x*, *z* and *â*) that do not have their own phonetic representations, but whose pronunciations map to the native Finnish phonemes. For example, *c* maps to [s] in some situations and to [k] in others.

Table 2-1. The TIMIT 48-phoneme set with example pronunciations.

<i>aa</i>	car	<i>ey</i>	ma <u>k</u> e	<i>r</i>	ray
<i>ae</i>	hat	<i>f</i>	far	<i>s</i>	sea
<i>ah</i>	cut	<i>g</i>	agree	<i>sh</i>	<u>sh</u> e
<i>ao</i>	sc <u>o</u> re	<i>hh</i>	hay	<i>t</i>	tea
<i>aw</i>	hou <u>s</u> e	<i>ih</i>	bit	<i>th</i>	<u>th</u> ink
<i>ax</i>	<u>a</u> bout	<i>ix</i>	acc <u>i</u> dent	<i>uh</i>	<u>u</u> h
<i>ay</i>	pie	<i>iy</i>	se <u>a</u>	<i>uw</i>	<u>u</u> w
<i>b</i>	bed	<i>jh</i>	joke	<i>v</i>	voice
<i>ch</i>	<u>ch</u> oke	<i>k</i>	key	<i>w</i>	way
<i>d</i>	day	<i>l</i>	lay	<i>y</i>	yard
<i>dh</i>	<u>th</u> at	<i>m</i>	mill	<i>z</i>	zone
<i>dx</i>	dir <u>t</u> y	<i>n</i>	nine	<i>zh</i>	meas <u>u</u> re
<i>eh</i>	get	<i>ng</i>	sing	<i>cl</i>	<closure>
<i>el</i>	bott <u>l</u> e	<i>ow</i>	bo <u>o</u> t	<i>vcl</i>	<closure>
<i>en</i>	butt <u>o</u> n	<i>oy</i>	bo <u>y</u>	<i>epi</i>	<silence>
<i>er</i>	bir <u>d</u>	<i>p</i>	put	<i>sil</i>	<silence>

¹ TIMIT is a fully transcribed multi-speaker speech database developed by Texas Instruments (TI) and the Massachusetts Institute of Technology (MIT). It is commonly used in developing and evaluating speech recognizers.

Visemes are the visual equivalents of phonemes: the smallest visually distinguishable units of speech. There are far fewer visemes in a language than there are phonemes, because many of the acoustically distinguishable sounds are visually ambiguous. For example, the phonemes [p], [b] and [m] are all produced with the mouth closed. Similarly, [k] and [g] belong to the same visual group. Hence, there is a many-to-one mapping between phonemes and visemes.

Not all phonemes can be classified as unambiguously as the above examples. For instance, the phonemes [k] and [t] may or may not belong to the same visual class. Their difference lies in the place of articulation: [k] is *velar* (tongue is against the soft palate) while [t] is *alveolar* (tongue is against the dental ridge). This difference, however, may not be visible. Some studies classify the two phonemes in the same group while others do not.

As is the case with phonemes, there is no general agreement over the alphabet of visemes in English. One attempt in standardizing a set of visemes for English is included in the MPEG-4 standard [10]. The MPEG-4 viseme set includes 15 mouth shapes: nine for consonants, five for vowels and one for a relaxed mouth position. The classes are defined by one or more phonemes each: for example, viseme #1 is the mouth shape that occurs when pronouncing [b], [p] or [m].

Our classification of the TIMIT 48-phoneme set into the MPEG-4 visemes is shown in Table 2-2. Most phonemes belong to a specific class by definition, but the others have been classified using phonetic knowledge (see Jones [13]), intuition and experimentation. In many cases, there is no single appropriate class for a phoneme: preferring one group to another is largely a matter of taste. Also, diphthongs must be represented by a sequence of two consecutive visemes, and some phonemes have no associated facial expression at all.

Table 2-2. TIMIT phonemes grouped into MPEG-4 visemes. Phonemes that define a viseme are in bold. Note that [oh] is not in the TIMIT set.

MPEG-4 Viseme	Single Phonemes	Viseme Pair	Diphthong
0 (relaxed mouth)	sil	<10, 12>	ay
1 (bilabial; lips closed)	b, p, m	<10, 14>	aw
2 (labio-dental)	f, v	<13, 12>	oy
3 (dental)	dh, th	<13, 14>	ow
4 (alveolar)	d, t, dx		
5 (velar)	g, k, ng		
6 (palato-alveolar)	ch, jh, sh, zh		
7 (blade-alveolar)	s, z		
8 (alveolar)	l, n, el, en		
9 (post-alveolar)	r		
10 (open, neutral lips)	aa, ah, ae, ax, er		
11 (half-open, spread lips)	eh, ey		
12 (half-closed, spread lips)	iy, ih, ix, y		
13 (half-open, rounded lips)	(oh), ao		
14 (half-closed, rounded lips)	uw, uh, w		
skip (no facial expression)	hh, cl, vcl, epi		

2.2.2 Acoustic-to-visual speech conversion

There are two essential requirements that speech-driven facial animation must fulfill in order to look satisfactory. First, the mouth movements must be synchronized with the speech. The mouth must start moving when the sound starts, and stop moving when the sound stops. Second, the mouth movements must imitate those of a real, speaking person. For example, when the sound [uw] is uttered, the lips should be half-closed and rounded. Of these two criteria, the first is perhaps the more important one, because synchronization errors can be easily seen even without knowing the language. Minor errors in mouth shape, such as having the mouth wide open when it should be rounded, go more easily unnoticed. (This can be experimentally verified by watching movies dubbed in a language you do not know.)

There are two primary approaches to acoustic-to-visual speech conversion. In the *functional* approach, one attempts to find an optimal mapping from the space of acoustic observations into the space of parameters that describe the mouth shape. Using audio-video footage of speaking people, a mapping from acoustic observations into mouth shape parameters can be devised as follows.

Short intervals (~10 ms) of speech are analyzed to yield a set of acoustic features that describe the sound during the analysis interval. Meanwhile, the video images corresponding to the same

interval are analyzed to find values for a selected set of parameters that describe the mouth shape. With large amounts of training material, correlations between the acoustic and visual observation spaces can be identified, and the required statistical predictor function formed. The function can be realized using, for example, hidden Markov models or artificial neural networks. This type of acoustic-to-visual speech conversion is discussed by, e.g., McAllister, *et al.* [21], Lavagetto, *et al.* [17, 18], Rao, *et al.* [26] and Luo & King [20].

The functional methods work very well, but there are some drawbacks, too. First, rather large amounts of video footage must be collected. Second, image analysis techniques must be applied (and developed) that extract mouth shape parameters from the video so that associations between acoustic and visual observations can be established during the system training phase. Third, the mapping may not always work very well for people other than those in the training videos, or in different acoustic conditions. This is perhaps because only very low-level analysis of the speech signal is performed, compared to speech recognition systems (or humans).

In the *phonetic* approach to audio-to-visual speech conversion, the transformation is done via the intermediate step of *phoneme recognition*. The phonemes, in turn, are converted into visemes (as described in the previous section). The advantage of this approach is that the information present in the speech signal (and even other linguistic information) can be utilized fully. This is helpful in abstracting over variations in speakers, environmental conditions, speaking styles, and so on. Another advantage is that no video footage or image analysis systems are needed. Also, speech recognizers are publicly available, so that they only need to be trained or adapted to the task at hand, instead of building a system up from scratch.

The downsides of the phonetic approach include the fact that no current phoneme recognizer is able to attain very high levels of recognition accuracy for unrestricted speech. The best results so far (for English) are no better than about 60% in accuracy. This means that there will always be errors in the recognized phoneme string, which will reflect to the visemes and consequently to the observed visual quality. Another problem is caused by delays in recognition: a phoneme can not possibly be recognized the moment it starts, but only after sufficient acoustic data has been obtained. This has implications for real-time applications.

A phonetic approach to audio-to-visual speech conversion has been adopted by, for example, Bothe [3] and Jones & Dlay [12]. The visual quality of their systems is apparently good. Their methods are not geared towards communications applications, however, and do not operate in real time.

3 Speech Recognition

Speech recognition is the process of transcribing acoustic speech into text – or more generally, into a sequence of labels. That talent comes so naturally for us humans that the difficulty of endowing a computer with an equal capability has repeatedly been underestimated.

Speech recognition systems were built in the 1950's for vowel recognition and digit recognition, yielding creditable performance. It was thought that these results could be extended in a natural way to more sophisticated systems. Unfortunately, the techniques did not scale up – a situation all too familiar from many other frontiers of artificial intelligence. The real world with its immense proportions and diversity proved impossible to handle for techniques that were developed for simple, constrained tasks.

In the first section of this chapter, we will discuss some of the reasons that make speech recognition so difficult. Then, in section 3.2, we will introduce a mathematical formulation of the speech recognition problem. The formulation is based on probabilities, leading to a convenient decomposition into easier to treat subproblems: *speech analysis*, *language modeling* and *acoustic modeling*. The first two subproblems will be introduced briefly in sections 3.3 and 3.4. Acoustic modeling will be discussed in more detail in chapter 5.

3.1 *The challenge in speech recognition*

At first glance, speech recognition might not seem so difficult. The definition does not even appear to require that the spoken words be understood in any way. Solving the problem should be a matter of deciding which phonemes were uttered and then looking up the corresponding words from a pronunciation dictionary.

Unfortunately, there is much more variability in speech than one might think. The variability is caused by different speakers, speaking styles, environments, vocabularies, task constraints, and so on. No current method is capable of dealing with all these sources of variability; the task must somehow be simplified to reduce variability. Of course, these simplifications also limit the type of input that the system can handle, so there is a tradeoff involved. In the following, we will present several dimensions along which variability can be reduced, and along which the intrinsic difficulty of a specific recognition task can be evaluated. We will conclude by characterizing several typical tasks in terms of these dimensions.

3.1.1 Dimensions of difficulty

The most influential decision in speech recognizer design is whether to allow *continuous* speech or to require pauses between words. Continuous speech is much more difficult to recognize than discrete words, for three main reasons. First, word boundaries are very hard to detect. Second, the pronunciation of a phoneme may depend on the preceding and following phonemes in a complex manner (this is called *coarticulation*). Third, articulation in everyday speech is not very precise to begin with: common words are abbreviated, vowels are formed sloppily, and so on.

As an example that illustrates why continuous speech is so difficult to recognize, consider the words "did you". Articulated clearly, with a silence in between, they are pronounced [d ih d sil y uw]. In fluent speech, however, a common pronunciation is [d ih jh ax]. All of the three phenomena discussed above occur in this example. First, the words are blended together so that there is no silence between them. Second, coarticulation transforms [d y] into [jh]. Last, [uw] is replaced with the neutral vowel, that is easier to articulate quickly. These changes make no difference for a human listener, but a computer algorithm will have a hard time finding a matching word for [d ih jh ax] in a pronunciation dictionary. Worse still, the phonemes may not be very accurately recognized to begin with, making the task even more difficult.

Significant variability in acoustic speech is caused by variations in people's voices and speaking styles. For example, the voice and speaking style of a ten-year-old girl are remarkably different from those of a man in his seventies. *Speaker-dependent* recognizers are trained exclusively for a single speaker, whereas *speaker-independent* systems attempt to handle input from anyone. The former need only consider within-speaker variabilities (such as the speaking rate or physiological state), whereas speaker-independent systems must also abstract over variabilities among different people: dialect, gender, vocal tract features, and so on. Obviously, speaker-dependent systems are potentially much more accurate.

Perhaps the most evident source of performance degradation in speech recognition is *noise*. Noise can be classified as either *environmental* (traffic, rain, other people talking) or *speaker-induced* (coughing, sneezing, swallowing, breathing, chewing a gum). Additional noise is introduced in the recording and digitization process (static hum, quantization error). If a recognizer is to be used in noisy conditions, noise compensation and elimination schemes must be explicitly employed in its design. Also, the speech data used in training the system should be noisy; a system trained only on clean speech will not perform very well when noise is introduced.

Another important factor in recognizer performance is the *vocabulary*. It is rather easy to correctly recognize a sentence if there are only a few possible words to choose from. If there are, for example, 10 words in a vocabulary, then there are 10^N possible sentences of N words. Searching through all of them is a realistic possibility for small N . Further, it is possible to separately design an acoustic model for every word, if there are not too many of them. With large vocabularies, on the other hand, an exhaustive search through all possible sentences is out of question, as is constructing word models by hand. Most commercial applications of speech recognition are based on a small vocabulary (in the order of 100 words), but large vocabulary systems are emerging.

The number of possible sentences can be restricted by imposing syntactic constraints using a *grammar*. For example, a grammar can be defined that only accepts command phrases. A grammar that restrictive would reduce the search space dramatically, but at the expense of severely limiting the style of input. For general purpose dictation and many other tasks, a grammar must allow more freedom in formulating sentences. Ideally, ungrammatical phrases should be permitted as well.

The combined effect of vocabulary size and grammar can be measured with *perplexity*: the average number of words that can occur at any decision point (see Jelinek [11] for a precise definition). The greater the perplexity, the harder the task.

Table 3-1. Dimensions of difficulty in speech recognition.

	EASIEST TASK SPECIFICATION	HARDEST TASK SPECIFICATION
Continuity of speech	isolated words	normal fluent speech
Speaker variability	single known user	multiple unknown users
Amount of noise	clean speech	speech corrupted by noise
Vocabulary size	small	large or unlimited
Sentence structure	restricted	unrestricted

3.1.2 Typical recognition tasks

Speech recognition systems and tasks can be characterized along the described dimensions of difficulty (summarized in Table 3-1). For example, a database front end for processing spoken queries ("*Show all employees with salary greater than ten thousand*") should be speaker-independent and allow continuous speech. On the other hand, the vocabulary need not be very large, and a restrictive grammar can be employed to reduce the number of possible interpretations. This type of systems are already commercially available and in productive use.

Another commercial application of speech recognition is voice dialing, used in some mobile phones. The perplexity of a voice dialing application is typically rather low, due to a restrictive grammar and a small vocabulary. What makes the task challenging is the requirement to recognize fluently spoken digit strings and commands in environments with different kinds of noise.

Recently, several commercial large-vocabulary dictation systems have also been introduced. These systems are basically speaker-independent, but often require an adaptation period for each new user. Vocabulary size in dictation systems is typically in the order of 10000 words, but with only a subset of the vocabulary active at a time. Continuous speech is generally allowed as input, but recognition accuracy is better if words are articulated more clearly than usual.

Finally, humans are capable of accurately recognizing fluent speech in noisy environments, spoken by anyone, with a potentially unlimited vocabulary and a very diverse sentence structure. Performance of that level is far beyond the capabilities of any current speech recognizer, and is likely to remain so for several more years.

3.2 *Speech recognizer design*

Early speech recognizers used a method known as *template matching*. It is based on perhaps the most obvious idea of solving the speech recognition problem. First design a *prototype* (or *template*) for each word in the vocabulary, then compare the unknown input utterance to the prototypes and find the best match. A word template can be derived by recording multiple spoken instances of the word and then averaging them in some suitable way.

This simple scheme produced very good performance for isolated word recognition with small vocabularies. Unfortunately, it does not scale up to handle continuous speech and long sentences. It would be impossible to even construct a template for every word in a large vocabulary, not to mention the computational cost of comparing the input with all the templates. The lack of detectable word boundaries would make the search space astronomically large.

In order to derive a more formal and scalable solution than template matching, we need to define the speech recognition problem mathematically. The fundamental insight behind the definition, to be presented shortly, is that *words cause signals*. The speech recognition problem, then, is to diagnose the cause of a perceived signal – that is, the spoken words. The problem is hard because the causal relationship between words and signals is nondeterministic, involving many uncertain factors. These are best dealt with using a probabilistic formulation, as follows.

Let \mathbf{A} denote the acoustic evidence (percepts) based on which the recognizer will make its decision about which words were spoken. Without loss of generality, we may assume that \mathbf{A} is a sequence of symbols drawn from some (possibly very large) alphabet \mathcal{A} :

$$\mathbf{A} = a_1, a_2, \dots, a_m \quad a_i \in \mathcal{A}. \quad (3.1)$$

The symbols can be thought to have been generated in time, as indicated by the index i . Let

$$\mathbf{W} = w_1, w_2, \dots, w_n \quad w_i \in \mathcal{Q} \quad (3.2)$$

denote a string of n words, each belonging to a fixed and known vocabulary \mathcal{Q} . The task of the recognizer, then, is to find the most probable word string, given the acoustic evidence. In other words, the recognizer should decide in favor of a word string satisfying

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} P(\mathbf{W} | \mathbf{A}), \quad (3.3)$$

where $P(\mathbf{W}|\mathbf{A})$ denotes the conditional probability of \mathbf{W} , given \mathbf{A} . The above formula can not be evaluated as such, because there are just too many possible pairs (\mathbf{W}, \mathbf{A}) to allow finding the most probable one in any reasonable time. However, using the well-known Bayes' formula, the right-hand side probability can be rewritten as

$$P(\mathbf{W} | \mathbf{A}) = \frac{P(\mathbf{W})P(\mathbf{A} | \mathbf{W})}{P(\mathbf{A})}, \quad (3.4)$$

where $P(\mathbf{W})$ is the prior probability that the word string \mathbf{W} will be uttered, $P(\mathbf{A}|\mathbf{W})$ is the probability that when \mathbf{W} is uttered the acoustic evidence \mathbf{A} will be observed, and $P(\mathbf{A})$ is the prior probability of observing \mathbf{A} . Because $P(\mathbf{A})$ is constant with respect to the maximization in (3.3), its value need not be computed. This simplifies the recognizer's task to find the following:

$$\hat{\mathbf{W}} = \arg \max_{\mathbf{W}} P(\mathbf{W})P(\mathbf{A} | \mathbf{W}). \quad (3.5)$$

Now, what do we need to solve the task? First, we need to decide what the alphabets \mathcal{A} and \mathcal{Q} will look like. Then, we need to find means to compute $P(\mathbf{W})$ and $P(\mathbf{A}|\mathbf{W})$ for individual \mathbf{W} and \mathbf{A} . Last, we need a search algorithm, guided by the computed probabilities, for finding the maximizing word string.

Choosing the alphabet \mathcal{A} amounts to designing a *front end* that converts the pressure waveform, which is what sound is, into the symbols a_i that the recognizer can deal with. The front end thus includes a microphone, a means of sampling the signal, and a method for processing the se-

quence of samples to produce the symbols a_i . Speech analysis methods and front end design will be discussed in the next section.

The nature of the speech recognizer's output is determined by the vocabulary \mathcal{V} . Usually, the vocabulary consists of ordinary words found on a dictionary. However, the words may as well be labels of any kind that suit a particular purpose. For example, the vocabulary might consist of syllables or phonemes (as is the case in this work), or of larger units such as short phrases.

The most critical stage in speech recognition is estimating the probabilities $P(\mathbf{A}|\mathbf{W})$ for all possible pairs of \mathbf{A} and \mathbf{W} . Clearly, the number of combinations is far too large to permit the values to be precomputed and stored in a table. The probabilities $P(\mathbf{A}|\mathbf{W})$ must therefore be computable at run time. For this purpose, we need an *acoustic model* of the speaker's interaction with the front end. This interaction includes factors like pronunciation, environmental noise, the nature of processing performed by the front end, and so on. Acoustic modeling will be discussed in chapter 5, after having presented the basic theory of hidden Markov models.

Apart from the acoustic model, we need a *language model* to provide the recognizer with values of $P(\mathbf{W})$ for all \mathbf{W} . Again, the values cannot be computed accurately but must be estimated at run time, using, for example, statistical techniques. Language modeling on a general level will be discussed in section 3.4. Later, in chapter 5, we will apply certain language modeling techniques to the hidden Markov modeling framework.

Finally, a *search algorithm* for finding the maximizing word sequence must be developed. An exhaustive, unguided search through all possible \mathbf{W} is certainly out of question. Thus, some kind of heuristics are needed to focus the search on the most promising alternatives – those that are in some way suggested by the acoustic evidence and are not considered impossible by the language model. The search algorithm to use depends on what kind of acoustic models and language models are employed. If both models are based on HMMs, the search can be conducted using a very efficient technique known as the *Viterbi algorithm* (to be introduced in chapter 4).

The described mathematical formulation of speech recognition gives rise to the interpretation given in Figure 3-1, due to Jelinek [11]. The human speaker is seen as consisting of two parts: the brain and the vocal apparatus. The message \mathbf{W} is first formulated in the mind, and then sent to the speech producer, which transforms the words into sound waves. At the speech recognizer, the acoustic processor analyzes the sound in order to produce a sequence of acoustic evidence \mathbf{A} . Finally, this evidence is processed by the linguistic decoder to find the most likely word string $\hat{\mathbf{W}}$.

In this communication theoretic view, the speaker's mind is seen as the *source*, the linguistic decoder as the *receiver*, and the speech producer and acoustic processor together as the *channel*. The receiver models the source, using a language model, in an attempt to predict the message \mathbf{W} . It also models the channel, by means of an acoustic model, to find out the channel's transmission characteristics (that is, how a message may be modified and distorted along the way).

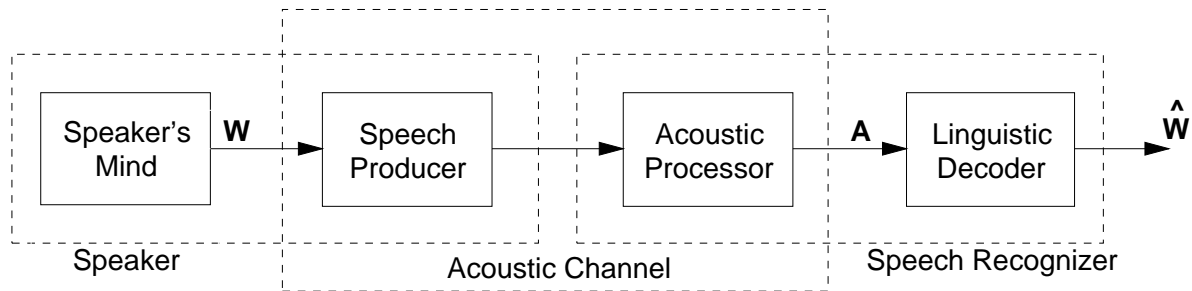


Figure 3-1. The source-channel model of speech recognition.

3.3 *Speech analysis*

In principle, a pure digitized speech waveform could serve as the input to a speech recognizer. The waveform, however, contains information that is considered redundant or even harmful for speech recognition purposes. The perceptually relevant features of the signal are effectively hidden beneath the massive amounts of samples (ten thousand or more per second). The computational load of processing the raw data would also be prohibitive. Encoding schemes are therefore needed that reduce the amount of data, while retaining and even enhancing the relevant features. The encoding process – known as *speech analysis*, *speech coding* or *feature extraction* – forms the *front end* of a speech recognizer.

There exist a variety of techniques to extract features from a speech signal. The most simple ones are those that operate directly on the waveform, that is, in the time domain. These include energy measurement and autocorrelation analysis, which will both be described in section 3.3.1.

Very useful information can be derived from a speech signal by first transforming it into the frequency domain. The result of such transformation is the *spectrum* of the signal, and it displays the amount of energy at selected frequency bands. In section 3.3.2, we will discuss spectral analysis and how it can be used in classifying speech sounds.

In the last section, the processing stages in a typical speech recognizer front end will be outlined. A front end usually performs not only one, but several types of analyses, and consequently produces several different types of features, as well.

Speech analysis is a science of its own with hundreds of publications each year. It is not possible to cover, even superficially, the whole field in just a few pages. We will thus present, on a somewhat abstract level, only those ideas that constitute the most essential background for this work. Most of the material in this chapter is collected from Rabiner's articles and books [24, 25, 29]. To keep the presentation concise, references to those sources will be omitted.

3.3.1 Time domain analysis

One of the simplest representations of a signal is its *energy*. Denoting a discrete-time signal by a sequence of samples $x(n)$, the energy of the signal at time n can be defined as

$$E(n) = \sum_{m=0}^{N-1} (x(n-m))^2, \quad (3.6)$$

where N is the number of most recent samples to include in the energy calculation. The function is hence computed over a finite number of consecutive samples, or a *frame*. As more samples of the signal are received, the frame is moved forward and the function re-evaluated. The same concept is used in other analysis methods, as well. Frame lengths and step sizes of 5-30 milliseconds are commonly used, with a short overlap between adjacent frames (see Figure 3-2).

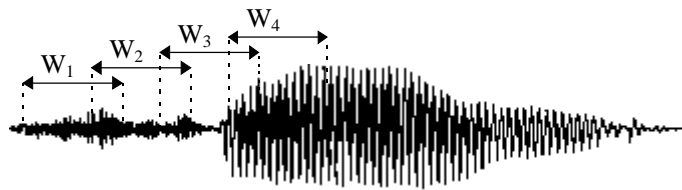


Figure 3-2. A segment of speech that consists of an unvoiced sound followed by a voiced sound. Framing is performed with roughly one-half total overlap between successive frames.

Windowing (or *weighting*) can be applied to a frame to minimize signal discontinuities at the beginning and end of the frame. The most common weighting scheme is the *Hamming window*, which is illustrated in Figure 3-3. Samples at both ends of the frame are tapered towards zero by applying small weights, whereas the samples near the middle are scaled down only slightly.

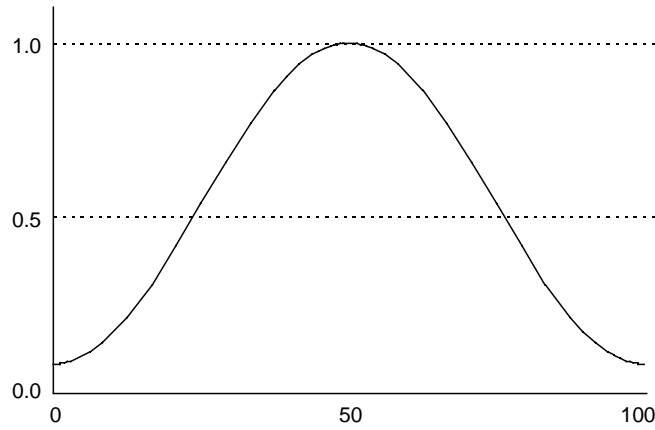


Figure 3-3. A typical weighting function known as the Hamming window. In this example, the frame length is 100 samples.

The idea behind the energy function $E(n)$ is that it displays the time varying amplitude properties of the speech signal. A segment of speech with high amplitude has high energy, and a low-amplitude region has low energy. Because voiced sounds (such as the vowels) have considerably higher energy than breathed sounds, the energy function is a reliable aid in classifying a sound as either voiced or unvoiced.

A more elaborate time domain analysis method is the *autocorrelation*. The autocorrelation function of a signal $x(n)$ is generally defined as

$$\varphi(m) = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{n=-N}^N x(n)x(n+m). \quad (3.7)$$

In speech processing, autocorrelation is computed over a segment of samples at a time, as in energy measurement. The function is useful in displaying structure in a waveform. For example, if a signal is periodic with period P , i.e. $x(n) = x(n+P)$ for all n , then it can be shown that $\varphi(m) = \varphi(m+P)$. Periodicity in the autocorrelation function thus indicates periodicity (with the same period) in the signal. The periodic peaks are usually far more evident in the autocorrelation function than in the original signal. On the other hand, lack of predictable structure in the signal is indicated by an autocorrelation function that is sharply peaked around $m = 0$ and falls off rapidly to zero as m increases. Sondhi [31] gives a decision algorithm that formalizes the process of locating these peaks from a sequence of correlation functions.

Periodicity and lack of structure are important criteria in voiced-unvoiced classification, because voiced sounds are periodic (see the figure below) and breathed sounds are not. Autocorrelation also helps in estimating the fundamental frequency of a periodic sound, allowing the sound to be classified more accurately.

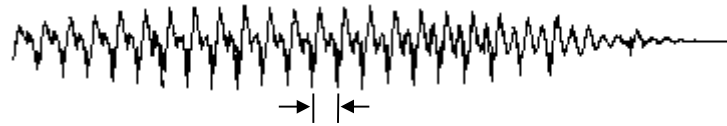


Figure 3-4. A periodic (voiced) interval of speech. The arrows indicate the fundamental period of the waveform.

3.3.2 Spectral analysis

The most important information to be extracted from a speech signal is the *short-time spectrum*, which specifies the energy of the signal over a range of frequencies during a given interval of time.

The basic idea in short-time spectrum analysis is that although speech is not a stationary signal, it is, however, relatively constant over short intervals (in the order of centiseconds). A sequence of short-time spectra can thus represent the signal quite faithfully. Such a sequence can be graphically displayed as a *spectrogram*, which is a time-frequency plot of signal energy (see Figure 3-5).

Each phoneme is characterized by a distinguished set of spectral properties. These properties are, for the most part, visible in a spectrogram display. It is therefore possible to identify distinct phonemes by inspecting a spectrogram. In fact, it is possible for a human expert to decipher an unknown utterance by means of spectrogram reading (see Zue [37]). This means that a spectrogram contains, indeed, all the acoustic information needed for speech recognition purposes.

As an example of spectrogram reading, consider the section of speech after two seconds from the start in Figure 3-5. The signal's energy at that time is concentrated on the upper frequencies (3-5 kHz), with no energy at all below 1 kHz. This indicates that either the unvoiced fricative [s] or its voiced counterpart [z] was uttered; a phonetician could tell the difference. The vowels, on the other hand, manifest themselves as relatively long, dark bands concentrated under 3 kHz; see for example the long [iy] in "knees" (between 1.8 and 2.0 seconds).

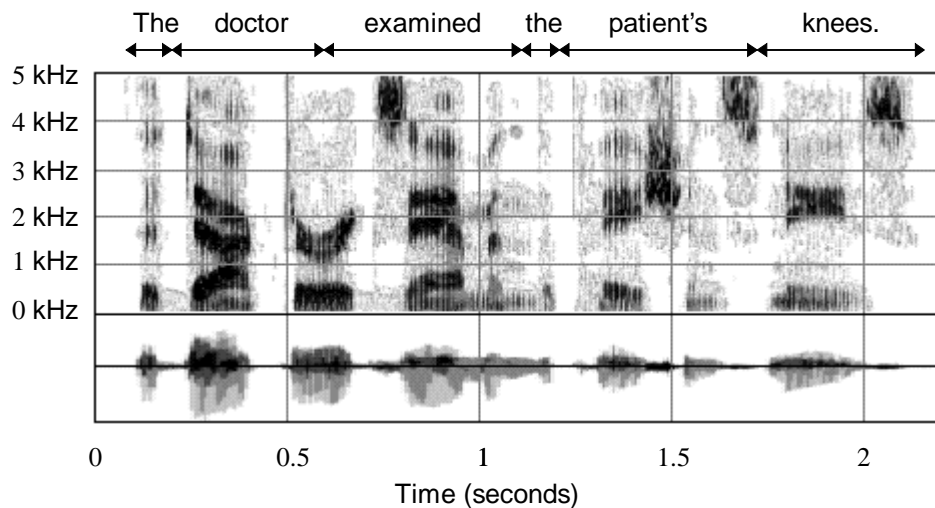


Figure 3-5. The waveform and spectrogram of an utterance [23]. Signal energy is represented by shades of gray: dark color indicates high energy.

Even though the spectrogram is a good representation of the speech signal in the sense that it contains all the necessary acoustic information, it is rather fuzzy and obscure in character, as can be seen from the above figure. Various kinds of smoothing operations and further transformations can be applied on the spectrum to make the critical features more concrete. These operations do not necessarily make any difference for a human, because our brain is good at interpreting fuzzy images by nature, but algorithmic processing may become more tractable.

The human auditory system is also based on spectrum analysis, but in a more complex manner than the traditional algorithmic methods. One basic observation is that human perception of high-frequency sounds does not follow a linear, but a logarithmic scale. For example, a sound with a pitch of 4 kHz is perceived to be *less* than twice as high as a 2 kHz sound. This knowledge has been utilized in speech analysis by, for example, designing perceptually based frequency scales.

A recent trend in speech analysis is the attempt to emulate the human auditory system more accurately. This requires very detailed knowledge of how the human ear works. If the human ear could be accurately replicated, it is reasonable to expect that such a system would far surpass the conventional techniques in performance. Unfortunately, not nearly all of the needed knowledge to replicate the human auditory system is available yet.

One of the first attempts in human auditory modeling was made by Seneff [30]. Since then, the idea has attracted an ever-growing amount of interest. Today, the best auditory-based methods already provide better performance than conventional front ends, especially in noisy conditions (see Rabiner & Juang [24], 134-139).

3.3.3 A complete speech recognizer front end

A speech recognizer front end accepts as input a sequence of speech samples and produces a sequence of *feature* (or *observation*) *vectors*. Each observation vector attempts to characterize the sound that occurred during the interval that it corresponds to. Perceptually similar sounds should produce similar observations, while dissimilar sounds are supposed to produce dissimilar observations.

In a *time-synchronous* front end, the analysis interval is fixed in length, whereas a *segmenting* front end attempts to segment the input signal into acoustically homogeneous intervals of varying length. Time-synchronous front ends are prevailing in practical systems. This is partly because fixed-length frames are easier to handle with current acoustic modeling methods (particularly HMMs), and partly because segmentation is difficult as such (and thus not always reliable).

The basic processing steps in a typical time-synchronous speech recognizer front end include the following (see Figure 3-6):

1. **Framing and windowing.** The incoming waveform is segmented into frames of N samples each. Adjacent frames are separated by M samples, with $M \leq N$. The samples contained in a frame are weighted by a Hamming window, or similar, to reduce signal discontinuities.
2. **Feature extraction.** Several types of features are extracted from a frame. Typically, these include a number of spectral parameters that specify the signal energy at selected frequency bands, and an overall energy measure.
3. **First derivative of features.** The basic features alone are not adequate for phonetic classification: often the characteristics of a signal at a particular time are less important than the way these properties are changing. The differentials are usually computed with respect to both of the neighboring frames, not only the preceding frame. This yields better estimates, but at the expense that the values can be computed only after the next frame has been analyzed.
4. **Second derivative of features.** Acceleration of change is crucial for detecting very rapidly changing features. Acceleration is usually computed with respect to frames two or three steps away in both directions. Of course, this causes further lag in output production.

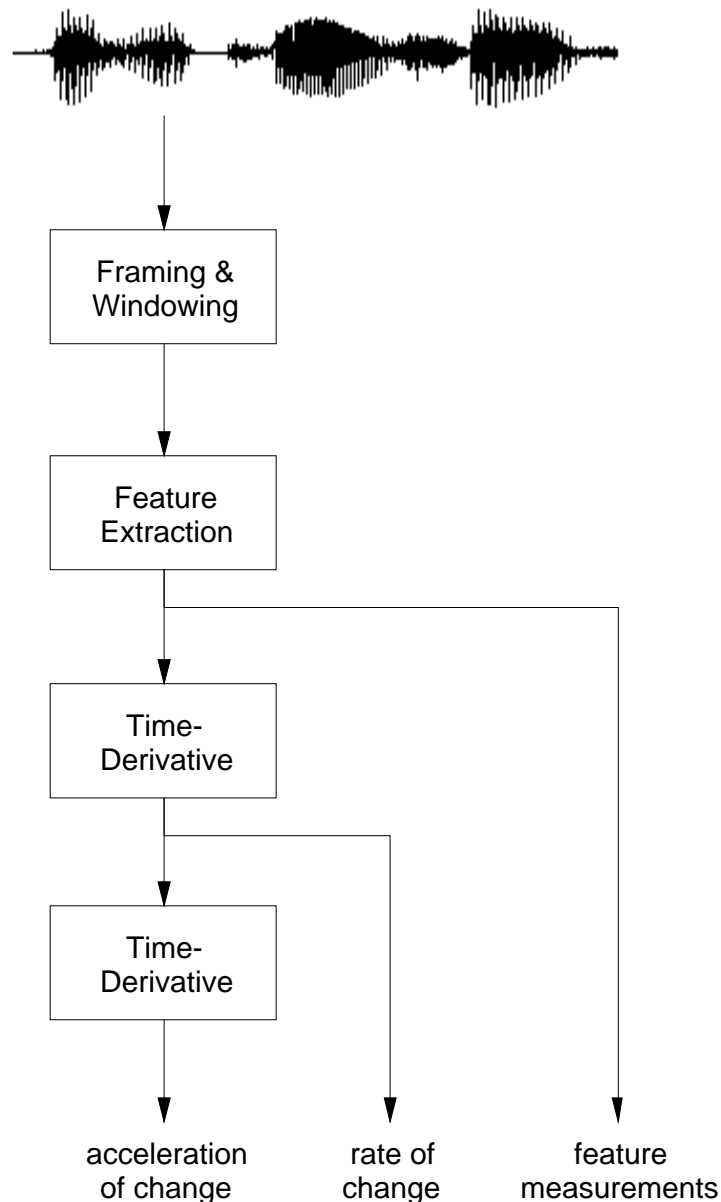


Figure 3-6. Simplified architecture of a typical, time-synchronous speech recognizer front end. Given a waveform, the front end outputs a sequence of feature vectors at fixed intervals.

Feature vectors can be sent to the acoustic modeler as such, if the modeler is capable of handling vector-valued observations. If this is not the case, however, the observations must first be processed by a *vector quantizer*. Vector quantization is the process of mapping a vector into a single integer, as briefly described below (see Gray [7] for a complete introduction).

Let the dimension of the observation vectors be L . The vectors can thus be regarded as points in the L -dimensional observation space. By assumption, points that represent similar sounds will be located close together, while points representing different sounds will be located farther away

from each other. *Clusters* of points will thus be formed in the observation space, with each cluster (ideally) representing a distinct type of sound.

In vector quantization, the observation space is partitioned into non-overlapping regions according to the clusters of points. A representative vector, called a *codeword*, is then chosen for every region. Usually, the centroid of a region is selected as the representative. Collectively, the codewords form a *codebook*. Once a codebook has been constructed, the actual mapping of a given vector into a single integer is easy: just replace the vector with the codebook index of the closest codeword.

Of course, vector quantization in general is not reversible, and important information may be lost in the process. To avoid this unnecessary loss of information, most speech recognizers today are capable of handling vector-valued observations directly. Nonetheless, the use of vector quantization is sometimes justified as a means of data compression or of reducing the running time of the acoustic modeler.

3.4 Language modeling

Acoustic modeling of phonemes (or some other speech units) is not sufficient by itself, if word-level transcriptions of spoken sentences are needed: there must also be some means to combine the recognized tokens into words, and words into sentences. This is a very difficult task for continuous speech (see section 3.1.1), even if the phonetic transcriptions were 100 % correct. Yet, the transcriptions are far from accurate. There is no way that the spoken words can be recovered from error-ridden phonetic transcriptions without using some external information. That information – linguistic knowledge, statistics – is contained in the *language model*, which guides the recognizer by ranking hypothesized word strings according to how likely they are to occur.

The role of language modeling in practical speech recognizers is remarkable. In fact, not many of the existing commercial speech recognition applications would be possible without powerful language models and restrictive grammars.

In this section, we will first describe the properties of a good language model and then present a mathematical formulation that supports those characteristics. Then, we will briefly evaluate different methods that have been proposed for language modeling.

3.4.1 Definition and requirements

The task of a language model is to assign a probability to each of a (possibly infinite) number of word strings. Mathematically, the language model should estimate $P(\mathbf{W})$ for all word strings \mathbf{W} . The value of $P(\mathbf{W})$ is interpreted as the prior probability that the speaker will say the string \mathbf{W} , and is used to guide the search of the recognizer among competing (partial) text hypotheses.

How to derive the required probability estimates, then? At first, it might seem that a conventional grammar would do: all ungrammatical sentences would be assigned a probability of zero. But what probability should be given to the valid sentences? Clearly not all of them are equally likely. Moreover, a recognizer that simply rejects all ungrammatical sentences would be very inconvenient to use: just one hesitation or mispronunciation in mid-sentence, and one would have to start all over again. No intermediate recognition results could be provided, either, because a parser needs a whole sentence to do anything. Therefore, traditional parsers can at best serve as final filters for word strings that were arrived at using a more appropriate language model (for example, Tsukada, *et al.* [33] propose re-evaluation of competing final hypotheses using a context-free grammar).

A more suitable language model can be developed by decomposing the probability estimation problem into parts. Using elementary rules of probability theory, the language model's probability $P(\mathbf{W})$ can be written as

$$P(\mathbf{W}) = P(w_1)P(w_2 | w_1)P(w_3 | w_1, w_2) \cdots P(w_n | w_1, w_2, \dots, w_{n-1}), \quad (3.8)$$

which states that the choice of w_i depends only on the *history* of words, not in any way on the future. This sequential decomposition is an appropriate one, because it leads to the development of natural intermediate decision criteria, allowing a minimal delay in the response of the recognizer to the progressing speech.

Now, the language modeling problem is reduced to estimating the conditional probabilities in the above formula. Suitable estimates can be derived by analyzing a large *training corpus* of text using statistical techniques. Of course, the probabilities $P(w_i | w_1, w_2, \dots, w_{i-1})$ would be impossible to estimate as such for even moderate values of i . For example, with a vocabulary of 10000 words, there are $10000^3 = 10^{12}$ different three-word histories. Only a tiny fraction of these histories will ever occur in practice. Thus, there is no data for most of the histories to support any estimates.

It follows that the various possible conditioning histories must be distinguished as belonging to some manageable number of different *equivalence classes*. Let Φ be a many-to-one mapping of histories into some number M of equivalence classes. If $\Phi[w_1, \dots, w_i]$ denotes the equivalence class of the string w_1, \dots, w_i , then the probability $P(\mathbf{W})$ can be approximated by

$$P(\mathbf{W}) = P(w_1)P(w_2 | \Phi[w_1])P(w_3 | \Phi[w_1, w_2]) \cdots P(w_n | \Phi[w_1, w_2, \dots, w_{n-1}]). \quad (3.9)$$

The selected classification scheme Φ will necessarily be a compromise between two conflicting requirements: (1) the classification must be sufficiently refined to provide adequate information about the history so it can serve as a basis for prediction; (2) when applied to histories of a given training corpus, the scheme must yield the M possible classes frequently enough so that the probabilities for each word conditioned on each class can be reliably estimated.

3.4.2 Methods

The prevailing paradigm in language modeling is *n-gram modeling*. It is based on a very simple equivalence classification: histories are equivalent if they end in the same $n-1$ words. In the most common case of $n = 3$, we have the *trigram* model:

$$P(\mathbf{W}) = \prod_{i=1}^n P(w_i | w_{i-2}, w_{i-1}). \quad (3.10)$$

Most trigrams never occur in any given training corpus, even a large one, but are still not impossible or even unlikely. To avoid assigning a zero probability to a trigram that is not encountered in the training phase, the trigram probabilities must be averaged in some way with *bigram* (one-word history, or $n = 2$) and *unigram* (no history) probabilities (see Jelinek [11], chapter 4).

The purpose of a language model for speech recognition is not meaning extraction, but an apportionment of probability among alternative futures. In this sense, language modeling closely resembles text compression, where the aim is to guess the next character based on past characters. *Variable-order n-gram models* (i.e., n -gram models where n is not fixed) have proven effective in text compression, so it is not surprising that similar techniques have been introduced to language modeling. For instance, the General Dynamic Markov Compression (GDMC) algorithm [32] is used in a recent language modeling scheme called Refined Probabilistic Finite Automata, or RPFA [9].

Another type of approach is to use *decision trees* in the equivalence classification of histories. Building a decision tree in the training phase is time-consuming and complicated, but the result-

ing language model appears to be more powerful than simple bigram and trigram models (Jelinek [11], chapter 10).

Perhaps the most obvious equivalence classification method would be to categorize words according to their *grammatical function* (determiner, noun, preposition, and so on). The process of assigning grammatical labels or tags to words in a sentence is called *part-of-speech tagging*. The tags may also be based on semantics (e.g., proper names and addresses could be identified).

To illustrate the way that tags can be used in history classification, consider the case where the two previous words are *in the*. These would be tagged as a preposition and a determiner, respectively. Now, the next word is very likely to be a noun, but definitely not a verb or a determiner. Using this type of knowledge and grammatical inference rules, perhaps coupled with a standard *n*-gram model, a more powerful language model can be built than with statistics alone.

The problems with part-of-speech tagging in language modeling are similar to those of formal grammars. First, a complete sentence is needed for best results. Trigram-based statistical taggers, which provide for intermediate decisions, display an order of magnitude higher error rate than the state-of-the-art sentence-based linguistic tagger of Voutilainen, *et al.* [28]. Second, both stochastic and linguistic taggers have a problem with ungrammatical sentences that commonly occur in spoken language. Consequently, tag equivalence classification is perhaps best suited for utterance postprocessing only.

As a conclusion, the simple trigram modeling scheme is still far from being obsolete. It is the most commonly used and practical method, with a solid foundation of more than twenty years of published research. It also integrates very nicely into the hidden Markov modeling framework, as will be shown later. An issue worth noting is that part-of-speech tagging, parsing and some of the other methods only apply if we want to recognize normal words – not when we need to recognize phonemes or other subword units. On the other hand, *n*-gram models have no such limitation.

4 Hidden Markov Models

Since their invention in the late 1960's, hidden Markov models have seen increasing utility in various scientific fields, including computational biology, speech recognition, speech enhancement and pattern recognition. Hidden Markov modeling is based on sound mathematical principles of probability theory, information theory, formal automata, optimization, algorithmics, and others. The basic definition of the HMM is often augmented, adapted and generalized in various ways when applying the theory into practice, but the underlying ideas remain the same.

In this chapter, we will formally define the hidden Markov model, as well as present several related algorithms and ideas that are crucial to using HMMs in modeling real-world phenomena. Our presentation draws from several sources, including the classic tutorial article by Rabiner [25] and the more recent textbooks by Rabiner & Juang [24] and Jelinek [11].

4.1 Definitions

4.1.1 Markov chains

Let $\{S_0, S_1, \dots, S_i, \dots\}$ be a sequence of discrete random variables assuming values in a finite alphabet $\mathcal{O} = \{1, 2, \dots, N\}$. The random variables are said to form a *Markov chain*, if for all values of i greater than zero

$$P(S_i = s_i | S_0 = s_0, \dots, S_{i-1} = s_{i-1}) = P(S_i = s_i | S_{i-1} = s_{i-1}), \quad (4.1)$$

where $s_i \in \mathcal{O}$. This assumption is known as the *Markov property*. It states that the conditional probability of an event depends only on the previous event, not the ones before that. A Markov chain is thereby a random process that has the minimum amount of memory. The output of the process, or the *observations*, are the actual values s_i that the random variables assume from the alphabet \mathcal{O} .

We can think of the elements of \mathcal{O} as *states*. Thus, if $S_t = j$, the Markov process is said to be in state j at time t . Since the process periodically moves from one state to another, the conditional probabilities

$$a_{ij} = P(S_t = j | S_{t-1} = i) \quad \text{for all } i, j \in \mathcal{O} \quad (4.2)$$

are called *transition probabilities*. If the probabilities are independent of time, as is the case with all models in this work, the transitions are *stationary*. The probabilities a_{ij} can be conveniently presented as a *transition matrix*,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix} \quad (4.3)$$

where N is the number of states or, alternatively, the size of the alphabet \mathcal{S} . The i :th row of \mathbf{A} is actually the conditional probability distribution of S_t , given that $S_{t-1} = i$. Therefore, the following stochastic constraints must hold:

$$\sum_{j=1}^N a_{ij} = 1 \quad \text{for all } i \in \mathcal{S}, \quad (4.4a)$$

$$a_{ij} \geq 0 \quad \text{for all } i, j \in \mathcal{S}. \quad (4.4b)$$

To fully define a Markov model, one more parameter is needed: the *initial state distribution*. The initial state distribution Π specifies, for all states, the probability that the process starts from that state:

$$\pi_i = P(S_0 = i) \quad \text{for all } i \in \mathcal{S}, \quad (4.5a)$$

$$\Pi = (\pi_1, \pi_2, \dots, \pi_N). \quad (4.5b)$$

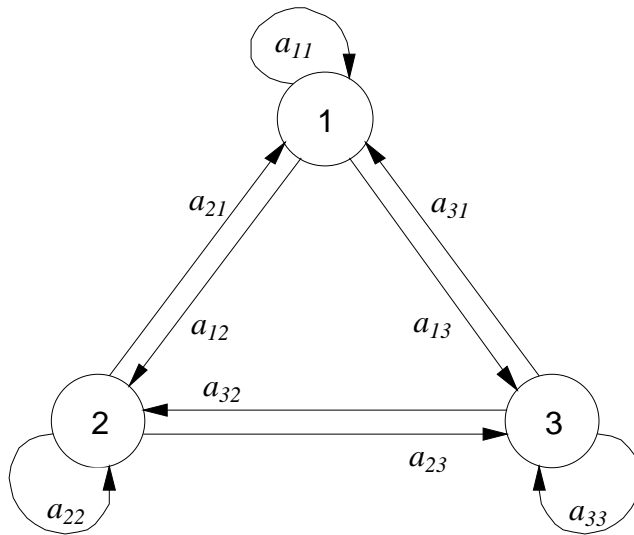


Figure 4-1. An ergodic Markov model with three states.

A Markov model can be displayed by an intuitive diagram, such as the one in Figure 4-1. The type of model in the figure is called *ergodic*: every state can be reached from every other state in a finite number of steps (assuming that $a_{ii} \neq 1$ for $i = 1, 2, 3$). Other types of models will be discussed later.

Summarizing, a Markov model $\lambda = (\Pi, \mathbf{A})$ is characterized by the initial state distribution Π and the state transition matrix \mathbf{A} . The state space \mathcal{S} need not be mentioned, because it is implicitly defined by either of the two parameters. The model generates as its output the state sequence that it goes through.

4.1.2 Hidden Markov models

Markov models are only good for very simple modeling tasks, because a given state always generates the same output. This is too restrictive for many problems of interest. In this section, we will extend the concept of Markov models to include the case where a state is no longer associated with just one observation, but with a probability distribution over all possible observations.

A (discrete) *hidden Markov model (HMM)*, denoted by $\lambda = (\Pi, \mathbf{A}, \mathbf{B})$, is characterized by the initial state distribution Π , the state transition matrix \mathbf{A} , and the *emission probability matrix* \mathbf{B} . The emission probability matrix specifies, for each state, a probability distribution over the output alphabet. The output alphabet need no longer be the same as the state space. Denoting the output alphabet with $\mathcal{O} = \{1, 2, \dots, M\}$ we get a matrix with N rows and M columns,

$$\mathbf{B} = \begin{pmatrix} b_1(1) & b_1(2) & \cdots & b_1(M) \\ b_2(1) & b_2(2) & \cdots & b_2(M) \\ \vdots & \vdots & \ddots & \vdots \\ b_N(1) & b_N(2) & \cdots & b_N(M) \end{pmatrix} \quad (4.6)$$

where $b_i(k)$ is the probability of symbol k being emitted from state i . The emission probability matrix is another stochastic matrix, in the sense that each row sums up to one, and all elements are greater than or equal to zero.

If we denote the random process that generates the state sequence by $\{S_i\}$, $i \geq 0$, then we can think of the emitted symbols as being selected by another random process $\{O_i\}$, $i \geq 0$. The two processes are connected to each other by the emission probabilities:

$$b_i(k) = P(O_t = k | S_t = i) \quad \text{for all } i \in \mathcal{S}, k \in \mathcal{O} \text{ and } t \geq 0. \quad (4.7)$$

One further simplifying assumption is needed to make hidden Markov modeling mathematically tractable: the emitted symbols are assumed to be *conditionally independent*² of each other, given the state sequence. The choice of a symbol to be emitted is thus solely determined by the probability distribution at the current state; previously emitted symbols have no effect. Formally,

$$P(O_0 = o_0, \dots, O_T = o_T \mid S_0 = s_0, \dots, S_T = s_T) = \prod_{i=0}^T P(O_i = o_i \mid S_i = s_i) = \prod_{i=0}^T b_{s_i}(o_i). \quad (4.8)$$

Without conditional independence, the simple multiplicands $P(o_i \mid s_i)$ in the above formula would have to be replaced with much more complicated terms of the form $P(o_i \mid o_0, \dots, o_{i-1}, s_i)$. That is, the whole history of emitted symbols would need to be considered. It is also worth noting that $\{ S_i \}$ being a Markov process does not imply that $\{ O_i \}$ also were one; i.e., we cannot write $P(O_i \mid O_0, \dots, O_{i-1})$ as $P(O_i \mid O_{i-1})$.

A hidden Markov model thus consists of an underlying Markov chain with a finite number of states, and a set of probability distributions, each associated with its respective state. At discrete instants of time, the Markov chain changes state according to the transition probability matrix \mathbf{A} . An observation $o \in \mathcal{O}$ is then generated according to the probability distribution at the new state. This is illustrated in Figure 4-2, where a total of seven observations are emitted from three states.

An outside observer can only perceive the generated observations, not the states where they came from. Because the same symbols can be emitted by any of a number of states, it is not generally possible to deduce with certainty the state sequence from observations. The state sequence is therefore *hidden*.

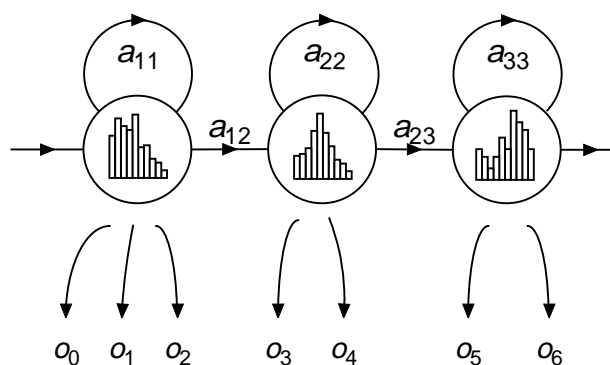


Figure 4-2. A discrete hidden Markov model generating observations.

² The symbols O_{i-1} and O_i are conditionally independent, given the state S_i , if and only if $P(O_i \mid O_{i-1}, S_i) = P(O_i \mid S_i)$.

4.1.3 Three basic problems

Having formally defined hidden Markov models, we may now turn to the three classical problems that must be solved in order to apply hidden Markov modeling to real-world tasks. These problems are the following:

- (1) **Evaluation**, or computing $P(\text{Observations} \mid \text{Model})$. This allows us to find out how well a model matches a given observation sequence. The main concern here is computational efficiency: finding an algorithm with only a polynomial running time. Such an algorithm is presented in section 4.2.
- (2) **Decoding**, or finding the hidden state sequence that best corresponds to the observed symbols. Because there are generally many sequences that give rise to the same symbols, there is no "correct" solution to be found in most cases. Thus, some optimality criterion must be chosen. The most widely used criterion is to find a path through the model that maximizes $P(\text{Path} \mid \text{Observations}, \text{Model})$. The *Viterbi algorithm*, presented in section 4.3, finds such a path in time linear on the number of observations.
- (3) **Training**, or finding the model parameter values $\lambda = (\Pi, \mathbf{A}, \mathbf{B})$ that specify a model most likely to produce a given sequence of training data. In other words, the objective is to construct a model that best fits the training data (or best represents the source that produced the data). There is no known way to analytically solve for the best model, but an iterative algorithm exists that often yields sufficiently good approximations. The training problem is discussed in section 4.4.

4.2 Probability evaluation

Computing $P(\text{Observations} \mid \text{Model})$ is trivial for observable Markov models, because the state sequence $S = (s_0, s_1, \dots, s_T) \in \mathcal{S}^{T+1}$ is equal to the observation sequence $O = (o_0, \dots, o_T) \in \mathcal{O}^{T+1}$. That is, $\mathcal{S} = \mathcal{O}$ and $s_i = o_i$ for all i . Thus, all that needs to be done is to follow the T -length path through the model and multiply together the transition probabilities a_{ij} along the way:

$$P(O \mid \lambda) = P(S \mid \lambda) = \pi_{s_0} \prod_{i=1}^T a_{s_{i-1}s_i}. \quad (4.9)$$

Things get more complicated with hidden Markov models, as the state sequence is no longer known. A straightforward solution would be to enumerate all possible paths $S \in \mathcal{S}^{T+1}$, compute

the probability of each using equation (4.9), then multiply in the observation probabilities for each individual path using equation (4.8), and finally sum up the path probabilities:

$$P(O | \lambda) = \sum_{S \in \mathcal{S}^{T+1}} \pi_{s_0} b_{s_0}(o_0) \prod_{i=1}^T a_{s_{i-1}s_i} b_{s_i}(o_i). \quad (4.10)$$

Although this gives the correct answer, the computational load is prohibitive. If there are N states and the model is fully connected, the number of T -length paths is N^{T+1} . If a model does not have a transition from every state to every other state, the number of paths is smaller, but still generally exponential on T . Because $O(T)$ multiplications are needed to evaluate the probability of one path, equation (4.10) has a running time of $O(TN^T)$.

Fortunately, there is a better way: the *forward algorithm*. We first describe the algorithm for observable Markov models to fix ideas, and then extend it in a straightforward way to handle hidden Markov models. Let us first define the *forward variable* for Markov chains:

$$\alpha_j(t) = P(S_t = j | \lambda) \quad \text{for all } j \in \mathcal{S} \text{ and } t \geq 0, \quad (4.11)$$

i.e., the probability of being in state j at time t , given the model (the conditioning model is omitted in the following to clarify notation). For Markov chains (but not for HMMs), $\alpha(t)$ is a probability distribution over the state space. At time $t = 0$, $\alpha(t)$ is obviously equal to the initial state distribution Π .

A computationally efficient formula can be derived for the forward variable as follows. First, using Bayes' rule, we write the definition in terms of the previous states. Then, we notice that all but the immediately preceding state can be forgotten, due to the Markov property. Finally, we get a formula (illustrated in Figure 4-3) for $\alpha(t)$ that only depends on $\alpha(t-1)$ and the transition probabilities a_{ij} . Formally,

$$\begin{aligned} \alpha_j(t) &= P(S_t = j) \\ &= P(S_t = j | S_0, S_1, \dots, S_{t-1}) P(S_0, S_1, \dots, S_{t-1}) \\ &= P(S_t = j | S_{t-1}) P(S_{t-1}) \\ &= \sum_{i \in \mathcal{S}} P(S_t = j | S_{t-1} = i) P(S_{t-1} = i) \\ &= \sum_{i \in \mathcal{S}} \alpha_i(t-1) a_{ij} \end{aligned} \quad (4.12)$$

This so-called forward algorithm is based on the realization that all paths ending in a particular state have exactly the same possibilities of taking the next step. Extending a path that ends in a certain state automatically extends all the other paths ending in that state, as well. Since there

are N states and at most N transitions from each, the number of operations needed to extend *all* paths is $O(N^2)$. The total running time of the algorithm is thus merely $O(TN^2)$, in contrast to the exponential running time of the naïve algorithm (4.10). The forward algorithm can be thought to proceed through a *trellis* structure (see Figure 4-4), whereas the naïve algorithm searches through a tree with a branching factor of N .

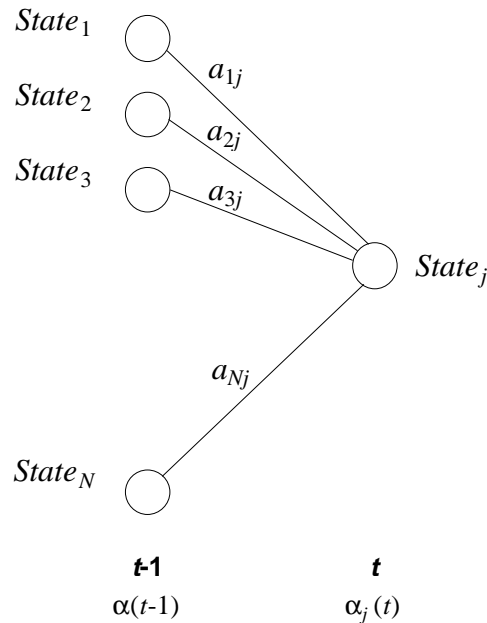


Figure 4-3. The forward algorithm. The probability of being in state j at time t can be computed using just $\alpha(t-1)$, the state probability distribution at time $t-1$, and the transition probabilities a_{ij} .

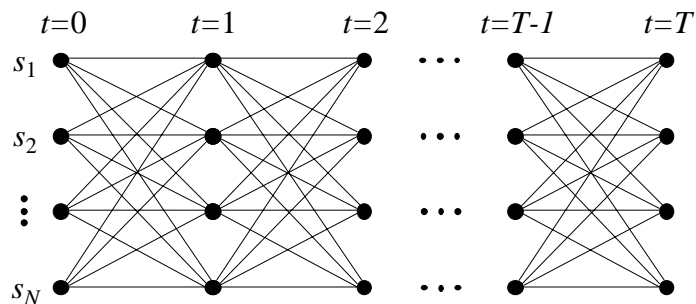


Figure 4-4. The trellis structure of the forward algorithm. All paths ending in a particular state are equivalent with respect to taking the next step. The algorithm proceeds from left to right in a breadth-first manner.

In order to finally solve the evaluation problem, we need to extend the forward procedure to handle hidden Markov models. We first redefine the forward variable as

$$\alpha_j(t) = P(O_0 = o_0, \dots, O_t = o_t, S_t = j | \lambda), \quad (4.13)$$

so that also the observations up to time t are accounted for. Note that the forward probabilities at time t do not generally sum up to one. This means that the variables no longer constitute a probability distribution over the state space, as was the case with Markov chains.

We can solve for the forward variable inductively in the same manner as before; the emission probabilities just need to be multiplied in. We also introduce a terminating condition: once we have the forward probabilities for all states at time T (the final time frame) we sum over the state space to yield the quantity that we were looking for in the first place – $P(O | \lambda)$. This is the fastest known way to compute the likelihood of an HMM, with a running time in the order of N^2 .

(1) **Initialization:**

$$\alpha_j(0) = \pi_j b_j(o_0) \quad \text{for all } j \in \mathcal{S} \text{ and } t = 0. \quad (4.14a)$$

(2) **Induction:**

$$\alpha_j(t) = \sum_{i \in \mathcal{S}} \alpha_i(t-1) a_{ij} b_j(o_t) \quad \text{for all } j \in \mathcal{S} \text{ and } t \geq 1. \quad (4.14b)$$

(3) **Termination:**

$$P(O | \lambda) = \sum_{i \in \mathcal{S}} \alpha_i(T) \quad \text{for } t = T. \quad (4.14c)$$

4.3 Decoding: finding the best path

The forward algorithm allows the probability of an HMM to be evaluated with respect to a given observation sequence, but it does not give any indication as to the underlying state sequence. As a remedy, we will present the *Viterbi algorithm*, which can be used to find the most likely path through a model (this is known as *sequential decoding*).

The Viterbi algorithm is similar in structure to the forward procedure presented in the previous section. Both have a similar kind of induction step, but whereas the forward algorithm sums probabilities over the state space, the Viterbi algorithm takes the maximum. Since the basic idea is already familiar, we may proceed directly to the hidden Markov model case, without separately explaining the algorithm for observable models.

The Viterbi algorithm requires keeping track of two quantities for every state and every time frame. These are the highest probability at time t along any single path that ends in state j (and accounts for the observations up to time t):

$$\delta_j(t) = \max_{(s_0, \dots, s_{t-1})} P(S_0 = s_0, \dots, S_{t-1} = s_{t-1}, S_t = j, O_0, \dots, O_t | \lambda), \quad (4.15)$$

and the maximizing path $S = (s_0, s_1, \dots, s_{t-1}, j) \in \mathcal{O}^{t+1}$ itself. The path is kept track of implicitly in the form of *back pointers*, as described shortly. Note that the path must, indeed, be a legal state sequence in the HMM: there must be a transition from state s_t to s_{t+1} for all $t \geq 0$.

Initially (at $t = 0$) there is only one path leading into state j : the zero-length path whose only element is j itself. The probability of this path is the initial state probability of state j , times the probability of emitting the first symbol o_0 while in state j :

$$\delta_j(0) = \pi_j b_j(o_0) \quad \text{for all } j \in \mathcal{S} \text{ and } t = 0.$$

An inductive formula for $\delta_j(t)$ can now be derived in the same way as for the forward variable in (4.12). The terminating condition occurs when the final observation at time T is processed, and the $\delta_j(T)$ values have been calculated for all j . Then, the probability of the best path through the model, denoted $\delta^*(T)$, is simply the maximum of the $\delta_j(T)$'s:

$$\delta_j(t) = \max_{i \in \mathcal{S}} \delta_i(t-1) a_{ij} b_j(o_t) \quad \text{for all } j \in \mathcal{S} \text{ and } t \geq 1,$$

$$\delta^*(T) = \max_{j \in \mathcal{S}} \delta_j(T) \quad \text{for } t = T.$$

The above scheme finds the probability of the most likely path, but does not enable the path itself to be identified. This is easily remedied by saving the maximizing state(s) at every step in an auxiliary variable:

$$\psi_j(0) = \emptyset \quad \text{for all } j \in \mathcal{S} \text{ and } t = 0,$$

$$\psi_j(t) = \arg \max_{i \in \mathcal{S}} \delta_i(t-1) a_{ij} b_j(o_t) \quad \text{for all } j \in \mathcal{S} \text{ and } t \geq 1,$$

$$\psi^*(T) = \arg \max_{i \in \mathcal{S}} \delta_i(T) \quad \text{for } t = T.$$

Since there can be several preceding states i that maximize the probability, the auxiliary variable $\psi_j(t)$ is actually a set of states. In practice, it suffices to select just one of the maximizing states and discard the rest. Now, $\psi^*(T)$ specifies the state (s_T) where the most likely path ends. The whole path can be retrieved by following the back pointers:

$$s_T = \psi^*(T), s_{T-1} = \psi_{s_T}(T-1), s_{T-2} = \psi_{s_{T-1}}(T-2), \dots, s_0 = \psi_{s_1}(0).$$

The Viterbi algorithm is indeed very similar to the forward procedure. They both carry out a breadth-first search in a trellis structure (see Figure 4-5). The major difference is the maximization over previous states instead of summation. Also, to retrieve the state sequence, a backtracking stage is needed. The computational cost of Viterbi search is the same as that of the forward algorithm, $O(TN^2)$, because backtracking has only a linear cost. As a summary, we formally state the Viterbi algorithm for hidden Markov models as follows:

(1) Initialization:

$$\delta_j(0) = \pi_j b_j(o_0) \quad \text{for all } j \in \mathcal{S} \text{ and } t = 0, \quad (4.16a)$$

$$\psi_j(0) = \emptyset \quad \text{for all } j \in \mathcal{S} \text{ and } t = 0. \quad (4.16b)$$

(2) Induction:

$$\delta_j(t) = \max_{i \in \mathcal{S}} \delta_i(t-1) a_{ij} b_j(o_t) \quad \text{for all } j \in \mathcal{S} \text{ and } t \geq 1, \quad (4.16c)$$

$$\psi_j(t) = \arg \max_{i \in \mathcal{S}} \delta_i(t-1) a_{ij} b_j(o_t) \quad \text{for all } j \in \mathcal{S} \text{ and } t \geq 1. \quad (4.16d)$$

(3) Termination:

$$\delta^*(T) = \max_{j \in \mathcal{S}} \delta_j(T) \quad \text{for } t = T, \quad (4.16e)$$

$$\psi^*(T) = \arg \max_{j \in \mathcal{S}} \delta_j(T) \quad \text{for } t = T. \quad (4.16f)$$

(4) Path backtracking:

$$\psi^*(t) = \psi_{\psi^*(t+1)}(t+1) \quad \text{for } t = T-1, T-2, \dots, 1, 0. \quad (4.16g)$$

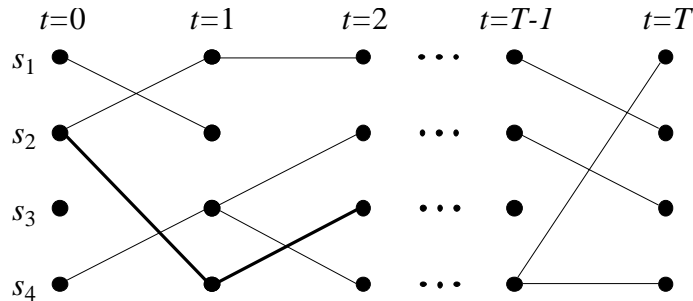


Figure 4-5. The Viterbi algorithm. At every time step, the single most probable path is retained for each state. In this example, the best path ending in state s_3 at time $t = 2$ is (s_2, s_4, s_3) .

4.4 Training: estimating model parameters

In the preceding sections, it was assumed that appropriate Markov models were available, but it has not been told how such models can be constructed. Not surprisingly, constructing models is more difficult than using them.

In this section, we will discuss the problem of making the parameters (Π , \mathbf{A} , \mathbf{B}) of a hidden Markov model to fit some given training data as well as possible. This is known as *parameter estimation* or *training*. We assume that the state space \mathcal{S} , the output symbols \mathcal{O} and the acceptable transitions $a_{ij} \neq 0$ have been defined beforehand, because that must be done experimentally. There is no generic algorithm to choose a suitable topology or an appropriate output alphabet for a given task. We will return to this problem in the next chapter.

4.4.1 Maximum likelihood recognition and training

The training problem for hidden Markov models is to estimate the transition probabilities, the initial state distribution and the emission probability distributions from sample data. The idea is that when the model is later presented with data from an unknown source, it will *recognize* the data if it has the same characteristics as the training data. Typically, we have multiple models, all trained to represent a separate source. In the recognition phase, the models compete against each other to see which one of them best matches the unknown data. The data can then be classified as having been produced by the source corresponding to the best matching model.

Suppose, now, that we have several different HMMs and a sequence of observations generated by some process. Which of the models best represents that source, or recognizes the observation sequence? A natural choice is the model with the highest probability with respect to the observations, i.e., the one that maximizes $P(\text{Model} | \text{Observations})$:

$$\hat{\lambda}_{\text{MAP}} = \arg \max_{\lambda} P(\lambda | O) = \arg \max_{\lambda} \frac{P(O|\lambda) P(\lambda)}{P(O)} = \arg \max_{\lambda} P(O|\lambda) P(\lambda). \quad (4.17)$$

This is called the *maximum a posteriori (MAP)* estimate. The second equality follows from Bayes' rule and the definition of conditional probability. The denominator $P(O)$ can be omitted because it is constant over the models, and so does not affect the maximization.

If meaningful values for the priors $P(\lambda)$ cannot be found, as is often the case, they can be assumed to have the uniform distribution. This makes also the term $P(\lambda)$ constant over the models,

effectively removing it from the above formula. The only term that is left, $P(O|\lambda)$, is called the *likelihood function* of the model, and correspondingly

$$\hat{\lambda}_{\text{ML}} = \arg \max_{\lambda} P(O|\lambda) \quad (4.18)$$

is called the *maximum likelihood (ML)* estimate. The best model in maximum likelihood sense is therefore the one that is most probable to *generate* the given observations.

For the training problem, several different mathematical solutions can be derived. The task is to optimize a model or a set of models with respect to some *optimization criterion*. If the optimization criterion is to maximize the likelihood function $P(O|\lambda)$, O being the training observations, a practical solution called the *Baum-Welch algorithm* applies. We will present the algorithm on a schematic level, omitting technical details and mathematical proofs. More thorough explanations can be found in any tutorial article or textbook on hidden Markov models (e.g., Rabiner [25] or van Alphen [34]). For a proper mathematical treatment, refer to Koski [14].

The Baum-Welch algorithm is an iterative procedure that progressively refines the model parameters Π , \mathbf{A} and \mathbf{B} until a maximum of the likelihood function is reached. This is illustrated in Figure 4-6. Notice that the parameters from the previous round are always used as the basis for the next iteration. For this reason, the procedure is often called *reestimation*. Note, also, that the likelihood of the model is strictly increasing: at every step, the likelihood is either improved or the procedure is terminated. This, unfortunately, implies that there is no way to back off from a local maximum (by means of temporarily decreasing the likelihood) to find higher peaks elsewhere.

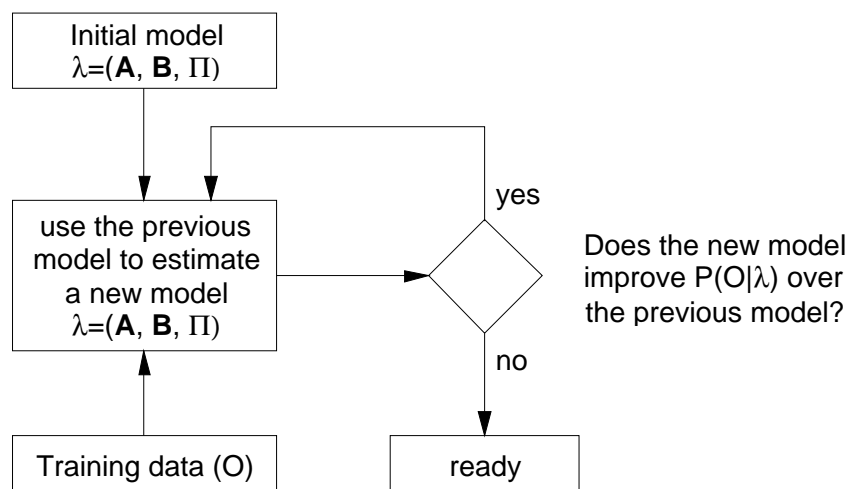


Figure 4-6. Maximum likelihood training of hidden Markov models [34].

The convergence of the Baum-Welch algorithm to a local maximum of the likelihood function can be proven [14]. However, a global maximum is attained only if the initial parameter values lie in its attraction domain. Even then, the model will not necessarily be optimal performance-wise: maximum likelihood does not equal maximum recognition performance. Further, no training procedure can overcome an unsuitable model topology or observation alphabet, or the effects of poorly chosen training data.

The following set of reestimation formulas can be shown to find parameter values that yield a local maximum of the likelihood function (given sufficiently many iterations):

$$\tilde{\pi}_i = \text{Expected number of times in state } i \text{ at time } t = 0 \quad (4.19a)$$

$$\tilde{a}_{ij} = \frac{\text{Expected number of transitions from state } i \text{ to state } j}{\text{Expected number of transitions from state } i} \quad (4.19b)$$

$$\tilde{b}_j(k) = \frac{\text{Expected number of times in state } j \text{ and observing symbol } k}{\text{Expected number of times in state } j} \quad (4.19c)$$

All the expectations are of course conditioned on the (current) model λ and the particular observation sequence O that is used in the training.

To solve the reestimation formulas we first need to define the *backward variable*. Recall that the forward variable $\alpha_i(t)$ denotes the probability of the joint event that (o_0, \dots, o_t) are observed and the system stops in state i at time t . The corresponding backward variable $\beta_i(t)$ denotes the probability that (o_{t+1}, \dots, o_T) are observed and the system *starts* from state i at time t . The backward variable can be evaluated recursively (proceeding backwards from time T) in the same way as the forward variable.

The expectations in the reestimation formulas can be written in terms of the current model parameters and the forward and backward variables, yielding a computationally efficient parameter estimation procedure. We will not derive these formulas for all of the required expectations, but will give an example to illustrate how it can be done.

Consider the probability of being in state i at time t while observing the whole observation sequence O , $P(O_0 = o_0, \dots, O_T = o_T, S_t = i | \lambda)$. This can be written with the forward and backward variables as $\alpha_i(t)\beta_i(t)$. Dividing by $P(O | \lambda)$, we get $P(S_t = i | o_0, \dots, o_T, \lambda)$, which is the probability of being in state i at time t , given the observations and the model:

$$P(S_t = i | O, \lambda) = \frac{P(O, S_t = i | \lambda)}{P(O | \lambda)} = \frac{\alpha_i(t) \cdot \beta_i(t)}{P(O | \lambda)} = \frac{\alpha_i(t) \cdot \beta_i(t)}{\sum_{j \in \mathcal{S}} \alpha_j(T)}. \quad (4.20)$$

The first equality holds by basic rules of probability theory, the second by the definitions of α and β , and the third by equation (4.14c) from section 4.2.

By summing the values of (4.20) over all t , we get a quantity that can be interpreted as the expected number of times that state i is visited – that is, the denominator of (4.19c). The remaining expectations can also be written in terms of just α , β , \mathbf{A} , \mathbf{B} and Π .

4.4.2 Alternative criteria for parameter estimation

The basic philosophy of hidden Markov modeling is that a signal (or observation sequence) can be well modeled if the parameters of an HMM are carefully and correctly chosen. The problem with this philosophy is that it is sometimes inaccurate – either because the signal does not obey the constraints of the HMM, or because it is too difficult to get reliable estimates of all HMM parameters. To alleviate these problems, at least two alternatives to the maximum likelihood optimization criterion have been proposed.

The first alternative – *maximum mutual information (MMI)* – is based on the idea that several HMMs are to be designed at the same time, in such a way so as to maximize each model's ability to distinguish between the output of the correct source and that of alternative sources (that correspond to other models). The model set is therefore optimized as a whole, rather than treating all models individually. A concept from information theory, *mutual information*, is used as a measure of discriminative power among the models. Unfortunately, no analytical or reestimation type solutions for the MMI optimization problem are known [25].

The second alternative optimization criterion is known as *minimum discrimination information (MDI)*. This criterion allows us to relax the – often false – assumption that the source to be modeled obeys the Markov property. Each model is, again, optimized in isolation. *Discrimination information*³, another concept from information theory, is used to measure the distance between the set of observed signal probability densities and the set of HMM probability densities. Techniques for minimizing that distance by choosing suitable HMM parameters are highly non-trivial, and will not be discussed here.

The idea of MMI optimization was first proposed by Bahl, *et al.* [1], and MDI optimization by Ephraim, *et al.* [6]. The three modeling approaches – ML, MMI and MDI – and the relationships

³ Also known as *cross entropy*, *relative entropy*, *divergence* or *Kullback distance*.

between them are discussed by Ephraim & Rabiner [5]. Note, finally, that the different criteria in parameter estimation do not affect the recognition phase. The recognition decisions can be made on maximum likelihood basis, regardless of the training philosophy.

5 Hidden Markov Models in Speech Recognition

Hidden Markov models have enjoyed widespread popularity in speech recognition research for the past two decades. The success of HMMs in speech processing is mostly explained by three factors: (1) they have a sound mathematical basis and practical algorithms for training and use; (2) they are good at modeling one-dimensional time-varying signals in general; (3) they are good at modeling uncertainty and do not require too many assumptions to be made about the source or process that is to be modeled.

In this chapter, the theory and equations presented in the previous chapter will be adapted for speech observations. A front end similar to the one described in 3.3.3 is assumed to produce the observations. We will start by applying HMMs for the most essential task in speech recognition, namely acoustic modeling.

Language models and grammars may also be represented as Markov chains, as described in section 5.2. In section 5.3, we will devise an integrated *recognition network*, encompassing both the acoustic models and the language model. Computationally efficient searching through the recognition network will also be discussed. We will conclude by identifying several drawbacks and limitations of HMM-based speech recognition.

5.1 Acoustic modeling

As described in section 3.2, the task of an acoustic model in speech recognition is to estimate, at run time, the probabilities $P(\mathbf{A}|\mathbf{W})$ for any acoustic data string $\mathbf{A} = a_1, a_2, \dots, a_m$ and hypothesized word string $\mathbf{W} = w_1, w_2, \dots, w_n$. In other words, the task is to estimate the probability that when the speaker utters \mathbf{W} , the acoustic processor outputs \mathbf{A} .

In this section, we will apply hidden Markov models to the acoustic modeling task. This is not simply an issue of setting up some arbitrary model and then training it with a sufficiently large speech database. Instead, the system designer must use his or her knowledge and experience to make a number of critical design choices. These include the following:

- (1) What kind of model topology to use?
- (2) What kind of speech units to model?
- (3) What kind of observations to use?

5.1.1 Model topology

Unless the states of a hidden Markov model correspond directly to some known physical states, the model topology must be chosen in an *ad hoc* manner. There is no generic algorithm to decide the number of states a Markov model should have, or which states should be connected to each other. Practical experience in the application area is the main factor in determining the type of model to use.

In many fields, such as speech recognition and biological sequence analysis, the custom is to use so-called *left-to-right* models. Left-to-right models obviously have no transitions from right to left, or more formally, from higher-index states to lower-index states. This implies that the rightmost state is an *absorbing* state: once it has been reached, there is no way to get out. Every other state is *transient*, because the chain will eventually leave that state without ever returning to it.

Left-to-right models have several advantages. For time-varying signals like speech, the nature of the signal is reflected in the model topology. The number of possible paths through the model is also reduced significantly (compared to ergodic models), simplifying computations. Because the number of states and transitions is relatively small, estimating transition and emission probabilities becomes more tractable – fewer parameters to estimate implies more reliable estimates. Finally, practical experience in speech recognition shows that the left-to-right topology performs better than any other topology.

Figure 5-1 shows a left-to-right model of the most simple kind. Each path through the model must visit each state at least once. This means that sequences shorter than the number of states in the model can not be generated. Alternative (shorter) routes through a model are often added by *state skipping*. This is illustrated in Figure 5-2.

Usually, the number of states in a phonetic HMM is small (from 3 to 7), and the same for all models. The length of a left-to-right HMM may also be decided according to the average duration of the unit that it represents, or manually by phonetic considerations. In practice, these kinds of enhancements have not increased recognition accuracy. Other, more fundamental flaws in HMM-based recognition apparently undermine any minor improvements in model topology.

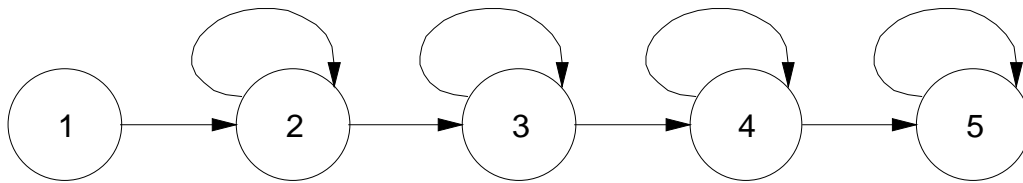


Figure 5-1. A simple left-to-right model with five states and no state skipping. The last state is *absorbing*, the others *transient*.

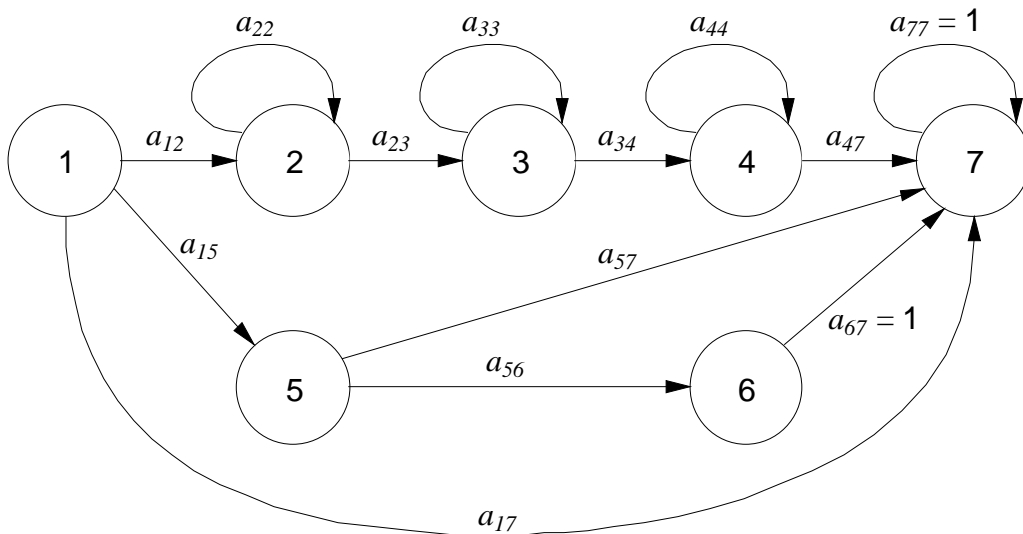


Figure 5-2. A left-to-right model with skip states. There are shortcuts to the final state either directly, via state 5, or via both 5 and 6.

5.1.2 Subword modeling units

In continuous-speech recognition, *subword models* are required to account for differences in pronunciation and for the possibly infinite vocabulary. The subword models are then used as building blocks for whole-word models. For example, an acoustic model for the word *candy* can be constructed by concatenating the five phoneme models for *k*, *ae*, *n*, *d* and *ih*, or the two syllable models for *kaen* and *dih*. This is illustrated in Figure 5-3.

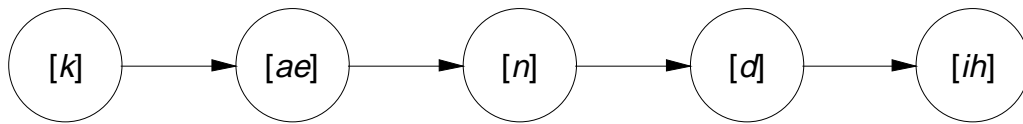


Figure 5-3. A word HMM composed by concatenating phoneme HMMs. Each state in the word model is actually an HMM representing a specific phoneme.

Subword units should be *consistent* and *trainable*. Consistency means that different instances of the same subword unit have similar acoustic characteristics. Consistency is important because it improves the discrimination between different subword units (the better the discrimination, the higher the accuracy). Trainability means that each unit can be trained on sufficiently many example pronunciations.

If we have infinite training data, consistency is the only property of interest. However, because training data is not only finite, but often limited, trainability becomes an important issue, as well. Trainability can be achieved either by using very general units – at the cost of inconsistency – or by sharing among units.

Monophone models are the most obvious choice for a subword unit: each model represents one phoneme. Since there are only about 50 phonemes in English, monophone models can be sufficiently trained with a relatively small number of sentences. However, monophone models assume that a phoneme in any context is equivalent to the same phoneme in any other context. Yet, this is far from the truth. The acoustic realization of a phoneme is strongly affected by its neighboring phones (coarticulation), as discussed earlier.

There are two principal ways of dealing with coarticulatory effects: (1) using larger units of speech, such as syllables, and (2) using an allophonic alphabet. With syllable-sized models, any allophonic variations of the central phonemes in syllables (e.g. the [æ] in *bat*) are well accounted for. The central phonemes are therefore consistent. On the other hand, the starting and ending portions are still susceptible to coarticulation effects, and thus not consistent. The main problem, however, is with trainability: there are over 10000 syllables in English. Parameters for that many models are very difficult to estimate reliably.

Constructing a consistent, yet sufficiently refined allophonic alphabet by hand would require considerable phonetic skill and a lot of hard work. In addition, a large amount of training data would have to be manually labeled with the allophones. Thus, the only practical option is to

automatically select the allophones in a data-driven manner. *Triphones* and *biphones* can be used for this purpose.

A triphone is the acoustic realization of a phoneme when surrounded by two specific phones. For example, the triphone $[b-ae-t]$ denotes the acoustic realization of $[ae]$ when preceded by $[b]$ and followed by $[t]$. A biphone is similar, but with only the left or the right context involved. A good exposition of triphones and other context-dependent models is given by Lee [19].

With a phonetic alphabet of 50 symbols, each phoneme has theoretically $50^2 = 2500$ different contexts, yielding 125000 triphones in total. Only a fraction of these actually occur in speech, but even the remaining couple of thousand are too many for reliable training. Fortunately, many of the acoustic contexts are so similar to each other that they can be modeled as one. Consider, for instance, the triphones $[v-ae-n]$ and $[f-ae-n]$. The only difference between them is in voicing: $[v]$ is voiced while $[f]$ is not. Because the vocal tract configurations are the same, both triphones represent the same allophone of $[ae]$.

Triphone models need not be clustered manually. Information theoretic measures can be used to find similar triphone models from a larger set. The identified triphones can then be combined, and the resulting model retrained using all the training data of the original triphones. Typically, clustering allows the number of triphones to be reduced by an order of magnitude, without significantly degrading consistency.

To further increase the trainability of triphone models, they can be *interpolated* with biphone and monophone models. Biphones and monophones are easier to train reliably, because they occur more often. For example, the reliability of the triphone model $[X-ae-Y]$ can be improved by computing a weighted average of the original $[X-ae-Y]$, the biphones $[X-ae]$ and $[ae-Y]$, and the monophone $[ae]$. Interpolation is particularly beneficial for uncommon triphones, for which not enough training material is otherwise available.

Triphone models have proven to yield better recognition accuracy than monophone models. However, the difference is not always decisive; other factors may dominate the overall performance. Monophone models are also easier to construct, so using them instead is often well justified.

5.1.3 Observation densities

An issue of considerable practical importance is the nature of the observations. So far, we have only considered *discrete* HMMs, where the observations are drawn from a finite alphabet. This is, however, a suboptimal design for speech recognition, where the incoming data is inherently continuous. Quantizing the continuous feature vectors into single integers obviously introduces distortion and loss of information (as pointed out in section 3.3.3). This can be avoided by using *continuous* HMMs, in which a continuous probability density function (PDF) is attached to each state. Such an HMM no longer generates discrete symbols only, but continuous values from the observation space.

In order to use a continuous observation density, some restrictions have to be placed on the form of the probability density function to ensure that its parameters can be estimated in a consistent way. A simple way to form a more generalizable PDF that still obeys some parametric shape is to use a weighted sum of two or more PDFs. This is referred to as a *mixture distribution*. Formally, the PDF of a weighted mixture distribution is

$$f(x) = \sum_{i=1}^N c_i f_i(x), \quad (5.1)$$

where the mixture gains c_i are non-negative and sum up to one, and the f_i 's are probability density functions. This ensures that the resulting PDF is properly normalized. Usually, the mixture components are Gaussian densities, so that we have

$$f(x) = \sum_{i=1}^N c_i \mathcal{N}(\mu_i, \sigma_i), \quad (5.2)$$

where $\mathcal{N}(\mu_i, \sigma_i)$ is the normal distribution with mean μ_i and variance σ_i . As an example, a four-component mixture of Gaussian densities is shown in Figure 5-4.

Specifying a one-dimensional continuous density is not enough, however. The feature vectors are multidimensional, so we need a multidimensional PDF, as well. Theoretically, this causes no complications: the reestimation formulas can be readily adapted to handle multivariate Gaussian mixtures (see Jelinek [11], chapter 9), of the form

$$f(x) = \sum_{i=1}^N c_i \mathcal{N}(\boldsymbol{\mu}_i, \mathbf{U}_i), \quad (5.3)$$

where the one-dimensional mean has been replaced with a *mean vector* and the variance with a *covariance matrix*. The mixture gains must still satisfy the usual stochastic constraints, so that

the (N -dimensional) volume of the PDF is normalized. To illustrate, a two-dimensional mixture density is shown in Figure 5-5.

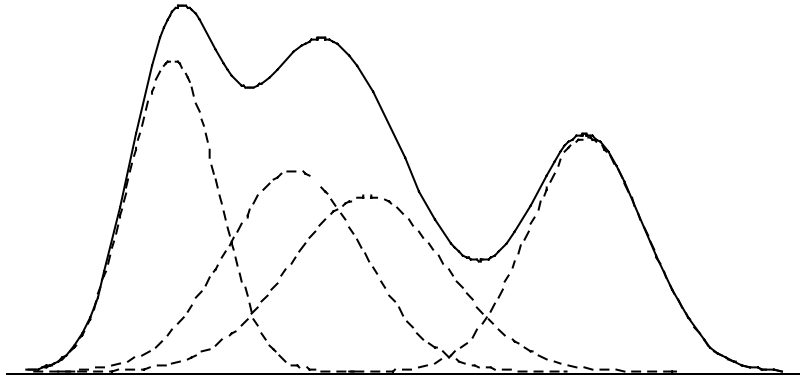


Figure 5-4. A continuous probability density function specified as a sum of four Gaussian PDFs. The four Gaussians are weighted uniformly by 0.25, as shown, so that the mixture density has an area of 1.

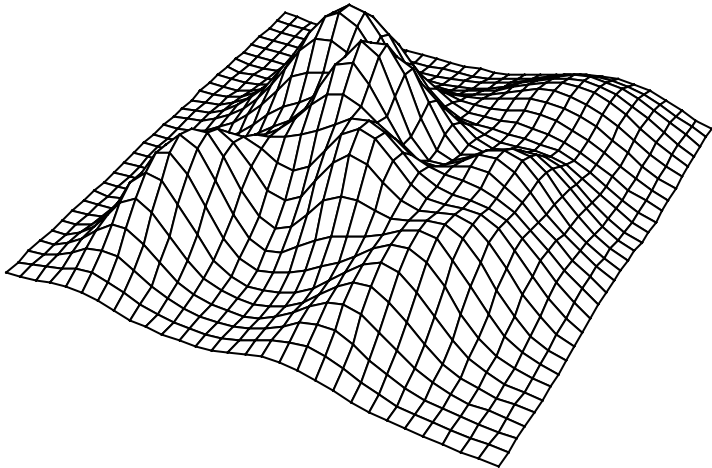


Figure 5-5. A two-dimensional probability density function specified as a mixture of eight two-dimensional Gaussian PDFs. The mixture components are not shown.

The main practical difficulty with multivariate densities is the large number of parameters to estimate. With typical 30-component feature vectors, we need to estimate 30 means and 30 x 30

= 900 covariances for every mixture component in every state. This has proven to be too much. One solution is to use diagonal covariance matrices, i.e., setting all the off-diagonal elements to zero. This causes no major degradation in accuracy, but decreases the amount of required training data significantly.

Another possibility to reduce the number of parameters is to use *semi-continuous* HMMs (also known as *tied mixtures*). This method tries to cover the observation space with a set of independent Gaussian densities. The resulting set of means and covariances is stored in a codebook, while the state-specific PDFs are defined as mixtures of selected Gaussian densities from that codebook. Semi-continuous HMMs have proven to give results no worse than fully continuous models, and consistently better than discrete models.

5.2 Language modeling

As suggested in section 3.4, some of the language modeling techniques can be conveniently integrated into the hidden Markov modeling framework. In this section, we will show how to represent certain types of language models as Markov chains. In the next section, we will integrate them with acoustic models.

A regular grammar can be represented as a finite state automaton (FSA) [8]. A finite state automaton, on the other hand, is actually a Markov chain without the probabilities. Thus, an FSA can be converted to a Markov model by imposing the uniform distribution on the transitions. For example, if a state has eight outgoing arcs, each of them will get the probability of $\frac{1}{8}$. This way, a task grammar that can be specified with a regular grammar (or, equivalently, with regular expressions), can be represented as a Markov model and hence integrated into the overall HMM recognition network. As an example, Figure 5-6 specifies a simple task grammar as an FSA.

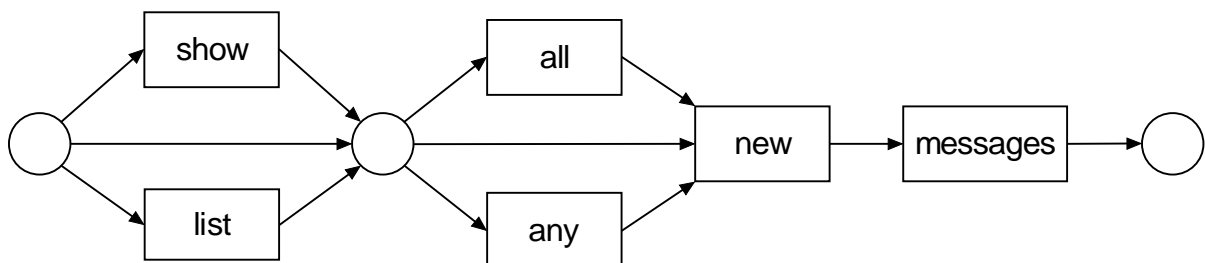


Figure 5-6. A regular grammar for a command-and-control task, represented as a finite state automaton (with symbols attached to states instead of transitions). Null states are used to reduce the number of transitions.

The n -gram language models (as defined in section 3.4.2) can also be represented as Markov models. For $n = 2$, the conversion from a tabular form is straightforward: every word in the vocabulary gets an own state, and the bigram probabilities are attached to the transitions between states. A separate initial state is also required, as well as transitions from the initial state to the other states. If there are N words in the vocabulary, the number of states will be $O(N)$ and the number of transitions $O(N^2)$. A bigram language model for a three-word vocabulary is shown in Figure 5-7.

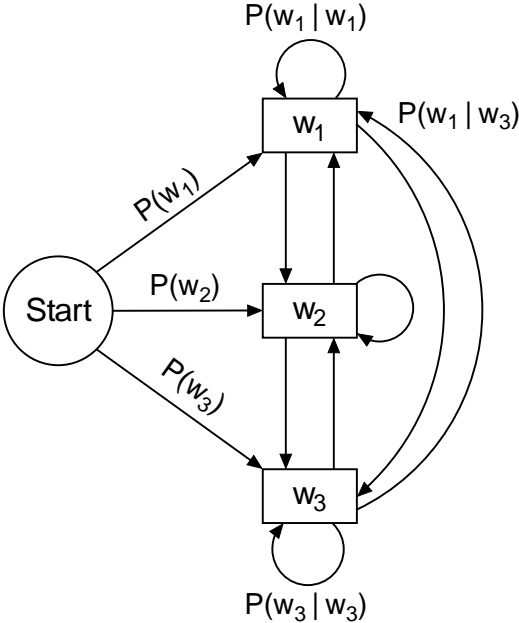


Figure 5-7. A bigram language model for a three-word vocabulary represented as a Markov chain. Some of the state transition probabilities are omitted for clarity.

Representing a trigram language model as a Markov chain is more complicated. This is because the conditioning history is now two words instead of one, and each possible history must have its own state. There will hence be $O(N^2)$ states in the resulting Markov model. The transitions are defined as follows: there is a transition from state (w_i, w_j) to state (w_k, w_l) if and only if $w_k = w_j$. If this is the case, the transition will have the probability $P(w_l | w_i, w_j)$. This implies that the number of transitions in the model will be $O(N^3)$: each of the $O(N^2)$ states will originate $O(N)$ transitions. This is illustrated in Figure 5-8 for a two-word vocabulary.

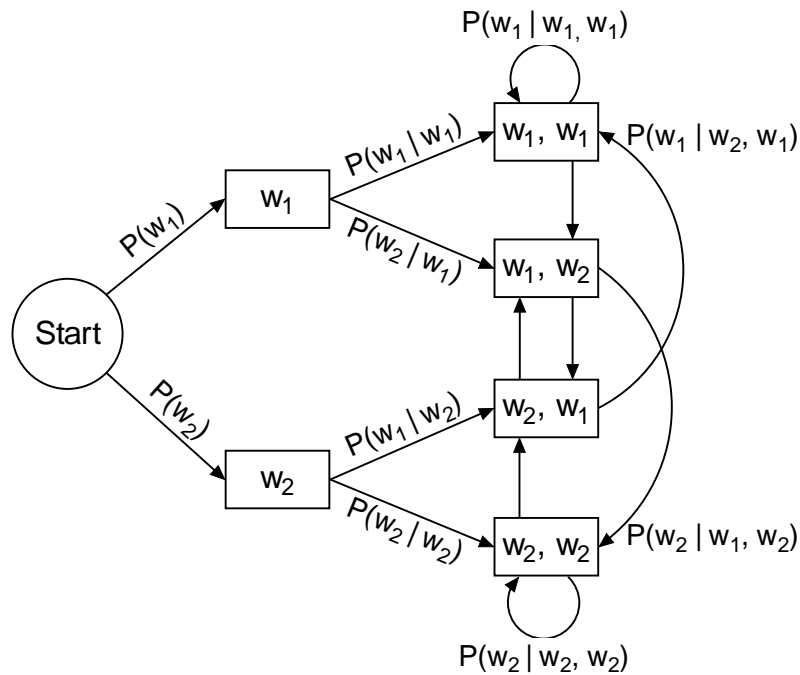


Figure 5-8. A trigram language model for a two-word vocabulary represented as a Markov chain. Some of the state transition probabilities are omitted for clarity.

It is also possible to integrate a language model with a regular grammar (or the corresponding Markov model). The grammar component then defines the set of permissible sentences, while the language model allows these sentences to be ranked according to their probability. For a bigram model, the integration is easy: just attach the bigram probabilities to the transitions. For trigrams, additional intermediate states and transitions have to be used in a similar fashion as in Figure 5-8.

5.3 Integrated hypothesis search

Taking advantage of the fact that embedding HMMs into an HMM yields a new HMM, we may build a model that conveniently integrates acoustic modeling and language modeling. Although the model is actually just an HMM, it is often called a *recognition network*. In the following, we will illustrate how to construct such a network, and then how to carry out a hypothesis search in it.

The recognition network is composed hierarchically by first concatenating subword models to form word models, and then replacing the states of an integrated grammar/language model with the word models. Figure 5-9 illustrates this process for monophone acoustic models and a very simple grammar with associated bigram probabilities.

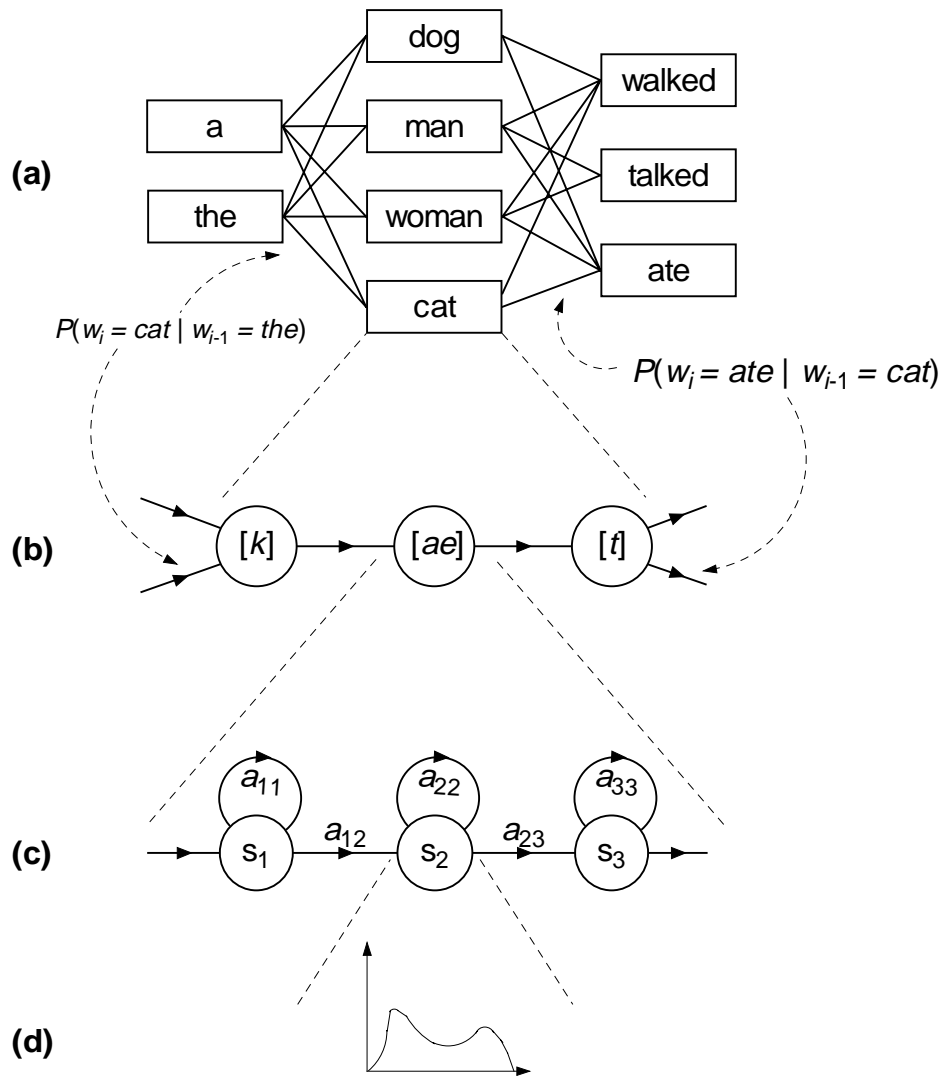


Figure 5-9. Constructing a recognition network by hierarchical composition of Markov models. (a) A regular grammar with bigram probabilities, represented as a Markov chain. (b) A word in the network is a Markov chain specifying the word's phonetic spelling. (c) Each phoneme is modeled with a three-state left-to-right HMM. (d) Each HMM state has an associated probability density for acoustic features.

How do we get any recognition results from the network, then? The optimal solution would be to find for each candidate word string \mathbf{W} the probability of the *set* of paths that correspond to that \mathbf{W} , and then identify the string whose set of paths has the highest probability. Unfortunately, this is not computationally feasible with a large network or a long utterance.

An efficient, although not optimal, solution is to use the Viterbi algorithm instead. As we know from section 4.3, the Viterbi algorithm finds the most likely path through an HMM. The particular word string corresponding to that path is then returned as the recognition result. This

string is not necessarily the same as the one whose set of paths has the highest probability. In practice, however, it is very rare that the word string corresponding to the most probable set of paths is the one actually spoken but the one corresponding to the most probable path is not. Consequently, the error introduced by the Viterbi approximation will not be significant in most cases.

It should be obvious that the number of states and transitions in a recognition network may grow very large indeed. To get an idea, consider a vocabulary of ten thousand words and a tri-gram language model: there will be roughly 10^8 states and 10^{12} transitions at the highest level of the network. If the average word length is five phonemes, and each phoneme is modeled with a four-state HMM, the number of states in the network expands to $5 \cdot 4 \cdot 10^8 = 2 \cdot 10^9$.

With state spaces this large (and with long observation sequences), even the Viterbi algorithm requires too much memory and processing power for practical use. The *beam search*, a modified Viterbi search, has been proposed to ease this problem. The beam search allows trading optimality (in the sense of finding the best path for sure) for a drastically reduced search space.

The idea of beam search is to prune, at each step, all paths that fall below a certain *probability threshold*. The threshold should be dynamic, based on the path with the highest probability at the time. This is because the path probabilities never increase with more observations; they either decrease or (rarely) stay the same. Using equation (4.16e) from the previous chapter, we obtain the probability of the most likely path at time t , $\delta^*(t)$. A dynamic threshold can now be defined, for example by dividing $\delta^*(t)$ by a suitably chosen constant K . Those states j whose score $\delta_j(t)$ falls below the threshold will not be considered when taking the next step. That is, all paths ending in those states will be purged.

The idea of beam search now becomes clear: paths falling outside an imaginary beam of light (i.e., a probability range) are pruned. Setting the rejection threshold close to the probability of the best path will terminate most of the paths, while a lower threshold allows more paths to survive. Beam search thus permits the search space to be reduced considerably, but at the risk of pruning a path that would later become more probable – or even the most probable. With a high enough threshold, it may even occur that only one path remains at some point, and that one path hits a dead end so that we get no results at all. In practice, though, the degradation in recognition accuracy caused by the beam search is rather small.

Several other approaches have been proposed for dealing with very large search networks, including, for example, different multi-pass strategies and tree-based searching. Jelinek [11] gives an introduction to these and other hypothesis search algorithms.

5.4 Limitations of HMMs in speech recognition

There are several drawbacks involved with the hidden Markov modeling approach to speech recognition. In the following, we will present some of the major limitations and briefly review the various solutions that have been proposed to overcome them:

- (1) **Weak duration modeling.** Given that a Markov chain is in state i at time t , the probability that it stays there for exactly d time frames in total is $(a_{ii})^{d-1}(1-a_{ii})$, as is easily verified. The self-transition probabilities therefore impose a geometric distribution on the state duration. This is inappropriate for most physical signals. Incorporating *explicit state duration densities* allows for more accurate modeling, but at the expense of more complicated decoding, evaluation and reestimation formulas [25]. The number of parameters to be estimated also increases.
- (2) **The assumption of conditional independence** of observations, given the state sequence. In reality, successive observations are rarely independent of each other. In fact, the reason that the observation space is usually augmented with feature derivatives (section 3.3.3) is that at least some of the time dependencies in the incoming signal can be captured in that manner. Relaxation of the assumption of conditional independence is widely recognized to be useful, but tractable methods for doing it have yet to emerge.
- (3) **Constant length observation frames.** The requirement that all observations represent a fixed-length segment of speech restricts the possibilities on feature extraction (front end processing). Better representations could potentially be extracted if the frame length were decided dynamically by the front end. Then it would be possible for the front end to segment the incoming signal into acoustically homogeneous – instead of arbitrary – intervals. Ostendorf, *et al.* [22] have proposed generalized HMMs known as *segment models (SMs)* to mitigate this problem. Whereas in an HMM, a state always generates one observation at a time, a state in an SM may generate a *sequence* of observations. In addition to allowing segmental feature extraction, the SM paradigm also enables better duration modeling than HMMs, as well as explicit modeling of correlation between observations. The drawback of segment modeling is, as expected, a substantial increase in computational complexity.
- (4) **The Markov assumption.** Hidden Markov modeling is based on the assumption that the process to be modeled obeys the Markov property, which is not always the case. The MDI optimization criterion discussed earlier allows this assumption to be relaxed, but again the

reestimation formulas become more complicated. So far, only modest gains in recognition performance have been achieved with MDI training, although there appears to be potential for more improvement.

- (5) **Lack of formal methods for choosing a model topology.** Even though the left-to-right architecture has been shown to perform better than the ergodic, a number of design decisions remain that must be made in an *ad hoc* manner. For example, whether to have alternative paths through a model, whether to use the same topology for all models in the set, and so on. There is really no general rule to solve these problems – except trial and error.
- (6) **Restricted output PDFs.** The output PDFs of a continuous density HMM must adhere to some parametric shape in order for the reestimation formulas to work. This means that the shape of the PDF must be decided beforehand, and then the models trained as if this initial assumption was correct. Inaccuracies will result if the selected shape does not fit very well to the actual observed distribution. Mixture densities are commonly used to compose more flexible shapes. Unfortunately, they are still parametric, whereas the output of a real-world process need not obey any parametric shape. A nonparametric, arbitrary density would apparently be useful in some situations. This can be accomplished by estimating the densities using neural networks (Kurimo [15]), thus resulting in an HMM/ANN hybrid. This kind of hybrid solutions have actually shown real improvement in recognition accuracy [12].
- (7) **Large amounts of training data required.** There are so many parameters to estimate in a typical set of acoustic HMMs that enough training data is hard to obtain. The various kind of HMM extensions and generalizations introduced above do not help; in fact, they make matters worse by introducing even more parameters. Techniques discussed earlier, such as semi-continuous HMMs, triphone clustering and interpolation have been used successfully to alleviate the adverse effects of insufficient training data. Numerous other solutions have also been proposed. The most obvious solution, of course, would be to collect even larger public speech databases than are available now.

6 Experiments

In this chapter, we will discuss our experimental work based on the principles introduced in the preceding chapters. A facial animation prototype, based on the phonetic approach to audio-visual speech conversion, is developed and evaluated. The purpose of the prototype is to assess the visual quality achievable with the phonetic approach, as well as the feasibility of the approach for real-time applications.

In the first section, the requirements for the facial animation prototype are laid out. In section 6.2, the functionality and implementation of the prototype system is discussed. An overview of the system's data flows is given, as well as a brief description of its software architecture. Characteristics of the selected speech recognizer, and those of the facial animation module, are outlined. Experimental results on the prototype's performance and its visual quality are presented in section 6.3. In the final section, we identify and analyze several important problems that degrade the performance and the observed quality.

6.1 *Prototype requirements*

A facial animation prototype based on phoneme recognition is to be developed and evaluated. Special attention should be paid to the applicability of the phonetic approach to real-time visual communications.

6.1.1 Input speech characteristics

The task of real-time phoneme recognition for facial animation can be characterized along the classical dimensions of difficulty as shown in Table 6-1. First of all, the mode of speaking must be continuous. A person-to-person communication device that requires leaving pauses between spoken words would be useless. The system must be able to handle normal, fluent speech.

The vocabulary of the system is small: just the phonemes. Perhaps some non-speech sounds, such as coughing, could be included as well. There is no need to produce a word string, so the problems with matching phoneme strings to words are avoided altogether. No syntactic or semantic analysis is needed, either. The phonetic alphabet is, on the other hand, very ambiguous, including many phonemes that are easy to confuse with each other, such as $[p]$, $[b]$ and $[t]$.

There must be no grammatical or other restrictions on the style of speaking: people do not always speak in grammatical or even comprehensible sentences. The system need not, and must not, try to correct any errors that the speaker makes. Basically, there should be no restrictions on which phonemes may follow one another.

The system should be speaker-independent. A short adaptation period for new users could be tolerated, though. Finally, an ability to work in noisy conditions is not required. The system can be used in fairly noiseless conditions with a good microphone.

Table 6-1. Dimensions of difficulty in the phoneme recognition task.

DIMENSION	REQUIREMENT
Continuity of speech	normal fluent speech
Vocabulary size	small (phonetic alphabet)
Sentence structure	unrestricted
Speaker variability	multiple unknown users
Amount of noise	fairly noiseless conditions

6.1.2 Response time constraints

In addition to the conventional sources of difficulty discussed above, there are added complications due to the requirement of real-time response. It is not enough that the system can keep up with fluent speech; the *latency* or *recognition delay* must also be kept at the minimum.

There is an *inherent delay* between the time that the speaker starts uttering a phoneme, and the time when there is enough acoustic data to enable that phoneme to be recognized. Once a sound can be recognized and the artificial face animated, the speaker has already moved his or her lips to a different posture. It is therefore not possible – not even in theory – to keep the animated face synchronized with the original speaker. However, animation and audio can be synchronized with each other, if not with the speaker, by delaying sound playback suitably.

Unlike inherent delays, *algorithmic delays* can be reduced by design changes or by trading off accuracy. An example of such delays is the computation of forward feature differences between speech frames, as described in section 3.3.3. By computing those differences only with respect to previous frames, the latency can be reduced, but some relevant information might be lost.

Aside from the latency issue, there is the normal question of *algorithmic complexity*. Speech processing and recognition algorithms are very complicated, and not all of them can keep up with

the incoming speech. Perhaps the most efficient solution for speech recognition, in terms of algorithmic complexity and recognition accuracy, is hidden Markov modeling. Thus, an HMM-based phoneme recognizer is probably the most suitable choice for our purposes.

6.2 *Prototype implementation*

The speech-driven facial animation prototype is based on two primary components: a speech recognition toolkit from an external party, and the custom-made Talking Heads animation software. These are insulated from each other by modular object-oriented structures, so that either component may be changed or updated without affecting the other.

6.2.1 System overview

An overview of how the prototype system works is shown in Figure 6-1. The incoming speech is forwarded to the phoneme recognizer, which outputs phoneme timing information. The phoneme timings are then converted to mouth shape timings. Finally, an animation stream is generated (this is covered more thoroughly in section 6.2.3).

The software architecture of the prototype application is illustrated in Figure 6-2. The arrows denote functional dependencies. For example, the Recognizer Interface component depends on the Result Buffer and on the particular recognizer that is being used. Notice that the recognizer can be changed without modifying any existing code: the recognizer interface is just expanded (by inheritance) to handle the new recognizer engine.

6.2.2 Phoneme recognition

The speech recognizer we are using is called *HTK API (HAPI)*. It is developed by Entropic, Ltd., which is a Cambridge University spin-off company. HTK, or *Hidden Markov Model Toolkit*, is a general speech recognition toolkit well suited for research and development purposes [37], and HAPI is the associated programming interface.

HTK supports continuous, multi-dimensional mixture distributions for emission probabilities, so that real-valued feature vectors (as produced by the acoustic front end, also included in HTK) can be used without vector quantization.

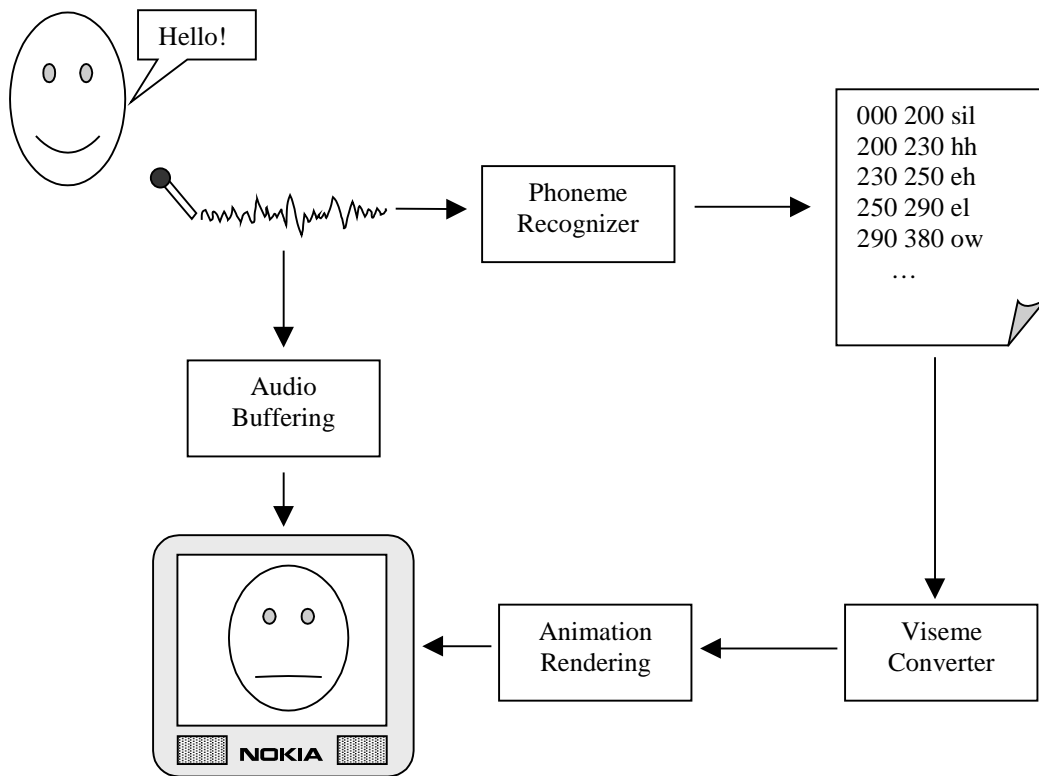


Figure 6-1. Prototype system overview.

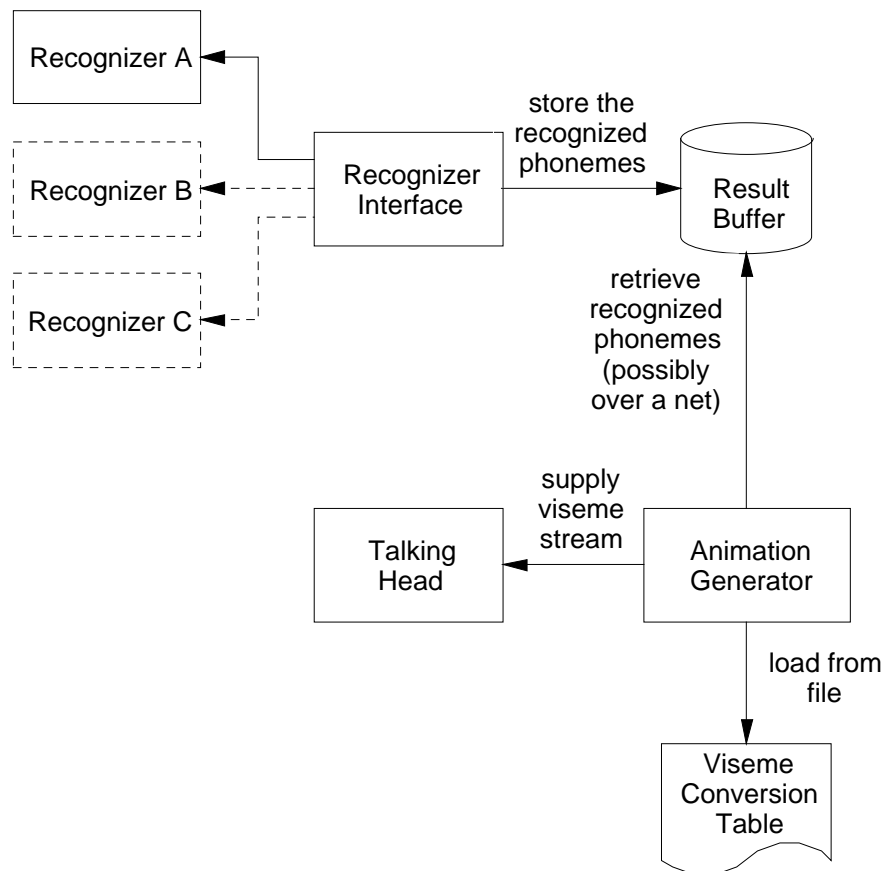


Figure 6-2. Functional dependency graph of the prototype system.

The recognizer supplies time-stamped phonemes (start-end-label triplets) progressively while recognition proceeds. The phonemes are then converted to visemes by means of a lookup table. The viseme set consists of 10 predefined mouth shapes. The viseme set does not adhere to any standards, for historical reasons, but the MPEG-4 viseme set is to be implemented in the future.

6.2.3 Facial animation

The three-dimensional head models used in the prototype are captured with a system, developed earlier, that is based on stereo imaging. Therefore, only the frontal parts of the heads are covered. The models support a predefined set of distinct mouth shapes, and linear interpolation between those. Currently, no other kind of animation except mouth movement is supported.

An animation sequence is generated from phoneme timings as follows. At phoneme start time, the corresponding mouth shape is displayed. The mouth posture is then deformed (interpolated) towards the next mouth shape, which will be displayed at the next phoneme's starting time. The mouth shape at any time during the animation is a linear combination of the current shape and the next shape:

$$Shape = c_1 \times Shape_1 + c_2 \times Shape_2, \quad (6.1)$$

where c_1 and c_2 are weighting coefficients ($0 \leq c_i \leq 1$ and $c_1 + c_2 = 1$). The weights are adjusted dynamically to interpolate from one mouth shape to another. In the table below, an animation sequence that deforms shape 3 into shape 5 is shown.

Table 6-2. Linear interpolation from shape 3 to shape 5.

$t = 50$ ms:	Shape 3, weight 1.0	Shape 5, weight 0.0
$t = 70$ ms:	Shape 3, weight 0.8	Shape 5, weight 0.2
$t = 90$ ms:	Shape 3, weight 0.6	Shape 5, weight 0.4
$t = 110$ ms:	Shape 3, weight 0.4	Shape 5, weight 0.6
$t = 130$ ms:	Shape 3, weight 0.2	Shape 5, weight 0.8
$t = 150$ ms:	Shape 3, weight 0.0	Shape 5, weight 1.0

6.3 Performance

In this section, we will present and evaluate our experimental results with the prototype. The results will be analyzed from three perspectives: (1) The recognizer's accuracy; (2) The recognition latency incurred by HAPI; and (3) The observable visual quality of the resulting facial animation.

We will start by evaluating the recognizer's accuracy, but first we need to define what we mean by "accuracy".

Recognition results are evaluated by *string alignment*: the reference transcription is aligned with the recognizer's transcription using dynamic programming (see Cormen, *et al.* [4]). Then, the differences are counted. There can be three different types of errors: *substitutions*, where the recognizer has mistaken one phoneme for another, *deletions*, where the recognizer has failed to recognize a phoneme altogether, and *insertions*, where the recognizer has recognized extraneous phonemes. These cases are illustrated below.

Ref:	dh	ow	z	sil	m	y	uw	z	ih	sh	ih	n	z	hh	aa	m	ax	n	ay	z		
Rec:	dh	ow	z	sil	m		uw	z	ih	sh	eh	n	z	hh	ax	m	ax	n	ax	ay	ih	z
						D					S				S				I	I		

Once we know the number of deletions, substitutions and insertions, and the total number of phonemes in the reference transcription, we can compute two informative values: *accuracy* and *correctness*. This is done as follows:

$$Accuracy = \frac{N - S - D - I}{N}, \tag{6.2}$$

$$Correctness = \frac{N - S - D}{N}, \tag{6.3}$$

where N is the total number of phonemes in the reference transcriptions, S is the number of substitution errors, D is the number of deletion errors and I is the number of insertion errors. Note that correctness is the proportion of recognized phonemes that are actually correct, whereas accuracy takes into account also those cases where the recognizer has inserted excess phonemes.

The recognizer characteristics in the experiment were as follows. We use a 39-element subset of the TIMIT 48-phoneme set (a few pairs of very similar phonemes, such as $[ix]$ and $[ih]$, are mapped together). There are 39 acoustic models, one for each phoneme in the phonetic alphabet. The models are conventional 3-state left-to-right monophone HMMs with no state skipping. The emission probabilities are specified by a weighted mixture of several Gaussian densities. Every state has its own output PDF, i.e., none of the states are tied together. The models were trained on a subset of the TIMIT database.

The front end was set to output 39-component, real-valued feature vectors: 12 components for perceptually based spectral coefficients and one for signal energy, 13 for feature deltas and 13 for delta-deltas. These settings are obviously fixed: changing the feature vector type would require training new acoustic models.

No language model or grammar is currently being used: any phoneme can follow any other, and all choices have an equal probability. In reality, of course, not all phoneme combinations are equally likely; for example, [m ay] is much more probable than [m g]. This suggests that a simple bigram or trigram language model for phonemes could be employed. Transition probabilities for the model could be estimated by counting the relative frequencies of each phoneme pair or triplet in a large, phonetically labeled database, such as TIMIT. Whether such a language model would improve accuracy remains to be tested.

Figure 6-3 shows recognition accuracy and correctness for several test runs of 192 utterances from 24 different speakers, taken from the TIMIT database. The utterances consisted of roughly 7000 phonemes in total. None of the test utterances was used in the training phase.

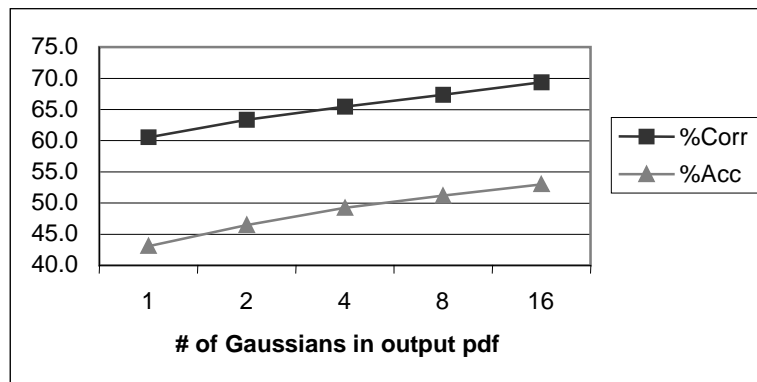


Figure 6-3. Recognition accuracy and correctness with the output probability densities approximated by a mixture of 1, 2, 4, 8 and 16 Gaussians ($N = 7215$ phonemes to be recognized).

The results indicate that increasing the number of Gaussians in the mixture densities improves recognition accuracy. With more mixture components, the estimated output PDFs can be made to fit the actual statistics of the training material better. However, with more than 16 components a phenomenon known as *overfitting* occurred: the estimated PDFs matched the training statistics too closely, degrading recognition accuracy. The best results – 70 % correctness and 53 % accuracy – were obtained with 16-component mixtures.

The following table shows the distribution of substitution errors among selected consonants (plosives, nasals and some fricatives). As might be expected, the table shows that a significant amount of [b]'s is recognized as [p]'s, and vice versa. Also, [t]'s and [d]'s are mixed up a lot. These errors are not too bad, because the phonemes map to the same visemes, anyway. On the other hand, the matrix also shows that [p]'s are often recognized as [t]'s, which is bad – this error

causes the mouth to be open when it should be shut. For the same reason, the numerous confusions of [v] vs. [b], [dh] vs. [b], [th] vs. [p] and [m] vs. [n] are harmful.

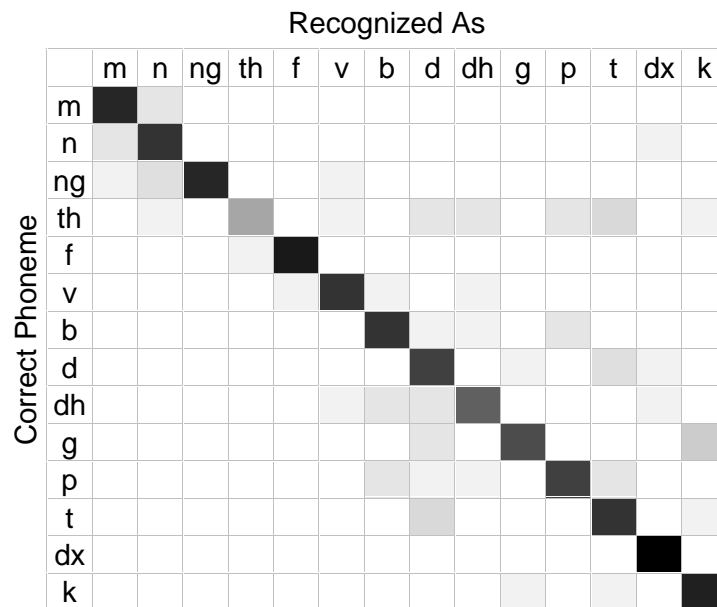


Figure 6-4. Confusion matrix for selected consonants.

Confusions among vowels are shown in the next table. The table shows that all of the vowels except for [iy] and the diphthong [ey] are very often mistaken to be the neutral vowel, [ax]. Also, all but [aa] and [aw] are often recognized as the short [ih] (as in *bit*). A very disturbing observation is that the short [uh] is rarely recognized correctly; it is almost always mistaken for [ih]. However, this might be partly due to the very low number (less than 20) of [uh]’s in the test material. Considering mouth shapes, the following errors are among the most severe: [eh] mistaken as [ax], [uw] as [iy], [oy] as [ih], and [oy] as [aa].

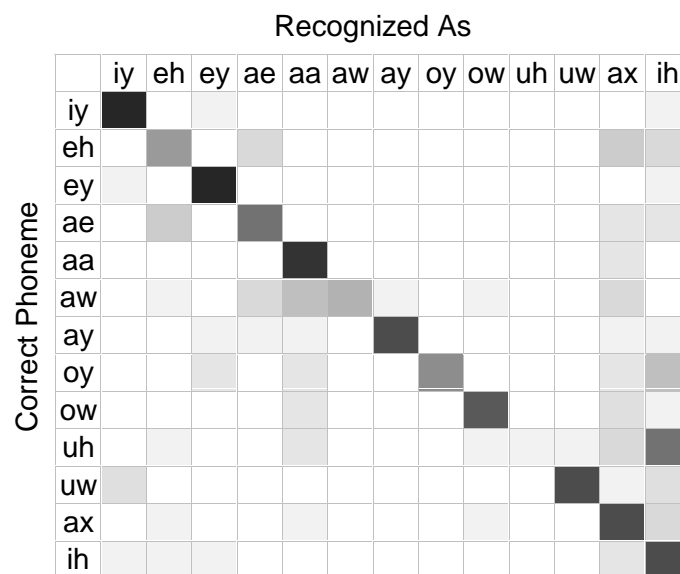


Figure 6-5. Confusion matrix for vowels.

To assess the system's applicability for real-time use, it is important to know how long does it take for the recognizer to recognize a phoneme. We have measured the delay to be about 100 ms on the average, relative to the ending time of a phoneme. Most of this delay is caused by HAPI – it would be technically possible, in the HMM framework, to guess a phoneme's identity while it is still being uttered, but HAPI just cannot do it.

The thing that matters in the end is how the mouth movements actually look. Unfortunately, the quality of the reproduced mouth movements has been deemed no better than "bearable" by observers. The quality is not sufficient to fool anyone into thinking that the Talking Head is an image of a real, speaking person. Also, the need to buffer the incoming audio and play it back with a delay of several hundred milliseconds, due to latencies in recognition and animation, would be annoying in real-time communications.

6.4 Discussion

In order to direct further research towards relevant subgoals, the most critical factors affecting the system's visual quality and its applicability to real-time use must be identified and analyzed. Some of the problems are related to the recognition phase, while others are due to the keyframe-based animation principle.

6.4.1 Problems in phoneme recognition

The main problem with phoneme recognition is the high error rate. The accuracy of HAPI with the current recognition network is less than 55 %, which means that nearly half of the phonemes are recognized either wrong or not at all. Accuracy at viseme level is somewhat better, because the mapping from phonemes to visemes is many-to-one. However, the recognizer's discrimination between certain visually distinctive phonemes should clearly be improved, as indicated by the confusion matrices. Triphone modeling or a simple language model could perhaps improve the results.

Another problem with phoneme recognition is the latency between the time when a phoneme begins and when that phoneme is recognized. This delay appears to be at least 200 milliseconds, even 400 ms, depending on the length of the phoneme. Of the delay, about 50-150 ms is caused by inherent reasons that cannot be avoided, such as waiting for enough acoustic data to become available. The rest of the delay, however, is mostly caused by HAPI. There seems to be no theo-

retical or technical reason for that delay; it just works that way. Some other recognizers would perhaps do a better job in this respect.

6.4.2 Problems in facial animation

Linear interpolation between predefined keyframes is not quite sufficient for realistic animation, because it produces too abrupt movement. The whole idea that a mouth shape can be defined as a combination of two predefined mouth shapes may not be sound, either. Furthermore, keyframe-based animation does not permit emotions to be displayed while speaking.

The above problems could be partially solved by parameterizing the mouth shape (in terms of width, height, roundness, jaw rotation, tongue position, and so on). This would require changes in both the face model construction system and the animation component. Designing a parametric facial animation system is, nonetheless, a task that needs to be undertaken in the future.

Another problem related to animation is the fact that we cannot start deforming the current mouth shape until we know the next shape, as well (see Table 6-2). That is, we need to have a *lookahead buffer* of one phoneme. This causes an additional latency of 50-300 ms (depending, again, on the phoneme length), and must also be compensated for by delaying audio playback.

For optimum quality animation, a one-phoneme lookahead is not even enough. To account properly for coarticulation (i.e., contextual effects), a lookahead of two or three phonemes would be preferable. The actual mouth shape would then be a function of more than just two visemes. Using a lookahead of multiple phonemes would, however, cause too much delay for real-time use.

7 Conclusions

We have explored the possibility of making a synthetic human head to imitate a real, speaking person by taking advantage of the correlation between the acoustic and visual components of human speech. Application areas that would benefit from such ability to convert acoustic speech into mouth movements include, for example, virtual reality, videotelephony, education, rehabilitation of the hearing-impaired, and entertainment.

We first discussed methods for capturing and animating three-dimensional models of human heads. A simple and cost-effective scheme based on *stereo imaging* and structured lighting was described for obtaining a static representation of a person's facial geometry and texture. For animation purposes, we adopted a set of 15 distinct mouth shapes (*visemes*) that relate to the basic speech sounds (*phonemes*). We then devised a mapping from a common phonetic alphabet to the set of visemes. This is known as the *phonetic* approach to speech-driven facial animation.

In order to extract phonetic information from acoustic speech, *phoneme recognition* is needed. Although full-scale speech recognition is not necessary for our purposes – we do not need the words, just the phonemes – it is useful to have an overall picture of the field. Thus, we started by analyzing the major sources of difficulty in speech recognition. We then presented the classical probabilistic formulation of the speech recognition problem. The formulation allows the task to be decomposed into three primary subtasks: *feature extraction*, *acoustic modeling* and *language modeling*. Of these, acoustic modeling is the most important in phoneme recognition.

The standard approach to speech recognition is to use *hidden Markov models (HMMs)*. They are generally effective when there are a lot of data but little knowledge. The mechanisms of human speech production and perception are still largely unknown (and very hard to formalize), but a large amount of speech data is available. Thus, HMMs are potentially well suited for the speech recognition task.

We formally defined hidden Markov models, also presenting the basic *evaluation*, *decoding* and *training* algorithms. We then applied HMMs to speech recognition, describing HMM-based acoustic models and language models, as well as their integration into a *search network*. Finally, we identified several key problem areas in HMM-based speech recognition. The shortcomings of HMMs are mostly related to the underlying modeling assumptions. Several recent proposals aim to relax some of these assumptions without incurring an exponential growth in algorithmic complexity. Practical implementations of these generalized models have yet to emerge, however.

A speech-driven facial animation prototype was developed to assess the feasibility of the phonetic approach to audio-to-visual speech conversion. The prototype consists of a phoneme recognizer, coupled with a custom facial animation module.

Several experiments were made with the prototype. The results indicate that the phonetic approach is not suitable when real-time response with a very low latency is required. First of all, there is an inherent delay in phoneme recognition of probably at least 50 milliseconds. This is because a certain amount of acoustic data is necessary to recognize a phoneme. Second, look-ahead information is needed when generating animation, due to interpolation from one mouth shape to another. This causes an additional lag of one or more phonemes (at least 100 ms on the average). To compensate for the latency, audio playback must be deferred correspondingly. If this is not an option, alternatives to the phonetic approach have to be examined.

Certain *functional* conversion schemes can map audio to mouth shapes with a lag of less than 50 milliseconds, at least in principle. No lookahead information is needed, either, because short (20-50 ms) speech segments are directly mapped to mouth shape parameters. Hence, there is no need to interpolate between the shapes. The functional approach potentially allows a much faster response to the incoming speech than the phonetic approach. However, it remains questionable how well the functional methods cope with unknown speakers, or variations in acoustic conditions.

Further research should be focused into three relatively independent areas. Firstly, the current prototype should be improved. Different phoneme recognizers should be evaluated in an attempt to find one with as little latency as possible. A simple *bigram* language model could be employed to increase recognition accuracy. Acoustic modeling could be improved by using *triphones* instead of *monophones*. Secondly, some of the functional conversion methods should be examined more closely. Finally – so that any functional method could be used in the first place – the head model capturing software and the facial animation module should be improved by implementing *parametric* mouth shapes.

Both speech recognition and facial animation are extremely challenging problems that are far from being solved. Considerable advances will have to be made in both fields until highly realistic facial animation imitating a human speaker can be generated from that person's speech alone. Nonetheless, the potential benefits of such technology, as well as the sheer challenge of realizing it, will continue to motivate research efforts in this area.

References

1. Bahl, L.R., Brown, P.F., de Souza, P.V. & Mercer, R.L., Maximum mutual information estimation of hidden Markov model parameters for speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing 1986*. 49-52.
2. Beymer, D., Vectorizing face images by interleaving shape and texture computations. A.I. Memo No. 1537, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1995.
3. Bothe, H.H., Audio to audio-video speech conversion with the help of phonetic knowledge integration. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 1997*. Vol. 2, 1632-1637.
4. Cormen, T., Leiserson, C. & Rivest, R., *Introduction to Algorithms*. MIT Press, 1990.
5. Ephraim, Y. & Rabiner, L.R., On the relations between modeling approaches for speech recognition. *IEEE Transactions on Information Theory*. Vol. 36, No. 2 (March 1990), 372-380.
6. Ephraim, Y., Dembo, A. & Rabiner, L.R., A minimum discrimination information approach for hidden Markov modeling. *IEEE Transactions on Information Theory*. Vol. 35, No. 5 (September 1989). 1001-1013.
7. Gray, R.M., Vector quantization. In Waibel & Lee [35], 75-100.
8. Harju, T., Automaatit, formaaliset kielet ja laskettavuus. Lecture notes, University of Turku, 1998.
9. Hu, J., Turin, W. & Brown, M., Language modeling with stochastic automata. *Proceedings of the IEEE International Conference on Spoken Language Processing, 1996*. 406-409.
10. ISO/IEC 14496 (MPEG-4): *Coding of Audio-Visual Objects*.
11. Jelinek, F., *Statistical Methods for Speech Recognition*. MIT Press, 1997.

12. Jones, C.M. & Dlay, S.S., Automated lip synchronisation for human-computer interaction and special effect animation. *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, 1997. 589-596.
13. Jones, D., *An Outline of English Phonetics*. (9th edition.) Heffer, Cambridge, 1967.
14. Koski, T., Hidden Markov models with applications in computational biology. Lecture notes, University of Turku, 1998.
15. Kurimo, M., *Using Self-Organizing Maps and Learning Vector Quantization for Mixture Density Hidden Markov Models*. PhD thesis, Helsinki University of Technology, 1997.
16. Lanitis, A., Taylor, C.J. & Cootes, T.F., Automatic interpretation and coding of face images using flexible models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 19, No. 7 (July 1997), 743-755.
17. Lavagetto, F., Arzerello, D. & Caranzano, M., Lipreadable frame animation driven by speech parameters. *Proceedings of the International Symposium on Speech, Image Processing and Neural Networks*, 1994. Vol. 2, 626-629.
18. Lavagetto, F., Converting speech into lip movements: a multimedia telephone for hard of hearing people. *IEEE Transactions on Rehabilitation Engineering*. Vol. 3, No. 1 (March 1995), 90-102.
19. Lee, K.-F., Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition. In Waibel & Lee [35], 347-365.
20. Luo, S.-H., King, R.W., A novel approach for classifying continuous speech into visible mouth-shape related classes. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1994. Vol. 1, 465-468.
21. McAllister, D.F., Rodman, R.D., Bitzer, D.L. & Freeman, A.S., Speaker independence in automated lip-sync for audio-video communication. *Computer Networks and ISDN Systems*. Vol. 30 (1998), 1975-1980.
22. Ostendorf, M., Digalakis, V. & Kimball, A., From HMM's to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Transactions on Speech and Audio Processing*. Vol. 4, No. 5 (September 1996), 360-378.

23. Picone, J., Fundamentals of Speech Recognition. Lecture notes, Department of Electrical and Computer Engineering, Mississippi State University, 1996.
24. Rabiner, L.R. & Juang, B.-H., *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.
25. Rabiner, L.R., A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. Vol. 77, No. 2 (February 1989), 257-286.
26. Rao, R., Mersereau, R. & Chen, T., Using HMM's in audio-to-visual conversion. *Proceedings of the IEEE First Workshop on Multimedia Signal Processing, 1997*. 19-24.
27. Russell, S. & Norvig, P., *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.
28. Samuelsson, C. & Voutilainen, A., Comparing a Linguistic and a Stochastic Tagger. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL), 1997*.
29. Schafer, R. & Rabiner, L.R., Digital representations of speech signals. In Waibel & Lee [35], 49-64.
30. Seneff, S., A joint synchrony/mean-rate model of auditory speech processing. *Journal of Phonetics*. Vol. 16, No. 1 (January 1988), 55-76.
31. Sondhi, M.M., New methods of pitch detection. *IEEE Transactions on Audio Electroacoustics*. Vol. 16 (June 1968), 262-266.
32. Teuhola, J. & Raita, T., Application of a finite-state model to text compression. *The Computer Journal*. Vol. 36 (1993), 607-614.
33. Tsukada, H., Yamamoto, H., Takezawa, T. & Sagisaka, Y., Reliable utterance segment recognition by integrating a grammar with statistical language constraints. *Speech Communication*. Vol. 26, No. 4 (December 1998), 299-309.
34. van Alphen, P., *HMM-based continuous-speech recognition: Systematic evaluation of various system components*. PhD thesis, University of Amsterdam, 1990.

35. Waibel, A. & Lee, K.-F., *Readings in Speech Recognition*. Morgan Kaufmann, 1990.
36. Watt, A., *3D Computer Graphics*. Addison-Wesley, 1993 (2nd edition).
37. Young, S., Odell, J, Ollason, D, Valtchev, V. & Woodland, P., *The HTK Book*. Reference manual for Hidden Markov Model Toolkit (HTK) V2.1. Cambridge University, 1997.
38. Zue, V., The use of speech knowledge in automatic speech recognition. In Waibel & Lee [35], 200-213.