7

Speech Recognition

This chapter is about the technologies used to allow computers to recognize the words in human speech. It describes the basic components of all speech recognition systems and illustrates these with an example of a simple recognizer typical of several inexpensive commercial products. After this discussion of basic recognition, the chapter details a larger range of features that can be used to differentiate various recognizers according to the styles of speech interactions they support. Several more advanced recognition techniques are then introduced followed by brief descriptions of selected research projects in large vocabulary and speaker independent word recognition.

BASIC RECOGNIZER COMPONENTS

There are three basic components of any speech recognizer.

1. A speech **representation** that is computationally efficient for pattern matching. The representation is the form into which the recognizer converts the speech signal before it begins analysis to identify words. Typical representations include the output of a bank of filters (similar to a spectrogram), Linear Predictive Coding (LPC) coefficients,[1] and zero crossings of the speech waveform. Recognizers of

[1]See Chapter 3.

increased sophistication incorporate more abstract representations of speech such as phonemes or distinctive spectral features. Hidden Markov Models, described later in this chapter, are a statistical representation based on the various ways words or phonemes may be pronounced.

2. A set of **templates** or **models,** which are descriptions of each word to be recognized, in the representation of speech used by the recognizer. The templates describe the words in the recognizer's vocabulary, i.e., those words that the recognizer can identify. They are reference models against which an input can be compared to determine what was spoken.
3. A **pattern matching** algorithm to determine which template is most similar to a specimen of speech input. This element of the speech recognizer must determine word boundaries, locate the most similar template, and decide whether the difference between the input and the selected template is minor enough to accept the word. In very large vocabulary recognizers, the pattern matching technique usually includes access to more advanced knowledge of language such as syntax and semantics, which constrain how words can be combined into sentences.

To identify a word the recognizer must capture incoming speech and convert it to the chosen internal representation (see Figure 7.1). The pattern matcher then selects the template that most closely matches the input or rejects the utterance if no template is a close enough match.

SIMPLE RECOGNIZER

This section describes a simple recognizer typical of many inexpensive, commercially available devices. This recognizer is designed to identify a small number of

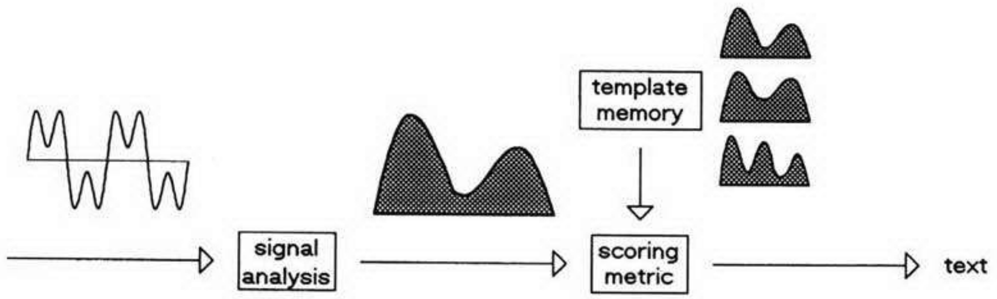

Figure 7.1. The functional elements of a speech recognizer. The user's speech is digitized and converted to the recognizer's internal representation. The captured speech is then compared with the words stored in the recognizer's template memory. The pattern matching algorithm then determines which template, if any, is the closest match.

words spoken in isolation by a specific individual. The purpose of the description that follows is not to provide details of a particular product but rather to offer a sample implementation of the basic components described previously.

Representation

A simple recognizer digitizes incoming audio by means of a codec[2] and then uses a digital signal processing algorithm to extract frames of Linear Predictive Coding (LPC) coefficients every 20 milliseconds. The LPC coefficients are a concise representation of the articulatory characteristics of speech as they capture both the resonances of the vocal tract as well as its voicing characteristics. The 20 millisecond sampling interval results in 50 LPC frames per second, providing significant data reduction to simplify later pattern-matching.

Templates

Templates are gathered by prompting the user from a word list and then saving a set of LPC frames for each word generated as just described. To build a template, the recognizer must determine when the user begins and finishes speaking to know which LPC frames to include in the template. Since each LPC frame or set of coefficients in one 20 millisecond sampling period includes an energy value, this task can be accomplished easily. As shown in Figure 7.2, once the audio exceeds the threshold of background noise, LPC frames are saved until the audio drops below the threshold. It is necessary to wait until the signal has dropped below the threshold for a short period of time as silence or low energy can occur within a word, such as at stop consonants.

In the case of this simple recognizer, templates are trained by a user saying each word once. More sophisticated recognizers may take multiple specimens of a

[2]See Chapter 3.

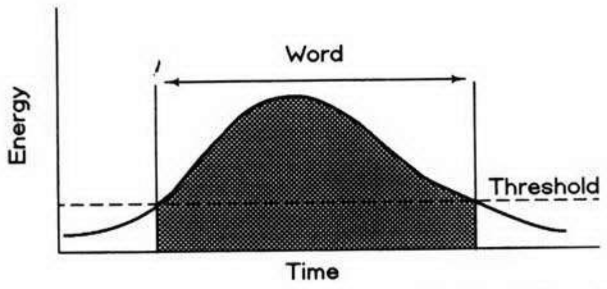

Figure 7.2. Energy thresholds can be used to find word boundaries if the words are spoken in isolation.

word to build a more robust template. Once templates are trained successfully, the user can save them as a disk file on the host computer and reload them before each session, to avoid template training each time the recognizer is used.

This recognizer also allows retraining of a single template, in which case the new LPC frames replace the old. Retraining is useful if the vocabulary changes, if the original training was poor, or if the user decides that a different pronunciation of the word is more comfortable. Retraining may also be necessary when the user has a cold, due to an obstructed nasal cavity.

Pattern Matching

Word recognition requires the detection and classification of speech input. Just as with template creation, a word is captured by detecting that the audio input exceeds the background noise threshold and then converting the input to LPC frames until the input again drops to the background level. The input word is then compared with each template; each frame or set of LPC parameters of the input is compared to the corresponding frame of the template. The frame-by-frame error is the sum of the differences between each of the dozen or so LPC parameters; the error for the word is the sum of the errors for each frame. The template with the smallest word error most closely matches the audio input. If the error exceeds a rejection threshold, a failed recognition attempt is reported. Otherwise the recognizer reports the word corresponding to the closest template.

Both the templates and the captured audio input are multidimensional vectors, with one degree-of-freedom per LPC parameter and one for time. But to illustrate pattern matching let us assume that the templates consist of only a pair of values that display on two axes. One parameter is assigned to the X-axis and the other parameter maps to the Y-axis so that each template occupies a point whose coordinates are the two parameters (see Figure 7.3). Speech input is converted into this representation, i.e., a pair of parameters specifying a point in the same parameter space as the templates. The closest template is the nearest neighbor template to the point representing the input. The four templates divide the space into four regions; any pair of parameters in each region most closely matches the template in that region.

Even the simplest of speech recognizers can improve its accuracy by employing a better pattern matching algorithm. Two additional refinements, illustrated in Figure 7.4, contribute to this recognition improvement. The first refinement is that the input cannot exceed a threshold distance (**r** in the figure) from the nearest template. When the audio input is further from the nearest template than this distance, a rejection error is reported. This is necessary to avoid inadvertently accepting noise, breath sounds, or a mistaken utterance as one of the acceptable words in the recognizer's vocabulary.

The second refinement is a requirement that the input word maps significantly closer to the best match template than any other. If a point is very nearly the same distance from two or more templates, the recognizer is confident that one of these was spoken but uncertain which particular word was spoken. This is depicted in Figure 7.4 as the shaded regions around the lines partitioning the

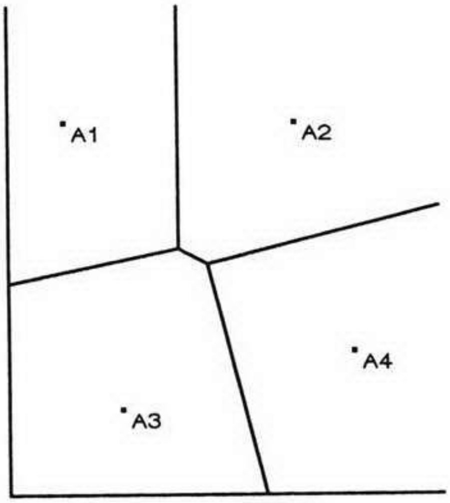

Figure 7.3. To illustrate pattern matching decisions, assume that each template represents two degrees of freedom defining a Cartesian coordinate space. Four templates occupy four points in this space.

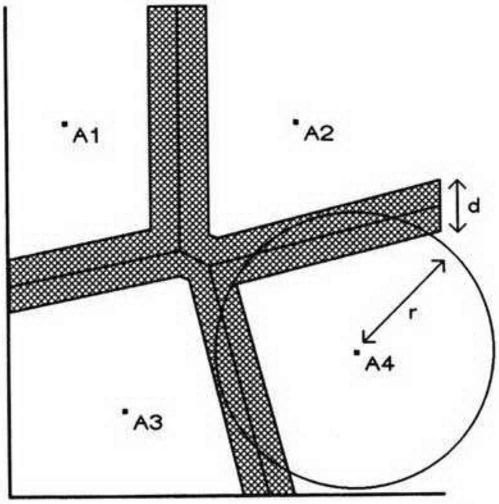

Figure 7.4. A more refined classifier has upper bounds **r** on the distance between an input specimen and a chosen template. It will also reject values which are closer to one template but also close to another, i.e., within distance **d.**

space between neighbors; the distance **d** defines the minimal difference required to differentiate candidate choices. Points within the shaded region are nearly the same distance from each of the two closest templates and will also cause a rejection error.

This section described an extremely simple recognizer that identifies only isolated words. The speech recognizer's simplicity is due in part to its unsophisticated processing of acoustical information: It does not extract significant features but merely considers LPC frames in isolation. Because each template may contain many frames, the pattern matcher has to make a large number of frame-by-frame difference calculations, making it computationally difficult to recognize a large number of words. More sophisticated recognizers detect and classify key acoustic features of an utterance and use this for the internal representation.

Additionally, the pattern matcher just described makes no provision for templates of different lengths and, more importantly, assumes that the input utterance will be of the same length as the correct template. If acoustic boundary detection operates slightly differently from word to word or if the words are spoken at different speeds, this pattern matcher will likely fail. A technique known as **dynamic time warping** or **dynamic programming** (discussed later in this chapter) is often used to compensate for time differences.

CLASSES OF RECOGNIZERS

Recognizers vary widely in their functionality. In addition to performance, other distinctions between recognition techniques are important in selecting a recognizer for a particular application. These distinctions are summarized in Figure 7.5, which displays a three-dimensional space representing the range of possible recognizer configurations. Some of these recognizers are commercially available and have been in use for some time, whereas others are the subject of ongoing research and are just beginning to emerge from research laboratories.

Who Can Use the Recognizer?

A **speaker-independent** recognizer is designed to recognize anyone, while a **speaker-dependent** recognizer can be expected to understand only a single particular speaker. A **speaker-adaptive** recognizer functions to an extent as a combination of the two; it accommodates a new user without requiring that the user train every word in the vocabulary.

Speaker-independent recognition is more difficult than speaker-dependent recognition because we all speak the same words slightly differently. Although we usually understand each other, this variability in pronunciation and voice quality among talkers plays havoc with the pattern matching algorithms of simple recognizers. Speaker-independent recognition requires more elaborate template generation and a clustering technique to identify various ways a particular word may be pronounced.

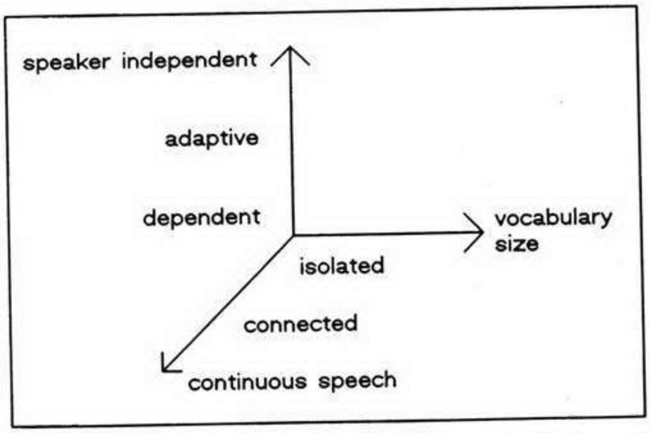

Figure 7.5. A three-dimensional space defined by the different functionalities provided by a recognizer.

This process of classifying the acceptable variations in word pronunciations is the equivalent of the training method just described for the simple illustrative recognizer. Since speaker-independent recognition is such a difficult task, most speaker-independent recognizers are limited to small vocabularies (10 to 20 words) and recognize only single words spoken in isolation. Several commercially available speaker-independent recognizers accept the digits "0" through "9" plus "yes" and "no." Speaker-independent recognition for a larger vocabulary or with words spoken together must rely on additional constraints such as a heavily restricted grammar to achieve adequate recognition performance.

An individual pronounces a single word in a much more consistent fashion than that of a number of different people; this is why speaker-dependent recognition is less difficult. Most speaker-dependent recognizers use a training method similar to the one described for the simple recognizer in the previous section. Multiple templates may be trained and merged together to give an average pronunciation for each word in the vocabulary.

A speaker-adaptive recognizer learns to adapt to a particular speaker either by calibrating itself to a known sample of that person's speech or by incorporating user feedback when it recognizes a word correctly or incorrectly. Such a recognizer has a basic acoustic model for each word that it refines to fit a particular speaker. One common technique is to have the user read a passage of training data; the recognizer can then determine how this particular person speaks.

If a large vocabulary size is needed, a recognizer must be either speaker-adaptive or independent because training many templates is tedious and time consuming. Some adaptive recognizers require many hours of computer time to derive the user model from the training data, but once the data has been acquired the user is freed from the task. The resulting user model can be saved in a host computer and downloaded later, similar to the set of word templates for the speaker-dependent recognizer.

Speaking Style: Connected or Isolated Words?

A **discrete speech** recognizer requires pauses between each word to identify word boundaries. A **connected speech** recognizer is designed to recognize a short series of words spoken together as a phrase. A **continuous speech** recognizer is capable of identifying words in a long string of ordinary speech without the talker pausing between groups of words. A **keyword spotting** recognizer can locate a few words in the midst of any amount of speech.

"Connected speech" for recognition purposes is not natural speech. Users of a connected speech recognizer can speak a few words or perhaps a whole sentence at a time but then must pause to let the recognizer catch up. Users must also speak distinctly. Pausing after each sentence provides reliable boundary points for the first and last word, which facilitates recognition of the entire phrase. But we do not pause between sentences in fluent speech. Human listeners can keep up, and this is the goal of continuous speech recognition, which may become a reality as computer speeds increase.

Keyword spotting searches for a small number of words in a stream of continuous speech. For example, a keyword recognizer might be designed to recognize the digits from an utterance such as "The number is three five seven . . . um . . . four one, please." Successful keyword spotting is comparatively new [Wilpon *et al.* 1990] and more difficult than simply separating speech from non-speech background noise.

Most currently-available commercial products recognize isolated or connected speech. Connected speech is much faster to use because it eliminates the need for unnatural pauses and requires less attention to speaking style on the part of the user. But because connected recognition is more difficult, it may manifest higher error rates, which could outweigh the speed advantage. It can also be argued that speaking discretely is more effective because it requires the user to keep in mind the need to speak clearly from a limited vocabulary while using speech recognition [Biermann *et al.* 1985].

Connected word recognition is much more difficult than isolated word recognition for several reasons.

- Coarticulation changes the pronunciation of a word as a function of its neighbors. Initial and final syllables are particularly subject to modification. Words spoken in isolation do not suffer from coarticulation effects between words.
- It is difficult to find word boundaries reliably from within fluent speech. There is no pause between words, nor is there a significant decrease in speech energy at word boundaries; low energy syllables containing stop consonants are often more discernible than word boundaries.
- The probability of error increases with the number of words in an utterance. If the first word is incorrectly matched against a template which is either too long or too short, then the data to be analyzed for the second word will be incorrect, making the error propagate to subsequent words.

Because of these factors many current applications of speech recognition are based on isolated word recognizers.

Vocabulary Size

Another criterion by which to differentiate recognizers is vocabulary size, which can be grossly categorized as small, medium, or large. Small vocabulary recognizers with less than 200 words have been available for some time. Medium size recognizers (200 to 5000 words) are being developed, usually based on the same algorithms used for smaller vocabulary systems but running on faster hardware. Large vocabulary recognizers aim for the 5000 to 10,000 word level. Much ordinary office language could be handled with a vocabulary of this breadth, which marks the range being aimed for in "listening typewriters" designed to accommodate dictation of business correspondence.

Several issues conspire to complicate the recognition of large vocabularies. One factor is the computational power required for the pattern matching algorithm. The input must be compared with each template, so classification time is a function of the number of templates, i.e., vocabulary size. The requirement of an acceptable response time therefore puts an upper limit on vocabulary size. As microprocessor speeds increase, this computational limit will become less of an issue. Some search-pruning techniques can also be employed to quickly remove candidate templates from the search if they are obviously poor choices, such as if they are much too long or short. If the recognizer employs a grammar, it can use this knowledge of how words can be combined to eliminate syntactically incorrect candidates at each point in the search.

A more serious limitation to vocabulary size is simply that as the number of words increases, it is more likely that the recognizer will find some of them similar. To return to the two parameter template model discussed earlier, compare Figure 7.3 with Figure 7.6. As the number of templates increases, the average distance between them decreases, allowing for a smaller margin of difference in pronunciation between the input sample and its associated template.

Decreased distance between templates is confounded by the fact that as templates or words are added to the recognizer's vocabulary, they are not uniformly distributed throughout the recognizer's representation space. In fact, some words in the vocabulary are likely to sound so similar that to distinguish among them on acoustic evidence alone is extremely difficult. For example, if a vocabulary consisted of the words, "sun," "moon," and "stars," we might expect that distinguishing which word was spoken to be easy. But if we add "Venus," "Earth," and "Mars" to the vocabulary, we might anticipate confusion between "stars" and "Mars."[3]

[3]This is a simplistic example. The strong fricative and stop at the beginning of "stars" should be easily differentiated from the nasal of "Mars" if suitable acoustic features are used for the recognizer's representation. It is hard to predict which words will sound similar without intimate knowledge of the recognizer's internal speech representation.

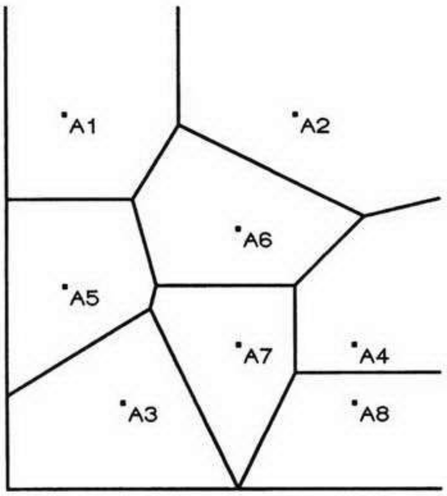

Figure 7.6. As the number of words increases, the mean distance between templates decreases.

ADVANCED RECOGNITION TECHNIQUES

This section explores several techniques to enhance speech recognition. The simplest recognizers, supporting a very small speaker-dependent vocabulary of isolated words, occupy the region near the origin of the three-dimensional model depicted earlier in Figure 7.5. In moving away from the origin in any direction, recognition errors are more likely as the process becomes more complicated. But a large vocabulary, continuous speech, and speaker independence are precisely those attributes that make recognition more widely useful.

Unless users can be convinced to change their habits to speak more clearly and consistently, recognizers must be improved. More advanced recognizers generally employ one of two techniques to make pattern matching more powerful for dealing with variations in speech patterns. These techniques, **Dynamic Time Warping** and **Hidden Markov Models,** are the topics of the next two sections. The descriptions that follow provide an overview of the two techniques; the curious reader is encouraged to consult the references for a more rigorous description of the algorithms. A third approach to managing speech pattern variation uses **neural networks** for speech recognition; this approach is far less developed than the previous two and still speculative.

Data reduction can facilitate pattern matching by minimizing its computational requirements. **Vector Quantization** is a technique employed by many recognizers to capture important variations in speech parameters without overloading the classifier—so it will be described in this section as well. Finally, the last part of this section considers how nonspeech evidence could be used to improve large vocabulary speech recognition.

Dynamic Time Warping

Dynamic Time Warping is a technique that compensates for variability in the rate at which words are spoken. Dynamic Time Warping (DTW) was developed primarily as a mechanism to compensate for variable word duration in connected speech recognition [Sakoe and Chiba 1978]. This method can also help determine word boundaries when an unknown number of words are spoken together. DTW is based on a more general computational technique known as **dynamic programming.**

The duration of spoken words is quite variable, especially if we compare connected speech with the isolated words that may have been used for training a recognizer. Coarticulation may shorten words by combining their boundary syllables with the preceding or subsequent words. Words spoken in isolation (citation form) are longer and often more clearly enunciated. The stress pattern of the sentence lengthens some syllables, as stressed syllables are longer than unstressed syllables. Such changes in length are not linear; every phoneme is not lengthened by the same scale factor. DTW compensates for such nonlinear phenomena.

Consider the simple recognizer discussed earlier in this chapter. It computes an error between the input speech and a template by computing the frame-by-frame difference between the two. But if the talker speaks at a different rate, successive frames of input may not align with the same phoneme in the template, giving a deceptively large error. For example, if the word "fast" is trained but the talker lengthens the vowel ("faaast") during recognition, some frames of the lengthened vowel would be scored against the frames corresponding to the unvoiced "s" in the template. DTW detects that successive "a" frames of the input match the fewer "a" frames of the template better than the "s" frames that follow, and it computes an error based on the selected matchup.

DTW operates by selecting which frames of the reference template best match each frame of the input such that the resulting error between them is minimized. By allowing multiple frames of one to be matched against a single repeated frame of the other, DTW can compress or expand relative time. Because this decision is made on a frame-by-frame basis, the time-scaling is local; DTW may compress one portion of the input and expand another if necessary.

DTW provides a mapping between the sample and a reference template such that the error when the two are compared is minimized. This mapping defines a path between sample and reference frames; for each frame of the sample it specifies the template frame best matched to the next sample frame. In Figure 7.7, the *nth* sample frame is compared to the *mth* reference frame. If the sample is spoken more quickly, then multiple reference frames correspond to a single template frame; this case is indicated by the vertical path, which allows the *nth* sample frame to also match the *m + 1st* reference frame. If the reference and sample are proceeding at the same rate, the *m + 1st* reference frame will match the *n + 1st* sample frame forming a path at a 45-degree angle (slope = 1). Finally, if the sample is spoken more slowly than the reference, multiple sample frames must be compared to a single reference frame as indicated by the horizontal line.

Figure 7.8 shows how these frame-by-frame decisions combine to produce a path for comparing the sample to the reference. The DTW algorithm computes

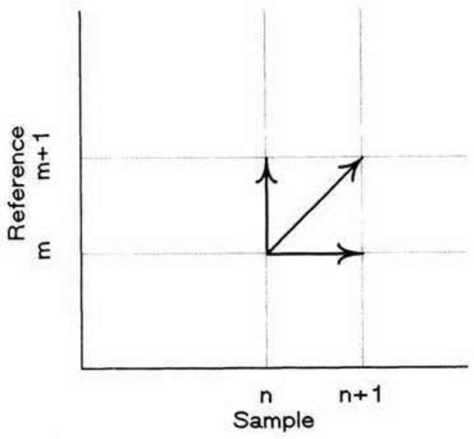

Figure 7.7. Dynamic Time Warping defines a path between frames of a sample utterance and a template such that the frame-by-frame error between the two is minimized. If sample point **m** matches reference point **m,** then reference point **m** + **1** may match either sample point **n** or **n** + **1.**

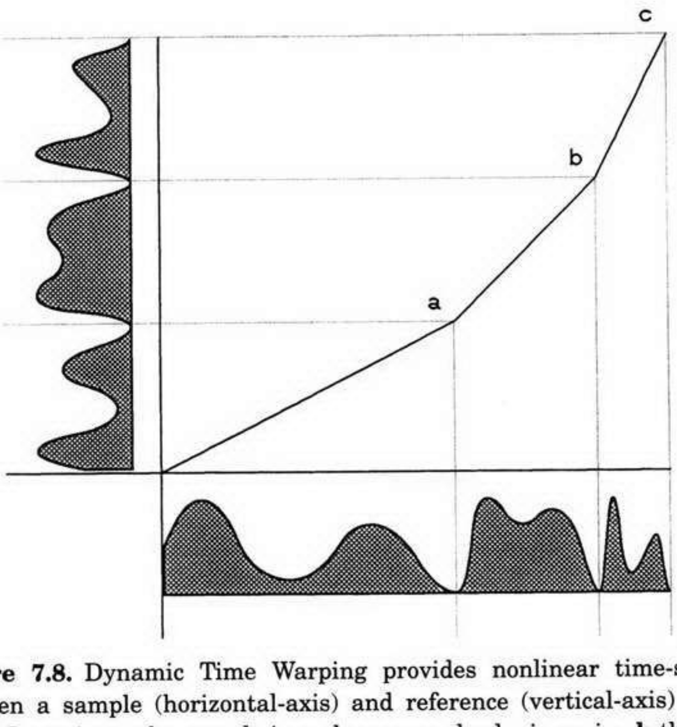

Figure 7.8. Dynamic Time Warping provides nonlinear time-scaling between a sample (horizontal-axis) and reference (vertical-axis) utterances. In region **a** the sample is spoken more slowly; in region **b** they are spoken at the same rate; and in region **c** the template is spoken more slowly.

the minimal error path between the input and template frames; relative speaking rate can be inferred from the slope of this path. A slope of 1 (45-degree line) is optimal when both are spoken at the identical rate.

It is necessary to place limits on the degree of time-scaling allowed by DTW. It would be senseless to compare 50 milliseconds of the sample to 1200 milliseconds of reference; normal speech does not exhibit such wide variability. Constraints on the maximum time-warping ratio are imposed by placing upper and lower bounds on the slope of the DTW path. The typical range of slopes is one-half to two, which accommodates a sample being spoken either half or twice as fast as the reference.

"Two-level" dynamic programming can be used to find word boundaries in a connected utterance containing an unknown number of words. At the lower level, DTW allows time-scaling of each word by computing the minimal error path matching the input to each template. At the higher level, these word-by-word error values are used to compute the best path through the set of all possible word combinations that could comprise a sentence. The two levels operate in concert to apportion the duration of the utterance among the set of selected templates. Figure 7.9 shows how a number of words of different lengths could consist of similar phones, or speech sounds.[4] It may be that the first three phones most closely match the short word "her," which would suggest that the utterance contained three words. But a better *overall* score might be obtained by matching these three phones against the beginning of "heartfelt" and selecting the utterance with only two words.

Dynamic programming is most beneficial for connected speech recognition. Word durations show greater temporal variation in connected speech than in isolated speech. Additionally, word boundaries must be found by considering how the continuous stream of phones can best be segmented into words in the absence of interword pauses. Dynamic programming is used in a number of commercial connected speech recognition systems, but its popularity for research purposes has faded in favor of Hidden Markov Models.

Hidden Markov Models

A **Hidden Markov Model (HMM)** is a two-stage probabilistic process that can be used as a powerful representation for speech. A Hidden Markov Model is a well-behaved mathematical construct, and a number of detailed algorithms exist for solving problems associated with HMMs; introductions to these can be found in [Rabiner and Juang 1986a]. This section first explains HMMs in the abstract and then demonstrates how they can be applied to speech recognition.

An HMM consists of a number of internal states; the model passes from an initial state to the final state as a step-by-step process generating an observable output at each step (state transition). For example, the states may correspond to phonemes contained in a word with the observable output corresponding to the

[4]Such a representation is called a **word lattice.**

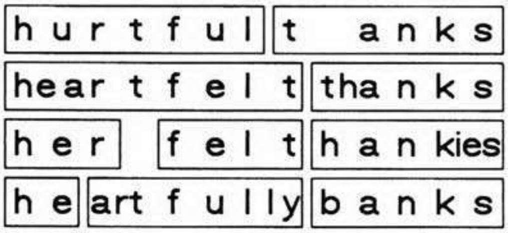

Figure 7.9. Different sequences of words could match an utterance; each sequence implies different word boundaries in the input utterance.

presence or absence of a number of acoustic features. At each step, the model can either move to a new state or stay in the current one. The model is "hidden" in that we cannot observe the state directly but only its output. From the series of observable outputs, we can attempt to guess when the model was in each state. Alternatively, we can say whether some series of outputs was likely to have been generated by a particular HMM.

For example, consider the arrangement shown in Figure 7.10, in which a box and a bowl are each full of black balls and white balls. The box has many more black balls than white while white balls dominate the bowl. Starting with the box, we remove one ball from it at random and pass it to an observer in another room who cannot see what is being done. Then a normal six-sided die is tossed. If the result is less than four, then the source of the next ball is the bowl. If the die is greater than three, then the next ball will be selected from the box. The cycle is then repeated.[5] Whenever we select from the box, a die throw of less than four shifts attention to the bowl. Once we start selecting from the bowl, we continue selecting from it unless the die shows a one, at which point we are finished. This model is likely to spend a majority of its cycles selecting from the bowl since a state transition from the box has a probability ½ of shifting to the bowl, but a bowl transition has a probability of only ⅙ of terminating and ⅚ of continuing with the bowl.[6] We should expect this arrangement to initially present more black balls than white to the observer (while it is in the box state) and then to produce more white balls. We should also expect more white balls overall. But keep in mind that this is a probabilistic process: Not all black balls come from the box and not all selections come from the bowl.

The box-and-bowl setup is a Hidden Markov Model; it is characterized by a number of *states* and associated probabilities for each *transition* from any state. The box and bowl are each a state; the rules about throwing the die define the

[5]This pedagogic example was inspired by Rabiner and Juang [Rabiner and Juang 1986b].

[6]Probability is expressed as a number between zero and one. If an event occurs with probability ½, we would expect it to happen once in two trials. A probability of ⅙ implies that we can expect the event once out of six trials. A smaller probability indicates that an event is less likely to occur.

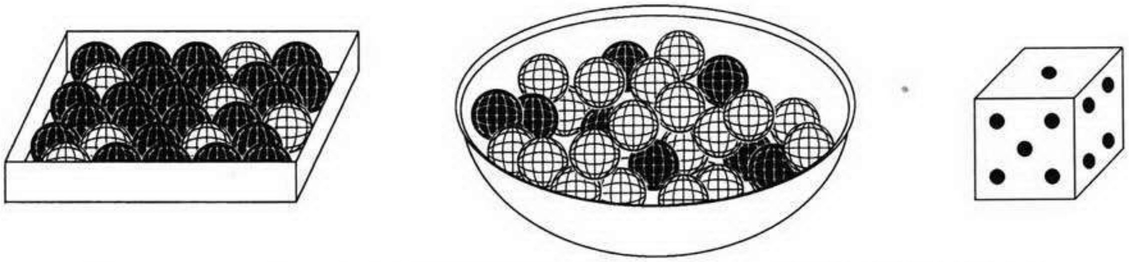

Figure 7.10. A box and a bowl full of colored balls illustrate a Hidden Markov Model.

transition properties. Figure 7.11 shows a more general HMM; in each state **S(n)** there is some probability **P(n,n)** of remaining in that state and some probability **P(n, n + 1)** of transitioning to the next state. In some models, it may be possible to bypass a state or a series of states as indicated by the dashed arc labeled P_{13} in the figure. The sum of the probabilities of all the arcs leaving a state is one; the model accounts for every possible path through its states.

How does this apply to speech recognition? An HMM can be used to represent a word with internal states representing characteristic acoustic segments, possibly phonemes or allophones. The output of a state is a frame or vector of acoustic parameters or features; this output is probabilistic to allow for variability in pronunciation and hence differences in the acoustic representation. The duration of an acoustic segment is a function of the number of steps in which the model is in the state corresponding to the segment. Staying in the same state, i.e. lengthening a phone, depends on the probability associated with the transition from that state to itself (P_{11} in Figure 7.11). Arcs such as P_{13} may be included to indicate that an intermediate state **S2** is optional during pronunciation, such as the second syllable in "chocolate."

In terms of the basic recognizer components, the "templates" consist of a set of HMMs with one HMM associated with every word. The acoustic representation of an input specimen of speech is *not* a collection of states but rather a set of acous-

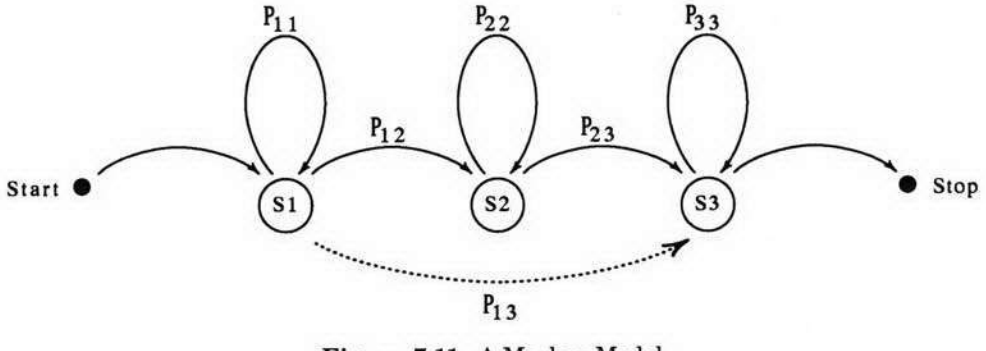

Figure 7.11. A Markov Model.

tic parameters corresponding to the possible observed sequences from the template HMMs. For the HMM-based recognizer, classification consists of determining which HMM has the highest probability of producing the observed acoustic sequence. The Viterbi algorithm is a commonly used method of solving this classification problem; a more detailed description is beyond the scope of this book but see [Lee and Alleva 1992] for an overview of its theory.

There are two related HMM problems which can also be solved mathematically. One is training the statistically defined HMMs that the recognizer requires as templates. Given some number of observation sequences (i.e., repetitions of the word), an HMM must be derived to represent that particular word. The second problem, not directly relevant during recognition, is to determine for some HMM and a set of observed outputs the internal states of the model that were most likely at each step.

Hidden Markov Models are a powerful representation of speech and consequently are used in many speech recognition systems currently under development. Although this section has described isolated word recognition, HMMs can also be used with connected speech to represent not only the phones of each word but also the probabilities of transitioning from one word to another. For connected speech recognition, the observed sequence would be generated by passing through a sequence of states in a sequence of HMMs with one HMM for each word spoken. The combination of words is also described statistically as another layer of HMMs in which each state corresponds to an entire word thereby encoding syntactic and semantic information.

Vector Quantization

Vector Quantization (VQ) is a technique employed for data reduction in speech coding and speech recognition. VQ is used to reduce a widely ranging input value into a smaller set of numbers representing typical values of the input. Because the input is classified as one of a small number of values instead of the value itself, less storage space is required. More importantly, the input is usually multidimensional (e.g., a set of LPC coefficients), and it is reduced to the single dimension identifying the nearest vector-quantized value.

In order to perform Vector Quantization, it is first necessary to decide how to efficiently cluster all possible input values. The analysis of sample data, which must be typical of the data to be encoded, determines how to cluster similar input values. The cluster's "center of gravity," or average value, then represents the entire set of data in the group. Each cluster is represented as one entry in a **codebook,** which lists the average value of each group. Having created a robust VQ codebook from specimen data, new data can be classified by selecting the codebook entry nearest in value to that data. The original value of the input data is lost; it is now represented by the single value associated with the codebook entry.

Building the codebook divides the space of possible values into regions and *typical* values. This is much like the process of selecting word templates for recognition as shown in Figure 7.6. The number of entries required for the codebook depends on how well the data clusters into groups and the accuracy with which the data must be represented. If the data points are widely scattered, it is difficult to cluster them efficiently. The range of values subsumed by each cluster determines the accuracy with which vector quantized data can be represented since the codebook entry is used to look up a value for the data and only one value is stored per cluster.

Representing an input as a codebook entry having a similar value significantly reduces both storage and computation during pattern matching. To vector quantize an input value based on a codebook of N entries, the input must be compared with at most N possible values. The error between each of the codebook values can be computed in advance and stored in a two-dimensional array. During pattern matching the error between an input value and a corresponding template value can be found by table lookup instead of direct computation. The sum of the errors for each frame is the error between the input and that template.

As a simple example, vector quantization can be used to represent the number of hours per day that light bulbs are left burning in a home. If daily use is represented as an integral number of hours from zero to twenty-four, five bits of storage are required for each bulb. Higher resolution, such as minutes in addition to hours, requires even more bits of storage per bulb. However, by observing the individual bulbs we discover some interesting patterns of usage. Many bulbs are rarely turned on. Some bulbs are always left on all night like an outdoor lamp or a child's night light. Other bulbs, in the living room and kitchen, may typically be on from dusk until the occupants go to sleep. Still other bulbs such as in a closet may be turned on occasionally for brief periods of a few minutes each day.

This situation can be described by a vector quantization scheme employing a codebook with four entries to cover the four typical cases described in the previous paragraph.[7] Each entry in the codebook (see Figure 7.12) corresponds to one case of light bulb use, and the value stored for that entry indicates how long such a bulb is likely to be used. Using this codebook, light bulb use is encoded in two bits at a savings of three bits per bulb.

Vector quantization is usually employed to represent multidimensional quantities so the compression to a single-dimensioned codebook offers further data compression. To continue with the light bulb example, we might observe that many of the lamps used rarely or left on all night are low wattage, e.g., 60 or 75 watts. The living room and kitchen lights are more likely to be 100 or 150 watts to illuminate work areas. So we might vector quantize this information as well as shown in Figure 7.13. This new codebook indicates that a bulb assigned value two is typically 120 watts and on for five hours a day. Note that a 150 watt bulb used seven hours a day and a 75 watt bulb used four hours a day will both be represented by codebook value 2 as well. It may be that there is actually no single bulb of 120 watts; this is an average that represents the least error when compared to all the bulbs in the sample set with this codebook value.

[7]Such a codebook is said to be "of size four."

codebook entry	codebook value
0	0 hours
1	15 minutes
2	5 hours
3	10 hours

Figure 7.12. A simple VQ codebook representing light bulb use.

Once quantized, detailed information about a particular light bulb is lost, but we still have a general idea of how much each is used. This would be useful for identifying which bulbs should be replaced by more expensive energy-efficient models. While the memory savings are small in this example, vector quantization can be used for much greater data reduction in speech coding applications where it represents complex acoustical information about an utterance.

How is vector quantization used in speech representation? Consider the output of an LPC coder; it includes precise indications of the locations of each of the formants, or resonances, in the vocal tract as conveyed by filter parameters. But for gross categorization of the position of the vocal tract, e.g., to differentiate the vowels by their F1/F2 ratio, such precision is not necessary. Furthermore, not all combinations of F1 and F2 are possible since they are constrained by the physical characteristics of the vocal tract. Vector quantizing these filter coefficients can capture the overall vocal tract configuration in a very concise form when great precision of each parameter is not necessary.

Employing Constraints

A recognizer's task becomes increasingly difficult as the number of words to be recognized increases. To improve recognition large vocabulary systems apply constraints to the set of all possible words that can make up an input utterance.

codebook entry	codebook value	
	time	wattage
0	0 hours	65
1	15 minutes	75
2	5 hours	120
3	10 hours	70

Figure 7.13. Vector quantization is usually used to represent multidimensional quantities. Here it encodes both the average duration of illumination as well as the wattage of bulbs.

Constraints limit the search space during word classification by ruling out improper combinations of words. Although we readily employ general syntactic and semantic constraints to rule out many ill-formed or nonsense sentences while trying to understand a talker, it is difficult to express the knowledge that underlies such decisions in computationally tractable terms. Current recognition systems incorporate limited constraints specific to a given task.

Some of these constraints are specific to the the input; for example, North American telephone numbers have seven digits (ten if the area code is included). Other sets of constraints can be derived from a specification of all the legal utterances for a set of words and a task. For example, simple rules of syntax tell us that every sentence must have a verb, and common sense dictates that an application for ordering pizza should expect to encounter the word "large" adjacent to "pepperoni." To define these constraints the application designer must specify every legal sentence to be recognized. This is facilitated by lumping groups of words into classes which have meaning in a particular task context, such as "digit," "size," or "topping." Sentences can then be described as series of words combined from particular classes instead of listing every possible word sequence.

Because they are derived by observing people perform a task and computing the likelihood of particular sequences of words, constraints can also be probabilistic. This requires a large body of data from which to derive the probabilities as there may be many possible sentences and statistical representations that require a large quantity of training data. Some language models have been based on analysis of a large corpus of text documents. Although spoken and written language differ, the advantage of this approach is that the text is both readily available and easily analyzed by a computer.

The effectiveness of constraints is related to the degree to which they limit the possible input utterances; this can be stated as the predictive ability of a string of *n* words on the *n + 1st* word. Alternatively, the constraints may be quantized as the **branching factor** associated with any node in a string of input words; this number indicates how many different words can follow one or more previous words. A question requiring a yes-or-no answer has a branching factor of two, for example, while one requiring a single digit has a branching factor of ten. **Perplexity** is a measure of the branching factor averaged across all possible word junctures in the set of legal utterances. The lower the perplexity, the more effectively lexical constraints can improve recognition.

Recognizers based on either *ad hoc* or statistical models of language and the task at hand may be subject to limited portability. The words a talker uses change as a function of the topic of discourse, and a recognizer may not be able to make such a shift. For example, a carefully constructed grammar for a task that allows the user to make inquiries into the state of ships in a navy's fleet will be rather different from one used in general business correspondence. Recognizers do not yet make use of language representations general enough to make specification of the task a simple problem. A more general discussion of methods of specifying both syntactic and semantic constraints is found in Chapter 9.

ADVANCED RECOGNITION SYSTEMS

This section briefly describes details of three advanced speech recognizers that are considerably more sophisticated than the simplistic example presented early in this chapter. These recognizers are research projects, not yet commercial products. This is hardly an exhaustive list of speech recognition research, but it is intended to be representative of the approaches being used in modern, large vocabulary recognition systems. These descriptions will almost certainly be out of date by the time this book is actually published so they should be taken as little more than a snapshot of a rapidly evolving field.

IBM's Tangora

IBM's Tangora speech recognizer is designed to recognize 5000 to 20,000 words of isolated speech [Jelinek 1985]. A real-time version of this research project was implemented on a single-slot personal computer board. The Tangora recognizer is speaker adaptive with training based on the user's reading a few paragraphs of text to calibrate its acoustic model.

The Tangora recognizer uses a vector quantization front end to classify acoustic input, which is then matched to words using discrete Hidden Markov Models of phoneme realization. A linguistic decoder then classifies the output of the acoustic front end producing a word for output. Tangora is targetted at automatic transcription of speech to text in a dictation context.

The linguistic constraints employed by the second stage of this recognizer are based on the probabilities of groups of two or three words occurring in sequence. This statistical model, called a **bigram** or **trigram grammar,** was derived through analysis of a large quantity of text from business correspondence. It should be readily apparent that such a representation of the language includes both syntactic and semantic information as well as indirectly encoded world knowledge. Not only is "late I slept" an unusual sequence due to constraints, but similarly "I slept furiously" is improbable due to its ill-formed semantics; "the green cat" is unlikely due to world knowledge that cats do not have green fur. But this model cannot distinguish the source of knowledge; it merely represents the end result of all factors that contribute to likely word sequences.

CMU's Sphinx

Carnegie-Mellon University's Sphinx recognizer is designed for speaker-independent connected speech recognition [Lee and Hon 1988, Lee 1988]. At the time of this writing, it operates in nearly real time using hardware multiprocessor peripherals to aid in the search process. Sphinx operates with a 1000 word vocabulary and achieves a recognition accuracy percentage in the mid-nineties when provided with a bigram language model; however, accuracy drops to the mid-fifties without the grammar model.

Sphinx makes extensive use of Hidden Markov Models: each phone is represented by an HMM, each word is a network of phones, and the language is a network of words. Phones in function words are modeled separately from other phones because of the higher degree of coarticulation and distortion in function words. Because function words are common to all vocabularies, they can be modeled using more thoroughly trained HMMs.

Sphinx uses multiple codebook vector quantization as its acoustic front end. Separate codebooks are used to encode energy, cepstral, and differential cepstral parameters. The cepstrum is a signal analysis construct gaining popularity with speech recognition because it differentiates voicing and vocal tract aspects of the speech signal. Differential cepstrum measurements indicate how the cepstrum changes from one sampling period to the next; this measure is particularly useful for analysis of the most dynamic aspects of speech such as many of the consonants.

MIT's SUMMIT

The SUMMIT [Zue *et al.* 1989a] system is a phoneme-based connected speech recognizer being developed by the Spoken Language Systems Group at M.I.T. In recognizing speech, SUMMIT first transforms the speech signal into a representation modeled after the auditory processing that occurs in our ears. This is a nonlinear transformation based on Seneff's auditory model [Seneff 1988]; it enhances acoustic information crucial for recognizing speech and suppresses irrelevant detail. Part of this analysis models the transduction between the hairs in the cochlea and their associated nerve cells to enhance temporal variation in the input emphasizing onsets and offsets crucial to detecting consonants. Another portion, which models the nerve cell firing rate related to the characteristic frequency of the cell, emphasizes spectral peaks; this is useful for vowel identification.

The next stage of recognition segments the transformed speech by finding acoustic landmarks in the signal. Regions between the landmarks are grouped into segments on the basis of similarity. A set of phonetic classifiers assigns probabilistic phonetic labels to each of these portions of the speech signal. Different classifiers may be invoked for differentiating among distinct classes of phonemes. Words are represented by a network of phonemes; several networks may be used to encode alternate pronunciations. A pattern matching algorithm similar to dynamic programming matches the phoneme labels of the input against the word networks, deducing word boundaries in the process. Coarticulation rules compensate for some phonemic aspects of connected speech.

SUMMARY

This chapter has detailed the process of recognizing speech by computers. It began with an overview of the recognition process as a set of three basic components: an acoustical representation, a vocabulary or set of templates, and a pattern matching process to measure the similarity of an input utterance with each of the set of templates. This process was illustrated by considering a simplistic, hypothetical recognizer. The chapter then discussed a full range of possible recognizer configurations differentiated by the specificity of the talker, the style of speaking, and the size of the vocabulary to be recognized. Several techniques important for connected and large vocabulary recognition were then introduced; these included Dynamic Time Warping, Hidden Markov Models, Vector Quantization, and language constraints. The chapter concluded with a descriptive overview of three well-known research projects on large vocabulary speaker independent recognizers.

This chapter was intended to introduce the technology of speech recognition and some of the difficulties it must overcome to be effective. Speech recognition algorithms are under continual development and are useful for an ever-expanding range of applications. Even more important for its deployment, the steady increases in microprocessor speeds enable even the more sophisticated algorithms to be implemented entirely as software thus encouraging the more widespread distribution of applications that make use of recognition.

The next chapter explores the utility of speech recognition, the classes of applications for which it may be well suited, and interaction techniques to make effective use of recognition. Of central concern is the issue of speech recognition errors; error rates remain the primary deterrent to the successful application of speech recognition. But in some specialized application niches speech recognition is already being used to improve personal productivity.

FURTHER READING

O'Shaughnessy covers many topics in speech recognition thoroughly. Furui and Sondhi is a collection of particularly current papers about all aspects of speech processing with particular emphasis on recognition. Although rather technical, it has chapters describing most of the concepts and some of the speech-recognition research projects discussed in this chapter. Waibel and Lee present a collection of previously published research papers about speech recognition. Dixon and Martin is a similar collection of key papers from the 1970s. Another good source of very good working papers are conference proceedings of the DARPA Speech and Natural Language Workshops.