SPEECH RECOGNITION

Speech is our preferred medium for everyday human-to-human communication. Thanks to the recent developments in speech recognition technology, this medium is now becoming our premier choice for human-to-machine communication as well. Speech recognition technology enables a computer to transcribe spoken words. Gone are the days when we need to master a computer keyboard to prepare a letter. Instead, we can create it via our voice. We can issue normal formatting

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*. Copyright © 1999 John Wiley & Sons, Inc.

commands as well, such as "next paragraph" and "capitalize," simply by speaking them. Indeed, this technology has already found its way in a wide variety of application. Medical practitioners, such as radiologists, pathologists, and internists, use it to prepare diagnostic reports for their patients. Legal professionals depend on it to produce documents and briefs for their clients. Students benefit by writing their homework essays with the help of this technology. An author can speed up the production of a manuscript by dictating rather than typing it. We expect this trend to accelerate and this technology to take its legitimate place as an indispensable computer interface such as a mouse or a keyboard.

A speech recognition system is configured in a variety of ways depending on its intended application (1–9). In addition to dictation, other applications include automated operator-assisted call handling, home banking over the telephone, package sorting, assembly line quality control, aid for the handicapped, and educational software and voice-driven games. To introduce the basic concepts of this technology, we concentrate on describing one application—that of desktop dictation—in detail. The block diagram of Fig. 1 illustrates it schematically. We note that the layout for some of the other applications may differ considerably from this one. We point out some of these variations later in this article.

The *user interface* shown in Fig. 1 contains a microphone for speech input in addition to the usual computer peripheral hardware (not shown), such as a monitor, a keyboard, and a mouse. The other four components are called *signal processor, acoustic model, language model,* and *decoder*. When we speak into the microphone, the signal processor analyzes this input and derives a set of features. For instance, each feature may correspond to a measure of acoustic energy over a particular frequency bandwidth during a short time interval. The decoder acts upon these features with the help of the acoustic model and the language model components, searching through all possible outcomes to determine what was said. The decision of the decoder is displayed on the monitor built into the user interface.

The functions of the acoustic model and the language model components are complementary. The language model maintains some knowledge of the language. It helps the decoder by predicting what we are likely to say next at a given instant. For example, if we say "Roses are" the language model can guess that the next word is probably going to be "red" (or some other plausible color). But if we utter "Books are" the next likely word is probably not "red," but the homophonous word "read." The acoustic model, on the other

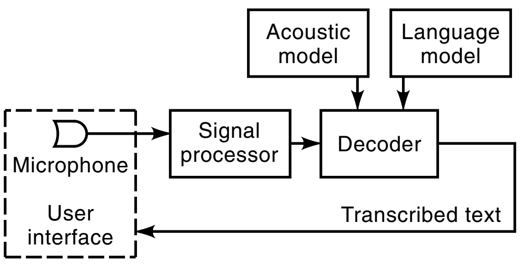

Figure 1. Schematic of a speech recognition system. Speech input picked up by the microphone is processed by the system components. The transcribed text is displayed at a monitor built into the user interface.

hand, attempts to identify different sounds in our speech from their acoustic characteristics. When we say "roses" or "books" it attempts to recognize the specific acoustic signature of each word. However, it cannot differentiate between "red" and "read" of this example, since they both sound the same, and it must call upon the language model to make the distinction. The two ingredients of a speech recognition system help each other out in this manner. We expound on this elemental description of a speech recognition system in the following section.

Before we conclude these introductory notes, let us clarify the distinction between speech recognition and some of the other closely related applications that operate on speech input. For instance, the disciplines of speaker verification and speaker identification seek to distinguish one speaker from another. Language identification deals with the problem of ascertaining a speaker's language by examining a given sample of his or her speech. These disciplines share many common speech recognition methodologies. However, the implementation details depend on the application at hand. We refer the readers for further study to other speech-related articles in this encyclopedia.

It is also important to distinguish between speech recognition and speech understanding. Unlike speech recognition, which deals with the problem of transcribing speech, a speech understanding system is faced with the task of making some sense out of the spoken words so that it can respond properly. In general, this is a complex unsolved problem, as a word or a sentence may be interpreted in a variety of ways depending on factors such as the domain of discourse and semantic context. For instance, a "bank" can mean either a financial institution or the side of a river. However, in certain cases, we can restrict the domain of discourse sufficiently to construct a class of limited but useful speech understanding systems. These are called conversational systems. For example, an airline travel information system (ATIS) can respond to natural spoken queries regarding flights between major US cities (10). A system called JUPITER developed at the Massachusetts Institute of Technology can provide weather information to more than 500 cities worldwide through a telephone conversational interface (1).

We now indicate how the remainder of this article is organized. The first section is targeted to all readers without any specialized scientific or technical background. We present the basic concepts of this technology, highlight some of its application potential, and indicate its current limitations. In the second section, we describe in considerable detail the theoretical underpinnings of this technology. We assume that a reader of that section has a thorough background in information theory and mathematics. For instance, we freely draw upon concepts established in information theory such as Markov modeling and search strategy for an optimal path. Finally, the third section of the article is meant for speech specialists who may be engaged in speech recognition or related activities.

FUNDAMENTALS

Description of a Speech Recognition System

We can implement a speech recognition system in a number of different ways. We describe most of the important approaches in the following section. However, our initial goal is to introduce some key concepts of this technology within a simple but standard framework shown in Fig. 1. To avoid complications, we defer discussions of alternative methodologies to a later subsection.

Signal Processor. Turning our attention to the signal processor of Fig. 1, one common strategy for implementing this function is to derive a fixed set of *spectral* features at a regular time interval. This time interval is on the order of 10 ms (one-hundredth of a second). The number of features in a set is typically about 20. Speech frequency spans the bandwidth from about 120 Hz to 8000 Hz. Each of the spectral features in a set represents the acoustic energy in a particular portion of this bandwidth in a given time interval. For instance, the first feature may indicate the energy during a 10 ms time interval in the 120 Hz to 150 Hz bandwidth range. Similarly, the last feature may correspond to the energy in the 7000 Hz to 8000 Hz range. Notice that the frequency spacing is not uniform. More information in speech is carried by the lower range of frequencies. Consequently, we use narrower bandwidths at lower ends of the frequency scale to capture the finer structure of a speech signal. The nonuniform partitioning of the speech bandwidth is called *mel frequency scaling*. It is interesting to note that human auditory mechanism carries out a similar frequency scaling. For our purpose, we can derive these spectral features by passing the input through a bank of filters of different bandwidths tuned to different center frequencies. Most modern implementation of the filter bank involves the use of *fast Fourier transform* (FFT) along with some pre- and postprocessing. Often these features are further transformed by a cosine function to a cepstrum (11) representation for improved performance.

Collectively all the features in a set produce a snapshot of the whole speech bandwidth during a 10 ms time frame. This collection of features is called a feature vector. If the utterance lasts for a second, for example, each of the 100 frames in that period is represented by a separate feature vector, leading to a total of 100 such feature vectors.

Another goal of signal processing is to capture a measure of dynamics of the speech signal in addition to the static snapshot every 10 ms. This is often done by computing a difference vector between the current vector and a previous one and appending this difference vector to the current one (12). Taking difference from a sample up to 90 ms in the past is fairly common.

Acoustic Model. We briefly discussed in the introduction the role played by the acoustic model in a speech recognition system. The acoustic model learns to perform its task of discrimination between different speech sounds through a design procedure called *training,* illustrated in Fig. 2. A speaker or

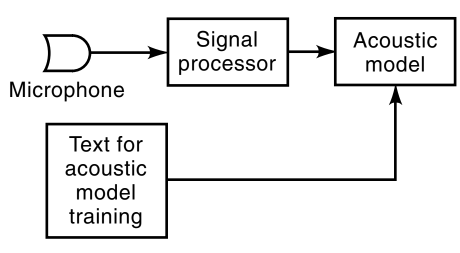

Figure 2. Acoustic model training. The text and the corresponding speech input are correlated to generate the acoustic model.

a group of speakers reads a given text. The signal processor analyzes the acoustic data and derives a feature vector for each 10 ms time frame, as described in the previous sub-subsection. This text and the feature vectors are supplied to the acoustic model as shown in Fig. 2, so that it can correlate the feature vectors with the text and learn to characterize the speech sounds.

A judicious choice for units of speech sounds to learn is based on the phonetic spelling of a word, rather than the word itself. In the English language, for example, there are only some 50 phonemes. If we characterize these phonemes, we can then handle all the words in the language. On the other hand, if we try to learn the acoustic signature of each word separately, we face the daunting task of categorizing several hundreds of thousands of words, along with all their dialectal variations. Our reliance on phonetic units as the basis for recognition makes it unnecessary for the training text to include every word of the vocabulary. Only sufficient examples are needed to model all the phonemes.

The first step in our acoustic model strategy is to install a table of phonetic spellings for each word under consideration. This is called the table or dictionary of baseforms. Each entry in this table represents a way a word is pronounced. There are often multiple entries for a single word in this table, corresponding to different possible pronunciations for that word. In addition, people from different regions of the country or nonnative speakers impart their own accents. Depending on the scope of our speech recognition system, we may want to include those phonetic spellings as well in our table of baseforms.

With the help of the baseform table, the acoustic model attempts to correlate the incoming feature vectors with the phonetic spelling of the corresponding text. However, the task is further complicated by the idiosyncrasies in our speech, as explained in the next paragraph. We are normally unaware of these peculiarities because our auditory mechanism effectively deals with them.

When we say a word, we usually do not produce all the sounds in that word at a uniform rate. In addition, if we repeat that word, the timings are likely to be different. Another artifact of our speech production process is that our vocal apparatus wavers imperceptibly when making the sounds. For instance, when we say "Books" the resulting string of phonemes with "oo" as the target will probably consist of several instances of "oo" mixed with instances of other sounds, such as "uh" and "eh," which are close to the target sound. Also, our vocal apparatus cannot instantly switch from producing one sound to the next. Consequently, the portions of our speech during a transition between two sounds do not precisely fit the characteristics of either the preceding or the following sound.

To handle this lack of precision in speech production we introduce the notions of observations and hidden states and deal with them within a probabilistic framework. The observations are the imprecise renditions that we notice directly by examining a string of feature vectors. The hidden states are an idealized representation that we assume exist and are related to the observable data. Our goal during training is to examine the observations and discover both the parameters of a mathematical model producing the hidden states and the relationships between the hidden states and the observations. A convenient approach to realize this goal is to cast this problem in terms of a representation called the *hidden Markov model* (HMM) (2–7). An HMM deals with two sets of probabilities. The first set, called the transition probabilities, specifies how transitions take place from one hidden state to another. The second set is termed the output probabilities. It indicates how an observation is probabilistically generated during a transition between two hidden states.

Now we return to our description of training the acoustic model component. Recall that a baseform for a word is a string of phonetic symbols. Let us view each phonetic symbol as an HMM. So the word is represented by a concatenation of HMMs. An illustration of HMMs concatenated for the three phonetic symbols of the word "one" is shown in Fig. 3. A self-loop signifies repetition of a state, and a transition is indicated by an arrow to the next state.

When a speaker says a word, we look up its baseform, construct the corresponding HMM structure, and reconcile the observations with the output generated by this structure. We utilize a substantial body of text, consisting of some 1000 to 2000 words, and its corresponding feature vectors during training. The purpose of training is to estimate the transition and output probabilities so that the probability of the observed data is maximized. This is called the *maximum likelihood principle.* We cannot directly estimate the transition and output probabilities in a straightforward manner, due to the hidden nature of the states. The technique used for estimation is called the *estimation–maximization* (EM) algorithm. It consists of repeated application of a procedure in which the probabilities are successively adjusted to maximize the likelihood score over the whole body of training text and the corresponding feature vectors. When applied to HMM parameter estimates, the algorithm is also known as the *forward–backward* or *Baum–Welch* procedure (13).

The decoder depends upon the acoustic model to furnish it with an acoustic match score corresponding to each input word. The score reflects how well the observations corresponding to the input match the states predicated by the transition and output probabilities associated with a word in the vocabulary. We provide a more detailed discussion of HMM and the EM algorithm in the section on detailed theory.

Language Model. Design of the language model component depends on the application under consideration. For example, consider a voice-driven banking application. The first step of a typical transaction may consist in uttering a sequence of digits as the password. The second step, after password verification, may call for the customer to say one of three words "savings," "checking," or "loan." In this situation, the speech recognition system is programmed to expect nothing but a fixed number of digits during the first step and one of only three words in the second step. This type of simple language model is called a *finite-state grammar.*

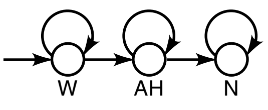

Figure 3. HMM representation of the word "one" with its three phonetic symbols. Repetition of a state is shown by a self-loop. Transition is indicated by an arrow to the next state.

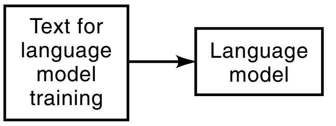

Figure 4. Language model training. A large body of text is used for this purpose.

In contrast, the language model for more complex applications, such as dictation, operates within a probabilistic framework. We train it, as shown in Fig. 4, by utilizing a large corpus of text, typically on the order of tens of million of words, which we assume is a representative sampling of the language. Our goal is to model the contextual information inherent in this text by counting how often a word occurs in a given context of previous words. Due to cost and robustness considerations, we typically limit our context lookup to a maximum of two previous words. This type of count, made for a word within the context of two previous words, is called a trigram count. In practice, we normalize these counts by expressing them as probability values, called trigram probabilities. Similarly we calculate bigram and unigram probabilities by examining the frequencies of occurrence of pairs of words and of each word by itself. The decoder typically uses a weighted sum of these three probabilities as the language model score.

Decoder. Let us now discuss the operation of the decoder. There are many possible structures for decoding speech. For instance, a dictation system designed to handle a vocabulary of tens of thousands of words typically uses a two-step procedure, the steps being called the *fast match* and the *detailed match.* For an input word, we cannot afford to examine all words in the vocabulary thoroughly for the best match. Consequently, the fast match carries out an approximate search to trim down the list of possibilities quickly. The detailed match acts upon the trimmed list.

On the other hand, if the speech recognition system deals with a small vocabulary, as in the banking application mentioned in the previous section, we can skip the fast match and carry out only a detailed match.

The decoder evaluates alternate hypothesized word strings in parallel by maintaining a stack of top string choices. This is called the *stack decoder* algorithm (14). Each time a string is extended by another word, after consulting both the acoustic model and the language model scores, the top choices are reshuffled to reflect the current best selections. In the meantime, the choices made several words earlier begin to firm up, that is, not change with the latest extensions. We consider these words as the ones recognized by the system. When we reach the end of a sentence, we can firm up the whole string and finish recognizing the whole sentence. We describe further details of our decoder strategy in the section on detailed theory.

This completes a general description of our speech recognition system. We saw how the signal processor generates a set of feature vectors to represent the speech input, how the acoustic model and the language model components are trained, and how they help the decoder to decide on the string of recognized words.

We can apply a database of new sentences never used for training to “test” the efficacy of a recognition system. A speaker or a group of speakers utters the test sentences. We compare the recognized output words against the known text and determine recognition accuracy. Commercial recognition systems with a vocabulary size in the range of 20,000 to 30,000 words claim typical word accuracy rates on the order of 90% to 98%. However, some users fare better than the others. Often poor performance can be traced to a specific factor such as accented speech, background noise, or improper microphone placement. But in some cases, the culprit may turn out to be more elusive.

Configurations and Compromises

We can configure a speech recognition system in a number of different ways to strike a compromise between accuracy and convenience. Consider a speaker-dependent speech recognition system that is specifically tailored to each user of the system. Compared to a speaker-independent system, it tends to have better accuracy. However, a speaker-independent system is more convenient, since customization for each user is unnecessary. A popular middle ground is to customize a speaker-independent system by a rapid initial adaptation step. A typical approach is to ask the user to read a short text so that the acoustic model parameters can be adjusted. Language model adaptation is also possible, for instance, by monitoring a user’s preferences for words and sentence constructions.

Another common alternative is to resort to *isolated* speech input for performance gain. In this type of system, the user pauses briefly, at least on the order of a tenth of a second, between successive words. The pauses provide helpful cues to the recognition system regarding word boundaries. However, this mode of speech entry is somewhat unnatural. *Continuous* speech recognition systems boast of a more natural mode of speech entry without such artificial pauses. But the accuracy may suffer, particularly if the speech is spontaneous. Spontaneous speech tends to be more casual and often contains extraneous sounds, unconsciously produced by us, such as “uh” and “um,” which are problematic for a recognition system.

There are other external factors that can hurt the performance of a speech recognition system. We discuss them in the next subsection, along with some common remedies.

Robustness Issues

A speech recognition system must not only be as accurate as possible, but also be resistant to external detrimental factors such as background noise and channel distortion. Background noise such as conversational babble and automobile and aircraft din tend to degrade recognition accuracy. A telephone line can introduce channel distortion, as its electrical characteristics may fluctuate from one instant to another.

We try to counteract the damaging effects of such extraneous factors by a host of techniques. We can use directional or noise-canceling microphones to reduce background noise. An algorithmic approach is to learn to recognize various types of noise from their acoustic signatures. That is, we deliberately subject our recognition system to these noise sources during training so that it can acoustically distinguish them from true speech input. Subsequently, the decoder will utilize this knowledge to try to ignore similar noise in the input stream.

A simple but effective compensation technique that serves a dual purpose is called mean normalization. In it, each feature vector is scaled, mapping its mean value to a fixed quantity. Its chief advantages are that it tends to take care of loudness variations in both speech and background noise.

On-line adaptation is yet another method for combatting the effects of noise and distortion. For instance, we can characterize the speech and noise by a set of reference templates and update them as needed. The goal is to compare the corrupted input speech against the updated templates and derive a suitable correction factor.

User Interface

Despite all such efforts towards robustness, a speech recognition system is likely to make some transcription errors. Techniques for handling these errors in an efficient manner constitute an integral part of the design of the user interface.

Consider the case of a typical dictation system. We may want to correct an error made by the system as soon as we see it, or, we may want to do it later at our own convenience, for example, after dictating the whole document. There are other relevant issues. How do we position the cursor on the erroneous word: by voice, by mouse clicks, or by keyboard strokes? After positioning the cursor do we say the desired word again, or do we type it in to make the correction? If the recognition system maintains a list of alternate top choices for each spoken word, we can search through that list for the correct one. Some recognition systems save the audio data, usually in a compressed format, and can play them back later on demand to refresh our memory. Some can play a synthesized version of the written text as well. This is helpful if we prefer to listen to rather than visually examine a document to make our corrections or changes. The audio playback features are essential for a young child or a learning-disabled adult user with reading problems.

Similarly, the user interface for a conversational system is tailored to its specific needs. For example, if an ATIS system recognizes “Houston” for “Boston” and provides flight information for that city in Texas, we need an efficient and least irritating way of correcting that mistake, receiving travel data for our desired destination, the city in Massachusetts.

Alternative Strategies

We discussed a standard configuration for a speech recognition system in the previous paragraph. However, there are a number of other variants (3,5–7,15). For instance, instead of representing the acoustic energy in a particular frequency band, a feature may relate to some form of articulatory or auditory parameter. Inclusion of pitch frequency as a feature is important for some languages, such as Mandarin Chinese, where tonal information is vital. A speech recognition system designed to operate over a telephone line would contain special signal processing and acoustic modeling features to handle the line noise.

Instead of using the FET technique for signal processing, as described in a previous section, some systems resort to a time-domain based approach called linear predictive coding (5–7). We used the unit of phoneme for our acoustic modeling. Other possible choices include larger units such as diphones, demisyllables, syllables, and words. We mentioned stack decoding in describing our decoder strategy. Other possibilities include a beam search technique, which is discussed later in the Detailed Theory section.

Artificial neural networks, described in the section on exploratory work, can be used to implement an acoustic model. They learn to characterize different sounds by studying some speech data tagged with their class affiliations and generating complex decision surfaces while adjusting a set of innate weights and thresholds. Language models may be customized to handle specific tasks. For example, a voice-enabled telephone dialer is programmed to accept a fixed number of digit strings only.

History

The idea of speech recognition has attracted and fascinated mankind since ancient times. An early example is in a tale from the *Arabian Nights* where a secret door to a treasure-filled cave is activated by uttering the phrase "Open sesame."

Initial attempts to develop speech recognition systems (16) tended to rely heavily on heuristic methodologies. Most modern recognition systems trace their roots to HMM-based work done in the IBM Corporation and Carnegie Mellon University in the early 1970s (17,18). More recently, researchers from several countries—including the United States, Canada, England, France, Germany, Italy, Spain, Japan, China, and Russia—have made significant contributions to the development of this technology (5–8,15,19).

DETAILED THEORY

Algorithms for Automatic Speech Recognition: Overview

Most of the speech recognizer components described above are in practice implemented as software algorithms. Although the task of an automatic speech recognizer (ASR) is ostensibly one of decision making—choosing the correct sequence of words from a vocabulary—the algorithms do not consist of simple decision rules but require numerically intensive floating-point computation. Natural human speech does not obey simple engineering specifications, but exhibits a complexity more typical of biological systems. The success of early manually designed, rule-based ASRs was therefore very limited. Modern ASRs rely instead on large mathematical models with millions of parameter values empirically estimated through numerical optimization. Probability theory plays an important role in the design of these systems.

Figure 5 shows the overall concept of a speech recognizer. The user speaks into a microphone; the signal a goes to an ASR, which is usually a computer program; and the recognized message w emerges to be displayed as text or to cause some requested action such as the closing or opening of a program window.

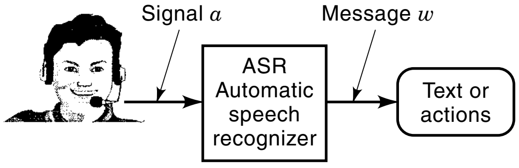

Figure 5. The task of an Automatic Speech Recognition system is to translate an acoustic speech signal a into a computer- or human-readable message w.

The important data entities are

- The acoustic speech signal a, for example a spoken sentence.
- The text or message w.

If $\mathscr{A}$ is the space of acoustic signals, and $\mathscr{W}$ the space of messages, then an ASR is a function

$$D : \mathscr{A} \mapsto \mathscr{W} \tag{1}$$

The function D, unfortunately, is not a simple or obvious encoding. It cannot be derived entirely from theory, nor entirely from empirical data. The approach usually taken is to assume that there exists a joint probability distribution $p(a, w)$ that belongs to some parametric family $p(a, w|\theta)$. By statistical estimation methods, a value $\hat{\theta}$ is then obtained for the parameter vector on the basis of a sample known as the *training corpus,* which, for a high-accuracy ASR, should contain at least tens of hours of speech. The mapping D is then constructed from the estimated distribution:

$$D(a) = \underset{w}{\operatorname{argmax}}\; p(a, w|\hat{\theta}) \tag{2}$$

On the training corpus, a and w are known, say a_{R} and w_{R}, and $\hat{\theta}$ can be estimated for example by the maximum likelihood (ML) method (see ESTIMATION THEORY) as

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\; p(a_{\mathrm{R}}, w_{\mathrm{R}}|\theta) \tag{3}$$

Equations (2) and (3) constitute a deceptively simple statement of the entire procedure for designing an ASR system. The reality is much more complicated, because the distribution $p(a, w|\theta)$ is not simple like a Gaussian or a gamma distribution. It is, in fact, too complicated to write down in one piece in mathematical notation, and is ultimately defined by the software that implements the ASR under real world constraints on bytes and flops. The software typically performs a cepstral analysis on the incoming waveform to convert it to a sequence of vectors, and then defines distributions in the resulting space of vector sequences by means of a multilayered hidden Markov source model, consisting of a language model layer, a pronouncing-dictionary layer, and an acoustic-realization layer. The last contains probability distributions of thousands of allophone segments, each distribution being a mixture of tens of multivariate Gaussians in a space of tens of dimensions. All the means and covariances of these Gaussians, as well as their mixture weights, and the many other parameters in other layers and other parts of the system, including the language model, together constitute the parameter vector $\boldsymbol{\theta}$, which may have well over 1,000,000 components. ML estimation, Eq. (3), is used for some of these components, but more complicated methods are needed for other components.

In the following, we first take an intuitive approach to describing the operations that a typical ASR performs to implement D. To motivate some of the rather elaborate algorithms found in actual ASRs, we start with simpler alternatives, then describe their deficiencies and possible remedies to demonstrate the need for more complicated schemes.

After describing the operation of an ASR, we turn to the problem of training (i.e., optimizing) it, again considering simpler algorithms first. We discuss both the philosophy and specific methods of optimization.

Operation of an Automatic Speech Recognizer

To extract the message from the audio signal, a typical ASR performs the steps shown in Fig. 6:

1. Analog-to-digital conversion.
2. Signal processing.
3. Acoustic processing.
4. Statistical decoding.

Analog-to-Digital Conversion. Step 1 is implemented in hardware. Its output is an electrical signal representing the acoustic waveform sampled at a rate of 8000 s^{-1} or more, with each sample represented as a binary 8 bit or 16 bit number. This data format is known as PCM (pulse code modulation). The number of possible PCM waveforms is very large: even a short segment of only 0.01 s constains at least 640 bits, yielding $2^{640} \approx 4.6 \times 10^{192}$ different possible waveforms. With such signal-space complexity, an ASR will never encounter the same waveform twice—not only does every person have a unique voice, but even every reading of the same sentence by the same person presents a unique waveform.

Signal Processing

Acoustic Vectors. Step 2, signal processing, reduces the size of the signal space but still leaves it large enough that no two utterances are identical. The output of this step is a point, or vector, in a multidimensional *acoustic feature space.* As the sound changes, the point moves. The signal processor recomputes the acoustic feature vector typically 100 times per second. Each such computation constitutes a frame.

Each acoustic feature vector may be a set of mel frequency cepstral coefficients (MFCCs) along with delta-cepstrum and delta-delta-cepstrum coefficients. See SPEECH PROCESSING for motivation and details on MFCC and other types of feature vector computation, and CEPSTRAL ANALYSIS OF SPEECH for delta-cepstrum formulas. A typical acoustic vector might contain 39 components: 13 each of cepstral coefficients, delta-coefficients and delta-delta-coefficients. The cepstrum describes the short-term power spectrum of the sound; its perceptual correlate is sound quality or timbre. Not only does the cepstrum of a violin note differ from that of a human voice, but a voice saying "ee" has a cepstrum different from a voice saying "aw." In fact, every speech sound has a somewhat different cepstrum, as does the same sound spoken by a different person. Even the type of microphone and the acoustics of the room can affect the cepstrum. Furthermore, because frames are not phase-locked to components of the signal and because the signal may itself be a stochastic process, acoustic vectors will exhibit some random fluctuation even when the input is nominally steady-state, as with white noise or a periodic tone. Each type of sound, therefore, produces a little cloud of points in the acoustic space. The location of the cloud is different for each sound and thus can serve to identify the sound, but there is some overlap between the clouds, preventing unambiguous identification.

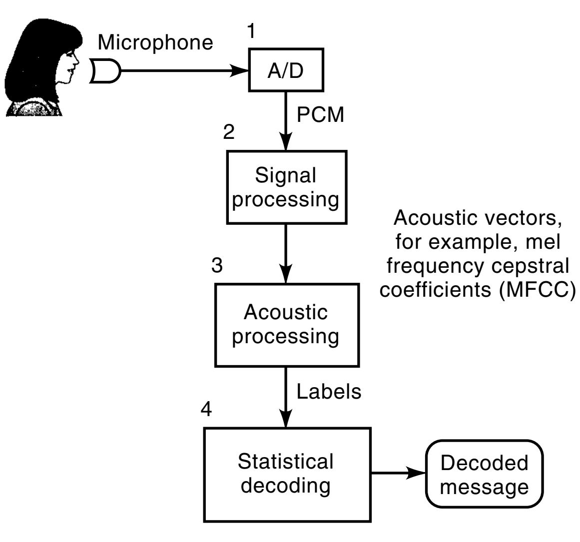

Figure 6. To accomplish its task, a speech recognition system performs the four major processing steps shown here.

Speech Sounds. Neither human listeners nor mathematical algorithms can definitely identify short segments of sound, of the order of 10 ms to 20 ms in duration. As a general rule, the longer a speech segment is, the easier it is to recognize. A word is easier to recognize than is a single speech sound. A sentence is easier to understand than a single word—you can fill in poorly heard words from context. The number of possible sentences, however, is enormous; hence a recognizer must break down sentences into smaller components. Even the number of words is too large to allow us to get an adequate sample of pronunciations of each word in a variety of contexts. For this reason, large-vocabulary ASRs today use sound units shorter than words.

To a first approximation, a spoken word is a sequence of sounds, each of which corresponds to a cloud or region in the acoustic space. Every language has its own set of distinctive sounds, called *phonemes,* which suffice to differentiate the words of that language from each other. In American English, for example, the vowels in the words "eat" and "it" are different, representing different phonemes, but the consonant "t" in both words represents the same phoneme. Even though every utterance of the word "eat" is actually slightly different, these differences are not phonemic, and an ASR must ignore them, but must detect phonemic differences such as those between "eat" and "it." If an ASR could partition the acoustic space into regions corresponding exactly to phonemes, it could generate a phonetic transcription containing just the information necessary for speech decoding. This, unfortunately, is impossible because the physical quality of each phoneme's sound depends strongly on context. Inertia prevents articulatory organs—lips, tongue, jaw, etc.—from making step-function changes; hence a given phoneme will be influenced by what came before or after it. This phenomenon is called *coarticulation.* The "r" sound in "tree," for example, is physically quite different from that in "through" and may, in fact, resemble a "sh" sound. Similarly, the "i" sounds in the words "bit" and "bib" are different, as shown in the spectrograms in Fig. 7. Different physical realizations of the same phoneme in different contexts are called *allophones.* Coarticulation causes each phoneme to have so many allophonic variations that there ex-

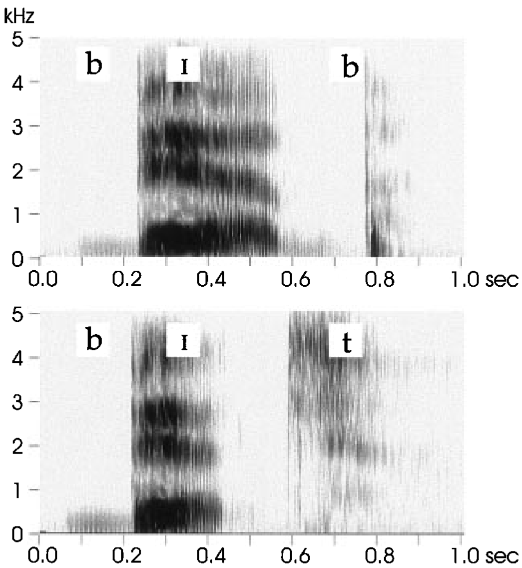

Figure 7. Sound spectrograms show that the same speech sound—the vowel "I" in the words "bib" and "bit"—differs in duration and spectral pattern in different contexts. The horizontal axis represents time and the vertical axis frequency. Darker areas represent higher energy.

ists no reliable frame-by-frame mapping from the acoustic space to phonemes.

The next step, acoustic processing, nevertheless attempts to partition the acoustic space into linguistically or phonetically informative subregions.

Acoustic Processing

Labeling. Acoustic processing, step 3, partitions the acoustic space into subregions, again reducing the complexity of the signal. At each frame (i.e., each $\frac{1}{100}$ s) the acoustic processor compares the acoustic vector with a set of *acoustic prototypes,* selects one or more of the best-matching ones, and thus attaches a label to each frame, or a list of labels ordered according closeness of match, possibly accompanied by a numerical measure of the closeness for each label. The resulting label stream may be thought of as an approximate phonetic transcription, but its accuracy is limited by coarticulation and random variations. The labels are an intermediate representation of the signal, closer to the message than is the original PCM, but not so close that a simple table lookup could extract words of text from it.

Simple Vector Quantization. In the simplest labeling scheme, each acoustic prototype is just a point in the acoustic vector space—perhaps the center of a cloud representing some particular sound. For each frame, the acoustic processor simply finds the prototype that is closest to the current acoustic vector. Thus, if π_i is the point representing the ith prototype and $\boldsymbol{x}$ is the current acoustic vector, then the label ℓ for the current frame is

$$\ell(\boldsymbol{x}) = \operatorname*{argmin}_i |\boldsymbol{x} - \pi_i| \tag{4}$$

This operation, known as *vector quantization* (VQ), maps the acoustic vectors from a continuous domain into a finite set of codes. The set of point prototypes is called the *codebook.* Each prototype can be identified simply by a number, e.g. i in Eq. (4), or by some mnemonic label such as the name of a speech sound frequently associated with that prototype.

Symbols for Speech Sounds. Linguists have traditionally labeled speech sounds with specialized, language-independent symbols, such as those of the International Phonetic Alphabet (IPA), loosely based on the Latin alphabet and Latin pronunciation but with many additional symbols and diacritics. In ASR technology, however, it has become customary to construct phoneme or sound labels from standard ASCII alphabetic characters, incorporating vernacular spelling conventions. Thus the pronunciation of the word "one" might be represented in IPA symbols as "wʌn" but in a typical ASR system as ``W AH N.''

A More Powerful Labeling Algorithm. The simple VQ scheme just described, which defines each prototype by means of a single point, is not powerful enough for large-vocabulary recognizers. In those ASRs each prototype is, instead, a probability density function in acoustic space. The labeler then selects the prototype that has the highest likelihood. Let $f_i(\cdot)$ be the density function that describes the ith prototype, and let $\boldsymbol{x}$ be the acoustic vector for some frame. The label ℓ for that frame is then

$$\ell(\boldsymbol{x}) = \operatorname*{argmax}_i f_i(\boldsymbol{x}) \tag{5}$$

or, if the prototypes are not equally probable and p_i is the prior probability of the ith prototype, then

$$\ell(\boldsymbol{x}) = \operatorname*{argmax}_i p_i f_i(\boldsymbol{x}) \tag{6}$$

These probability densities are usually modeled as mixtures of multivariate Gaussians,

$$f_i(\boldsymbol{x}) = \sum_{j=1}^{n_i} \frac{p_{ij}}{(2\pi)^{m/2}\sqrt{|\Sigma_{ij}|}}\, e^{-\frac{1}{2}(x-\mu_{ij})'\Sigma_{ij}^{-1}(x-\mu_{ij})} \tag{7}$$

where n_i is the number of mixture components in the ith prototype, p_{ij} is the weight (prior probability) assigned to the jth component of the ith prototype, m is the dimensionality of the acoustic space, Σ_{ij} is the covariance matrix of the jth Gaussian mixture component of the ith prototype (an $m \times m$ symmetric positive definite matrix), and μ_{ij} is the center of that component (a vector of m elements). To reduce the computational load and the size of the required training corpus, the covariance matrixes Σ_{ij} are usually assumed to be diagonal. This assumption is reasonably accurate in cepstrum space, but not in the power spectrum space, which is the main reason for using cepstra as acoustic vectors. Other transformations of the power spectrum space, specifically designed to reduce errors due to suppression of the off-diagonal terms in Σ_{ij}, are also sometimes used.

The values of p, μ, and Σ in Eq. (7) must be estimated on the basis of a representative sample of speech—the training corpus. Methods for doing this are described in a later subsection under "Training of an automatic speech recognizer."

Allophones and Allophone Segments. The size of the label alphabet is typically in the thousands. English has only about 40 phonemes, or possibly 60 if vowels with primary stress are considered phonemes separate from unstressed ones, but the labels are associated with allophones rather than phonemes.

The number of allophones is somewhat arbitrary, depending on sensitivity to small pronunciation differences. In a large-vocabulary ASR the number may well exceed 1000. Allophone rules in these systems are usually implemented as decision trees. Given an hypothesized phoneme string, the tree decides which allophone to use at a given position in the string by asking questions about several phonemes preceding and following the target. In this way, the hypothesized phoneme string is translated into an allophone string by deterministic rules. The advantage of decision trees as opposed to simple table lookup of contexts is that the trees can handle contexts never seen in the training data. The decision tree is constructed by an automatic optimization scheme described in a later subsection under "Training of an automatic speech recognizer." The label alphabet usually contains about three labels for each allophone—the decoder expects each allophone to consist of three segments: beginning, middle, and end. This segmentation is necessary because the beginning is most strongly influenced by preceding phonemes and the ending by succeeding ones, and also because some phonemes are inherently dynamic. The vowel in "eight," for example, is usually a diphthong, and the consonant "t" in "two" consists of a silence followed by a burst of noise followed by an aspiration.

Discrete Versus Continuous Labeling. The acoustic processor can produce either one label per frame, or a list of the most likely labels, or a list of labels together with the likelihood of each. Whatever data it produces serve as input to the next step—decoding. The decoder sees neither the original PCM signal nor the acoustic vectors, only the labels and possibly the likelihoods of the labels as estimated by the acoustic processor. If the acoustic processor passes at least some of the likelihoods $f(x)$ to the decoder, then the latter is known as a *continuous-parameter decoder;* otherwise it is a *discrete-label decoder.*

Decoding. The final step, 4, determines the most probable message, given the sequence of labels.

Rigid Templates. A simple scheme for decoding a message might look for matches between the label stream and a set of rigid templates—fixed sequences of labels to be matched exactly. This, in effect, would be a table lookup scheme. The template for the word "one" might, for example, be the sequence `W W W AH AH AH N N N`. At a frame rate of 100 s^{-1}, this template would require that the word should last exactly 0.09 s, and that each of the three sounds in that word should last exactly 0.03 s. This scheme fails if the user speaks faster or slower. It can also fail because the actual labels generated by the acoustic processor are not always in strict one-to-one correspondence with phonemes. A more typical real label sequence might look something like `W W UW AH AH EH AH N M N M`. The chief advantage of rigid templates was computational simplicity, and the cost of computing has dropped enough to make this consideration now irrelevant. Rigid templates are not used in practical ASRs today.

Linear Time Warping. To cope with varying durations, a simple remedy would be to stretch or compress the template linearly. Thus, if the length of the template is 9 frames, but the spoken word is 12 frames long, then each third template frame could be repeated, so that the template would turn into `W W W W AH AH AH AH N N N N`. This is also computationally and conceptually simple, but is still a crude approximation to reality, because not all sounds change their durations equally when a talker speeds up or slows down.

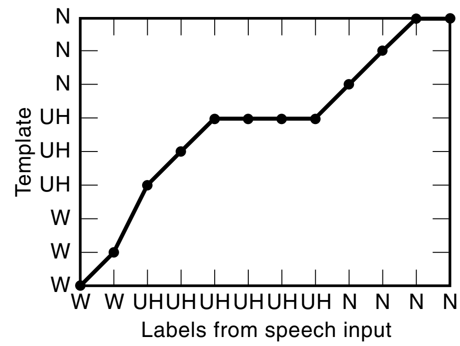

Figure 8. "Dynamic time warping" is necessary when matching actual speech to idealized templates.

Dynamic Time Warping. A technique more powerful than linear time adjustment is dynamic time warping (DTW). Although seldom used today, this technique introduces the important concept of a finite-state machine (FSM) for modeling a speech source. Here prototype frames are skipped or repeated as required, within prescribed limits, to match the observed label string, as illustrated in Fig. 8. The rules for permitted skips and repetitions can be formulated in terms of an FSM such as is shown in Fig. 9. Each frame of the prototype is considered a state. Transitions are allowed from each state to its successor. For some states, self-loops back to the same state are also allowed, permitting the state to be repeated and the sound to be lengthened. From some states, transitions skipping a state are allowed, making it possible for the sound to be shortened.

Dynamic Time Warping with Penalties. The additional flexibility that DTW introduces allows more of the variations and distortions of real speech to be matched, but also increases the danger that a template might match a wrong word, perhaps by skipping over some sounds entirely. To counteract this danger, a refinement of the DTW model introduces a penalty cost C for each transition. For example, each transition to the next state might have $C = 0$, but a transition skipping a state could have $C = 1$. In this way the template is able to match words that have distorted duration patterns, but at a cost that increases with the magnitude of the distortion. This means that even if a template is able to match an incorrect word, the correct template for that word will match better (with a lower penalty), and the decoder will then choose the correct word. Figure 10 shows an FSM with penalty costs attached to transitions. The decoder now has the additional burden of finding the match that has the lowest penalty, but it can do this efficiently by means of dynamic programming (see DYNAMIC PROGRAMMING). Even a DTW with penalties,

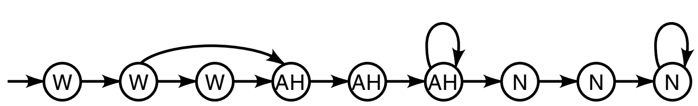

Figure 9. This finite-state machine is one example of a model capable of dynamic time warping.

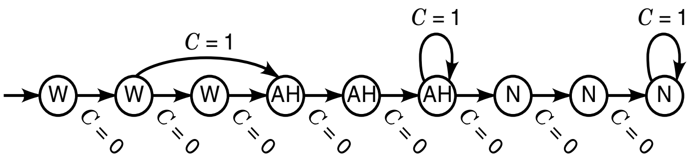

Figure 10. A finite-state machine with penalties exhibits a preference for the more probable warping patterns.

however, is still too limited for high-performance ASR use, because it can handle only changes in timing, not changes in labeling. If, for example, the acoustic processor generates the label `UW` (the final sound in "you") from one of the frames that are labeled `W` in the template, then the FSM of Fig. 10 fails to find any match at all.

To permit greater flexibility, each state might be permitted to match several different labels. Again, penalties could be levied against the less probable labels. With such an arrangement, however, the number of possible combinations of penalties becomes very large, and their effects difficult to foresee. For these reasons, parameters such as transition penalties in a large-vocabulary ASR must be adjusted by automated methods. Ways for doing this are discussed in a later subsection under "Training of an automatic speech recognizer," but the task is greatly facilitated if the penalties can be interpreted as logarithms of probabilities. The model then becomes a Markov source.

Hidden Markov Model. An FSM with penalties, such as the one shown in Fig. 10, is closely related to a Markov source. The latter is an FSM in which each transition has a probability attached to it, such that the probabilities of all transitions out of a given state are normalized to unity, and similarly each output alternative has a probability. The Markovian assumption states that these probabilities are independent of past history, i.e., independent of the path by which the current state was reached and independent of outputs of earlier frames. Let the transition probability from state q to state s be $p_t(s|q)$, normalized such that

$$\sum_s p_t(s|q) = 1 \tag{8}$$

Then the probability of a path S starting at some state S_0, going through state S_t at time t, and ending at S_T at time T is

$$p(S) = p(S_0) \prod_{t-1}^{T} p_t(S_t|S_{t-1}) \tag{9}$$

where $p(S_0)$ is the probability that the system starts in state S_0. The logarithm of this probability is

$$\log p(S) = \log p(S_0) + \sum_{t=1}^{T} \log p_t(S_t|S_{t-1}) \tag{10}$$

For an FSM with transition penalties, going through the same path, the total penalty is

$$C_{\text{tot}} = \sum_{t=1}^{T} C(S_t, S_{t-1}) \tag{11}$$

where $C(S_t, S_{t-1})$ is the penalty for the transition from S_{t-1} to S_t. Comparing the right-hand sides of Eqs. (10) and (11), it is evident that the penalty plays a role analogous to the logarithm of the transition probability.

Treating the label stream as if generated by a Markov source, that is, giving the penalties a probabilistic interpretation, has significant advantages for training the ASR, as discussed in later subsections. For this reason, current ASRs almost invariably take the Markov source approach to speech modeling. Because only the output of the Markov source, i.e. the label stream, is observable, and the states themselves are hidden from direct observation, the model is called a hidden Markov model (HMM).

The diagram in Fig. 10 shows a label associated with each state, but in an alternative, more widely used version of the HMM outputs are associated with transitions rather than states. Figure 11 illustrates this scheme. Here, each phoneme is shown as a sequence of three states, to accommodate differences in the sound between the beginning, middle, and end of a phoneme. All transitions actually have transition probabilities, but to reduce clutter, only a few are shown in the diagram. Similarly, only one output is shown for each arc, but there could be several, each with its own output probability value.

Alternatively, in a continuous-parameter decoder, the acoustic vector $\boldsymbol{x}$ is considered the output. Each arc then has an output probability density in the acoustic space. When such an arc is labeled `W1`, for example, the `W1` is the name of a prototype, corresponding to a probability density $f_{\texttt{W1}}(\boldsymbol{x})$ in the acoustic space. Although the decoder does not see the acoustic vector $\boldsymbol{x}$, it receives the value of $f_{\texttt{W1}}(\boldsymbol{x})$ from the labeler provided the value is sufficiently high, i.e. provided the `W1` prototype is sufficiently close to the top of the list. The labeler calculates the probability density according to a formula such as Eq. (7).

Viterbi Decoder. If the decoder were to hypothesize a path S through the state space such that at time t the state is $s = S_t$, for $t = 0, 1, 2, \ldots, T$, and if $\boldsymbol{X}$ is the acoustic vector sequence such that at time t the acoustic vector is $\boldsymbol{x} = \boldsymbol{X}_t$, then the prior probability of that state-space path, before looking at the acoustic signal, would be given by Eq. (9). Given that path, the probability of the observed sequence $\boldsymbol{X}$ of acoustic vectors is

$$p(\boldsymbol{X}|S) = \prod_{t=1}^{T} p_o(\boldsymbol{X}_t|S_t, S_{t-1}) \tag{12}$$

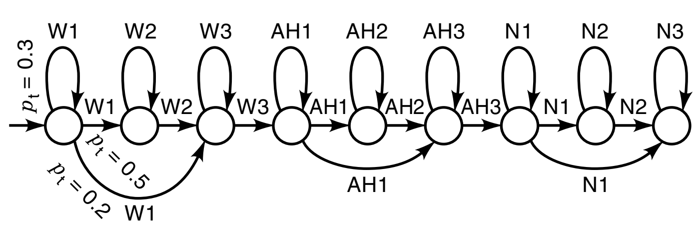

Figure 11. A "Hidden Markov Model" is a special kind of finite-state machine with penalties, one that is particularly suitable for automatic optimization. This figure shows a possible model for the word "one."

where $p_o(\boldsymbol{x}|S_t, S_{t-1})$ is the probability density $f(\boldsymbol{x})$ corresponding to the transition from S_{t-1} to state S_t.

The posterior probability of the hypothesized path S, then, according to Bayes's formula, is

$$p(S|\boldsymbol{X}) = \frac{p(S)p(\boldsymbol{X}|S)}{p(\boldsymbol{X})} \tag{13}$$

The most probable path then is

$$\hat{S} = \underset{S}{\operatorname{argmax}}\ p(S)p(\boldsymbol{X}|S) \tag{14}$$

where the denominator $p(\boldsymbol{X})$ in Eq. (13) has been ignored because it does not depend on S. Substituting from Eqs. (9) and (12), the above becomes

$$\hat{S} = \underset{S}{\operatorname{argmax}}\ p(S_0)\prod_{t=1}^{T} p_{\mathrm{t}}(S_t|S_{t-1})p_{\mathrm{o}}(\boldsymbol{X}_t|S_t, S_{t-1}) \tag{15}$$

This is the most probable path through the state space after a given acoustic-vector sequence or label sequence has been observed. The path specified by Eq. (15) can be found without exhaustive search by means of the Viterbi algorithm (20,21).

Beam Search. Although the Viterbi algorithm is an efficient method for finding the most probable path through the state space for a given set of observations, it can still require a prohibitive amount of computation if the number of states is large. A technique known as *beam search* is faster, although it is not guaranteed to always find the most probable path. At any given time point in the computation, beam search ignores those paths that are less probable than the best by some predetermined margin, and it only extends the most probable paths. Confining the computation thus to a narrow "beam" can greatly reduce the amount of work without seriously degrading accuracy.

Summing over Alternative Alignments. A hypothesized path S represents one possible time alignment of one hypothesized message—a word or word sequence—and $\hat{S}$ is the most probable time alignment for that message. To decide which message was spoken, the decoder needs to compare the posterior probabilities of all possible messages without, however, necessarily deciding on the alignment. For decoding a message, the alignments are nuisance parameters. The decoder is only interested in the marginal probabilities, after the nuisance parameters have been summed or integrated out. Let $\mathscr{S}_w$ be the set of all paths S corresponding to a particular message w. The marginal posterior probability that $S \in \mathscr{S}_w$, ignoring the alignment, is obtained by summing the posterior probabilities of all alignment hypotheses in that set:

$$\begin{aligned} p(S \in \mathscr{S}_w|\boldsymbol{X}) &= \sum_{S\in\mathscr{S}_w} p(S|\boldsymbol{X}) \\ &= \sum_{S\in\mathscr{S}_w} \frac{p(S)p(\boldsymbol{X}|S)}{p(\boldsymbol{X})} \end{aligned} \tag{16}$$

The most probable message $\hat{w}$ is then

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{S\in\mathscr{S}_w} p(S)p(\boldsymbol{X}|S) \tag{17}$$

where the denominator $p(\boldsymbol{X})$ that appeared in Eq. (16) is omitted, being independent of S and w. Substituting again from Eqs. (9) and (12), we get

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{S\in\mathscr{S}_w} p(S_0)\prod_{t=1}^{T} p_{\mathrm{t}}(S_t|S_{t-1})p_{\mathrm{o}}(X_t|S_t, S_{t-1}) \tag{18}$$

As written above, the right-hand side calls for summation over all possible alignments, i.e. all possible paths S for each message w. The number of terms in such a sum grows exponentially with the length T of the message. The sum can be factored, however, to yield a recursive computation

$$\sum_{S\in\mathscr{S}_w} p(S_0)\prod_{t=1}^{T} p_{\mathrm{t}}(S_t|S_{t-1})p_{\mathrm{o}}(X_t|S_t, S_{t-1}) = \sum_{s} \alpha_T(s) \tag{19}$$

where $\alpha_T(s)$ is the sum over all paths ending at $S_T = s$. The following recursion then holds:

$$\alpha_t(s) = \sum_{s'} \alpha_{t-1}(s')p_{\mathrm{t}}(s|s')p_{\mathrm{o}}(\boldsymbol{X}_t|s', s) \tag{20}$$

and

$$\alpha_0(s) = p(S_0 = s) \tag{21}$$

The quantity $\alpha_t(s)$ represents the joint probability of the acoustic vector sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$ and the path going through state s at time t.

In Eqs. (19), (20), and (21) the states s are to be restricted to those appearing in the set of paths allowed for a particular message w.

The recursion in Eq. (20) makes it feasible to calculate the sum in Eq. (17). This is the method used in ASRs when sufficient computing power is available. When not, then the sum is replaced by the probability of the most probable path $\hat{S}$ defined by Eq. (15).

Composite Hidden Markov Model. In a large-vocabulary ASR, any word can follow any other word. We could then imagine a composite HMM for the entire language where a transition would be permitted from the last state of any word to the first state of any word. With a vocabulary size of the order of tens of thousands of words, such an HMM is far too large to represent in a drawing, but Fig. 12 illustrates the idea visually for a vocabulary of only three words. Even such a large model, however, has a serious limitation: it allows the probability of a given word to depend on the previous word, but not on any earlier words, because the Markov model has no memory beyond the most recent state. A more complicated Markov model could be constructed in which the probability of the next word would depend on, say, two preceding words, but such a model would have many more states and transitions than the one illustrated.

The model in Fig. 12 leaves out another important detail: each phoneme, such as the `W` at the start of "one," is actually an allophone depending on the preceding and following phonemes. Each word may therefore need several alternative beginnings and endings depending on the adjacent words. Thus, although conceptually the entire operation of the decoder could be described by one large HMM, in practice such a model would be far too large to be precomputed and stored.

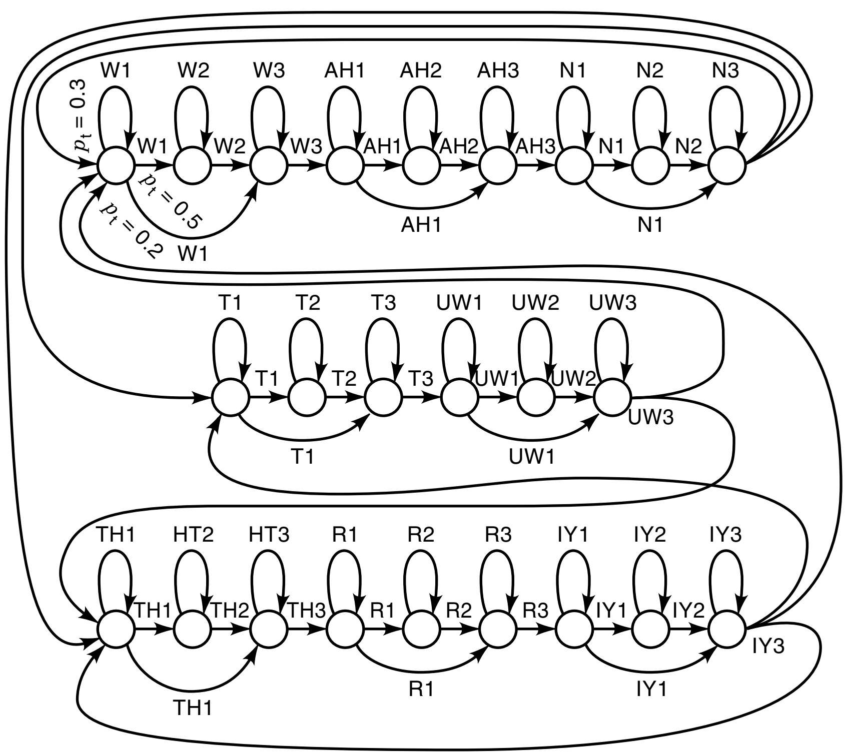

Figure 12. A "Hidden Markov Model" for only a three-word vocabulary already requires a large number of states and transitions.

In small-vocabulary applications, the composite HMM may be small enough to permit probability values for all the states to be computed, so that decoding can then be done by the Viterbi algorithm. For larger models, however, sequential decoding is used because it requires probabilities for only a small fraction of all the states to be computed.

Sequential Decoding. In a large-vocabulary ASR, the HMM is not prestored, but the decoder constructs portions of it dynamically as needed for those hypotheses that the decoder is actually testing. At any time point in a hypothesis where a new word might start, the decoder first uses an abbreviated acoustic match computation ("fast match") to eliminate most vocabulary words quickly from consideration. For each of the remaining words, the decoder constructs a branch of the HMM, consulting the language model (described in a later subsection) to determine the probability of the transition into the first state of the word. It then looks up the pronunication of the word in the *baseform dictionary*. An excerpt from a typical baseform dictionary might look like Table 1, using the vernacular phonetic symbols common in ASR technology.

Table 1.

Word Spelling	Baseform							
⋮								
AIRCRAFT'S	EH	AXR	K	R	AE	F	TS	
ALIGNS	AX	L	AY	N	Z			
ALLEYS	AE	L	IY	Z				
ALLOCATOR	AE	L	AX	K	EY	DX	AXR	
ALLOCATORS	AE	L	AX	K	EY	DX	AXR	Z
⋮								

The entries are called baseforms because they are the basis from which the actual allophone string is derived. The latter may be different for the same word in different contexts. The decoder uses a binary decision tree to decide which allophone to substitute for each phoneme in the new word, taking into account the context in which the new phoneme now appears. This context may include phonemes in a preceding word. Having determined the allophones, the decoder then replaces each one with a small (e.g., three state) HMM specifying the detailed acoustic structure of that allophone.

After appending new branches to its hypothesis network, the decoder calculates the acoustic match for each branch, performing essentially the computation specified by Eq. (17), by means of the recursive algorithm of Eq. (20). This computation is the same as the forward pass of the Baum–Welch algorithm and is described in greater detail in the section on statistical training later in this article.

After performing the forward computation, the decoder selects the branches that have the best match scores, and extends them further in the same way, starting again with the fast match.

Whereas the structure of the HMM inside a word is determined by the baseform dictionary and the allophone models, the transition probabilities between words come from the language model.

Language Model. Traditional syntactic analysis relies on rules constituting a grammar, but for ASR use such a deterministic approach with manually written rules has enjoyed little success, yielding instead to probabilistic n-gram language models. Part of the power of the latter models comes from their ability to incorporate limited semantic as well as syntactic information. Furthermore, actual speech, especially informal conversation, does not follow rules of grammar rigorously. Probabilistic n-gram models are able to accommodate arbitrary deviations from strict rules while still giving higher weights to more commonly occurring constructions. These advantages apparently outweigh the inherent limitations of this simple concept.

Let the message w be a text consisting of a sequence of N words W_i (or syllables in languages such as Mandarin):

$$w = (W_1, W_2, \ldots, W_N) \tag{22}$$

If we assume that the first word of every message is a special "start-of-message" word (not actually written in the text) and the last word similarly is "end-of-message," then the following identity is true:

$$p(w) = \prod_{i=1}^{N} p(W_i|W_1, \ldots, W_{i-1}) \tag{23}$$

The n-gram language model approximates this relationship by assuming that the probability of the ith word depends not on all the preceding words, but only at most $n - 1$ preceding words:

$$p(w) = \prod_{i=1}^{N} p(W_i|W_{k_i}, \ldots W_{i-1}) \quad \text{where} \quad k_i = \max(1, 1+i-n) \tag{24}$$

The probabilities $p(W_i|W_{k_i}, \ldots W_{i-1})$ appearing in the right-hand side of Eq. (24) are not known exactly; hence estimated values must be substituted. These are estimated during language model training on the basis of a training corpus, which need not contain any acoustic data, only samples of text. Several tens of millions of words are needed to get good estimates of these probabilities. Even then, special treatment is needed for trigrams that have low counts or do not appear in the training corpus at all. These procedures are described in greater detail in the subsection on ASR training.

Typical values for n are 2 (bigram model) or 3 (trigram model).

Training of an Automatic Speech Recognizer

Statistical Training versus Human Knowledge. Designing an ASR is a two-stage procedure consisting of preliminary algorithm design followed by fine tuning through *statistical training:*

1. If $\mathscr{D}$ is the space of all possible speech recognition algorithms, then an ASR implements a subset of these—a parametric family $\mathscr{D}_0 \in \mathscr{D}$ of functions $D: \mathscr{A} \mapsto \mathscr{W}$, where $\mathscr{A}$ is the space of acoustic signals, $\mathscr{W}$ the space of messages, and $D \in \mathscr{D}_0$.
2. A training algorithm, after extracting evidence from the training corpus, selects one specific element $D \in \mathscr{D}_0$, i.e. one set of parameter values, from those that the ASR is capable of implementing. The training algorithm may start with some nonuniform prior probability distribution in the parameter space, thus biasing it hopefully toward the correct solution.

Increasing the size of the parametric family $\mathscr{D}_0$ gives the training algorithm more freedom to find the best recognizer D, potentially improving the accuracy, but also requiring a larger training corpus. Finite computing speeds always impose a practical limit on the size of the training corpus. If that corpus is too small for a given family $\mathscr{D}_0$, that is, if $\mathscr{D}_0$ has too many degrees of freedom for the available training corpus, then training may reduce the error rate on that corpus to a very low value or even zero, without achieving any improvement at all on new test data. This phenomenon is called *overtraining.* Historically, as computing speeds have risen, larger speech corpora have become accessible. ASR design has thus come to depend more on empirical speech data and less on prior human knowledge, and as a consequence, ASR accuracy has improved. Complete removal of the human ingredient from the design procedure, however, is not plausible.

Human knowledge is incorporated into the ASR mainly through the choice of the parametric family $\mathscr{D}_0$ and through the design of the training algorithm. If we could make $\mathscr{D}_0 = \mathscr{D}$, with a uniform probability distribution over $\mathscr{D}_0$, then the entire burden of ASR design would be shifted to the training algorithm, and no human knowledge would be built in at all. In that case, however, the training algorithm would have no prior knowledge about the nature of the acoustic space $\mathscr{A}$ and would be unable to judge the similarity between a new waveform and any that it had already seen. It would then need to see in the training sample all waveforms that it might ever encounter. This is clearly impossible; hence prior human knowledge will always remain an important ingredient of ASR design.

As computing power increases, however, the nature of the human knowledge required becomes more and more abstract. Instead of hard-coding numeric decision boundaries, we now introduce prior information through algorithm structure, through design of statistical models, and through the selection of training corpora. Nevertheless, humans still possess kinds of information that has not been fully utilized in ASR systems. Among these are semantic knowledge, rules of grammar, and knowledge about the physical structure of the vocal tract. Active research is continuing towards incorporating more of this knowledge into the automatic algorithms.

Training and Adaptation. A third phase, *adaptation,* can be added to the two discussed above. ASRs perform better if they are adjusted for a specific user rather than for the general population. One way to adjust the ASR would be to have an entire training corpus spoken by one user, but this is generally neither feasible nor necessary. If an ASR is trained on a number of speakers of the same language and broad dialect (e.g. US English or UK English), then this speaker-independent recognizer can be adapted to a specific user with a relatively small amount of additional speech data. Such adaptation can be modeled as Bayesian estimation of the recognizer parameters. Let θ be the parameter vector, consisting for example of transition probability values, output probability values, etc. Then let $p_0(\theta)$ be the initial, prior distribution on θ, built into the recognizer before any training is done. This is frequently a uniform or maximum-entropy (maximally uninformative) distribution. Now if a particular multispeaker training corpus consisting of acoustic signals A_1 and corresponding messages (e.g. word sequences) W_1, is observed, then a new distribution on $p_1(\theta)$ is given by Bayes's rule as

$$\begin{aligned} p_1(\theta) &= p(\theta|A_1, W_1) \\ &= \frac{p_0(\theta)p(A_1, W_1|\theta)}{p(A_1, W_1)} \end{aligned} \tag{25}$$

This new distribution $p_1(\theta)$ is the result of speaker-independent training of the recognizer. It serves as the prior for speaker-specific adaptation. The adapted distribution $p_2(\theta)$ is then obtained again by Bayes's rule:

$$p_2(\theta) = \frac{p_1(\theta)p(A_2, W_2|\theta)}{p(A_2, W_2)} \tag{26}$$

where A_2 and W_2 constitute the adaptation corpus, spoken by one user. Because $p_1(\theta)$ has a lower entropy, i.e., is more informative, than $p_0(\theta)$, the adaptation corpus (A_2, W_2) does not need to be as large as the speaker-independent training corpus (A_1, W_1).

Statistical Model. As briefly stated in the beginning of this article, the usual approach to the training task is to first assume that the joint probability of signals and messages, $p(a, w)$, belongs to some parametric family $p(a, w|\theta)$. The selection of this family is the step that injects most of the prior knowledge into the ASR system. Additional prior information can be introduced by specifying a prior probability distribution for the parameter vector θ. During the initial design of the system, a maximum entropy prior is usually assumed, at least implicitly. Such a prior provides the least amount of information and hence requires the largest training corpus. During *adaptation* to a new user, however, a much more informative prior is derived from the initial training, and the needed number of adaptation data from the new user, therefore, is significantly smaller.

Separation into Language and Acoustic Models. The form of the parametric family $p(a, w|\theta)$ that has been found particularly useful in ASR technology splits θ into two subvectors θ_w and θ_a by factoring $p(a, w|\theta)$ as follows:

$$p(a, w|\theta_w, \theta_a) = p(w|\theta_w)p(a|w, \theta_a) \tag{27}$$

This formulation partitions the optimization problem effectively into two more or less independent parts: language model optimization and acoustic model optimization. The language model $p(w|\theta_w)$ deals only with word sequences, not their pronunciations. The acoustic model $p(a|w, \theta_a)$ deals with pronunciations, including acoustic prototypes and allophone HMMs. Instead of trying to minimize the overall error rate directly, this approach gives each submodel its own objective function and its own part of the parameter vector θ, and then estimates these parts separately by standard statistical estimation methods. One of the advantages of partitioning the problem in this way is that the language model can now be trained on text only, without requiring corresponding acoustic signals. Acoustic models, on the other hand, can be trained on speech samples for which the text is already known, so that a language model is not required for acoustic training. In this way much larger text corpora become available for language model training if needed, or, for small-vocabulary interactive dialog applications where the language model is simple finite-state grammar (FSG), the model can be designed manually without statistical training. In either case the language model can now be created without reference to spoken text and without the need for trained acoustic models. Meanwhile, acoustic training can proceed in parallel without the need for a trained language model.

Statistical Language Model

Objective Function for Language Model Training. A suitable objective function for the language model is the entropy of the language, given the model, Consider a vocabulary $\mathfrak{W}$ of words $\mathfrak{w} \in \mathfrak{W}$. The language model, having seen a string of words (W_1, W_2, . . ., W_{t-1}), predicts a probability distribution for the next word, $p_t(\mathfrak{w}|\theta_w)$. When the actual text word W_t is revealed, the amount of information conveyed by that relevation is according to Shannon's theory, $-\log_2 p_t(W_t|\theta_w)$ bits. The entropy of the language, conditioned on the model, is the expected value of this quantity:

$$H(\theta_\mathrm{w}) \equiv \mathsf{E}[-\log_2\, p_t(W_t|\theta_w)] \tag{28}$$

This quantity is also known as the cross entropy between the true probabilities and those predicted by the model. It is a minimum when the predictions agree with the true probabilities. The value of this entropy is a measure of the performance of the language model. It is customary to convert this to the *perplexity* Q by exponentation:

$$Q = 2^{H(\theta_w)} \tag{29}$$

The perplexity is frequently used as an objective function for optimizing a language model. Because Q is a monotonically increasing function of H in Eq. (29), minimizing either quantity leads to the same result. The parameter vector θ_w of the language model, therefore, is to be adjusted to minimize the entropy and the perplexity. Depending on the application, this may require statistical training on hundreds of millions of words of text, or it may require no training data at all, as in the case of manually designed finite-state grammars.

An n-gram Language Model. An n-gram language model predicts the probabilities of the next word on the basis of the n most recent preceding words. Typical values for n are 2, corresponding to a bigram model, and 3, for a trigram model.

In a trigram language model, the parameters of the model are the trigram probabilities $p(W_i, W_{i-2}, W_{i-1})$. The maximum likelihood estimate of these parameters makes them equal to the empirical probabilities on the training corpus. If $n_\mathrm{R}(W_i, W_{i-2}, W_{i-1})$ is the number of times the trigram (W_i, W_{i-2}, W_{i-1}) appears in the training corpus, and N_R is the total number of trigrams in the training corpus, then the maximum likelihood estimate is

$$p(W_i, W_{i-2}, W_{i-1}) = \frac{n_\mathrm{R}(W_i, W_{i-2}, W_{i-1})}{N_\mathrm{R}} \tag{30}$$

These values of the trigram probabilities minimize the entropy and perplexity of the language on the training corpus, but they result in serious overtraining: Eq. (30) gives zero probability to trigrams that do not occur in the training corpus. If such a trigram subsequently appears in test data, the decoder is guaranteed to make a mistake, since it assumes that these trigrams can never occur.

To correct the overtraining, it is necessary to incorporate additional prior knowledge into the training algorithm. In Bayes's formula, prior information is explicitly contained in a prior probability distribution, but in ASR design prior knowledge is often qualitative, not expressible as precise probability values. Language model design illustrates this: We do not know the probabilities of the unseen trigrams, but at least we know that all trigrams are possible, none has a zero probability. Beyond this, we know from previous experiments that a bigram language model is not as good as a trigram model, but is better than nothing at all. Even a unigram model is better than no model. This suggests that when the trigram model is clueless, because its empirical count is zero, a bigram or simpler model might still provide useful information. Even when the trigram count is nonzero but low, it seems reasonable to assume that the bigram count may give a better estimate or at least some additional useful information.

To translate these purely qualitative arguments into an algorithm, we need quantitative reasoning but must not make any strong new assumptions. Let P be the probability of some trigram in the population, and N_R the size of the training corpus, so that the expected count for this trigram in the training corpus is $\overline{n} = N_R P$. Then the observed count n obeys, to a first approximation, a Poisson distribution

$$P(n) = \frac{\overline{n}^n e^{-\overline{n}}}{n!} \tag{31}$$

and its standard deviation is $\sqrt{\overline{n}}$. For large $\overline{n}$ the standard deviation of log n is then approximately $1/\sqrt{\overline{n}}$. As the expected count $\overline{n}$ decreases, the standard deviation (the sampling error in n) increases. Bigram counts are typically larger than trigram counts, but bigram probabilities are poorer predictors of the next word than trigram. If the trigram count is very small, then its sampling error may be so large that the bigram probability is a better predictor. In such situations, a linear combination of the two may be a still better predictor then either alone. Furthermore, if even the bigram count is low, then unigram and zerogram probabilities can be invoked to improve the estimate. Thus we could write

$$P = \lambda_3 P_3 + \lambda_2 P_2 + \lambda_1 P_1 + \lambda_0 P_0 \tag{32}$$

where P_i is the empirical i-gram probability and λ_i is its weight. If we knew the relative magnitudes of the systematic and random erros in each predictor, we could calculate λ_i values to minimize the error in P. Because we do not know these magnitudes and do not want to make arbitrary assumptions about them, we must estimate the weights empirically by taking the P_i values from one part of the training corpus and then optimizing the λ_i on another part of the corpus (held-out data). The optimization can be done over repeated trials using different partitionings of the training corpus, so that all sample points are used in held-out data equally often. The resulting values of λ depend, in principle, on all the counts from unigrams through trigrams, though some simplified scheme is used in practice.

Another way to deal with low counts is to categorize words into syntactic categories such as verbs, nouns, etc., or semantic ones such as numbers, names, colors, etc., and to combine counts for these with word trigram counts in the manner of Eq. (32). Automatic methods exist for defining such categories.

Statistical Acoustic Model

Objective Function for Acoustic Model Training. The goal of acoustic processing is to generate output that contains as much information as possible about the message. Technically, this is equivalent to maximizing the mutual information between the message and the output of the acoustic processor. Because mutual information is symmetric, we can alternatively maximize the information that the message contains about the acoustic processor output.

For a given message, w, the acoustic model calculates the probability $p(a|w, \theta_a)$ that appears in Eq. (27). This function is defined for all possible acoustic signals a, but we are interested in the a that was actually associated with the message. If the acoustic model could predict a exactly, given w, then we would have $p(a|w, \theta_a) = 1$ and $-\log p(a|w, \theta_a) = 0$. The conditional entropy of the acoustic signal, given the message, would then be zero. The higher the entropy, the less information the acoustic model provides. The objective function to be minimized by acoustic model training is therefore

$$F_a = \mathsf{E}[-\log p(a|w, \theta_a)] \tag{33}$$

Baum–Welch Reestimation Algorithm. The goal of the model is to predict the acoustic signal as accurately as possible, given the message, because this also maximizes the information that the acoustic processor is able to extract from the signal about the message.

During training, the text and the acoustic signal are both available, but the exact time alignment of the speech sounds is not known. The training program, effectively, constructs an HMM in the same way that the decoder dynamically constructs the HMM for each hypothesis. During training, however, the text is known, so that the resulting HMM is much simpler. All paths through this model correspond to the same message, though to different time alignments, but the paths will have different probabilities. The training algorithm calculates the sum of the probabilities over all paths, using a recursion formula similar to Eq. (20). A two-pass modification of this procedure, including a similar computation in the reverse direction, permits a separate sum to be obtained for paths going through each transition. This permits the training algorithm to reestimate the probability of each transition, and also to reestimate the output probability distribution for each transition. In this way, the training algorithm updates the parameters of the HMM, which constitute the vector θ_a. It then repeats the computations starting from the updated model. Four iterations typically suffice to converge to a reasonably accurate estimate of θ_a. This procedure is known as the Baum–Welch reestimation algorithm, or the forward–backward algorithm. It is a special case of the *estimate–maximize* (EM) algorithm.

To illustrate the computations involved in the Baum–Welch algorithm, we consider the simple example shown in Fig. 13, consisting of only three time frames and only three

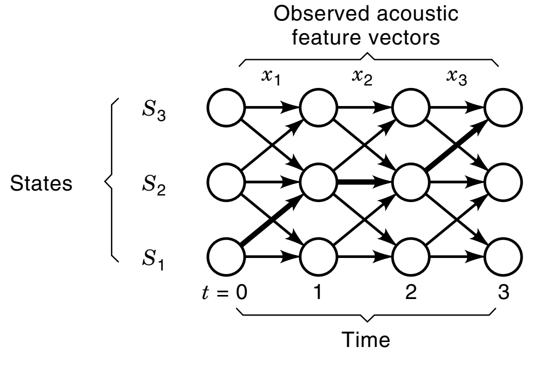

Figure 13. A trellis, in which each circle represents one possible state of the speech source at one instant of time, is the basis of a fast computational algorithm for speech decoding.

HMM states, with transitions between states 1 and 3 forbidden and all others permitted.

At time $t = 0$ there have been no observations. The probabilities of the states at that time are the initial prior probabilities: $p(s_1)$, $p(s_2)$, and $p(s_3)$. The probability of any one specific path is given by Eq. (9). For example, let S be the path (s_1, s_2, s_2, s_3) shown in heavy arrows. Its probability is:

$$\begin{aligned} p(S) &= p(S_0)\prod_{t=1}^{T} p_{\mathrm{t}}(S_t|S_{t-1}) \\ &= p(s_1)p_{\mathrm{t}}(s_2|s_1)p_{\mathrm{t}}(s_2|s_2)p_{\mathrm{t}}(s_3|s_2) \end{aligned} \tag{34}$$

Given this path, the conditional probability of observing the acoustic feature vector sequence $\boldsymbol{X} = (x_1, x_2, x_3)$ is, according to Eq. (12),

$$\begin{aligned} p(X|S) &= \prod_{t-1}^{T} p_{\mathrm{o}}(\boldsymbol{X}_t|S_t, S_{t-1}) \\ &= p_{\mathrm{o}}(x_1|s_1, s_2)p_{\mathrm{o}}(x_2|s_2, s_2)p_{\mathrm{o}}(x_3|s_2, s_3) \end{aligned} \tag{35}$$

The joint probability of the path and the output is

$$\begin{aligned} p(X, S) &= p(\boldsymbol{X}|S)p(S) \\ &= p(s_1)[p_{\mathrm{t}}(s_2|s_1)p_{\mathrm{o}}(x_1|s_1, s_2)] \\ &\quad \times [p_{\mathrm{t}}(s_2|s_2)p_{\mathrm{o}}(x_2|s_2, s_2)][p_{\mathrm{t}}(s_3|s_2)p_{\mathrm{o}}(x_3|s_2, s_3)] \end{aligned} \tag{36}$$

To train the model, we want to adjust the parameters of the model, that is, the output probabilities p_{o} and the transition probabilities p_{t}, so as to maximize the probability of the observed output, $p(X)$. This marginal probability can be computed by summing the joint probability $p(X, S)$ over all possible paths:

$$p(X) = \sum_S p(X, S) \tag{37}$$

The number of terms in this sum is the number of possible paths, which is very large in any realistic model. The sum, however, can be factored to obtain a fast algorithm for its computation.

To factor the sum, note first that each term in it is the product of factors such as those in square brackets in Eq. (36). Each such factor depends only on the feature vector at one time frame and the corresponding state transition. Now consider a set of paths that differ only at time $t = 0$, for example all paths that follow the heavy arrows in Fig. 13 from time $t = 1$ onward, but differ at time $t = 0$. Then the last two square-bracketed factors in Eq. (36) are the same for all three of these paths, and can be factored out. Therefore, the sum of the probabilities $p(\boldsymbol{X}, S)$ over these three paths can be written

$$\begin{aligned} \sum p(\boldsymbol{X}, S) = \{&p(s_1)[p_{\mathrm{t}}(s_2|s_1)p_{\mathrm{o}}(x_1|s_1, s_2)] \\ &+ p(s_2)[p_{\mathrm{t}}(s_2|s_2)p_{\mathrm{o}}(x_1|s_2, s_2)] \\ &+ p(s_3)[p_{\mathrm{t}}(s_2|s_3)p_{\mathrm{o}}(x_1|s_1, s_2)]\} \\ &\times [p_{\mathrm{t}}(s_2|s_2)p_{\mathrm{o}}(x_2|s_2, s_2)][p_{\mathrm{t}}(s_3|s_2)p_{\mathrm{o}}(x_3|s_2, s_3)] \end{aligned} \tag{38}$$

Note that the expression in the braces depends only on the part of the path up to time $t = 1$ and the other factors depend only on the part of the path from $t = 1$ onwards.

Expressions such as that enclosed in curly braces in Eq. (38) play an important role in the fast computational algorithm, and are commonly designated $\alpha_t(s)$. Each represents a sum over all paths that pass through state s at time t, but the sum is only over products of those terms that depend on the path up to that time. Thus, the factor appearing in Eq. (38) is designated $\alpha_1(2)$ because it represents paths going through state 2 at time 1. At time $t = 0$, the alphas are just the initial probabilities, so that e.g. $\alpha_0(1) \equiv p(s_1)$. Hence, the expression in the curly braces in Eq. (38) can now be written as

$$\begin{aligned} \alpha_1(2) = \{&\alpha_0(1)[p_{\mathrm{t}}(s_2|s_1)p_{\mathrm{o}}(x_1|s_1, s_2)] \\ &+ \alpha_0(2)[p_{\mathrm{t}}(s_2|s_2)p_{\mathrm{o}}(x_1|s_2, s_2)] \\ &+ \alpha_0(3)[p_{\mathrm{t}}(s_2|s_3)p_{\mathrm{o}}(x_1|s_1, s_2)]\} \end{aligned} \tag{39}$$

The above example illustrates the basic concept in the fast algorithm for computing the sum in Eq. (37). In general, the alphas are computed according to Eq. (20). The sum in Eq. (37) is then simply the sum over the alphas for the last time sample:

$$\begin{aligned} p(X) &= \sum_S p(\boldsymbol{X}, S) \\ &= \sum_{S_T} \alpha_T(S_T) \end{aligned} \tag{40}$$

where T is the value of t at the last frame, and S_T ranges over the possible states at that frame.

The object of training is to adjust the output probabilities p_{o} and the transition probabilities p_{t} so as to maximize the probability of the observed output, $p(\boldsymbol{X})$. Equation (40) permits this objective value to be calculated, but does not tell us how to adjust p_{o} and p_{t}. If we knew the actual path S, then we could estimate transition probabilities by counting the actual transitions in the path, and we could similarly estimate the output probabilities by collecting statistics on the observed output for each possible transition. Because we do not know the actual path, we consider all possible paths, but weight them according to their a posteriori probability, given the observed feature vector sequence.

In Fig. 13, for example, we know that the feature vector x_2 was observed at time $t = 2$, but we do not know in what state the system was at times 1 and 2; thus we do not know what transition was associated with that feature vector. We consider, therefore, all possible transitions at this time step, among them the one shown by the heavy arrow, from state 2 to state 2. In this example, there are evidently nine possible paths that are in state 2 at both times 1 and 2; in a realistic model the number of such paths would be very much larger. The path consisting of the three heavy arrows is one such path. The a posteriori probability of such a path is

$$p(S|X) = \frac{p(\boldsymbol{X}, S)}{p(\boldsymbol{X})} \tag{41}$$

A fast algorithm similar to the alpha recursion is available for summing over all paths that go through specified states at a specified pair of adjacent time values, but it involves a computation in both the forward and backward directions. The forward computation is done first, using Eq. (20), and all the values of $\alpha_t(s)$ are stored. The backward computation is then done according to an analogous formula:

$$\beta_t(s) = \sum_{s'} \beta_{t+1}(s') p_t(s'|s) p_o(X_t|s, s') \tag{42}$$

It can be shown that the sum of the joint probabilities $p(X, S)$ over all paths going through state S_{t-1} at time $t-1$ and through state S_t at time t is equal to

$$p(S_{t-1}, S_t, \boldsymbol{X}) = \alpha_{t-1}(S_{t-1})\beta_t(S_t) p_t(S_t|S_{t-1}) p_o(\boldsymbol{X}_t|S_{t-1}, S_t) \tag{43}$$

During the backward computation, each newly computed β value is combined with previously stored α values according to Eq. (43) to obtain $p(S_{t-1}, S_t, \boldsymbol{X})$. From these joint probabilities, the conditional probabilities $p(S_{t-1}, S_t|\boldsymbol{X})$ can be obtained by appropriate normalization:

$$p(S_{t-1}, S_t|\boldsymbol{X}) = \frac{p(S_{t-1}, S_t, \boldsymbol{X})}{\sum_{S'_{t-1}, S'_t} p(S'_{t-1}, S'_t, \boldsymbol{X})} \tag{44}$$

With the aid of these probabilities, new estimated values for p_t are then calculated as follows:

$$p_t(s|s') = \frac{1}{T} \sum_t p(S_{t-1} = s,\ S_t = s'|\boldsymbol{X}) \tag{45}$$

The statistics for the output probabilities are collected by distributing each observed feature vector x_t among the possible transitions in proportion to the probabilities of each transition at time t as given by Eq. (44). Thus, for example, if we are interested in the average value of $\mu_x(s_1, s_2)$ for the transition (s_1, s_2), we calculate

$$\mu_x(s_1, s_2) = \frac{\sum_{t=1}^{T} p(S_{t-1} = s_1,\ S_t - s_2|\boldsymbol{X}) x_t}{\sum_{t=1}^{T} p(S_{t-1} = s_1,\ S_t = s_2|\boldsymbol{X})} \tag{46}$$

In an analogous manner, new covariances for the feature vectors can be estimated, and from these means and covariances, new probability densities can be obtained. Using these new output probability densities and the new transition probabilities from Eq. (45), new alphas and betas can be computed. Four or five such iterations are typically needed to obtain reasonably good estimates of the transition probabilities and output probabilities.

Allophone Tree Optimization. The objective of the allophone tree is the same as the objective of the overall HMM, the function specified in Eq. (33). At each stage in tree construction, the design algorithm makes two decisions:

1. Which nodes to split.
2. What question to ask at that node.

If the number of samples at any node is too small, that node is automatically removed from the list of candidates for splitting. After that, the algorithm tries all questions in its repertory on all remaining leaves, and for each combination calculates the reduction in the objective function F_a defined by Eq. (33). For each leaf, it chooses the question that brings the greatest reduction, and if that reduction is sufficiently large, it then appends the question to the tree, splitting the node into two new leaves.

Figure 14 shows part of such an allophone decision tree for the beginning segment of the phoneme "r." There may be a separate tree for each of the typically three segments of a phoneme. Because this tree is for the beginning segment, most questions deal with the preceding phoneme. The first question asks whether the preceding phoneme is in the list of sonorants shown. If it is, the next question asks whether that sonorant is a low back vowel. If the answer again is yes, the tree then asks whether it is the specific phoneme ɑ, and if it is, the next question then asks about the phoneme following the "r"—whether it is one of the stops or affricates. On the other hand, if the answer to the first question is no, the tree then asks whether the preceding phoneme belongs to a specified list of consonants, and if the answer to this is also no then it asks the same question as at the start of the tree, but this time not about the preceding phoneme but about the one before that.

Only a small portion of the tree is shown; the entire tree in this example has more than 300 nodes. At the bottom of the tree each leaf represents one of the allophones for which this ASR has a separate acoustic density function. Each of these functions typically is a mixture of Gaussians as shown in Eq. (7).

EXPLORATORY WORK

Researchers are constantly exploring new ways to improve the performance of speech recognition systems. In this section, we briefly introduce some of the interesting approaches, citing references for further studies.

Artificial Neural Networks

We discussed the topic of phonetic probability estimation in the preceding section. An alternative strategy for estimating these probabilities is to employ the artificial neural network (ANN) technology (15,22). The term "neural" in ANN is derived from a tinuous similarity with the way neurons operate in the nervous system. The later operation is not fully understood. But we believe the basic building block of the ANN shown in Fig. 15 roughly duplicates its function.

We take an inner product of the inputs $x_1, x_2, \ldots, x_n$ with a set of weights $w_1, w_2, \ldots, w_n$, threshold it by θ, and apply it to a nonlinear element as given in

$$y = f\left(\sum_i w_i x_i - \theta\right) \tag{47}$$

The nonlinear function $f(x)$ is typically a sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{48}$$

Another nonlinear function used in many implementations is the softmax function defined by

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{49}$$

where the summation in the denominator is over all units in the output layer.

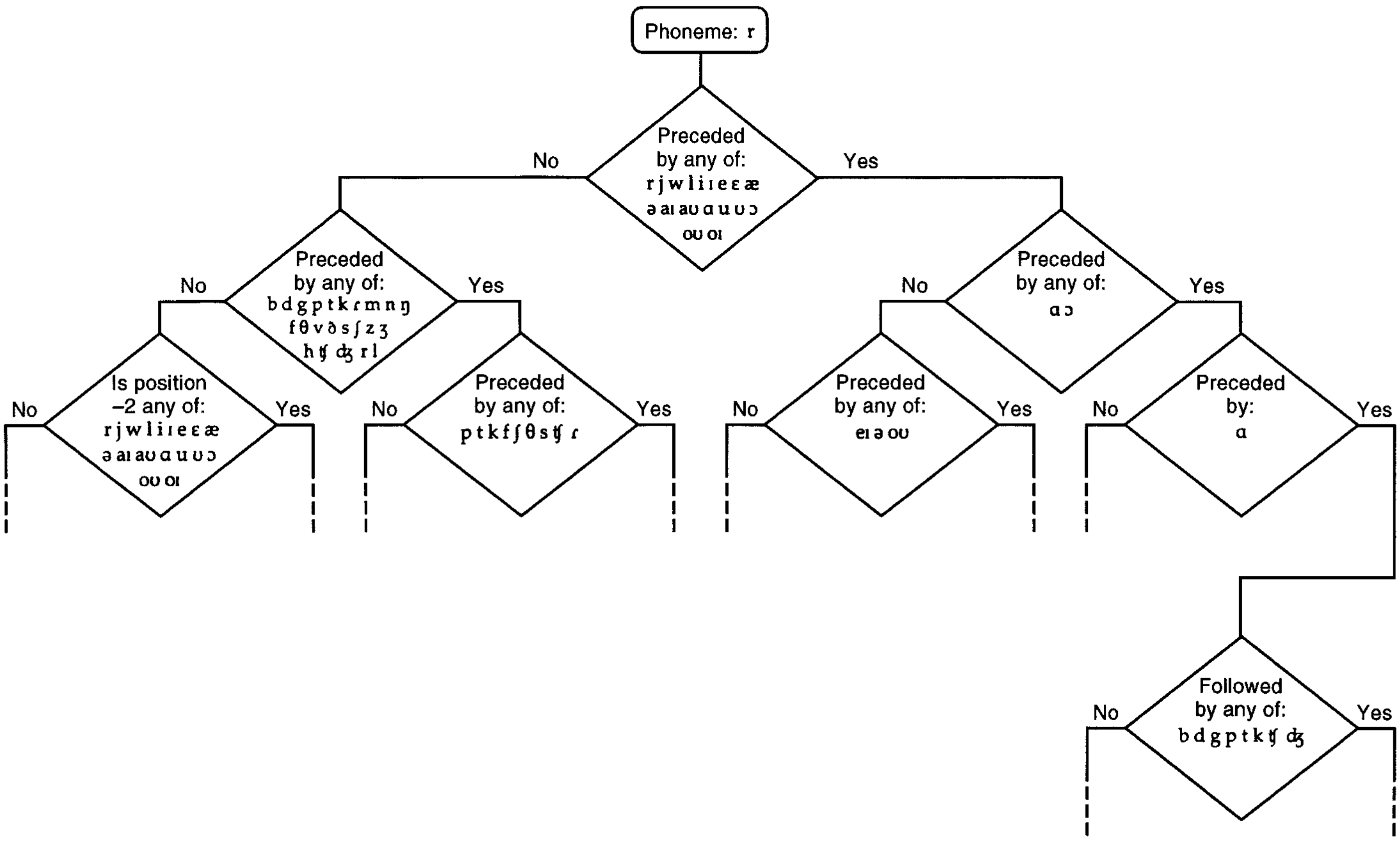

Figure 14. A decision tree enables the recognizer to cope with context-dependent variations of speech sounds. This is the beginning portion of the allophone decision tree for the phoneme "r."

There are a number of different ANN architectures using this type of building block. The nonlinear element permits us to construct complex decision surfaces. These surfaces can be designed to differentiate between classes of patterns, in our case, phonetic classes represented by cepstral data. A common ANN architecture is called the *multilayer perceptron* (MLP). It consists of an input and an output layer along with one or more optional intermediate layers. The intermediate ones are termed *hidden,* as they are not directly observable. Figure 16 illustrates an MLP with an input, a hidden, and an output layer.

We can compute the weights w_1, w_2, . . ., w_n and the threshold θ iteratively by utilizing some training data. The training data are presumed to be correctly labeled with their class affiliations. (This, for example, could be done by Viterbi alignment (20,21) using a previously trained system, as described in the Detailed Theory section.) The iterative procedure is called *backward error propagation,* since the error at the output layer is *propagated* back through the network towards the input for adjustments of the weights and the threshold at each layer. We start with an arbitrary assignment of the weights and the threshold to small values. We successively test each training sample on this ANN. We note the error at a given layer and apply a gradient scheme to adjust the weights in the preceding layer with the goal of minimizing this error. We repeat this calculation for all layers in the network. We iterate this procedure using all training data until we satisfy some convergence criteria.

In the preceding section we explained how several centisecond time frames are appended together to capture the dynamic nature of speech. An alternative procedure is to apply

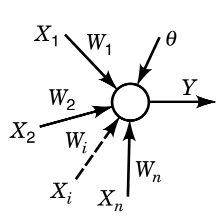

Figure 15. Artificial neural network element. The weighted sum of its inputs is thresholded and applied to a nonlinear element.

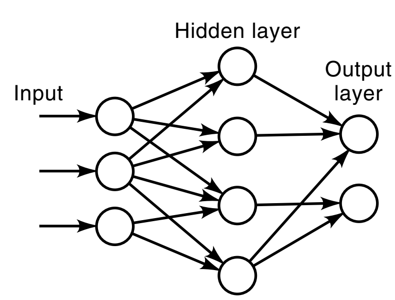

Figure 16. A multilayer perceptron (MLP) with an input, a hidden layer, and an output layer.

several consecutive frames of speech data directly to a neural network element. The resulting architecture is called a *time delay neural network* (tdnn).

Another important network architecture is called the *recurrent neural network* (rnn). In this case, conventional input to each element is supplemented by another set of input representing the *state*. Similarly, the usual output is accompanied by another state output, which is fed back with a time delay as the next state input. The purpose of these states is to retain some knowledge about the past, thereby furnishing some context-related information.

A prime example of this architecture is the one used in the ABBOT system (17a), shown in Fig. 17. This is a single-layer network where the output uses a softmax nonlinearity and the state nonlinearity is a sigmoid one. A delay of 4 time frames is used in the output so that the state mechanism can account for acoustic context.

Segment Model

In HMM formulation described earlier, a Markov state generates a single observable state, which is identified with a frame of speech data. Each frame, representing a time duration of typically 0.01 s, is usually not a meaningful entity in itself. In contrast, a segment model (23) is assumed to produce multiple states of duration l. We can identify these states with some meaningful units, such as subphones (24), phones (25), diphones (26), or syllables. Consequently, a segment model is associated with a duration distribution in addition to a family of output densities. In the case of HMM, we can use modeling distributions such as discrete distributions, full or diagonal covariance Gaussian densities, or a mixture of Gaussian densities. We can employ similar distributions in the case of segment models as well. However, the duration parameter provides extra degrees of freedom, necessitating a more generalized formulation for training and recognition algorithms. Thus, both duration distribution and a set of output densities corresponding to observation sequences of different lengths are used. For instance, Viterbi decoding (or dynamic programming) is a standard way of finding the most likely state in the case of HMM. For a segment model, the state includes both the segment label and duration. The search algorithm includes explicit evaluation of different segmentations as well. Speedup techniques such as the pruning used

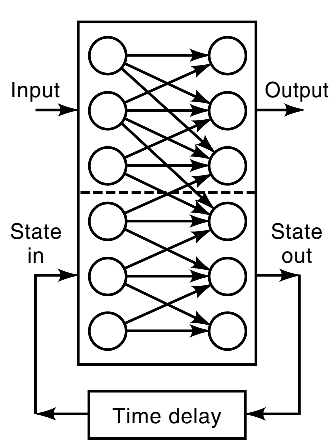

Figure 17. RNN architecture used in the ABBOT system. The time delay helps to account for the acoustic context.

in HMM can be employed in this case as well. Other computational savings may be obtained by eliminating unlikely phone candidates according to partial segment likelihoods, reducing the set of segmentations, and rescoring based on a multipass search strategy. In passing, we note that some researchers have incorporated neural network technology within the segment model framework (27).

Noise Robustness

Handling noise is a perennial problem in speech recognition systems. Noise may be introduced from ambient sources such as background conversation and automobile sounds. Other possibilities include line noise, e.g., in a telephone-based recognition system.

We can apply a host of techniques to increase the immunity of a speech recognition system to noise. One simple approach is to use a noise-canceling microphone. These microphones typically sample the speech from multiple elements and combine their outputs electronically to enhance the signal-to-noise ratio.

There are several algorithmic techniques to improve noise immunity as well. One straightforward algorithmic approach is called *spectral subtraction.* In this technique, we dynamically derive a spectral estimate of the background signal during silence portions of the speech signal and subtract it out from the speech data. Heuristics prevent any negative spectral values after subtraction. Other noise-immunity techniques include *codeword-dependent cepstral normalization* (CDCN) (28) and *parallel model combination* (PMC) (29).

OVERVIEW OF SPEECH RECOGNITION PRODUCTS

A wide spectrum of products driven by speech recognition technology is available in the market. For example, a number of manufacturers sell dictation products. Typically, they are large-vocabulary systems with a built-in vocabulary of some 20,000 to 30,000 words. In addition, custom words such as names of regular correspondents can be added to the list. Until recently, dictation products operated under a number of constraints. With the advent of newer technologies, these constraints are being relaxed. For instance, some of them were isolated-speech systems, requiring the user to pause briefly between successive words. Some others were speaker-dependent as well, so that a user had to go through an elaborate enrollment session before using the product. The latest products in this category are continuous-speech speaker-independent systems. The user can speak normally in a continuous manner. Some speakers with no discernible accents may be able to use the system without any enrollment. However, a short enrollment session is usually necessary to get a reasonable performance. The word accuracy rate is generally in the range of 90% to 98%. The performance tends to deteriorate if certain external factors are present, such as background noise, disfluencies, and severely accented speech. To combat background noise, some products feature noise-canceling microphones. Dysfluencies, such as “uh” and “um,” uttered during dictation can get confused with speech and produce errorful output. More generally, spontaneous speech, which tends to have a casual manner and contain ungrammatical and incomplete sentences, is problematic for recognition systems. Another drawback is that these systems usually work poorly on children’s voices.

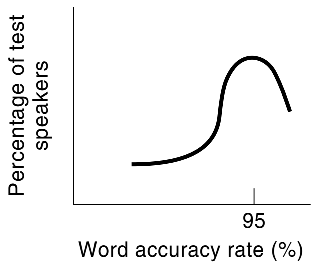

Figure 18. A sample performance plot for a hypothetical recognition system where a 95% word accuracy rate is observed for the majority of the users.

Some speech recognition products are targeted to specific segments of professionals, such as radiologists, pathologists, internists, and legal practitioners. For instance, a system designed for radiologists contains voice-driven boilerplate templates to aid in the task of filling out a diagnostic report for a patient. In addition, the vocabulary is enriched with medical terms frequently used by these professionals.

Another common application category is command and control. Usually the vocabulary size is smaller than that of a dictation system, on the order of several hundred to thousands of words and phrases. For instance, during a dictation session the user may issue a command "new paragraph" when desired. In fact, the possible reach of command and control functions is quite extensive. Examples range from personal computer desktop manipulation, to hands-free dialing of a telephone number, to control of home appliances such as televisions and VCRs. Command and control systems enhanced with natural language processing features can be used to build an airline travel reservation system. The natural language component lets an user ask a question in a variety of normal ways. If the system can handle reduced-bandwidth telephone-quality speech and is equipped with a speech synthesis feature for audio feedback, we can design a product for remote interactive information retrieval. Such a product, for example, would be capable of handling a natural language voice query regarding the location of a restaurant in a large city. Such voice controlled systems can also be used to retrieve E-mail, fax, and voice messages remotely from a personal computer.

TESTING SPEECH RECOGNITION SYSTEMS

We usually test the performance of a speech recognitioin system by aggregating the recognition scores over a set of test speakers. The testing process requires careful consideration of several factors. These include number and composition of speakers in the test pool and awareness of environmental conditions such as background and line noise. Word recognition accuracy is the usual criterion for performance evaluation. However, other considerations, such as decoding time and hardware configurations, may be important in specific circumstances.

We know that a speech recognition system does not perform equally well on all speakers. For various reasons, some known and some not so well understood, some speakers tend to fare better than the others. Thus, it is important to test a speech recognition system on a large pool of test speakers, so that a histogram of performance versus percentage of speakers reaching that performance benchmark, as shown in Fig. 18 for a hypothetical recognition system, can be plotted. In this example, the majority of speakers, 60 percent of them, perform at the level of 95% word accuracy. In other words, these speakers need to correct, on the average, 5 out of every 100 words they dictate. A smaller fraction of speakers achieve a higher recognition score. A significant portion of test speakers perform worse than the median value as well. For fairness in testing, none of the test speakers should be taken from the speaker pool used for designing the recognition system in the first place.

Composition of speakers in the test pool is also an important consideration. For instance, a recognition system for US English speakers should perform at an acceptable level for all native speakers from any part of the United States. Thus, the test pool should ideally reflect the dialectal demographics of the country.

Consideration of environmental conditions such as background and line noise may be an important issue in testing. For instance, a voice-driven car phone dialer needs to work satisfactorily under a plethora of noise conditions routinely encountered in an automobile. A recognition system designed to operate over telephone lines should be tested over a variety of line characteristics and noise conditions. Even dictation systems, ordinarily intended for operation in a relatively quite environment, should be somewhat resistant to common ambient noise, such as background conversation and telephone ringing. It is also desirable to test a dictation system on some extemporaneous speech to determine how its performance is influenced by disfluencies usually present in such data.

The National Institute of Standards and Technology (NIST) has taken an active role in the testing of some categories of speech recognition systems. The advantage is that recognition systems are evaluated on standardized sets of training and test data. Consequently, the strengths and weaknesses of competing systems are relatively easily judged. Two early examples of such standardized databases were TI digits, a collection of utterances of digits from a number of speakers recorded at Texas Instruments, and WSJ, consisting of read texts drawn from the *Wall Street Journal*. More recently, NIST has released the Hub 4 and Switchboard corpuses. Hub4 is a collection of speech databases recorded off the air from various television and radio news broadcasts (30). It is divided into seven different categories, called *focus* conditions, depending on factors such as dialect, fidelity and ambient noise conditions. For example, F0 focus deals with baseline high-fidelity broadcast data, from native English speakers, reading some prepared material in a relatively noise-free environment. In contrast, the F3 focus data are corrupted by background music and may include sections of spontaneous speech. Best word accuracy scores on F0 and F3 data are on the order of 81% and 67% respectively.

The switchboard database consists of conversations recorded off telephone lines. Consequently, they are subject to a number of factors detrimental to speech recognition performance. These factors include unpredictable line characteristics, line noise, speech disfluencies, and casually uttered speech. State-of-the-art speech recognition systems typically reach no more than 60% word accuracy on such data. These relatively poor performance scores on Hub4 and Switchboard data point out the need for further research in these areas.

APPLICATION PROGRAMMING INTERFACES

Application programming interfaces (APIs) handle the integration of speech recognition software in an operating system

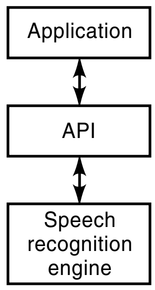

Figure 19. An API interface links an application to the speech recognition engine.

environment. They free the programmers from worries about differences in computer hardware. Three prevailing architectures are Speech API (SAPI) from Microsoft (31); Speech Recognition API (SRAPI), developed by a consortium of companies interested in speech recognition (32); and Speech Manager API (SMAPI) (33), used in a popular commercially available speech recognition program. An illustration of how an API interfaces a speech recognition system to a desired application is given in Fig. 19.

These APIs provide many support functions as well. For instance, SMAPI can carry out dynamic vocabulary handling and perform database operations to query and select system parameters such as users, languages, and domains and to augment an existing vocabulary.

BIBLIOGRAPHY

1. V. Zue, Conversational interfaces: Advances and challenges, in *Proc. Eurospeech '97,* September 1997, pp. 9–18.
2. L. R. Bahl, F. Jelinek, and R. L. Mercer, A maximum likelihood approach to continuous speech recognition, *IEEE Trans. Pattern Recog. Mach. Intell.,* **5**: 179–190, 1983.
3. L. R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, *Proc. IEEE,* **77**: 257–286, 1989.
4. J. Picone, Continuous speech recognition using hidden Markov models, *IEEE ASSP Magazine,* 26–41, July 1990.
5. L. R. Rabiner and B.-H. Huang, *Fundamentals of Speech Recognition,* Englewood Cliffs, NJ: Prentice-Hall, 1993.
6. J. R. Deller, Jr., J. G. Proakis, and J. H. L. Hansen, *Discrete-time Processing of Speech Signals,* New York: Macmillan Publishing Co., 1993.
7. C.-H. Lee, F. K. Soong, and K. K. Paliwal (eds.), *Automatic Speech and Speaker Recognition: Advanced Topics,* Boston: Kluwer Academic Publishers, 1996.
8. R. Ramachandran and R. Mammone (eds.), *Modern Methods of Speech Processing,* Boston: Kluwer Academic Publishers, 1995.
9. A. Averbuch et al., An IBM PC based large-vocabulary isolated-utterance speech recognizer, *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.,* **1**: 53–56, April 1986.
10. P. Price, Evaluation of spoken language systems: The ATIS domain, *Proc. DARPA Speech Nat. Language Workshop,* Morgan Kaufmann Publishers, pp. 91–95, June 1990.
11. Y. Tohkura and K. Aikawa, Cepstral analysis of speech, in J. G. Webster (ed.), *Encyclopedia of Electrical and Electronics Engineering,* New York: Wiley, 1999.
12. S. Furui, Speaker-independent isolated word recognition using dynamic features of speech spectrum, *IEEE Trans. Acoust., Speech Signal Process.,* **34**: 52–59, 1986.
13. L. E. Baum, An inequality and associated maximization technique in statistical estimation of probabilistic functions of Markov processes, *Inequalities,* **3** (1): 1–8, 1972.
14. F. Jelinek, A fast sequential decoding algorithm using a stack, *IBM J. Res. Development,* **13**: 675–685, November 1969.
15. D. P. Morgan and C. L. Scofield, *Neural Networks and Speech Processing,* Boston: Kluwer Academic Publishers, 1991.
16. S. R. Hyde, Automatic speech recognition: A critical survey and discussion of the literature, in N. R. Dixon and T. B. Martin (eds.), *Automatic Speech and Speaker Recognition,* New York: IEEE Press, 1979, pp. 16–55.
17. L. Bahl and F. Jelinek, Decoding for channels with insertions, deletions and substitutions with applications to speech recognition, *IEEE Trans. Inf. Theory,* **21**: 404–411, July 1975.

17a. G. Cook et al., Transcription of broadcast television and radio news: The 1996 ABBOT system, in *Proc. Speech Recognition Workshop,* February 1997, pp. 79–84.

18. J. Baker, The Dragon system—an overview, *IEEE Trans. Acoust. Speech Signal Process.,* **23**: 24–29, February 1975.
19. K.-F. Lee, *Automatic Speech Recognition—The Development of the Sphinx System,* Boston: Kluwer Academic Publishers, 1989.
20. A. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inf. Theory,* **13**: 260–269, 1967.
21. G. D. Forney, Jr., The Viterbi algorithm, *Proc. IEEE,* **61**: 268–278, 1978.
22. N. Morgan and H. A. Bourlard, Neural networks for statistical recognition of continuous speech, *Proc. IEEE,* **83**: 742–770, 1995.
23. M. Ostendorf, V. Digalakis, and O. Kimball, From HMM's to segment models: A unified view of statistical modeling for speech recognition, *IEEE Trans. Speech Audio Process.,* **4**: 360–378, 1996.
24. A. Kannan and M. Ostendorf, A comparison of trajectory and mixture modeling in segment-based word recognition, in *Proc. IEEE ICASSP,* **2**: April 1993, pp. 327–330.
25. M. Ostendorf et al., Continuous word recognition based on the stochastic segment model, in *Proc. DARPA Workshop CSR,* 1992.
26. O. Ghitza and M. Sondhi, Hidden Markov models with templates as nonstationary states: An application to speech recognition, in *Comput. Speech Language,* **2**: 101–119, 1993.
27. G. Zavaliagkos et al., A hybrid segmental neural net/hidden Markov model system for continuous speech recognition, *IEEE Trans. Speech Audio Process.,* **2**: 151–160, 1994.
28. A. Acero, *Acoustical and Environmental Robustness in Automatic Speech Recognition,* Norwell, MA: Kluwer Academic, 1993.
29. M. J. F. Gales and S. J. Young, Cepstral parameter compensation for HMM recognition in noise, *Speech Commun.,* **12**: 231–239, July 1993.
30. J. Garofolo, J. Fiscus, and W. Fisher, Design and preparation of the 1996 Hub-4 broadcast news benchmark test corpora, in *Proc. Speech Recognition Workshop,* February 1997, pp. 15–21.
31. http://www.amudsen.com/mstdg/book/mstdgbook.htm.
32. http://www.srapi.com.
33. http://www.software.ibm.com/voicetype/dev_home.html.

Lalit Bahl
Raimo Bakis
Subrata Das
Michael Picheny
Thomas J. Watson Research Center