# SPEECH SYNTHESIS AND RECOGNITION

## 2ND EDITION

John Holmes
and Wendy Holmes

# Speech Synthesis and Recognition

Second Edition

**John Holmes and Wendy Holmes**


London and New York

First edition by the late Dr J.N. Holmes published 1988 by Van Nostrand Reinhold
Second edition published 2001 by Taylor & Francis
11 New Fetter Lane, London EC4P 4EE

Simultaneously published in the USA and Canada
by Taylor & Francis
29 West 35th Street, New York, NY 10001

*Taylor & Francis is an imprint of the Taylor & Francis Group*



*Publisher's Note*
This book has been prepared from camera-ready copy provided by the authors.
Printed and bound in Great Britain by Biddles Ltd, Guildford and King's Lynn



*British Library Cataloguing in Publication Data*
A catalogue record for this book is available from the British Library

*Library of Congress Cataloging in Publication Data*

Holmes, J.N.
Speech synthesis and recognition/John Holmes and Wendy Holmes.--2nd ed
p.cm.
Includes bibliographical references and index.
ISBN 0-7484-0856-8 (hc.) -- ISBN 0-7484-0857-6 (pbk.)
1. Speech processing systems. I. Holmes, Wendy (Wendy J.) II. Title.

TK77882.S65 H64 2002
006.4'54--dc21

2001044279

ISBN 0-7484-0856-8 (hbk)
ISBN 0-7484-0857-6 (pbk)

# CONTENTS

# CHAPTER 8

# Introduction to Automatic Speech Recognition: Template Matching

## 8.1 INTRODUCTION

Much of the early work on **automatic speech recognition** (**ASR**), starting in the 1950s, involved attempting to apply rules based either on acoustic/phonetic knowledge or in many cases on simple *ad hoc* measurements of properties of the speech signal for different types of speech sound. The intention was to decode the signal directly into a sequence of phoneme-like units. These early methods, extensively reviewed by Hyde (1972), achieved very little success. The poor results were mainly because co-articulation causes the acoustic properties of individual phones to vary very widely, and any rule-based hard decisions about phone identity will often be wrong if they use only local information. Once wrong decisions have been made at an early stage, it is extremely difficult to recover from the errors later.

An alternative to rule-based methods is to use **pattern-matching** techniques. Primitive pattern-matching approaches were being investigated at around the same time as the early rule-based methods, but major improvements in speech recognizer performance did not occur until more general pattern-matching techniques were invented. This chapter describes typical methods that were developed for spoken word recognition during the 1970s. Although these methods were widely used in commercial speech recognizers in the 1970s and 1980s, they have now been largely superseded by more powerful methods (to be described in later chapters), which can be understood as a generalization of the simpler pattern-matching techniques introduced here. A thorough understanding of the principles of the first successful pattern-matching methods is thus a valuable introduction to the later techniques.

## 8.2 GENERAL PRINCIPLES OF PATTERN MATCHING

When a person utters a word, as we saw in Chapter 1, the word can be considered as a sequence of phonemes (the linguistic units) and the phonemes will be realized as phones. Because of inevitable co-articulation, the acoustic patterns associated with individual phones overlap in time, and therefore depend on the identities of their neighbours. Even for a word spoken in isolation, therefore, the acoustic pattern is related in a very complicated way to the word's linguistic structure.

However, if the same person repeats the same isolated word on separate occasions, the pattern is likely to be generally similar, because the same phonetic relationships will apply. Of course, there will probably also be differences, arising from many causes. For example, the second occurrence might be spoken faster or more slowly; there may be differences in vocal effort; the pitch and its variation during the word could be different; one example may be spoken more precisely

than the other, etc. It is obvious that the waveform of separate utterances of the same word may be very different. There are likely to be more similarities between spectrograms because (assuming that a short time-window is used, see Section 2.6), they better illustrate the vocal-tract resonances, which are closely related to the positions of the articulators. But even spectrograms will differ in detail due to the above types of difference, and timescale differences will be particularly obvious.

A well-established approach to ASR is to store in the machine example acoustic patterns (called **templates**) for all the words to be recognized, usually spoken by the person who will subsequently use the machine. Any incoming word can then be compared in turn with all words in the store, and the one that is most similar is assumed to be the correct one. In general none of the templates will match perfectly, so to be successful this technique must rely on the correct word being more similar to its own template than to any of the alternatives.

It is obvious that in some sense the sound pattern of the correct word is likely to be a better match than a wrong word, because it is made by more similar articulatory movements. Exploiting this similarity is, however, critically dependent on how the word patterns are compared, i.e. on how the 'distance' between two word examples is calculated. For example, it would be useless to compare waveforms, because even very similar repetitions of a word will differ appreciably in waveform detail from moment to moment, largely due to the difficulty of repeating the intonation and timing exactly.

It is implicit in the above comments that it must also be possible to identify the start and end points of words that are to be compared.

## 8.3 DISTANCE METRICS

In this section we will consider the problem of comparing the templates with the incoming speech when we know that corresponding points in time will be associated with similar articulatory events. In effect, we appear to be assuming that the words to be compared are spoken in isolation at exactly the same speed, and that their start and end points can be reliably determined. In practice these assumptions will very rarely be justified, and methods of dealing with the resultant problems will be discussed later in the chapter.

In calculating a distance between two words it is usual to derive a short-term distance that is local to corresponding parts of the words, and to integrate this distance over the entire word duration. Parameters representing the acoustic signal must be derived over some span of time, during which the properties are assumed not to change much. In one such span of time the measurements can be stored as a set of numbers, or **feature vector**, which may be regarded as representing a point in multi-dimensional space. The properties of a whole word can then be described as a succession of feature vectors (often referred to as **frames**), each representing a time slice of, say, 10–20 ms. The integral of the distance between the patterns then reduces to a sum of distances between corresponding pairs of feature vectors. To be useful, the distance must not be sensitive to small differences in intensity between otherwise similar words, and it should not give too much weight to differences in pitch. Those features of the acoustic signal that are determined by the phonetic properties should obviously be given more weight in the distance calculation.

### 8.3.1 Filter-bank analysis

The most obvious approach in choosing a distance metric which has some of the desirable properties is to use some representation of the short-term power spectrum. It has been explained in Chapter 2 how the short-term spectrum can represent the effects of moving formants, excitation spectrum, etc.

Although in tone languages pitch needs to be taken into account, in Western languages there is normally only slight correlation between pitch variations and the phonetic content of a word. The likely idiosyncratic variations of pitch that will occur from occasion to occasion mean that, except for tone languages, it is normally safer to ignore pitch in whole-word pattern-matching recognizers. Even for tone languages it is probably desirable to analyse pitch variations separately from effects due to the vocal tract configuration. It is best, therefore, to make the bandwidth of the spectral resolution such that it will not resolve the harmonics of the fundamental of voiced speech. Because the excitation periodicity is evident in the amplitude variations of the output from a broad-band analysis, it is also necessary to apply some time-smoothing to remove it. Such time-smoothing will also remove most of the fluctuations that result from randomness in turbulent excitation.

At higher frequencies the precise formant positions become less significant, and the resolving power of the ear (**critical bandwidth** – see Chapter 3) is such that detailed spectral information is not available to human listeners at high frequencies. It is therefore permissible to make the spectral analysis less selective, such that the effective filter bandwidth is several times the typical harmonic spacing. The desired analysis can thus be provided by a set of bandpass filters whose bandwidths and

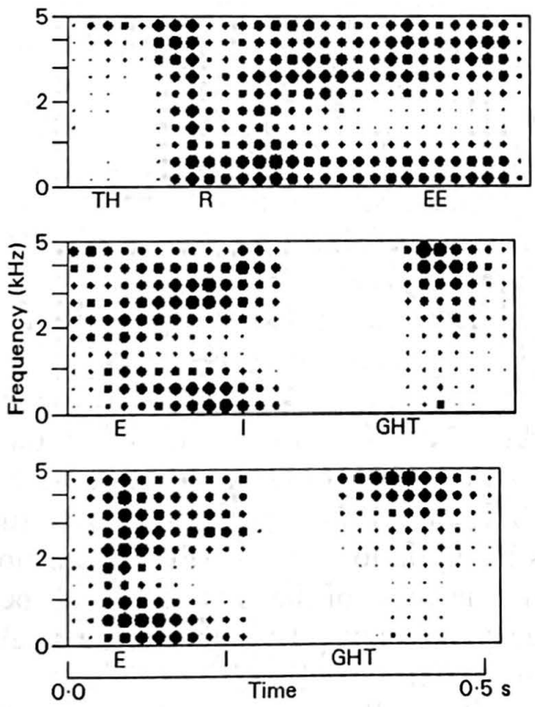



**Figure 8.1** Spectrographic displays of a 10-channel filter-bank analysis (with a non-linear frequency spacing of the channels), shown for one example of the word "three" and two examples of the word "eight". It can be seen that the examples of "eight" are generally similar, although the lower one has a shorter gap for the [t] and a longer burst.

spacings are roughly equal to those of critical bands and whose range of centre frequencies covers the frequencies most important for speech perception (say from 300 Hz up to around 5 kHz). The total number of band-pass filters is therefore not likely to be more than about 20, and successful results have been achieved with as few as 10. When the necessary time-smoothing is included, the feature vector will represent the signal power in the filters averaged over the frame interval.

The usual name for this type of speech analysis is **filter-bank** analysis. Whether it is provided by a bank of discrete filters, implemented in analogue or digital form, or is implemented by sampling the outputs from short-term Fourier transforms, is a matter of engineering convenience. Figure 8.1 displays word patterns from a typical 10-channel filter-bank analyser for two examples of one word and one example of another. It can be seen from the frequency scales that the channels are closer together in the lower-frequency regions.

A consequence of removing the effect of the fundamental frequency and of using filters at least as wide as critical bands is to reduce the amount of information needed to describe a word pattern to much less than is needed for the waveform. Thus storage and computation in the pattern-matching process are much reduced.

### 8.3.2 Level normalization

Mean speech level normally varies by a few dB over periods of a few seconds, and changes in spacing between the microphone and the speaker's mouth can also cause changes of several dB. As these changes will be of no phonetic significance, it is desirable to minimize their effects on the distance metric. Use of filter-bank power directly gives most weight to more intense regions of the spectrum, where a change of 2 or 3 dB will represent a very large absolute difference. On the other hand, a 3 dB difference in one of the weaker formants might be of similar phonetic significance, but will cause a very small effect on the power. This difficulty can be avoided to a large extent by representing the power logarithmically, so that similar power ratios have the same effect on the distance calculation whether they occur in intense or weak spectral regions. Most of the phonetically unimportant variations discussed above will then have much less weight in the distance calculation than the differences in spectrum level that result from formant movements, etc.

Although comparing levels logarithmically is advantageous, care must be exercised in very low-level sounds, such as weak fricatives or during stop-consonant closures. At these times the logarithm of the level in a channel will depend more on the ambient background noise level than on the speech signal. If the speaker is in a very quiet environment the logarithmic level may suffer quite wide irrelevant variations as a result of breath noise or the rustle of clothing. One way of avoiding this difficulty is to add a small constant to the measured level before taking logarithms. The value of the constant would be chosen to dominate the greatest expected background noise level, but to be small compared with the level usually found during speech.

Differences in vocal effort will mainly have the effect of adding a constant to all components of the log spectrum, rather than changing the shape of the spectrum cross-section. Such differences can be made to have no effect on the distance metric by subtracting the mean of the logarithm of the spectrum level of each frame

from all the separate spectrum components for the frame. In practice this amount of level compensation is undesirable because extreme level variations are of some phonetic significance. For example, a substantial part of the acoustic difference between [f] and any vowel is the difference in level, which can be as much as

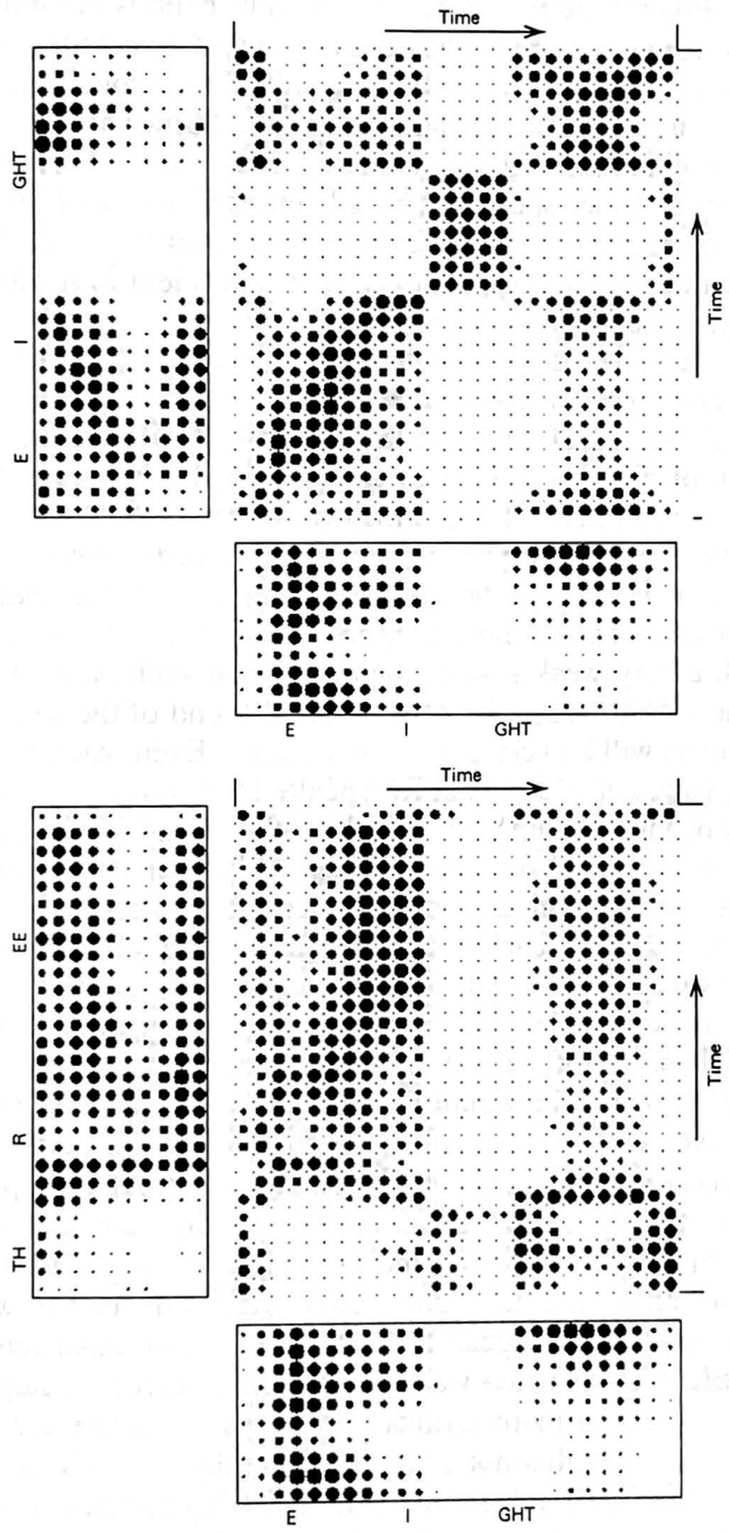



**Figure 8.2** Graphical representation of the distance between frames of the spectrograms shown in Figure 8.1. The larger the blob the smaller the distance. It can be seen that there is a continuous path of fairly small distances between the bottom left and top right when the two examples of "eight" are compared, but not when "eight" is compared with "three".

30 dB. Recognition accuracy might well suffer if level differences of this magnitude were ignored. A useful compromise is to compensate only partly for level variations, by subtracting some fraction (say in the range 0.7 to 0.9) of the mean logarithmic level from each spectral channel. There are also several other techniques for achieving a similar effect.

A suitable distance metric for use with a filter bank is the sum of the squared differences between the logarithms of power levels in corresponding channels (i.e. the square of the **Euclidean distance** in the multi-dimensional space). A graphical representation of the Euclidean distance between frames for the words used in Figure 8.1 is shown in Figure 8.2.

There are many other spectrally based representations of the signal that are more effective than the simple filter bank, and some of these will be described in Chapter 10. The filter-bank method, however, is sufficient to illustrate the pattern-matching principles explained in this chapter.

## 8.4 END-POINT DETECTION FOR ISOLATED WORDS

The pattern comparison methods described above assume that the beginning and end points of words can be found. In the case of words spoken in isolation in a quiet environment it is possible to use some simple level threshold to determine start and end points. There are, however, problems with this approach when words start or end with a very weak sound, such as [f]. In such cases the distinction in level between the background noise and the start or end of the word may be slight, and so the end points will be very unreliably defined. Even when a word begins and ends in a strong vowel, it is common for speakers to precede the word with slight noises caused by opening the lips, and to follow the word by quite noisy exhalation. If these spurious noises are to be excluded the level threshold will certainly have to be set high enough to also exclude weak unvoiced fricatives. Some improvement in separation of speech from background noise can be obtained if the spectral properties of the noise are also taken into account. However, there is no reliable way of determining whether low-level sounds that might immediately precede or follow a word should be regarded as an essential part of that word without simultaneously determining the identity of the word.

Of course, even when a successful level threshold criterion has been found, it is necessary to take account of the fact that some words can have a period of silence within them. Any words (such as "containing" and "stop") containing unvoiced stop consonants at some point other than the beginning belong to this category. The level threshold can still be used in such cases, provided the end-of-word decision is delayed by the length of the longest possible stop gap, to make sure that the word has really finished. When isolated words with a final unvoiced stop consonant are used in pattern matching, a more serious problem, particularly for English, is that the stop burst is sometimes, but not always, omitted by the speaker. Even when the end points are correctly determined, the patterns being compared for words which are nominally the same will then often be inherently different.

Although approximate end points can be found for most words, it is apparent from the above comments that they are often not reliable.

## 8.5 ALLOWING FOR TIMESCALE VARIATIONS

Up to now we have assumed that any words to be compared will be of the same length, and that corresponding times in separate utterances of a word will represent the same phonetic features. In practice speakers vary their speed of speaking, and often do so non-uniformly so that equivalent words of the same total length may differ in the middle. This timescale uncertainty is made worse by the unreliability of end-point detection. It would not be unusual for two patterns of apparently very different length to have the underlying utterances spoken at the same speed, and merely to have a final fricative cut short by the end-point detection algorithm in one case as a result of a slight difference in level.

Some early implementations of isolated-word recognizers tried to compensate for the timescale variation by a uniform time normalization to ensure that all patterns being matched were of the same length. This process is a great improvement over methods such as truncating the longer pattern when it is being compared with a shorter one, but the performance of such machines was undoubtedly limited by differences in timescale. In the 1960s, however, a technique was developed which is capable of matching one word on to another in a way which applies the optimum non-linear timescale distortion to achieve the best match at all points. The mathematical technique used is known as **dynamic programming** (DP), and when applied to simple word matching the process is often referred to as **dynamic time warping** (**DTW**). DP in some form is now almost universally used in speech recognizers.

## 8.6 DYNAMIC PROGRAMMING FOR TIME ALIGNMENT

Assume that an incoming speech pattern and a template pattern are to be compared, having $n$ and $N$ frames respectively. Some distance metric can be used to calculate the distance, $d(i, j)$, between frame $i$ of the incoming speech and frame $j$ of the template. To illustrate the principle, in Figure 8.3 the two sets of feature vectors of the words have been represented by letters of the word "pattern". Differences in timescale have been indicated by repeating or omitting letters of the word, and the fact that feature vectors will not be identical, even for corresponding points of equivalent words, is indicated by using different type styles for the letters. It is, of course, assumed in this explanation that all styles of the letter "a" will yield a lower value of distance between them than, say, the distance between an "a" and any example of the letter "p". To find the total difference between the two patterns, one requires to find the sum of all the distances between the individual pairs of frames along whichever path between the bottom-left and top-right corners in Figure 8.3 that gives the smallest distance. This definition will ensure that corresponding frames of similar words are correctly aligned.

One way of calculating this total distance is to consider all possible paths, and add the values of $d(i, j)$ along each one. The distance measure between the patterns is then taken to be the lowest value obtained for the cumulative distance. Although this method is bound to give the correct answer, the number of valid paths becomes so large that the computation is impossible for any practical speech recognition machine. Dynamic programming is a mathematical technique which guarantees to

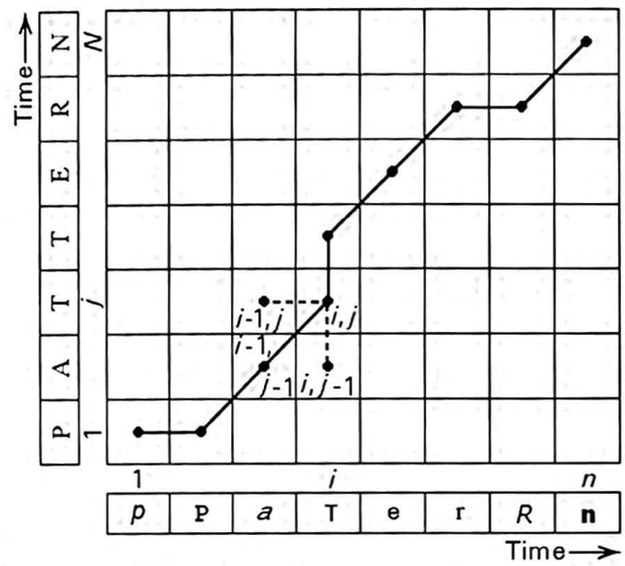



**Figure 8.3** Illustration of a time-alignment path between two words that differ in their timescale. Any point $i,j$ can have three predecessors as shown.

find the cumulative distance along the optimum path without having to calculate the cumulative distance along all possible paths.

Let us assume that valid paths obey certain common-sense constraints, such that portions of words do not match when mutually reversed in time (i.e. the path on Figure 8.3 always goes forward with a non-negative slope). Although skipping single frames could be reasonable in some circumstances, it simplifies the explanation if, for the present, we also assume that we can never omit from the comparison process any frame from either pattern. In Figure 8.3, consider a point $i,j$ somewhere in the middle of both words. If this point is on the optimum path, then the constraints of the path necessitate that the immediately preceding point on the path is $i-1,j$ or $i-1,j-1$ or $i,j-1$. These three points are associated with a horizontal, diagonal or vertical path step respectively. Let $D(i,j)$ be the cumulative distance along the optimum path from the beginning of the word to point $i,j$, thus:

$$D(i,j) = \sum_{\substack{x,y=1,1 \\ \text{along the} \\ \text{best path}}}^{i,j} d(x,y) \ . \tag{8.1}$$

As there are only the three possibilities for the point before point $i,j$ it follows that

$$D(i,j) = \min\left[D(i-1,j), D(i-1,j-1), D(i,j-1)\right] + d(i,j) \ . \tag{8.2}$$

The best way to get to point $i,j$ is thus to get to one of the immediately preceding points *by the best way*, and then take the appropriate step to $i,j$. The value of $D(1,1)$ must be equal to $d(1,1)$ as this point is the beginning of all possible paths. To reach points along the bottom and the left-hand side of Figure 8.3 there is only one possible direction (horizontal or vertical, respectively). Therefore, starting with the value of $D(1,1)$, values of $D(i,1)$ or values of $D(1,j)$ can be calculated in turn for increasing values of $i$ or $j$. Let us assume that we calculate the vertical column, $D(1,j)$, using a reduced form of Equation (8.2) that does not have to

consider values of $D(i-1,j)$ or $D(i-1,j-1)$. (As the scheme is symmetrical we could equally well have chosen the horizontal direction instead.) When the first column values for $D(1,j)$ are known, Equation (8.2) can be applied successively to calculate $D(i,j)$ for columns 2 to $n$. The value obtained for $D(n, N)$ is the score for the best way of matching the two words. For simple speech recognition applications, just the final score is required, and so the only working memory needed during the calculation is a one-dimensional array for holding a column (or row) of $D(i,j)$ values. However, there will then be no record at the end of what the optimum path was, and if this information is required for any purpose it is also necessary to store a two-dimensional array of back-pointers, to indicate which direction was chosen at each stage. It is not possible to know until the end has been reached whether any particular point will lie on the optimum path, and this information can only be found by tracing back from the end.

## 8.7 REFINEMENTS TO ISOLATED-WORD DP MATCHING

The DP algorithm represented by Equation (8.2) is intended to deal with variations of timescale between two otherwise similar words. However, if two examples of a word have the same length but one is spoken faster at the beginning and slower at the end, there will be more horizontal and vertical steps in the optimum path and fewer diagonals. As a result there will be a greater number of values of $d(i, j)$ in the final score for words with timescale differences than when the timescales are the same. Although it may be justified to have some penalty for timescale distortion, on the grounds that an utterance with a very different timescale is more likely to be the wrong word, it is better to choose values of such penalties explicitly than to have them as an incidental consequence of the algorithm. Making the number of contributions of $d(i,j)$ to $D(n, N)$ independent of the path can be achieved by modifying Equation (8.2) to add twice the value of $d(i,j)$ when the path is diagonal. One can then add an explicit penalty to the right-hand side of Equation (8.2) when the step is either vertical or horizontal. Equation (8.2) thus changes to:

$$D(i,j) = \min\big[D(i-1,j) + d(i,j) + hdp,\; D(i-1,j-1) + 2d(i,j),\; D(i,j-1) + d(i,j) + vdp\big]. \tag{8.3}$$

Suitable values for the horizontal and vertical distortion penalties, *hdp* and *vdp*, would probably have to be found by experiment in association with the chosen distance metric. It is, however, obvious that, all other things being equal, paths with appreciable timescale distortion should be given a worse score than diagonal paths, and so the values of the penalties should certainly not be zero.

Even in Equation (8.3) the number of contributions to a cumulative distance will depend on the lengths of both the example and the template, and so there will be a tendency for total distances to be smaller with short templates and larger with long templates. The final best-match decision will as a result favour short words. This bias can be avoided by dividing the total distance by the template length.

The algorithm described above is inherently symmetrical, and so makes no distinction between the word in the store of templates and the new word to be

identified. DP is, in fact, a much more general technique that can be applied to a wide range of applications, and which has been popularized especially by the work of Bellman (1957). The number of choices at each stage is not restricted to three, as in the example given in Figure 8.3. Nor is it necessary in speech recognition applications to assume that the best path should include all frames of both patterns. If the properties of the speech only change slowly compared with the frame interval, it is permissible to skip occasional frames, so achieving timescale compression of the pattern. A particularly useful alternative version of the algorithm is asymmetrical, in that vertical paths are not permitted. The steps have a slope of zero (horizontal), one (diagonal), or two (which skips one frame in the template). Each input frame then makes just one contribution to the total distance, so it is not appropriate to double the distance contribution for diagonal paths. Many other variants of the algorithm have been proposed, including one that allows average slopes of 0.5, 1 and 2, in which the 0.5 is achieved by preventing a horizontal step if the previous step was horizontal. Provided the details of the formula are sensibly chosen, all of these algorithms can work well. In a practical implementation computational convenience may be the reason for choosing one in preference to another.

## 8.8 SCORE PRUNING

Although DP algorithms provide a great computational saving compared with exhaustive search of all possible paths, the remaining computation can be substantial, particularly if each incoming word has to be compared with a large number of candidates for matching. Any saving in computation that does not affect the accuracy of the recognition result is therefore desirable. One possible computational saving is to exploit the fact that, in the calculations for any column in Figure 8.3, it is very unlikely that the best path for a correctly matching word will pass through any points for which the cumulative distance, $D(i, j)$, is much in excess of the lowest value in that column. The saving can be achieved by not allowing paths from relatively badly scoring points to propagate further. (This process is sometimes known as **pruning** because the growing paths are like branches of a tree.) There will then only be a small subset of possible paths considered, usually lying on either side of the best path. If this economy is applied it can no longer be guaranteed that the DP algorithm will find the best-scoring path. However, with a value of score-pruning threshold that reduces the average amount of computation by a factor of 5–10 the right path will almost always be obtained if the words are fairly similar. The only circumstances where this amount of pruning is likely to prevent the optimum path from being obtained will be if the words are actually different, when the resultant over-estimate of total distance would not cause any error in recognition.

Figures 8.4(a), 8.5 and 8.6 show DP paths using the symmetrical algorithm for the words illustrated in Figures 8.1 and 8.2. Figure 8.4(b) illustrates the asymmetrical algorithm for comparison, with slopes of 0, 1 and 2. In Figure 8.4 there is no time-distortion penalty, and Figure 8.5 with a small distortion penalty shows a much more plausible matching of the two timescales. The score pruning used in these figures illustrates the fact that there are low differences in cumulative

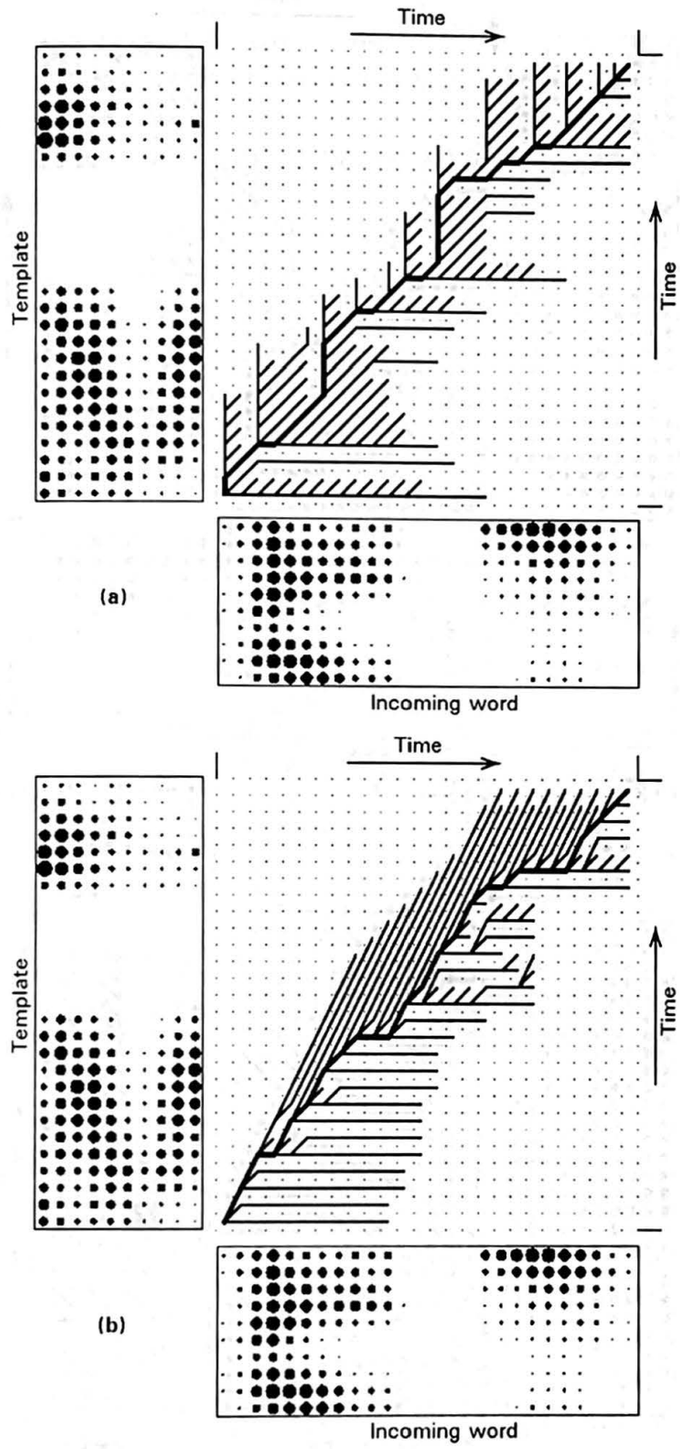



**Figure 8.4** **(a)** DP alignment between two examples of the word "eight", with no timescale distortion penalty but with score pruning. The optimum path, obtained by tracing back from the top right-hand corner, is shown by the thick line. **(b)** Match between the same words as in (a), but using an asymmetric algorithm with slopes of 0,1 and 2.

distance only along a narrow band around the optimum path. When time alignment is attempted between dissimilar words, as in Figure 8.6, a very irregular path is obtained, with a poor score. Score pruning was not used in this illustration, because any path to the end of the word would then have been seriously sub-optimal.

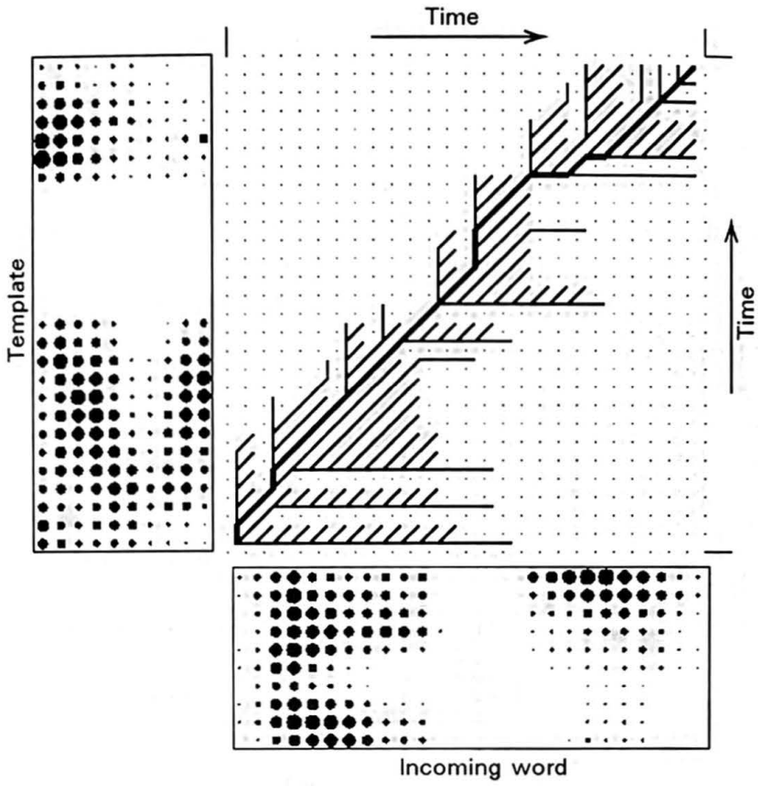



**Figure 8.5** As for Figure 8.4(a), but with a small timescale distortion penalty.

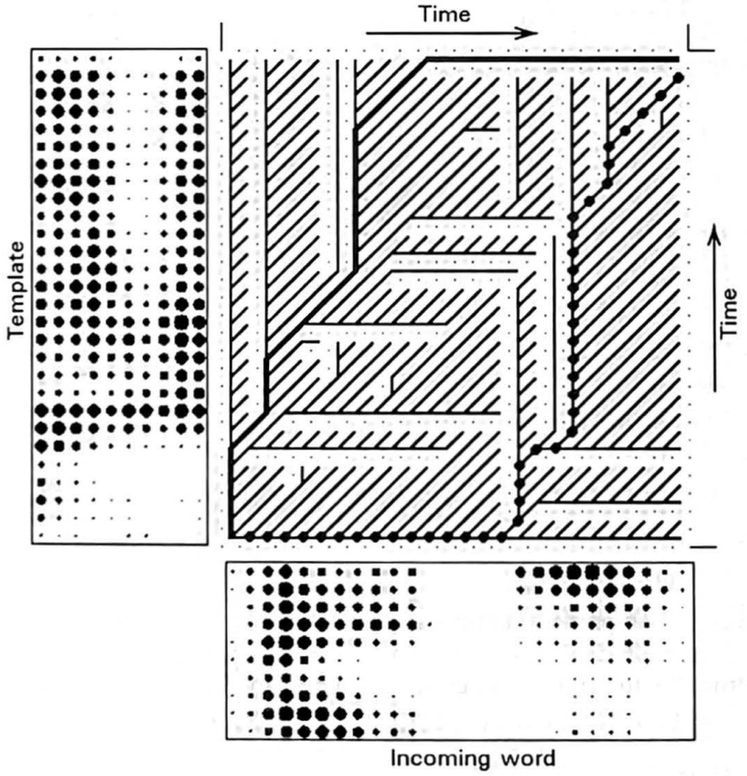



**Figure 8.6** The result of trying to align two dissimilar words ("three" and "eight") within the same DP algorithm as was used for Figure 8.5. The score pruning was removed from this illustration, because any path to the end of the word would then have been seriously sub-optimal. It can be seen that if the last frame had been removed from the template, the path would have been completely different, as marked by blobs.

## 8.9 ALLOWING FOR END-POINT ERRORS

If an attempt is made to match two intrinsically similar words when one has its specified end point significantly in error, the best-matching path ought to align all the frames of the two words that really do correspond. Such a path implies that the extra frames of the longer word will all be lumped together at one end, as illustrated in Figure 8.7. As this extreme timescale compression is not a result of a genuine difference between the words, it may be better not to have any timescale distortion penalty for frames at the ends of the patterns, and in some versions of the algorithm it may be desirable not to include the values of $d(i, j)$ for the very distorted ends of the path. If the chosen DP algorithm disallows either horizontal steps or vertical steps, correct matching of words with serious end-point errors will not be possible, and so it is probably better to remove the path slope constraints for the end frames.

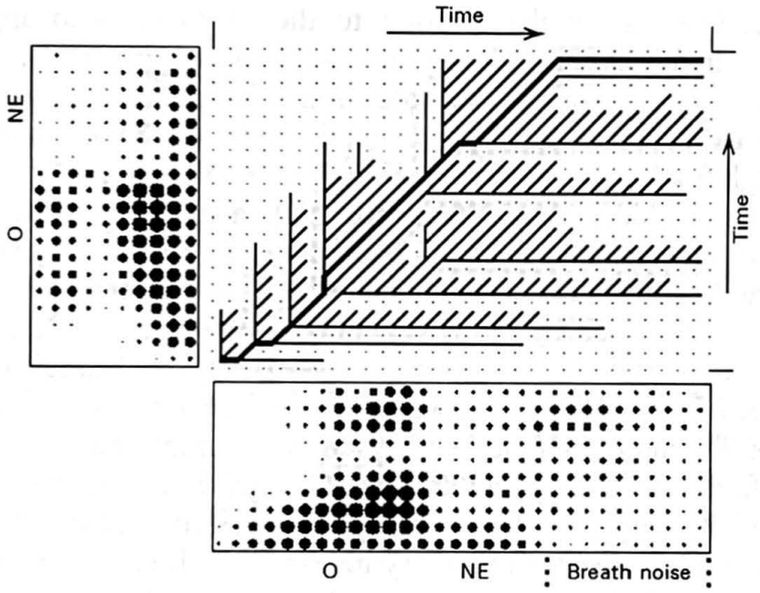



**Figure 8.7** An example of the word "one" followed by breath noise, being aligned with a "one" template. A timescale distortion penalty was used except for the beginning and end frames.

## 8.10 DYNAMIC PROGRAMMING FOR CONNECTED WORDS

Up to now we have assumed that the words to be matched have been spoken in isolation, and that their beginnings and ends have therefore already been identified (although perhaps with difficulty). When words are spoken in a normal connected fashion, recognition is much more difficult because it is generally not possible to determine where one word ends and the next one starts independently of identifying what the words are. For example, in the sequence "six teenagers" it would be difficult to be sure that the first word was "six" rather than "sixteen" until the last syllable of the phrase had been spoken, and "sixty" might also have been possible before the [n] occurred. In some cases, such as the "grade A" example given in Chapter 1, a genuine ambiguity may remain, but for most tasks any ambiguities are resolved when at most two or three syllables have followed a word boundary.

There is another problem with connected speech as a result of co-articulation between adjacent words. It is not possible even to claim the existence of a clear

point where one word stops and the next one starts. However, it is mainly the ends of words that are affected and, apart from a likely speeding up of the timescale, words in a carefully spoken connected sequence do not normally differ greatly from their isolated counterparts except near the ends. In matching connected sequences of words for which separate templates are already available one might thus define the best-matching word sequence to be given by the sequence of templates which, when joined end to end, offers the best match to the input. It is of course assumed that the optimum time alignment is used for the sequence, as with DP for isolated words. Although this model of connected speech totally ignores co-articulation, it has been successfully used in many connected-word speech recognizers.

As with the isolated-word time-alignment process, there seems to be a potentially explosive increase in computation, as every frame must be considered as a possible boundary between words. When each frame is considered as an end point for one word, all other permitted words in the vocabulary have to be considered as possible starters. Once again the solution to the problem is to apply dynamic programming, but in this case the algorithm is applied to word sequences as well as to frame sequences within words. A few algorithms have been developed to extend the isolated-word DP method to work economically across word boundaries. One of the most straightforward and widely used is described below.

In Figure 8.8 consider a point that represents a match between frame $i$ of a multi-word input utterance and frame $j$ of template number $k$. Let the cumulative distance from the beginning of the utterance along the best-matching sequence of complete templates followed by the first $j$ frames of template $k$ be $D(i, j, k)$. The best path through template $k$ can be found by exactly the same process as for isolated-word recognition. However, in contrast to the isolated-word case, it is not known where on the input utterance the match with template $k$ should finish, and for every input frame any valid path that reaches the end of template $k$ could join to the beginning of the path through another template, representing the next word. Thus, for each input frame $i$, it is necessary to consider all templates that may have just ended in order to find which one has the lowest cumulative score so far. This score is then used in the cumulative distance at the start of any new template, $m$:

$$D(i,1,m) = \min_{\text{over } k}\left[D(i-1, L(k), k)\right] + d(i,1,m)\,, \tag{8.4}$$

where $L(k)$ is the length of template $k$. The use of $i-1$ in Equation (8.4) implies that moving from the last frame of one template to the first frame of another always involves advancing one frame on the input (i.e. in effect only allowing diagonal paths between templates). This restriction is necessary, because the scores for the ends of all other templates may not yet be available for input frame $i$ when the path decision has to be made. A horizontal path from within template $m$ could have been included in Equation (8.4), but has been omitted merely to simplify the explanation. A timescale distortion penalty has not been included for the same reason.

In the same way as for isolated words, the process can be started off at the beginning of an utterance because all values of $D(0, L(k), k)$ will be zero. At the end of an utterance the template that gives the lowest cumulative distance is assumed to represent the final word of the sequence, but its identity gives no indication of the templates that preceded it. These can only be determined by storing pointers to the preceding templates of each path as it evolves, and then tracing back when the final point is reached. It is also possible to recover the positions in the input sequence

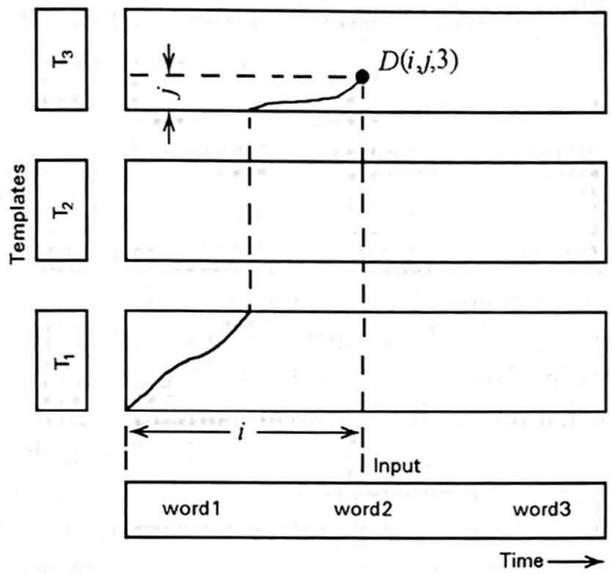



**Figure 8.8** Diagram indicating the best-matching path from the beginning of an utterance to the $j^{th}$ frame of template $T_3$ and the $i^{th}$ frame of the input. In the example shown $i$ is in the middle of the second word of the input, so the best path includes one complete template ($T_1$) and a part of $T_3$. The cumulative distance at this point is denoted by $D(i, j, 3)$, or in general by $D(i, j, k)$ for the $k^{th}$ template.

where the templates of the matching sequence start and finish, so segmenting the utterance into separate words. Thus we solve the segmentation problem by delaying the decisions until we have seen the whole utterance and decided on the words.

The process as described so far assumes that any utterance can be modelled completely by a sequence of word templates. In practice a speaker may pause between words, so giving a period of silence (or background noise) in the middle of an utterance. The same algorithm can still be used for this situation by also storing a template for a short period of silence, and allowing this **silence template** to be included between appropriate pairs of valid words. If the silence template is also allowed to be chosen at the start or end of the sequence, the problem of end-point detection is greatly eased. It is only necessary to choose a threshold that will never be exceeded by background noise, and after the utterance has been detected, to extend it by several frames at each end to be sure that any low-intensity parts of the words are not omitted. Any additional frames before or after the utterance should then be well modelled by a sequence of one or more silence templates.

When a sequence of words is being spoken, unintentional extraneous noises (such as grunts, coughs and lip smacks) will also often be included between words. In an isolated-word recognizer these noises will not match well to any of the templates, and can be rejected on this basis. In a connected-word algorithm there is no provision for not matching any part of the sequence. However, the rejection of these unintentional insertions can be arranged by having a special template, often called a **wildcard template**, that bypasses the usual distance calculation and is deemed to match with any frame of the input to give a standard value of distance. This value is chosen to be greater than would be expected for corresponding frames of equivalent words, but less than should occur when trying to match quite different sounds. The wildcard will then provide the best score when attempting to match spurious sounds and words not in the stored template vocabulary, but should not normally be chosen in preference to any of the well-matched words in the input.

## 8.11 CONTINUOUS SPEECH RECOGNITION

In the connected-word algorithm just described, start and finish points of the input utterance must at least be approximately determined. However it is not generally necessary to wait until the end of an utterance before identifying the early words. Even before the end, one can trace back along all current paths through the tree that represents the candidates for the template sequence. This tree will always involve additional branching as time goes forward, but the ends of many of the 'twigs' will not represent a low enough cumulative distance to successfully compete with other twigs as starting points for further branching, and so paths along these twigs will be abandoned. It follows that tracing back from all currently active twigs will normally involve coalescence of all paths into a single 'trunk', which therefore represents a uniquely defined sequence of templates (see Figure 8.9). The results up to the first point of splitting of active paths can therefore be output from the machine, after which the back-pointers identifying that part of the path are no longer needed, nor are those representing abandoned paths. The memory used for storing them can therefore be released for re-use with new parts of the input signal.

The recognizer described above can evidently operate continuously, with a single pass through the input data, outputting its results always a few templates behind the current best match. Silence templates are used to match the signal when the speaker pauses, and wildcards are used for extraneous noises or inadmissible words. The time lag for output is determined entirely by the need to resolve ambiguity. When two alternative sequences of connected words both match the input well, but with different boundary points (e.g. "grey day" and "grade A") it is necessary to reach the end of the ambiguous sequence before a decision can be reached on any part of it. (In the example just given, the decision might even then

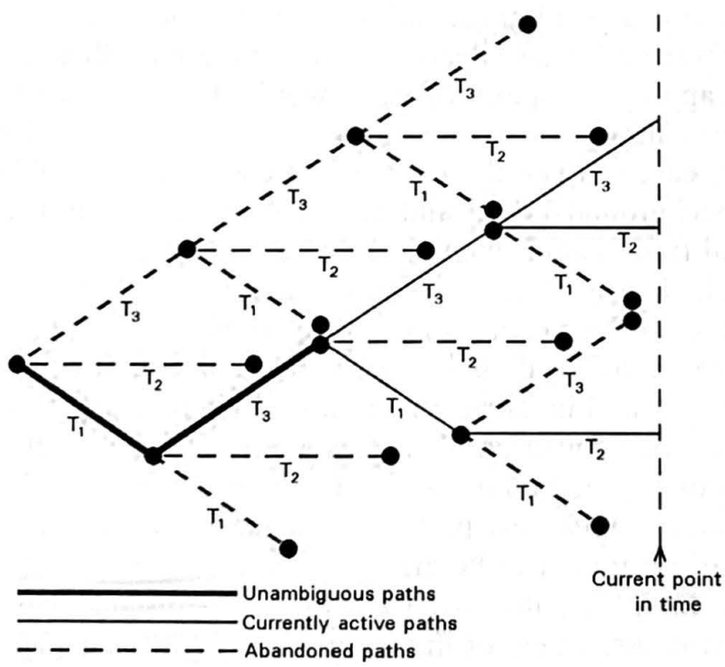



**Figure 8.9** Trace-back through a word decision tree to identify unambiguous paths for a three-word vocabulary continuous recognizer. Paths are abandoned when the cumulative distances of all routes to the ends of the corresponding templates are greater than for paths to the ends of different template sequences at the same points in the input. Template sequences still being considered are $T_1$–$T_3$–$T_3$–$T_3$, $T_1$–$T_3$–$T_3$–$T_2$ and $T_1$–$T_3$–$T_1$–$T_2$. Thus $T_2$ is being scored separately for two preceding sequences.

be wrong because of inherent ambiguity in the acoustic signal.) On the other hand, if the input matches very badly to all except one of the permitted words, all paths not including that word will be abandoned as soon as the word has finished. In fact, if score pruning is used to cause poor paths to be abandoned early, the path in such a case may be uniquely determined even at a matching point within the word. There is plenty of evidence that human listeners also often decide on the identity of a long word before it is complete if its beginning is sufficiently distinctive.

## 8.12 SYNTACTIC CONSTRAINTS

The rules of grammar often prevent certain sequences of words from occurring in human language, and these rules apply to particular syntactic classes, such as nouns, verbs, etc. In the more artificial circumstances in which speech recognizers are often used, the tasks can sometimes be arranged to apply much more severe constraints on which words are permitted to follow each other. Although applying such constraints requires more care in designing the application of the recognizer, it usually offers a substantial gain in recognition accuracy because there are then fewer potentially confusable words to be compared. The reduction in the number of templates that need to be matched at any point also leads to a computational saving.

## 8.13 TRAINING A WHOLE-WORD RECOGNIZER

In all the algorithms described in this chapter it is assumed that suitable templates for the words of the vocabulary are available in the machine. Usually the templates are made from speech of the intended user, and thus a **training session** is needed for enrolment of each new user, who is required to speak examples of all the vocabulary words. If the same user regularly uses the machine, the templates can be stored in some back-up memory and re-loaded prior to each use of the system. For isolated-word recognizers the only technical problem with training is end-point detection. If the templates are stored with incorrect end points the error will affect recognition of every subsequent occurrence of the faulty word. Some systems have tried to ensure more reliable templates by time aligning a few examples of each word and averaging the measurements in corresponding frames. This technique gives some protection against occasional end-point errors, because such words would then give a poor match in this alignment process and so could be rejected.

If a connected-word recognition algorithm is available, each template can be segmented from the surrounding silence by means of a special training syntax that only allows silence and wildcard templates. The new template candidate will obviously not match the silence, so it will be allocated to the wildcard. The boundaries of the wildcard match can then be taken as end points of the template.

In acquiring templates for connected-word recognition, more realistic training examples can be obtained if connected words are used for the training. Again the recognition algorithm can be used to determine the template end points, but the syntax would specify the preceding and following words as existing templates, with just the new word to be captured represented by a wildcard between them. Provided the surrounding words can be chosen to give clear acoustic boundaries where they

join to the new word, the segmentation will then be fairly accurate. This process is often called **embedded training**. More powerful embedded training procedures for use with statistical recognizers are discussed in Chapters 9 and 11.

## CHAPTER 8 SUMMARY

- Most early successful speech recognition machines worked by pattern matching on whole words. Acoustic analysis, for example by a bank of band-pass filters, describes the speech as a sequence of feature vectors, which can be compared with stored templates for all the words in the vocabulary using a suitable distance metric. Matching is improved if speech level is coded logarithmically and level variations are normalized.
- Two major problems in isolated-word recognition are end-point detection and timescale variation. The timescale problem can be overcome by dynamic programming (DP) to find the best way to align the timescales of the incoming word and each template (known as dynamic time warping). Performance is improved by using penalties for timescale distortion. Score pruning, which abandons alignment paths that are scoring badly, can save a lot of computation.
- DP can be extended to deal with sequences of connected words, which has the added advantage of solving the end-point detection problem. DP can also operate continuously, outputting words a second or two after they have been spoken. A wildcard template can be provided to cope with extraneous noises and words that are not in the vocabulary.
- A syntax is often provided to prevent illegal sequences of words from being recognized. This method increases accuracy and reduces the computation.

## CHAPTER 8 EXERCISES

**E8.1** Give examples of factors which cause acoustic differences between utterances of the same word. Why does simple pattern matching work reasonably well in spite of this variability?

**E8.2** What factors influence the choice of bandwidth for filter-bank analysis?

**E8.3** What are the reasons in favour of logarithmic representation of power in filter-bank analysis? What difficulties can arise due to the logarithmic scale?

**E8.4** Explain the principles behind dynamic time warping, with a simple diagram.

**E8.5** Describe the special precautions which are necessary when using the symmetrical DTW algorithm for isolated-word recognition.

**E8.6** How can a DTW isolated-word recognizer be made more tolerant of end-point errors?

**E8.7** How can a connected-word recognizer be used to segment a speech signal into individual words?

**E8.8** What extra processes are needed to turn a connected-word recognizer into a continuous recognizer?

**E8.9** Describe a training technique suitable for connected-word recognizers.

# CHAPTER 9

# Introduction to Stochastic Modelling

## 9.1 FEATURE VARIABILITY IN PATTERN MATCHING

The recognition methods described in the previous chapter exploit the fact that repeated utterances of the same word normally have more similar acoustic patterns than utterances of different words. However, it is to be expected that some parts of a pattern may vary more from occurrence to occurrence than do other parts. In the case of connected words, the ends of the template representing each word are likely to have a very variable degree of match, depending on the amount that the input pattern is modified by co-articulation with adjacent words. There is also no reason to assume that the individual features of a feature vector representing a particular phonetic event are of equal consistency. In fact, it may well occur that the value of a feature could be quite critical at a particular position in one word, while being very variable and therefore not significant in some part of a different word.

Timescale variability has already been discussed in Chapter 8. It must always be desirable to have some penalty for timescale distortion, as durations of speech sounds are not normally wildly different between different occurrences of the same word. However, there is no reason to assume that the time distortion penalty should be constant for all parts of all words. For example, it is known that long vowels can vary in length a lot, whereas most spectral transitions associated with consonants change in duration only comparatively slightly.

From the above discussion it can be seen that the ability of a recognizer to distinguish between words is likely to be improved if the variability of the patterns can be taken into account. We should not penalize the matching of a particular word if the parts that match badly are parts which are known to vary extensively from utterance to utterance. To use information about variability properly we need to have some way of collecting statistics which represent the variability of the word patterns, and a way of using this variability in the pattern-matching process.

The basic pattern-matching techniques using DTW as described in Chapter 8 started to be applied to ASR in the late 1960s and became popular during the 1970s. However, the application of statistical techniques to this problem was also starting to be explored during the 1970s, with early publications being made independently by Baker (1975) working at Carnegie-Mellon University (CMU) and by Jelinek (1976) from IBM. These more powerful techniques for representing variability have gradually taken over from simple pattern matching. In the period since the early publications by Baker and by Jelinek, there has been considerable research to refine the use of statistical methods for speech recognition, and some variant of these methods is now almost universally adopted in current systems.

This chapter provides an introduction to statistical methods for ASR. In order to accommodate pattern variability, these methods use a rather different way of

defining the degree of fit between a word and some speech data, as an alternative to the 'cumulative distance' used in Chapter 8. This measure of degree of fit is based on the notion of **probability**, and the basic theory is explained in this chapter. For simplicity in introducing the concepts, the discussion in this chapter will continue to concentrate on words as the recognition unit. In practice, the majority of current recognition systems represent words as a sequence of **sub-word units**, but the underlying theory is not affected by the choice of unit. The use of sub-word units for recognition, together with other developments and elaborations of the basic statistical method will be explained in later chapters. In the following explanation, some elementary knowledge of statistics and probability theory is assumed, but only at a level which could easily be obtained by referring to a good introductory textbook (see Chapter 17 for some references).

## 9.2 INTRODUCTION TO HIDDEN MARKOV MODELS

Up to now we have considered choosing the best matching word by finding the template which gives the minimum cumulative 'distance' along the optimum matching path. An alternative approach is, for each possible word, to postulate some device, or **model**, which can generate patterns of features to represent the word. Every time the model for a particular word is activated, it will produce a set of feature vectors that represents an example of the word, and if the model is a good one, the statistics of a very large number of such sets of feature vectors will be similar to the statistics measured for human utterances of the word. The best matching word in a recognition task can be defined as the one whose model is most likely to produce the observed sequence of feature vectors. What we have to calculate for each word is thus not a 'distance' from a template, but the *a posteriori* probability that its model could have produced the observed set of feature vectors. We do not actually have to make the model produce the feature vectors, but we use the known properties of each model for the probability calculations. We will assume for the moment that the words are spoken in an 'isolated' manner, so that we know where the start and end of each word are, and the task is simply to identify the word. Extensions to sequences of words will be considered in Section 9.11.

We wish to calculate the *a posteriori* probability, $P(w|\boldsymbol{Y})$, of a particular word, $w$, having been uttered during the generation of a set of feature observations, $\boldsymbol{Y}$. We can use the model for $w$ to calculate $P(\boldsymbol{Y}|w)$, which is the probability of $\boldsymbol{Y}$ conditioned on word $w$ (sometimes referred to as the **likelihood** of $w$). To obtain $P(w|\boldsymbol{Y})$, however, we must also include the *a priori* probability of word $w$ having been spoken. The relationship between these probabilities is given by Bayes' rule:

$$P(w|\boldsymbol{Y}) = \frac{P(\boldsymbol{Y}|w)P(w)}{P(\boldsymbol{Y})}. \tag{9.1}$$

This equation states that the probability of the word given the observations is equal to the probability of the observations given the word, multiplied by the probability of the word (irrespective of the observations), and divided by the probability of the observations. The probability, $P(\boldsymbol{Y})$, of a particular set of feature observations, $\boldsymbol{Y}$, does not depend on which word is being considered as a possible match, and therefore only acts as a scaling factor on the probabilities. Hence, if the

goal is to find the word $w$ which maximizes $P(w|\boldsymbol{Y})$, the $P(\boldsymbol{Y})$ term can be ignored, because it does not affect the choice of word. If for the particular application all permitted words are equally likely, then the $P(w)$ term can also be ignored, so we merely have to choose the word model that maximizes the probability, $P(\boldsymbol{Y}|w)$, of producing the observed feature set, $\boldsymbol{Y}$. In practice for all but the simplest speech recognizers the probability of any particular word occurring will depend on many factors, and for large vocabularies it will depend on the statistics of word occurrence in the language. This aspect will be ignored in the current chapter, but will be considered further in Chapter 12.

The way we have already represented words as sequences of template frames gives us a starting point for the form of a possible model. Let the model for any word be capable of being in one of a sequence of **states**, each of which can be associated with one or more frames of the input. In general the model moves from one state to another at regular intervals of time equal to the frame interval of the acoustic analysis. However, we know that words can vary in timescale. In the asymmetrical DP algorithm mentioned in Chapter 8 (Figure 8.4(b), showing slopes of 0, 1 and 2) the timescale variability is achieved by repeating or skipping frames of the template. In our model this possibility can be represented in the sequence of states by allowing the model to stay in the same state for successive frame times, or to bypass the next state in the sequence. The form of this simple model is shown in Figure 9.1. In fact, if a word template has a sequence of very similar frames, such as might occur in a long vowel, it is permissible to reduce the number of states in the model by allowing it to stay in the same state for several successive frames.

The mathematics associated with a model such as the one shown in Figure 9.1 can be made more tractable by making certain simplifying assumptions. To be more specific, it is assumed that the output of the model is a **stochastic process** (i.e. its operation is governed completely by a set of probabilities), and that the probabilities of all its alternative actions at any time $t$ depend only on the state it is in at that time, and not on the value of $t$. The current output of the model therefore depends on the identity of the current state, but is otherwise independent of the sequence of previous states that it has passed through to reach that state. Hence the model's operation is a **first-order Markov process**, and the sequence of states is a **first-order Markov chain**. Although the model structure shown in Figure 9.1 is quite appropriate for describing words that vary in timescale, the equations that represent the model's behaviour have exactly the same form in the more general case where transitions are allowed between all possible pairs of states.

At every frame time the model is able to change state, and will do so randomly in a way determined by a set of **transition probabilities** associated with the state it is currently in. By definition, the probabilities of all transitions from a

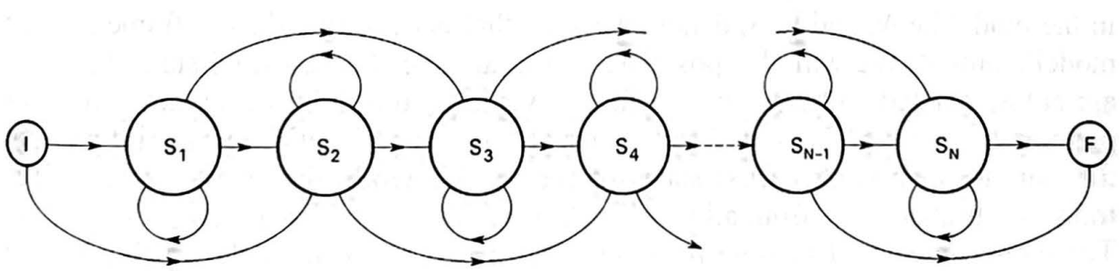



**Figure 9.1.** State transitions for a simple word model, from an initial state, I, to a final state, F.

state at any frame time must sum to 1, but the sum includes the probability of a transition that re-enters the same state. When the model is activated a sequence of feature vectors is emitted, in the same form as might be observed when a word is spoken. However, in the type of model considered here, observing the feature vectors does not completely determine what the state sequence is. In addition to its transition probabilities, each state also has associated with it a **probability density function** (**p.d.f.**) for the feature vectors. Each p.d.f. can be used to calculate the probability that any particular set of feature values could be emitted when the model is in the associated state. This probability is usually known as the **emission probability**. The actual values of the observed features are, therefore, probabilistic functions of the states, and the states themselves are hidden from the observer. For this reason this type of model is called a **hidden Markov model** (**HMM**).

The emission p.d.f. for a state may be represented as a discrete distribution, with a probability specified separately for each possible feature vector. Alternatively, it is possible to use a parameterized continuous distribution, in which feature vector probabilities are defined by the parameters of the distribution. Although there are significant advantages, which will be explained in Section 9.7, in modelling feature probabilities as continuous functions, it will simplify the following explanation if we initially consider only discrete probability distributions.

## 9.3 PROBABILITY CALCULATIONS IN HIDDEN MARKOV MODELS

In order to explain the HMM probability calculations, we will need to introduce some symbolic notation to represent the different quantities which must be calculated. Notation of this type can be found in many publications on the subject of HMMs for ASR. Certain symbols have come to be conventionally associated with particular quantities, although there is still some variation in the details of the notation that is used. In choosing the notation for this book, our aims were to be consistent with what appears to be used the most often in the published literature, while also being conceptually as simple as possible.

We will start by assuming that we have already derived good estimates for the parameters of all the word models. (Parameter estimation will be discussed later in the chapter.) The recognition task is to determine the most probable word, given the observations (i.e. the word $w$ for which $P(w|\boldsymbol{Y})$ is maximized). As explained in Section 9.2, we therefore need to calculate the likelihood of each model emitting the observed sequence of features (i.e. the value of $P(\boldsymbol{Y}|w)$ for each word $w$).

Considering a single model, an output representing a whole word arises from the model going through a sequence of states, equal in length to the number of observed feature vectors, $T$, that represents the word. Let the total number of states in the model be $N$, and let $s_t$ denote the state that is occupied during frame $t$ of the model's output. We will also postulate an initial state, $I$ and a final state, $F$, which are not associated with any emitted feature vector and only have a restricted set of possible transitions. The initial state is used to specify transition probabilities from the start to all permitted first states of the model, while the final state provides transition probabilities from all possible last emitting states to the end of the word. The model must start in state $I$ and end in state $F$, so in total the model will go through a sequence of $T+2$ states to generate $T$ observations. The use of non-

emitting initial and final states provides a convenient method for modelling the fact that some states are more likely than others to be associated with the first and the last frame of the word respectively[1]. These compulsory special states will also be useful in later discussions requiring *sequences* of models.

The most widely used notation for the probability of a transition from state $i$ to state $j$ is $a_{ij}$. The emission probability of state $j$ generating an observed feature vector $\boldsymbol{y}_t$ is usually denoted $b_j(\boldsymbol{y}_t)$.

We need to compute the probability of a given model producing the observed sequence of feature vectors, $\boldsymbol{y}_1$ to $\boldsymbol{y}_T$. We know that this sequence of observations must have been generated by a state sequence of length $T$ (plus the special initial and final states) but, because the model is hidden, we do not know the identities of the states. Hence we need to consider all possible state sequences of length $T$. The probability of the model generating the observations can then be obtained by finding the **joint probability** of the observations and any one state sequence, and summing this quantity over all possible state sequences of the correct length:

$$P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \sum_{\substack{\text{over all possible} \\ \text{state sequences} \\ \text{of length } T}} P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T, s_1, s_2, \cdots, s_T)$$

$$= \sum_{\substack{\text{over all possible} \\ \text{state sequences} \\ \text{of length } T}} P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T, | s_1, s_2, \cdots, s_T) P(s_1, s_2, \cdots, s_T), \tag{9.2}$$

where, for notational convenience, in the equations we are omitting the dependence of all the probabilities on the identity of the model.

Now the probability of any particular state sequence is given by the product of the transition probabilities:

$$P(s_1, s_2, \cdots, s_T) = a_{Is_1} \left( \prod_{t=1}^{T-1} a_{s_t s_{t+1}} \right) a_{s_T F}, \tag{9.3}$$

where $a_{s_t s_{t+1}}$ is the probability of a transition from the state occupied at frame $t$ to the state at frame $t+1$; $a_{Is_1}$ and $a_{s_T F}$ similarly define the transition probabilities from the initial state $I$ and to the final state $F$. If we assume that the feature vectors are generated independently for each state, the probability of the observations given a particular state sequence of duration $T$ is the product of the individual emission probabilities for the specified states:

$$P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T \mid s_1, s_2, \cdots, s_T) = \prod_{t=1}^{T} b_{s_t}(\boldsymbol{y}_t). \tag{9.4}$$

---

[1] Some published descriptions of HMM theory do not include special initial and final states. Initial conditions are sometimes accommodated by a vector of probabilities for starting in each of the states (e.g. Levinson *et al.*, 1983), which has the same effect as the special initial state used here. For the last frame of the word, approaches include allowing the model to end in any state (e.g. Levinson *et al.*, 1983) or enforcing special conditions to only allow the model to end in certain states. The treatment of the first and last frames does not alter the basic form of the probability calculations, but it may affect the details of the expressions associated with the start and end of an utterance.

Thus the probability of the model emitting the complete observation sequence is:

$$P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \sum_{\substack{\text{over all possible} \\ \text{state sequences} \\ \text{of length } T}} a_{Is_1} \left( \prod_{t=1}^{T-1} b_{s_t}(\boldsymbol{y}_t) a_{s_t s_{t+1}} \right) b_{s_T}(\boldsymbol{y}_T) a_{s_T F} \,. \tag{9.5}$$

Unless the model has a small number of states and $T$ is small, there will be an astronomical number of possible state sequences, and it is completely impractical to make the calculations of Equation (9.5) directly for all sequences. One can, however, compute the probability indirectly by using a recurrence relationship. We will use the symbol $\alpha_j(t)$ to be the probability[2] of the model having produced the first $t$ observed feature vectors and being in state $j$ for frame $t$. The recurrence can be computed in terms of the values of $\alpha_i(t-1)$ for all possible previous states, $i$.

$$\alpha_j(t) = P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_t, s_t = j) \tag{9.6}$$

$$= \left( \sum_{i=1}^{N} \alpha_i(t-1) a_{ij} \right) b_j(\boldsymbol{y}_t) \quad \text{for } 1 < t \leq T \tag{9.7}$$

The value of $\alpha_j(1)$, for the first frame, is the product of the transition probability $a_{Ij}$ from the initial state $I$, and the emission probability $b_j(\boldsymbol{y}_1)$.

$$\alpha_j(1) = a_{Ij} b_j(\boldsymbol{y}_1) \tag{9.8}$$

The value of $\alpha_j(T)$, for the last frame in the observation sequence, can be computed for any of the emitting states by repeated applications of Equation (9.7), starting from the result of Equation (9.8).

The total probability of the complete set of observations being produced by the model must also include the transition probabilities into the final state $F$. We will define this quantity as $\alpha_F(T)$, thus:

$$P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \alpha_F(T) = \sum_{i=1}^{N} \alpha_i(T) a_{iF} \,. \tag{9.9}$$

Equation (9.9) gives the probability of the model generating the observed data, taking into account all possible sequences of states. This quantity represents the probability of the observations given the word model (the $P(\boldsymbol{Y}|w)$ term in Equation (9.1)). Incorporating the probability of the word, $P(w)$, gives a probability that is a scaled version of $P(w|\boldsymbol{Y})$, the probability of the word having been spoken. Provided that the model is a good representation of its intended word, this probability provides a useful measure which can be compared with the probability according to alternative word models in order to identify the most probable word.

---

[2] In the literature, this probability is almost universally represented by the symbol $\alpha$. However, there is some variation in the way in which the $\alpha$ symbol is annotated to indicate dependence on state and time. In particular, several authors (e.g. Rabiner and Juang (1993)) have used $\alpha_t(j)$, whereas we have chosen $\alpha_j(t)$ (as used by Knill and Young (1997) for example). The same variation applies to the quantities $\beta$, $\gamma$ and $\xi$, which will be introduced later. The differences are only notational and do not affect the meaning of the expressions, but when reading the literature it is important to be aware that such differences exist.

## 9.4 THE VITERBI ALGORITHM

The probability of the observations, given the model, is made up of contributions from a very large number of alternative state sequences. However, the probability distributions associated with the states will be such that the probability of the observed feature vectors having been produced by many of the state sequences will be microscopically small compared with the probabilities associated with other state sequences. One option is to ignore all but the single most probable state sequence. Equation (9.2) can be modified accordingly to give the probability, $\hat{P}$, of the observations for this most probable state sequence:

$$\hat{P}(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \max_{\substack{\text{over all possible} \\ \text{state sequences} \\ \text{of length } T}} \left(P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T, s_1, s_2, \cdots, s_T)\right). \tag{9.10}$$

The probability associated with the most probable sequence of states can be calculated using the **Viterbi algorithm** (Viterbi, 1967), which is a dynamic programming algorithm applied to probabilities. Let us define a new probability, $\hat{\alpha}_j(t)$ as the probability of being in the $j^{\text{th}}$ state, after having emitted the first $t$ feature vectors and *having been through the most probable sequence* of $t-1$ preceding states in the process. Again we have a recurrence relation, equivalent to the one shown in Equation (9.7):

$$\hat{\alpha}_j(t) = \max_{\text{over } i} \left(\hat{\alpha}_i(t-1)a_{ij}\right) b_j(\boldsymbol{y}_t) \quad \text{for } 1 < t \leq T \ . \tag{9.11}$$

The conditions for the first state are the same as for the total probability, which was given in Equation (9.8):

$$\hat{\alpha}_j(1) = \alpha_j(1) = a_{1j} b_j(\boldsymbol{y}_1) \ . \tag{9.12}$$

Successive applications of Equation (9.11) will eventually yield the values for $\hat{\alpha}_j(T)$. Defining $\hat{\alpha}_F(T)$ as the probability of the full set of observations being given by the most probable sequence of states, its value is given by:

$$\hat{P}(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \hat{\alpha}_F(T) = \max_{\text{over } i} \left(\hat{\alpha}_i(T)a_{iF}\right). \tag{9.13}$$

The difference between the total probability and the probability given by the Viterbi algorithm depends on the magnitude of the contribution of the 'best' state sequence to the total probability summed over all possible sequences. If the feature-vector p.d.f.s of all states are substantially different from each other, the probability of the observations being produced by the best sequence might not be appreciably less than the total probability including all possible sequences. The difference between the total probability and the probability for the best sequence will, however, be larger if the best path includes several consecutive frames shared between a group of two or more states which have very similar p.d.f.s for the feature vectors. Then the probability of generating the observed feature vectors would be almost independent of how the model distributed its time between the states in this group. The total probability, which is the sum over all possible allocations of frames to states, could then be several times the probability for the

best sequence. This point will be considered again in Section 9.14. However, the design of models used in current recognizers is such that sequences of states with similar emission p.d.f.s generally do not occur. As a consequence, in spite of the theoretical disadvantage of ignoring all but the best path, in practice the differences in performance between the two methods are usually small. Some variant of the Viterbi algorithm is therefore usually adopted for decoding in practical speech recognizers, as using only the best path requires less computation. (There can also be considerable advantages for implementation, as will be discussed in Section 9.12.)

## 9.5 PARAMETER ESTIMATION FOR HIDDEN MARKOV MODELS

So far, we have considered the probability calculations required for recognition. We have assumed that the parameters of the models, i.e. the transition probabilities and emission p.d.f.s for all the states, are already set to their optimum values for modelling the statistics of a very large number of human utterances of all the words that are to be recognized. In the discussion which follows we will consider the problem of deriving suitable values for these parameters from a quantity of training data. We will assume for the moment that the body of training data is of sufficient size to represent the statistics of the population of possible utterances, and that we have sufficient computation available to perform the necessary operations.

The training problem can be formulated as one of determining the values of the HMM parameters in order to maximize the probability of the training data being generated by the models ($P(\boldsymbol{Y}|w)$ in Equation (9.1)). Because this conditional probability of the observations $\boldsymbol{Y}$ given word $w$ is known as the 'likelihood' of the word $w$, the training criterion that maximizes this probability is referred to as **maximum likelihood** (other training criteria will be considered in Chapter 11). If we knew which frames of training data corresponded to which model states, then it would be straightforward to calculate a maximum-likelihood estimate of the probabilities associated with each state. The transition probabilities could be calculated from the statistics of the state sequences, and the emission probabilities from the statistics of the feature vectors associated with each state. However, the 'hidden' nature of the HMM states is such that the allocation of frames to states cannot be known. Therefore, although various heuristic methods can be formulated for analysing the training data to give rough estimates of suitable model parameters, there is no method of calculating the optimum values directly.

If, however, one has a set of rough estimates for all the parameters, it is possible to use their values in a procedure to compute new estimates for each parameter. This algorithm was developed by Baum and colleagues and published in a series of papers in the late 1960s and early 1970s. It has been proved by Baum (1972) that new parameter estimates derived in this way always produce a model that is at least as good as the old one in representing the data, and in general the new estimates give an improved model. If we iterate these operations a sufficiently large number of times the model will converge to a locally optimum solution. Unfortunately, it is generally believed that the number of possible local optima is so vast that the chance of finding the global optimum is negligible. However, it is unlikely that the global optimum would in practice be much better than a good local optimum, derived after initialization with suitable starting estimates for the models.

Baum's algorithm is an example of a general method which has come to be known as the **expectation-maximization (EM) algorithm** (Dempster *et al.*, 1977). The EM algorithm is applicable to a variety of situations in which the task is to estimate model parameters when the observable data are 'incomplete', in the sense that some information (in this case the state sequence) is missing.

The detailed mathematical proofs associated with the derivation of the re-estimation formulae for HMMs are beyond the scope of this book, although Chapter 17 gives some references. In the current chapter, we will describe the re-estimation calculations and give some intuitive explanation. The basic idea is to use some existing estimates for the model parameters to calculate the probability of being in each state at every frame time, given these current estimates of the model parameters *and* the training data. The probabilities of occupying the states can then be taken into account when gathering the statistics of state sequences and of feature vectors associated with the states, in order to obtain new estimates for the transition probabilities and for the emission probabilities respectively. In the re-estimation equations we will use a bar above the symbol to represent a re-estimated value, and the same symbol without the bar to indicate its previous value.

### 9.5.1 Forward and backward probabilities

Suppose for the moment that we have just a single example of a word, and that this example comprises the sequence of feature vectors $\boldsymbol{y}_1$ to $\boldsymbol{y}_T$. Also, assume that the word has been spoken in isolation and we know that $\boldsymbol{y}_1$ corresponds to the first frame of the word, with $\boldsymbol{y}_T$ representing the last frame. In Equation (9.7) we showed how to compute $\alpha_j(t)$, which is the probability of the model having emitted the first $t$ observed feature vectors and being in state $j$. The values of $\alpha_j(t)$ are computed for successive frames in order, going **forward** from the beginning of the utterance. When estimating parameters for state $j$, we will need to know the probability of being in the state at time $t$, while the model is in the process of emitting *all* the feature vectors that make up the word. For this purpose we also need to compute $\beta_j(t)$, which is defined as the **backward** probability of emitting the remaining $T-t$ observed vectors that are needed to complete the word, given that the $j^{\text{th}}$ state was occupied for frame $t$ :

$$\beta_j(t) = P(\boldsymbol{y}_{t+1}, \boldsymbol{y}_{t+2}, \cdots, \boldsymbol{y}_T \mid s_t = j) . \tag{9.14}$$

When calculating the backward probabilities, it is necessary to start applying the recurrence from the end of the word and to work backwards through the sequence of frames. Each backward probability at time $t$ is therefore derived from the backward probabilities at time $t+1$. Because the notation convention is to move from state $i$ to state $j$, it is usual to specify the recurrence relationship for the backward probabilities with the $i^{\text{th}}$ state occupied at time $t$. Thus the value of $\beta_i(t)$ is computed in terms of the values of $\beta_j(t+1)$ for all possible following states $j$ :

$$\beta_i(t) = \sum_{j=1}^{N} a_{ij} b_j(\boldsymbol{y}_{t+1}) \beta_j(t+1) \qquad \text{for } T > t \geq 1 . \tag{9.15}$$

In contrast to Equation (9.7), it will be noticed that Equation (9.15) does not include the emission probability for frame $t$. This difference in form between the definitions of $\alpha_i(t)$ and $\beta_i(t)$ is necessary because of the way we will combine these quantities in Equation (9.17).

The first application of Equation (9.15) uses the fact that the model must be in the final state, $F$, at the end of the word. At this point all features will have been emitted, so the value of $\beta_i(T)$ is just the probability of a transition from state $i$ to state $F$:

$$\beta_i(T) = a_{iF} . \tag{9.16}$$

The probability of the model emitting the full set of $T$ feature vectors and being in the $j^{\text{th}}$ state for the $t^{\text{th}}$ observed frame must be the product of the forward and backward probabilities for the given state and frame pair, thus:

$$P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T, s_t = j) = \alpha_j(t)\beta_j(t) . \tag{9.17}$$

Although it is not relevant to parameter re-estimation, it is interesting to note that, as the probability of generating the full set of feature vectors and being in state $j$ for frame $t$ is given by $\alpha_j(t)\beta_j(t)$, the probability of the observations irrespective of which state is occupied in frame $t$ must be the value of this product summed over all states. We can write this probability as:

$$P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \sum_{i=1}^{N} \alpha_i(t)\beta_i(t) \qquad \text{for any value of } t, \tag{9.18}$$

where here we use $i$ as the state index for ease of comparison with Equation (9.9). Equation (9.18) is true for any value of the frame time, $t$, and Equation (9.9) is thus just a special case for the last frame, where $t = T$ and in consequence $\beta_i(T) = a_{iF}$.

### 9.5.2 Parameter re-estimation with forward and backward probabilities

In practice when training a set of models there would be several (say $E$) examples of each word, so the total number of feature vectors available is the sum of the numbers of frames for the individual examples. The re-estimation should use all the training examples with equal weight. For this purpose it is necessary to take into account that the current model would be expected to fit some examples better than others, and we need to prevent these examples from being given more weight in the re-estimation process. The simple product $\alpha_j(t)\beta_j(t)$ does not allow for these differences, as it represents the *joint* probability of being in state $j$ at time $t$ and generating a particular set of feature vectors representing one example. In order to be able to combine these quantities for different examples, we require the *conditional* probability of occupying state $j$ given the feature vectors.

We will define a quantity $\gamma_j(t)$, which is the probability of being in state $j$ for frame $t$, given the feature vectors for one example of the word. This quantity can be derived from $\alpha_j(t)\beta_j(t)$ using Bayes' rule, and it can be seen that the result involves simply normalizing $\alpha_j(t)\beta_j(t)$ by the probability of the model generating the observations.

$$\gamma_j(t) = P(s_t = j \mid \boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T) = \frac{P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T \mid s_t = j) P(s_t = j)}{P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T)}$$

$$= \frac{P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T, s_t = j)}{P(\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T)} = \frac{\alpha_j(t)\beta_j(t)}{\alpha_F(T)} \tag{9.19}$$

The normalization by $\alpha_F(T)$ thus ensures that when there are several examples of the word, all frames of all examples will contribute equally to the re-estimation.

The probability, $b_j(\boldsymbol{k})$, of observing some particular feature vector, $\boldsymbol{k}$, when the model is in state $j$ can be derived as the probability of the model being in state $j$ and observing $\boldsymbol{k}$, divided by the probability of the model being in state $j$. In order to take into account the complete set of training examples of the word, we need to sum both the numerator and the denominator over all frames of all examples. Hence, assuming $E$ examples of the word, the re-estimate for the emission probability is given by:

$$\bar{b}_j(\boldsymbol{k}) = \frac{\sum_{e=1}^{E} \sum_{\{t : \boldsymbol{y}_{te} = \boldsymbol{k},\, t = 1,2,\ldots,T_e\}} \gamma_j(t,e)}{\sum_{e=1}^{E} \sum_{t=1}^{T_e} \gamma_j(t,e)}. \tag{9.20}$$

In Equation (9.20), quantities for the $e^{\text{th}}$ example of the word are denoted by $T_e$ for the number of frames in the example and $\boldsymbol{y}_{te}$ for the feature vector at the $t^{\text{th}}$ frame of the example, with $\gamma_j(t,e)$ being used for the value of $\gamma_j(t)$ for the $e^{\text{th}}$ example.

The denominator in Equation (9.20) is the sum of the individual probabilities of being in state $j$ for each frame time, given the complete set of training data, and is sometimes referred to as the **state occupancy**. In some publications, the term **count** is also used when referring to this quantity. Although it is in fact a sum of probabilities, because it has been summed over the complete data set it is equivalent to the expected number, or count, of frames for which the state is occupied (although it will not in general be an integer number of frames).

In order to re-estimate the transition probabilities, we need to calculate the probability of a transition between any pair of states. This calculation is basically straightforward, but care needs to be taken to treat the start and end of the word correctly[3]. In the following explanation, transitions from the initial state and to the final state will be treated separately from transitions between emitting states.

Returning for the moment to considering only a single example of the word, let us define $\xi_{ij}(t)$ to be the probability that there is a transition from state $i$ to state $j$ at time $t$, given that the model generates the whole sequence of feature vectors representing the example of the word:

$$\xi_{ij}(t) = \frac{\alpha_i(t) a_{ij} b_j(\boldsymbol{y}_{t+1}) \beta_j(t+1)}{\alpha_F(T)} \quad \text{for } 1 \le t < T. \tag{9.21}$$

[3] The details of the equations given here apply to the use of special initial and final states and there will be slight differences if, for example, the model is allowed to end in any state (as in some publications).

Equation (9.21) can be applied to calculate the probability of a transition between any pair of emitting states at frame times starting from $t=1$ up until $t=T-1$. For the final frame, $t=T$, there cannot be a transition to another emitting state and the only possible transition is to the final state, $F$, with probability $\xi_{iF}(T)$, thus:

$$\xi_{iF}(T) = \frac{\alpha_i(T)a_{iF}}{\alpha_F(T)}. \tag{9.22}$$

For the initial state, we need to calculate the probability of a transition to each of the emitting states. This transition from the initial state is only possible at the start of the word, before any observations have been generated. If we regard this time as being $t=0$ then, given that the model must start in state $I$, another special instance of Equation (9.21) can be derived for all transitions out of state $I$, thus:

$$\xi_{Ij}(0) = \frac{a_{Ij}b_j(\boldsymbol{y}_1)\beta_j(1)}{\alpha_F(T)}. \tag{9.23}$$

The total probability of a transition between any pair of states $i$ and $j$ is obtained by summing the values of $\xi_{ij}(t)$ over all frames for which the relevant transition is possible. Dividing this quantity by the total probability $\gamma_i$ of occupying state $i$ gives the re-estimate for the transition probability $a_{ij}$. Assuming $E$ examples of the word, for a transition between any two emitting states we have:

$$\bar{a}_{ij} = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e-1}\xi_{ij}(t,e)}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_i(t,e)} \quad \text{for } 1 \le i, j \le N, \tag{9.24}$$

where $\xi_{ij}(t, e)$ denotes the value of $\xi_{ij}(t)$ for the $e^{\text{th}}$ training example. Note that the summation of $\xi_{ij}(t, e)$ over time only includes frames up until time $T_e - 1$. The last frame is not included as it cannot involve a transition to another emitting state, and so by definition the value of $\xi_{ij}(T, e)$ is zero for all pairs of emitting states.

Transitions from an emitting state to the final state $F$ can only occur at time $T_e$ and so the transition probability $a_{iF}$ may be re-estimated as:

$$\bar{a}_{iF} = \frac{\sum_{e=1}^{E}\xi_{iF}(T_e,e)}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_i(t,e)} \quad \text{for } 1 \le i \le N. \tag{9.25}$$

Transitions from the initial state $I$ can only occur at the start (time $t=0$), when the model *must* be in state $I$, so $\gamma_I(0, e) = 1$ for all examples and hence:

$$\bar{a}_{Ij} = \frac{\sum_{e=1}^{E}\xi_{Ij}(0,e)}{E} \quad \text{for } 1 \le j \le N. \tag{9.26}$$

The use of forward and backward probabilities to re-estimate model parameters is usually known either as the **forward–backward algorithm** or as the **Baum–Welch algorithm**. The second name in "Baum–Welch" recognizes the fact that Lloyd Welch was working with Baum on this subject in the early 1960s.

After re-estimation using the Baum–Welch algorithm, the probability of the training data given the new set of models is guaranteed to be higher than the probability for the previous model set, except at the **critical point** at which a local optimum has been reached and therefore the models (and hence the probability) are unchanged. The procedure can thus be repeated in an iterative manner until the difference between the new and old probabilities is sufficiently small that the training process can be regarded as being close enough to its local optimum.

It can be seen from the expression of Equations (9.24), (9.25) and (9.26) using the quantities defined in Equations (9.21), (9.22) and (9.23) that, if any of the $a_{ij}$ are initially given values of zero, their re-estimated values will also always be zero. Setting initial values of some transition probabilities to zero is thus a convenient way of constraining the structure of the word model to prevent it from producing intrinsically implausible state sequences. For example, it would not seem reasonable to allow the model to occupy a state early in the word, and then return to it after having been through several succeeding states. The sequence possibilities in Figure 9.1 are very limited, only allowing three non-zero values of $a_{ij}$ for any state $i$, yet this structure is very plausible as a word model. Constraining the possible state sequences by setting most of the initial values of the transition probabilities to zero has the added benefit of greatly reducing the computation required for both recognition and training.

Model initialization issues, including the choice of initial conditions for the emission p.d.f.s, will be discussed in more detail later on in this chapter.

### 9.5.3 Viterbi training

It is also possible to re-estimate the model parameters using only the most likely path through the states, as given by the Viterbi algorithm. The calculations are substantially simplified by just considering a single path. For any frame of input data the probability of a state being occupied can only be unity or zero, depending on whether that state is on the path. The most likely path can be found by calculating the values of $\hat{\alpha}_j(t)$ for all states and frames to the end of the word using Equation (9.11), and then tracing back from the final state in the same way as for the DTW method described in Chapter 8. In contrast to Baum–Welch re-estimation, the backward probabilities are not required.

Having identified the most likely path, each input frame will have been allocated to a single state to provide a state-level segmentation of the training data. It will therefore be known which state produced each observed feature vector, and also which states preceded and followed each state along the path. For the re-estimation it is then only necessary, for all examples of each training word, to accumulate the statistics of the feature vectors that occur for each occupied state, and of the transitions between states along the most likely path. Using the identified path, there will need to be counts of the following events, totalled over all $E$ examples of the word:

i. the number of frames for which each state gives rise to each of the possible feature vectors, with the count for state $j$ and feature vector $\boldsymbol{k}$ being denoted by $n_j(\boldsymbol{y}_t = \boldsymbol{k})$;
ii. the number of frames for which a transition occurs between each pair of states, which for transitions between states $i$ and $j$ will be denoted by $n_{ij}$;
iii. the number of occasions for which each state is occupied for the first frame of each example of the word, which for state $j$ will be denoted by $n_{Ij}$;
iv. the number of occasions for which each state is occupied for the last frame of each example of the word, which for state $i$ will be denoted by $n_{iF}$;
v. the number of frames for which each state is occupied, which will be denoted by $n_i$ and $n_j$ for states $i$ and $j$ respectively.

The re-estimation formulae are then simply given by:

$$\bar{b}_j(\boldsymbol{k}) = \frac{n_j(\boldsymbol{y}_t = \boldsymbol{k})}{n_j}, \tag{9.27}$$

$$\bar{a}_{ij} = \frac{n_{ij}}{n_i} \quad \text{for all pairs of emitting states, } 1 \le i, j \le N, \tag{9.28}$$

$$\bar{a}_{iF} = \frac{n_{iF}}{n_i} \quad \text{for all } i \text{ such that } 1 \le i \le N, \tag{9.29}$$

$$\bar{a}_{Ij} = \frac{n_{Ij}}{E} \quad \text{for all } j \text{ such that } 1 \le j \le N. \tag{9.30}$$

Note that the above re-estimation equations for Viterbi training are in fact equivalent to the corresponding Baum–Welch equations (9.20, 9.24, 9.25, 9.26) with the values of all the frame-specific state occupancy probabilities ($\gamma_j(t, e)$, etc.) set either to one or to zero, depending on whether or not the relevant states are occupied at the given frame time. As with the Baum–Welch re-estimation, the Viterbi training procedure (determination of the most likely state sequence followed by estimation of the model parameters) can be applied in an iterative manner until the increase in the likelihood of the training data is arbitrarily small.

Because the contribution to the total probability is usually much greater for the most likely path than for all other paths, an iterative Viterbi training procedure usually gives similar models to those derived using the Baum–Welch recursions. However, the Viterbi method requires much less computation and it is therefore often (and successfully) adopted as an alternative to full Baum–Welch training.

## 9.6 VECTOR QUANTIZATION

In the discussion above it was assumed that the data used for training the models include a large enough number of words for reliable values to be obtained for all the parameters. For any statistical estimation to give sensible results it is obvious that the total number of data items must be significantly larger than the number of separate parameters to be estimated for the distribution. If the number of possible feature vectors is very large, as a result of many possible values for each of several individual features, many feature vectors will not occur at all in a manageable

amount of training data. In consequence all the generation probabilities for these feature vectors will be estimated as zero. If such a feature vector then occurred in the input during operational use of the recognizer, recognition would be impossible.

The multi-dimensional feature space for any practical method of speech analysis is not uniformly occupied. The types of spectrum cross-section that occur in speech signals cause certain regions of the feature space, for example those corresponding to the spectra of commonly occurring vowels and fricatives, to be highly used, and other regions to be only sparsely occupied. It is possible to make a useful approximation to the feature vectors that actually occur by choosing just a small subset of vectors, and replacing each measured vector by the one in the subset that is 'nearest' according to some suitable distance metric. This process of **vector quantization (VQ)** is also used in systems for efficient speech coding (see Section 4.3.5).

Setting up a vector quantizer usually involves first applying a **clustering** algorithm to group similar vectors together, then choosing a representative quantized vector for each cluster. The performance of such a quantizer depends on the number of different vectors and how they are chosen, but the details of these decisions are outside the scope of this book. It is, however, clear that if a fairly small **codebook** of vectors is chosen to represent the well-occupied parts of the feature space, all of these quantized vectors will occur frequently in a training database of moderate size. For each model state it will thus be possible to obtain good estimates for the probability of all feature vectors that are likely to occur.

Even after vector quantization, a fully trained model for a particular word will often have some feature vectors that are given zero probability for all states of the word. For example, the word "one" would not be expected to contain any examples of a feature representing the typical spectrum of an [s] sound. It is, however, important not to allow the probabilities to remain exactly at zero. Otherwise there is the danger of error on an input word that matches fairly well to the properties of one of the models except for just one non-typical frame that is represented by a zero-probability feature vector. In such a case the model will yield zero probability for that sequence of vectors, and the recognizer will therefore not be able to choose the correct word. A simple solution is to replace the zero value by a very small number. The model will then yield a low probability of generating the observed features, but if the rest of the word is sufficiently distinctive even this low value can be expected to be greater than the probability of generating the same set of features from any of the competing models. Better estimates for the probability of an unseen feature vector can be obtained by using a measure of distance from the vectors that are observed for the word, so that the unseen vector is given a higher probability if it is similar to those vectors which do occur in the training examples.

## 9.7 MULTI-VARIATE CONTINUOUS DISTRIBUTIONS

Vector quantization involves an approximation which unavoidably loses some information from the original data, and any method for estimating the probability of an unseen feature vector will inevitably be somewhat *ad hoc*. These limitations associated with discrete distributions can be overcome by representing the distribution of feature vectors by some suitable parametric description. Provided

that an appropriate parametric distribution can be found for describing the true distribution of the features, a useful estimate can be computed for the probability of *any* feature vector that may occur in the training and recognition processes.

Many natural processes involve variable quantities which approximate reasonably well to the **normal** (or **Gaussian**) distribution. The normal distribution has only two independently specifiable parameters, the **mean**, $\mu$, and the **standard deviation**, $\sigma$. For a quantity $x$, the probability density, $\phi(x)$, is given by:

$$\phi(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right). \tag{9.31}$$

When quantities are distributed normally, this simple mathematical description of the distribution makes it possible to calculate the probability of the quantity lying in any range of values provided the mean and standard deviation of the distribution are known. To calculate the probability of one particular value (i.e. a measured acoustic feature vector) occurring, we need to consider the limiting case in which the size of the interval for the range of values is infinitesimally small.

The definition of the continuous probability density function, $\phi(x)$, of a variate, $x$, is such that the probability of an observation lying in an infinitesimal interval of size $dx$ centred on $x$ is $\phi(x)dx$, and is thus infinitesimally small. However, if continuous probability density functions are used instead of discrete probability distributions in the HMM equations given in Sections 9.3 to 9.5, the computation will still give the correct relative likelihoods of the different words, as the infinitesimal interval, $dx$, is common to all probability calculations. The probability of observing the features, $P(\boldsymbol{Y})$, independently of which word is spoken, is also affected in the same way by the size of $dx$. The probability of the word given the features is therefore still correctly given by the formula expressed in Equation (9.1), even if these probability densities are used instead of actual probabilities for $P(\boldsymbol{Y})$ and $P(\boldsymbol{Y}|w)$. Although their theoretical interpretations are different, it is thus equally suitable to use either discrete or continuous probability distributions in the calculations of word probability and in parameter re-estimation. In the following discussion of continuous distributions, it will be convenient to continue to use the term "probability" even where the quantities are, strictly speaking, probability densities.

## 9.8 USE OF NORMAL DISTRIBUTIONS WITH HMMS

It is obvious that many naturally occurring quantities are not normally distributed. For example, speech intensity measured over successive fixed time intervals of, say, 20 ms during continuous speech will certainly not approximate to a normal distribution because it clearly has a hard limit of zero during silences, will be low for much of the time during weak sounds, but will go up to quite high values during more intense vowels. The intensity on a logarithmic scale would have a more symmetrical distribution, which might be nearer to normal, but in this case the low-level end of the distribution will be very dependent on background noise level.

Normal distributions usually fit best to measurements which can be expected to have a preferred value, but where there are various chance factors that may cause

deviation either side of that value, with the probability progressively decreasing as the distance either side of the preferred value increases. Thus it might be reasonable to use a normal distribution to approximate a distribution of speech features which are derived from the same specific part of a specific word spoken in the same way by the same person. When it is assumed that features are normally distributed for each state of an HMM, the distributions are often termed **single Gaussian**.

When different speakers are combined in the same distribution the departures from normal will be greater, and for different regional accents there is a fairly high probability that the distribution will be multi-modal, and therefore much less suitable for modelling as a normal distribution. However, when multi-modal distributions are likely, as is the case with many current speech recognition systems, it is now almost universal to model the distributions with a weighted sum, or **mixture**, of several normal distributions with different means and variances (usually referred to as **Gaussian mixtures**). Provided that there is a sufficient number of mixture components, any shape of distribution can be approximated very closely. This characteristic of sums of Gaussian distributions, combined with the attractive mathematical properties of the Gaussian itself, is largely responsible for their widespread and successful use for describing emission probability distributions in HMM-based speech recognition systems.

The theory underlying the use of mixture distributions is a straightforward extension of the single-Gaussian case and will be discussed in Section 9.10, after first introducing the probability calculations and model parameter re-estimation equations using single Gaussian distributions.

### 9.8.1 Probability calculations

The features are multi-dimensional and so, in the case of single-Gaussian distributions, they will form a multi-variate normal distribution. In general the features may not vary independently, and their interdependence is specified by a **covariance matrix**. The entries along the main diagonal of this matrix represent the variance of each feature, while the remaining entries indicate the extent to which the separate feature distributions are correlated with each other.

Let us first consider the output probability $b_j(\boldsymbol{y})$ for the $j^{\text{th}}$ state, where $\boldsymbol{y}$ is a single feature vector. Assume that the column vector $\boldsymbol{y}$ comprises $K$ features, $y_1, y_2, \ldots, y_K$. Let $\boldsymbol{\mu}_j$ be the column vector of means, $\mu_{j1}, \mu_{j2}, \ldots, \mu_{jK}$, and $\boldsymbol{\Sigma}_j$ be the covariance matrix for the distribution of features associated with that state. The definition of the multi-variate normal distribution gives the output probability compactly in matrix notation:

$$b_j(\boldsymbol{y}) = \frac{1}{|\boldsymbol{\Sigma}_j|^{1/2}\,(2\pi)^{K/2}} \exp\left(\frac{-(\boldsymbol{y}-\boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j^{-1}(\boldsymbol{y}-\boldsymbol{\mu}_j)}{2}\right), \tag{9.32}$$

where $|\boldsymbol{\Sigma}_j|$ is the determinant of $\boldsymbol{\Sigma}_j$ and $(\boldsymbol{y}-\boldsymbol{\mu}_j)^T$ is the transpose of $(\boldsymbol{y}-\boldsymbol{\mu}_j)$. In the special case when the features are uncorrelated, the covariance matrix becomes zero except along its main diagonal (and is therefore often referred to as a **diagonal**

**covariance matrix**). The probability of a feature vector then reduces to a product of probabilities given by the univariate distributions of the separate features:

$$b_j(\boldsymbol{y}) = \prod_{k=1}^{K} \frac{1}{\sigma_{jk}\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y_k - \mu_{jk}}{\sigma_{jk}}\right)^2\right), \tag{9.33}$$

where $y_k$ is the $k^{\text{th}}$ feature of $\boldsymbol{y}$, and $\mu_{jk}$ and $\sigma_{jk}$ are the mean and standard deviation of the distribution of the $k^{\text{th}}$ feature for state $j$.

Equation (9.33) is evidently computationally simpler than Equation (9.32). The extent of the computational saving provides a strong motivation for choosing methods of speech analysis for which the features are substantially uncorrelated. Some of these methods will be described in Chapter 10. Most current speech recognition systems adopt such a method and use diagonal covariance matrices.

Having defined an expression for the emission probability in terms of the distribution parameters, recognition can be performed in the same way as when using discrete distributions. Thus, in the case of the Viterbi algorithm, the new definition of $b_j(\boldsymbol{y})$ is simply used in Equations (9.11) and (9.12).

### 9.8.2 Estimating the parameters of a normal distribution

When modelling emission probabilities with continuous distributions, the training task is to optimize the parameters of the feature distribution model, rather than the probabilities of particular feature vectors. If we had a set of $T$ feature vectors that were known to correspond to state $j$, then the maximum-likelihood estimates for the parameters of a normal distribution are easily calculated. The mean vector $\hat{\boldsymbol{\mu}}_j$ is equal to the average of all the observed vectors (i.e. the sample mean), and the covariance matrix $\hat{\boldsymbol{\Sigma}}_j$ is obtained based on the deviation of each of the observed vectors from the estimated mean vector (i.e. the sample covariance matrix):

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{y}_t , \tag{9.34}$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{1}{T}\sum_{t=1}^{T} (\boldsymbol{y}_t - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{y}_t - \hat{\boldsymbol{\mu}}_j)^T . \tag{9.35}$$

Obviously, in the case of HMMs, the state sequence is not known, but the standard methods for estimating mean and covariance given in Equations (9.34) and (9.35) can be extended for use in either Baum–Welch or Viterbi re-estimation procedures, as explained below.

### 9.8.3 Baum–Welch re-estimation

For the Baum–Welch algorithm, the parameters are re-estimated using contributions from all frames of all the $E$ examples of the word in the training data.

Each contribution is weighted by the probability of being in the state at the relevant frame time, as given by Equation (9.19). Therefore the re-estimates of the mean vector $\mu_j$ and the covariance matrix $\Sigma_j$ associated with state $j$ are given by:

$$\bar{\mu}_j = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_j(t,e)\,\mathbf{y}_{te}}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_j(t,e)}, \tag{9.36}$$

$$\bar{\Sigma}_j = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_j(t,e)\,(\mathbf{y}_{te}-\bar{\mu}_j)(\mathbf{y}_{te}-\bar{\mu}_j)^T}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_j(t,e)}, \tag{9.37}$$

where $\mathbf{y}_{te}$ is the feature vector for the $t^{\text{th}}$ frame of the $e^{\text{th}}$ example of the word.

Just as for discrete emission p.d.f.s, it can be shown that iterative application of the above formulae leads to a locally optimum solution. Baum's (1972) analysis included a proof for univariate normal distributions, which was later generalized by Liporace (1982) to a wider class of distributions, including multi-variate normal distributions.

Note that Equation (9.37) for re-estimating the covariance matrix is based on deviation of observed vectors from the *re-estimated* mean vector $\bar{\mu}_j$. In practice, when accumulating the contributions for covariance re-estimation it is easier to use the current estimate $\mu_j$ instead of the (yet to be computed) new value $\bar{\mu}_j$. It is then straightforward to correct for the difference between the old and new mean values at the end of the calculation.

### 9.8.4 Viterbi training

For Viterbi re-estimation, the requirement is just to use the state-level segmentation obtained from the most likely path according to the current set of models as the basis for collecting the statistics needed to apply Equations (9.34) and (9.35) for each model state. The statistics for state $j$ are therefore gathered over all examples of the word using all frames for which state $j$ is occupied. Using $s_{te}$ to denote the state occupied at frame $t$ of example $e$, the re-estimation formulae are as follows:

$$\bar{\mu}_j = \frac{1}{n_j}\sum_{e=1}^{E}\sum_{t\ni s_{te}=j}\mathbf{y}_{te}, \tag{9.38}$$

$$\bar{\Sigma}_j = \frac{1}{n_j}\sum_{e=1}^{E}\sum_{t\ni s_{te}=j}(\mathbf{y}_{te}-\bar{\mu}_j)(\mathbf{y}_{te}-\bar{\mu}_j)^T, \tag{9.39}$$

where, as in Section 9.5.3, $n_j$ is the number of frames for which state $j$ is occupied.

## 9.9 MODEL INITIALIZATION

There must be enough states in the model to capture all the acoustically distinct regions in the word. For example, a typical word of two or three syllables could need around 10–20 states to model the acoustic structure adequately. Because the training process only finds a local optimum, the initial estimates for the model parameters can have a strong influence on the characteristics of the final set of trained models. It is very important to give careful consideration to how the model parameters, including both transition and emission probabilities, should be initialized before training. The trained model for each word needs to capture the spectral and temporal characteristics of all spoken utterances of that word while at the same time, in order to minimize recognition errors, it must be a constraining model which does not allow inappropriate sequences of states for the word.

In an HMM, the probability of a path through the model is computed on a frame-by-frame basis and therefore cannot take into account any of the previous states occupied other than the one at the immediately preceding time frame. Thus, if a model allows many different transitions from each state, recognition errors can result if a sequence of frames gives a good acoustic match even if the complete state sequence is very inappropriate for a genuine example of the word. Even the limited degree of flexibility included in the model structure shown in Figure 9.1 can cause problems if used throughout a word.

The dangers associated with allowing flexibility of transitions within a word model are such that most current uses of HMMs only allow a very restricted set of possible transitions, by initializing most of the transition probabilities to zero. A popular HMM structure for speech recognition uses a left-to-right topology with the probability of all transitions set to zero except those to the next state or returning to the current state (i.e. as for Figure 9.1 but omitting the 'skip' transitions). If this model structure is used to represent a word, the word will be modelled as a sequence of acoustic regions which can vary in duration but which must always all occur and always in the same fixed order. With this strong temporal constraint provided by the model structure, re-estimation (using either the Baum–Welch or the Viterbi approach) can give a useful local optimum even with a simple initialization approach for the emission probabilities. One popular strategy is to start with a uniform segmentation of each training example, with the number of segments being equal to the number of states in the model. This segmentation can then be used to compute the required statistics for each state emission probability, with the allowed transition probabilities of all emitting states initialized to identical values.

In the case of Baum–Welch training, an even simpler initialization strategy may be used for the parameters of discrete or normal distributions. For this method, sometimes called a **flat start** (Knill and Young, 1997), all emission p.d.f.s for all states are set to average values computed over the entire training set, in addition to using identical transition probabilities for a limited set of allowed transitions. Thus all permitted paths through the model start with equal probability and the training algorithm is left to optimize the parameters from this neutral starting position with constraints imposed by the model structure. This approach has been found to work well if there are several utterances for each model unit (e.g. Paul and Martin, 1988).

An important advantage of the initialization approaches described above is that the training process can be carried out completely automatically, without

requiring any pre-segmented data. Alternatively, if there are any data available for which suitable state boundaries are 'known' (for example, the boundaries could be marked by hand for a small subset of the training corpus), this segmentation can be used as the basis for initializing some or all of the model parameters.

If all models use a restricted structure that only allows transitions back to the same state or on to the next state, it is implicitly assumed that such a model structure is appropriate for representing all words. There are many cases in human language where pronunciation varies from occasion to occasion, even for one speaker. The variations may be at the phonemic level: for example, in the word "seven" many speakers often omit the vowel from the second syllable and terminate the word with a syllabic [n]. Allophonic variations can also occur: for example, in words ending in a stop consonant, the consonant may or may not be released. If a word with alternative pronunciations is represented by a single sequence of states with the model structure described above, some states will have to cope with the different pronunciations, and so their p.d.f.s will need to be multi-modal to model the distributions well. In these cases a normal distribution will not be suitable, and Gaussian mixtures will be essential for good modelling of the data.

A rather different approach is to explicitly model alternative pronunciations as alternative state sequences, using either whole-word or sub-word models. Initialization then involves choosing a constraining topology separately for each word model to take into account the possible phonetic structure of the word and its expected variation. It will thus be necessary to decide on the number of states required to represent each phonetic event and on the allowed transitions between the states, with state skips being allowed only where a particular phonetic event is sometimes omitted. For this approach to work it is essential that the emission p.d.f.s of the models are initialized with values roughly appropriate for the phonetic events expected for each state, because otherwise the training frames may not be allocated to the states in the intended way. Such models can be initialized by carefully hand-labelling a few examples of the training words in terms of state labels, and collecting the statistics of these data to initialize the emission p.d.f.s. However, the whole method requires a lot of skilled human intervention, and a simpler model topology is usually adopted, with any limitations in this approach being addressed by using Gaussian mixtures for the p.d.f.s. Methods used for including some simple provision for alternative pronunciations will be considered further in Chapter 12.

## 9.10 GAUSSIAN MIXTURES

### 9.10.1 Calculating emission probabilities

The expression for the emission probability defined in Equation (9.32) is easily extended to include a weighted sum of normal distributions, where each component distribution has a different mean and variance. We will use the notation $N(\boldsymbol{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to represent the probability density of the observed vector $\boldsymbol{y}$ given a normal distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Thus, for an emission p.d.f. defined according to a Gaussian mixture distribution, the emission probability given by the $m^{\text{th}}$ component for state $j$ is:

$$b_{jm}(\boldsymbol{y}) = N\left(\boldsymbol{y}; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}\right). \tag{9.40}$$

The total emission probability for a distribution with $M$ components is defined as:

$$b_j(\boldsymbol{y}) = \sum_{m=1}^{M} c_{jm} b_{jm}(\boldsymbol{y}), \tag{9.41}$$

where $c_{jm}$ denotes the weight of the $m^{\text{th}}$ mixture component for state $j$. The mixture component weights can only take positive values, $c_{jm} \geq 0$, and must sum to 1:

$$\sum_{m=1}^{M} c_{jm} = 1 . \tag{9.42}$$

In the special case where there is only one mixture component, the emission probability specified by Equation (9.41) is defined in terms of a single Gaussian distribution with weight equal to 1 and is therefore equivalent to Equation (9.33).

Once the parameters of multiple-component mixture distributions have been trained, Equation (9.41) can be used as the basis for the recognition calculations in exactly the same way as with the simpler emission p.d.f.s that we have already discussed. Parameter estimation for mixture distributions requires more detailed consideration, and is discussed in the following sections. Firstly we will assume that initial estimates are available and address the re-estimation problem, before considering ways of obtaining suitable initial estimates in Section 9.10.4.

### 9.10.2 Baum–Welch re-estimation

Assuming that initial estimates are available for all the parameters of all the $M$ components of a Gaussian mixture representing the emission p.d.f. for state $j$, Baum–Welch re-estimation can be used to find new estimates for these parameters, $c_{jm}$, $\boldsymbol{\mu}_{jm}$ and $\boldsymbol{\Sigma}_{jm}$. When using Gaussian mixtures, the contribution from each observation $\boldsymbol{y}_t$ needs to be weighted by a probability that is specific to the mixture component $m$. By analogy with the quantity $\gamma_j(t)$ which was introduced in Section 9.5.2, let us define $\gamma_{jm}(t)$ to be the probability of being in state $j$ at time $t$ and using component $m$ to generate $\boldsymbol{y}_t$, given that the model generates the whole sequence of $T$ feature vectors representing an example of the word.

$$\gamma_{jm}(t) = \frac{\sum_{i=1}^{N} \alpha_i(t-1) a_{ij} c_{jm} b_{jm}(\boldsymbol{y}_t) \beta_j(t)}{\alpha_F(T)} \tag{9.43}$$

Now, if we have $E$ examples of the word, summing the values of $\gamma_{jm}(t, e)$ over all frames of all examples gives the total probability for the $m^{\text{th}}$ component of state $j$ generating an observation. Dividing this quantity by the corresponding sum of $\gamma_j(t, e)$ terms gives the re-estimate for the mixture component weight $c_{jm}$:

$$\bar{c}_{jm} = \frac{\sum_{e=1}^{E} \sum_{t=1}^{T_e} \gamma_{jm}(t,e)}{\sum_{e=1}^{E} \sum_{t=1}^{T_e} \gamma_j(t,e)} . \tag{9.44}$$

The re-estimation equations for the mean vector and covariance matrix are the same as for the single-Gaussian case given in Equations (9.36) and (9.37), but using the component-specific state occupation probabilities $\gamma_{jm}(t,e)$:

$$\bar{\mu}_{jm} = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_{jm}(t,e)\mathbf{y}_{te}}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_{jm}(t,e)}, \tag{9.45}$$

$$\bar{\Sigma}_{jm} = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_{jm}(t,e)(\mathbf{y}_{te}-\bar{\mu}_{jm})(\mathbf{y}_{te}-\bar{\mu}_{jm})^T}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\gamma_{jm}(t,e)}. \tag{9.46}$$

Juang (1985) extended Liporace's (1982) analysis to show that iterative application of the re-estimation formulae leads to a locally optimum solution when emission p.d.f.s are defined in terms of sums of normal distributions.

### 9.10.3 Re-estimation using the most likely state sequence

The use of a Gaussian mixture to represent the HMM emission p.d.f. incorporates another 'hidden' element in the model, as it is not known from the observations which mixture component generated each observation. The probability calculation in Equation (9.41) uses the total probability taking into account all the mixture components that could have produced the observation. As a result the Baum–Welch re-estimation formulae in Equations (9.44) to (9.46) use probabilities not only of state occupancy but also of mixture components. The formulae can be simplified if the state sequence is known, but the situation is more complex than for the single-Gaussian case because the mixture components are still unknown. Thus the state sequence alone does not lead to an analytic solution for these emission p.d.f.s.

One option is to retain the EM algorithm for estimating the parameters of the mixture distribution. Equations (9.44) to (9.46) can be simplified accordingly: the summations are now just over those frames for which state $j$ is occupied, and for each frame the component-dependent state occupation probability $\gamma_{jm}(t,e)$ simplifies to a component-dependent emission probability $c_{jm}b_{jm}(\mathbf{y}_{te})$. (The total state occupation probability $\gamma_j(t,e)$ is replaced by the emission probability $b_j(\mathbf{y}_{te})$.)

Alternatively, to estimate the distribution parameters without requiring an EM algorithm, each observation must be assigned to a single mixture component. This assignment can be achieved by using a clustering procedure to divide the observed feature vectors corresponding to any one model state into a number of groups equal to the number of mixture components for that state. ***K*-means clustering** is a well-established technique for dividing a set of vectors into a specified number of classes in order to locally minimize some within-class distance metric, and was originally applied to vector quantization (see Section 9.6). The term **segmental**

***K*-means** is often used to refer to the use of *K*-means clustering in conjunction with a Viterbi alignment procedure to identify the state-level segmentation.

After clustering, each frame will be labelled, not only with the state that was occupied, but also with the mixture component that generated the observation. The re-estimation formula for the weight associated with the $m^{\text{th}}$ mixture component of state $j$ is then given by:

$$\bar{c}_{jm} = \frac{n_{jm}}{n_j}, \tag{9.47}$$

where $n_{jm}$ represents the number of frames for which state $j$ was occupied and mixture component $m$ generated an observation. Using $s_t$ to denote the state occupied and $x_t$ to denote the mixture component used at time $t$, the re-estimation formulae for the mean feature vector and covariance matrix are straightforward extensions of the single-Gaussian case (Equations (9.38) and (9.39)), as follows:

$$\bar{\boldsymbol{\mu}}_{jm} = \frac{1}{n_{jm}} \sum_{e=1}^{E} \sum_{t \ni s_t = j, x_t = m} \boldsymbol{y}_{te}, \tag{9.48}$$

$$\bar{\boldsymbol{\Sigma}}_{jm} = \frac{1}{n_{jm}} \sum_{e=1}^{E} \sum_{t \ni s_t = j, x_t = m} (\boldsymbol{y}_{te} - \bar{\boldsymbol{\mu}}_{jm})(\boldsymbol{y}_{te} - \bar{\boldsymbol{\mu}}_{jm})^T. \tag{9.49}$$

### 9.10.4 Initialization of Gaussian mixture distributions

The segmental *K*-means procedure outlined above uses an initial set of models to obtain the state-level segmentation, but does not rely on any existing estimates for the mixture components. It therefore provides a convenient method for initializing the parameters of HMMs using mixture distributions. If no models are available, the process can even be started from a uniform segmentation, as described in Section 9.9. Once initial estimates have been obtained for all the mixture components, the estimates can be refined using further iterations of the segmental *K*-means procedure. At this point the models could be used for recognition, but they can be trained further using full Baum–Welch re-estimation, or even using the EM algorithm to update the mixture parameters without changing the segmentation.

A segmental *K*-means procedure is often used to initialize mixture models prior to Baum–Welch training. However, this approach requires the number of mixture components to be decided in advance. An alternative is to start with trained single-Gaussian models and to incrementally increase the number of mixture components using a method often referred to as **mixture splitting**. Starting with a single Gaussian distribution for the emission p.d.f., a two-component mixture model is initialized by duplicating the parameters of the original distribution and perturbing the means by a small amount in opposite directions (typically ±0.2 standard deviations). The variances are left unchanged and the mixture weights are set to 0.5 for both components. The means, variances and mixture weights are all re-estimated, and the mixture-splitting procedure is then applied to the component with the largest weight (setting the weights of both new components to half the

value for the component from which they were derived). The model parameters are re-estimated again, and so on until the desired level of complexity is reached.

For a given number of mixture components, Young and Woodland (1993) reported that an iterative mixture-splitting training procedure with Baum–Welch re-estimation gave similar results to using segmental *K*-means followed by Baum–Welch training. However, a useful advantage of the mixture-splitting approach is that the number of mixture components can be chosen for each state individually according to some objective criterion based on how well the data are modelled. Examples of useful criteria for deciding on the number of components are the magnitude of the increase in training-data likelihood from adding a new component, or the quantity of training data available for the model concerned. This flexibility of mixture modelling is particularly beneficial for modelling large vocabularies; its use will be discussed further in Chapter 12.

### 9.10.5 Tied mixture distributions

Increasing the number of components used in a Gaussian mixture distribution allows for greater flexibility in the shapes of distributions that can be modelled, but a larger quantity of training data is required to ensure that the parameters are trained robustly. In any practical recognizer there are often only limited data available for training each model, which imposes limitations on the number of state-specific mixture components that can be included. However, similarities between different speech sounds are such that many of the component distributions will be similar for several different states. One straightforward way of taking advantage of these similarities to provide more data for training the model parameters is to use the same Gaussian distributions to represent *all* the states of all the models, with only the mixture weights being state-specific. Thus the distribution parameters are tied across the different states, and this type of model is often referred to as a **tied mixture** (Bellegarda and Nahamoo, 1990). The term **semi-continuous HMM** has also been used (Huang and Jack, 1989), because the one set of continuous distribution parameters for all states can be regarded as an alternative to the VQ-generated codebook used with discrete emission probabilities.

When using tied mixtures, the emission probability $b_j(\boldsymbol{y})$ for any one state $j$ is calculated in the same way as for Equation (9.41), but although the mixture weights $c_{jm}$ are state-specific, the $b_{jm}(\boldsymbol{y})$ terms will be the same for all states.

Using the new definition of the emission probability, re-estimation formulae can be derived for the mean $\boldsymbol{\mu}_m$ and covariance matrix $\boldsymbol{\Sigma}_m$ of the $m^{\text{th}}$ component (the re-estimation of the mixture weights $c_{jm}$ is unchanged). For example, tied-mixture versions of the Baum–Welch formulae in Equations (9.45) and (9.46) are as follows:

$$\bar{\boldsymbol{\mu}}_m = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\sum_{j=1}^{N}\gamma_{jm}(t,e)\boldsymbol{y}_{te}}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\sum_{j=1}^{N}\gamma_{jm}(t,e)}, \tag{9.50}$$

$$\bar{\Sigma}_m = \frac{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\sum_{j=1}^{N}\gamma_{jm}(t,e)(\boldsymbol{y}_{te}-\bar{\boldsymbol{\mu}}_m)(\boldsymbol{y}_{te}-\bar{\boldsymbol{\mu}}_m)^T}{\sum_{e=1}^{E}\sum_{t=1}^{T_e}\sum_{j=1}^{N}\gamma_{jm}(t,e)} . \quad (9.51)$$

Thus the only difference from the original (untied) mixture-distribution re-estimation formulae is that the contributions are now summed over all states as well as over all frames of all examples.

Tied mixtures have been used with some success to address the problems associated with training a large number of model parameters from a limited quantity of training data. However, although any practical system will have some model states which share many similarities, there will obviously be others which are quite different, and this characteristic will be reflected in the mixture weights for the different states. Thus rather more parameters are being tied together than is necessary, and such extensive tying may not be desirable for maximum discrimination. It is important to note that tied mixtures are just one example of the much more general concept of **parameter tying**, whereby any parameters of any model states can be tied together and the only effect on the re-estimation formulae is in the nature of the summations and in the indexing of the model parameters. The ability to tie together the parameters of HMM states is a significant factor in the success of current large-vocabulary speech recognition systems, and this use of tying is explained in Chapter 12.

## 9.11 EXTENSION OF STOCHASTIC MODELS TO WORD SEQUENCES

In the same way as was described for the dynamic programming methods in Chapter 8, HMMs extend easily to connected sequences of words. For recognition the word sequences can be represented by a higher-level model in which the states correspond to whole words, and the transition probabilities are the language-model probabilities (recognition using a language model will be discussed in Chapter 12).

In the case of recognition of isolated words we were not interested in the state sequences as such, but only in the likelihood of each word model emitting the observed feature vectors. When applying HMMs to connected words, however, we need to know the most likely sequence of words, so at the word level the Viterbi algorithm is necessary. The word boundary procedure is then exactly analogous to that described in Section 8.10, making use of the back-pointers to determine the word sequences.

The HMM training algorithms can also be used when the training data are spoken as natural connected sequences of words. It is not generally necessary to segment the data into the individual words prior to training. Instead, an embedded **training** approach can be used, whereby a composite model is obtained for the whole utterance by concatenating the required sequence of word models. This concatenation is very easy if special non-emitting initial and final states are used for the individual models, as it simply involves linking the final state of one word to the initial state of the next. The parameters of the composite model are trained

using the same procedure that would be carried out if this composite model represented a single word. If a state occurs more than once in the composite model (i.e. if the utterance contains more than one example of any particular word), all occurrences of that state will contribute to the parameter re-estimation. Provided that each word is spoken in a variety of different contexts, embedded training is very successful (at least for a constrained left-to-right model structure), even with the simplest 'flat' initialization procedure of setting the parameters of all models to the same values. The ability of the HMM training framework to automatically find the patterns in the data to associate with individual models is fundamental to the successful use of HMMs for substantial recognition tasks.

## 9.12 IMPLEMENTING PROBABILITY CALCULATIONS

The calculation of the forward and backward probabilities for sequences of feature vectors involves multiplication of a very large number of probability components, the majority of which are much less than 1. The results will in general have very low values, and mostly will be smaller than the minimum size of floating point number that can be held in any normal computer.

One solution to the number range problem is to check the probabilities at each stage of the recursion, and to multiply them by a scale factor that will bring the numbers back into the centre of the available range. However, scale factors must be noted and taken into account in estimating the relative likelihoods that each frame of feature vectors has been generated by each word model.

An alternative way of avoiding problems with numerical underflow is to represent all probabilities in logarithmic form, so that no explicit scaling is necessary. The following sections will discuss the implementation of HMM probability calculations using logarithms of probabilities.

### 9.12.1 Using the Viterbi algorithm with probabilities in logarithmic form

Because the logarithmic function is monotonic and increasing, the task of maximizing a probability can be achieved by maximizing its logarithm, and the main Viterbi probability calculation given in Equation (9.11) can therefore be replaced by:

$$\hat{\alpha}_j^L(t) = \max_{\text{over } i}\left(\hat{\alpha}_i^L(t-1) + a_{ij}^L\right) + b_j^L(\boldsymbol{y}_t) \ , \tag{9.52}$$

where $\hat{\alpha}_j^L(t)$ is used for $\log(\hat{\alpha}_j(t))$, $a_{ij}^L$ for $\log(a_{ij})$ and $b_j^L(\boldsymbol{y}_t)$ for $\log(b_j(\boldsymbol{y}_t))$.

When using discrete emission p.d.f.s with vector quantization, the calculation of Equation (9.52) is very straightforward and can easily be made very efficient: the quantities $\log(a_{ij})$ and $\log(b_j(\boldsymbol{y}_t))$ are fixed for given values of $i, j$ and $\boldsymbol{y}_t$, and hence the logarithms need to be calculated just once and stored ready for use as required. The dynamic programming algorithm then only involves summations and comparisons, with no multiplications or logarithmic functions.

If normal distributions are used for the emission p.d.f.s, we must take the logarithm of the expression for the emission probability, but this is also

straightforward. For example, in the case of uncorrelated normal distributions for the individual features, taking logarithms[4] of Equation (9.33) gives:

$$b_j^L(\boldsymbol{y}_t) = \log(b_j(\boldsymbol{y}_t)) = -\frac{K}{2}\log(2\pi) - \sum_{k=1}^{K}\log(\sigma_{jk}) - \frac{1}{2}\sum_{k=1}^{K}\left(\frac{y_{kt}-\mu_{jk}}{\sigma_{jk}}\right)^2 \quad (9.53)$$

where we are now taking into account the fact that the observations are time-dependent, and are using the symbol $y_{kt}$ to denote the $k^{\text{th}}$ feature at the $t^{\text{th}}$ frame.

Comparing Equation (9.53) with (9.33), it can be seen that use of logarithms has eliminated the need for the observation-dependent exponential operation, while the logarithmic terms are independent of the observed feature values and so can be pre-computed. Thus, while the computational load when using normal distributions is somewhat greater than for discrete emission p.d.f.s, the use of logarithms leads to a considerable computational saving as well as solving the number range problem.

### 9.12.2 Adding probabilities when they are in logarithmic form

When calculating emission probabilities using Gaussian mixture distributions, and for all calculations of forward and backward probabilities in Baum–Welch re-estimation, probabilities must be summed as well as multiplied and so the use of logarithms is more complicated. If we consider two probabilities, $A$ and $B$, the task is to compute $\log(A + B)$ given $\log(A)$ and $\log(B)$. In theory, we could exponentiate both $\log(A)$ and $\log(B)$, add them and take the logarithm. However, aside from the computational issues, the exponential operation puts the probabilities back onto a linear scale and so presents problems for the wide range of probabilities that may be encountered. This difficulty can be addressed by first rewriting $\log(A + B)$ thus:

$$\log(A+B) = \log(A(1+B/A)) = \log(A) + \log(1+B/A)\,. \quad (9.54)$$

Assume that we have ordered the probabilities such that $A \geq B$. The issue is now one of evaluating the ratio $B/A$, which can be no greater than 1 and therefore the calculation will only present problems if this ratio is smaller than the smallest number which can be represented in the computer. This situation can only arise if $B$ is so much smaller than $A$ that it can safely be ignored by setting $\log(A + B) = \log(A)$. A procedure for finding $\log(A + B)$ is therefore as follows:

1. If $\log(B) > \log(A)$ then transpose $\log(A)$ and $\log(B)$.
2. Find $\log(B/A)$ by forming $\log(B) - \log(A)$. Store this value in $C$.
3. If $C <$ a suitable threshold, set $C = 0$.
4. Otherwise $C = \log(1 + \exp(C))$.
5. Add $C$ to $\log(A)$.

The threshold in step 3 is used to prevent underflow when taking the exponential in step 4. The smallest value to which this threshold can be set is the logarithm of the smallest number that can be represented in the computer.

The procedure described above for performing probability calculations in logarithmic form is effective and widely used. However, whenever there is the need

[4] When using normal distributions, the calculations are simplest if natural logarithms are used, and the use of natural logarithms has been assumed in Equation (9.53).

to add two probabilities, one exponential and one logarithmic operation are usually required. These operations can be avoided by using a method which allows the numbers to be added while in their logarithmic form[5]. Considering step 4 in the sequence of calculations described above, both the exponential and the logarithmic operation can be avoided by using a pre-computed look-up table to store the values of $\log(1 + B/A)$ in terms of $\log(B/A)$. Thus steps 3 and 4 can be replaced by a single table look-up operation, with $\log(B/A)$ as input (i.e. the value already stored in $C$ at step 2). The output is $\log(1 + B/A)$, which can again be stored as the new value of $C$. Moderate accuracy in the value of $\log(A + B)$ can be achieved with a small look-up table. For example, a 1% accuracy for $A + B$ enables values of $B/A$ of less than 0·01 to be ignored, and the look-up table for the larger values of $B/A$ only needs entries for 115 equally spaced values of $\log(B/A)$.

If the above method is implemented using a suitable scale factor for the logarithms, it is then even possible to make all the probability calculations for recognition and parameter estimation using integer arithmetic on logarithmically coded numbers. No multiplications would be required with the VQ method, and no exponential functions would be needed when using Gaussian distributions. The 1% error proposed above should have very little effect on the re-estimation, but the error could easily be reduced if necessary by using a larger look-up table.

## 9.13 RELATIONSHIP BETWEEN DTW AND A SIMPLE HMM

It is interesting to compare the Markov probability calculation with the cumulative distance formula for a simple asymmetric dynamic programming algorithm in which each input frame occurs exactly once in the distance calculation. If the DP uses a squared Euclidean distance metric, the recognition process can be regarded as a special case of HMM Viterbi decoding, in which the word models have one state per template frame, and the features are assumed to be normally distributed with unit variance.

To clarify this relationship, we will return to the Viterbi calculation using logarithms of probabilities, given in Equation (9.52). The value of $b_j^L(\boldsymbol{y}_t)$ according to an uncorrelated normal distribution is given by Equation (9.53), where we are now assuming that $\sigma_{jk} = 1$ for all states $j$ and for all features $k$. Hence

$$\sum_{k=1}^{K} \log(\sigma_{jk}) = 0 ,$$

and recursive calculation of $\hat{\alpha}_j^L(t)$ simplifies to:

$$\hat{\alpha}_j^L(t) = \max_{\text{over } i} \left( \hat{\alpha}_i^L(t-1) + a_{ij}^L \right) - \frac{K}{2} \log(2\pi) - \frac{1}{2} \sum_{k=1}^{K} \left( y_{kt} - \mu_{jk} \right)^2 . \qquad (9.55)$$

The term $K/2 \log(2\pi)$ is a constant, which will scale the likelihood calculation but will not affect the choice of optimal state sequence. Thus the only quantities that need be considered at each frame are the logarithms of the transition probabilities

[5] This method was described by Kingsbury and Rayner (1971) for a completely different application.

and the square of the Euclidean distance between the observed features at time $t$ and the means for model state $j$.

As maximizing $\hat{\alpha}_j^L(t)$ is equivalent to minimizing $-\hat{\alpha}_j^L(t)$, this recognition task can be regarded as one of minimizing a distance comprising $-a_{ij}^L$, the negative logarithm of the transition probability (which must itself be positive), *plus* the squared Euclidean distance of the features from their model mean values. Thus we have a simple DP algorithm in which the $-a_{ij}^L$ terms are interpreted as timescale distortion penalties. Where only slopes of 0, 1, and 2 are permitted, as is the case for the HMM in Figure 9.1, the time distortion penalties for other values of slope are –log(0), and are therefore infinite.

## 9.14 STATE DURATIONAL CHARACTERISTICS OF HMMS

The probability of a model staying in the same state, $i$, for successive frames is determined only by the transition probability, $a_{ii}$. The expected number of frames it will stay in state $i$ is $1/(1-a_{ii})$, so a value of $a_{ii} = 0{\cdot}9$ would be suitable for using one state to model, for example, a steady fricative sound whose expected duration is 10 frames. Although the expected total duration in state $i$ in this case is 10 frames, the most likely duration is only one frame, with a probability of 0·1. The probabilities for longer durations decrease exponentially, as shown in trace (i) of Figure 9.2(b). This distribution is often referred to as a **geometric distribution** because the probabilities for successive numbers of frames form a geometric progression.

For any state representing a particular phonetic event, this type of duration distribution is obviously not sensible. For any such event there will be a most probable duration, with reducing probability for both shorter and longer durations. If many more states are available, the durational characteristics of the model can be improved, but only if a long steady region is modelled by a sequence of states with very similar feature p.d.f.s and the total likelihood method is used to calculate the word probability. For example, consider a group of four identical states with a repeat probability of 0·6, as shown in trace (ii) of Figure 9.2. The expected duration for the group is 10 frames, as it is for the single state shown in trace (i). However, in the case of the group of states, the variation of probability with duration is much more appropriate for speech sounds within a word. This more realistic distribution arises because, while there is only one possible way of going through the states in the minimum number of frames, there are more possible paths for longer frame sequences. However, the improved shape of duration distribution given by this state-splitting approach relies on using total likelihood probability calculations, whereas the Viterbi algorithm is generally used for recognition.

A simple method which can be used with the Viterbi algorithm involves merely imposing a minimum and a maximum duration on state occupancy. Such duration constraints can be achieved with an easy modification to the recognition algorithm, and can give worthwhile performance benefits. Many other methods have been proposed for improving the duration characteristics of HMMs, including some that model duration distributions of each state explicitly. These methods generally give greater benefits than simple duration constraints, but at the expense of more computation and some increase in the number of model parameters.

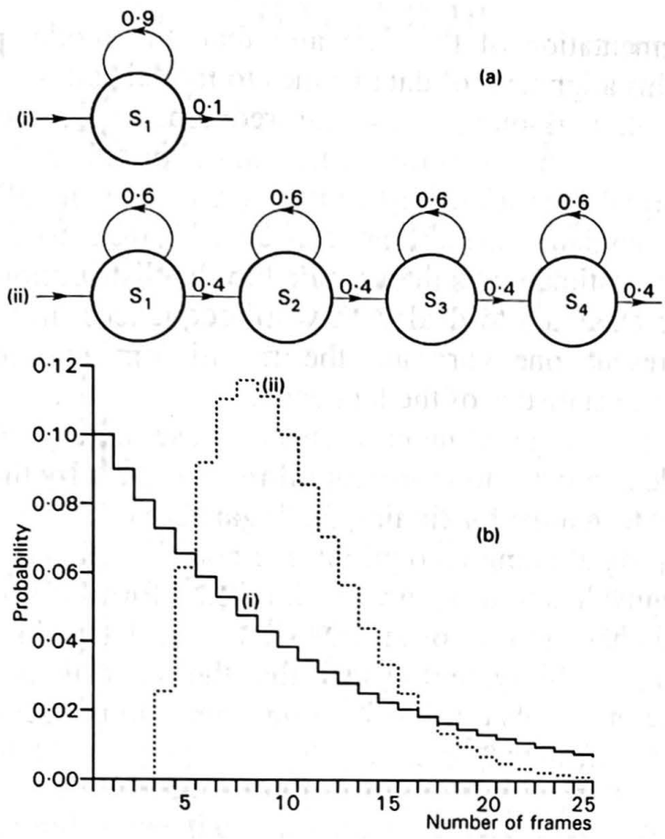



**Figure 9.2** **(a)** Two arrangements of states, each with an expected occupation time of 10 frames. **(b)** Probability of occupancy of groups of states in the model sections shown in (a).

## CHAPTER 9 SUMMARY

- The performance of pattern-matching speech recognizers is improved by representing typical characteristics of speech patterns in a way that also takes account of variability, which can be achieved by using a stochastic model of each word. Hidden Markov models (HMMs) represent each word as a sequence of states, with transition probabilities between each state and its permitted successors, and probability distributions defining the expected observed features for each state. A recursive formula can be used to calculate the probability that each word model will produce the observed data. The model with the highest probability is assumed to represent the correct word.
- Computation can be saved by using the Viterbi dynamic programming algorithm to calculate the probability of producing the data from only the most likely path through the states. This probability will always be less than the true probability, but the effect on recognition performance is usually very small and the Viterbi algorithm is generally adopted for HMM recognition.
- For each word model the transition probabilities and the probability distributions of the feature vectors can be found by the Baum–Welch re-estimation process. This process iteratively refines initial guesses to improve the model's representation of a set of training examples of the word, taking into account all possible paths through the states of the model.
- An alternative approach to estimating model parameters is to use a Viterbi training procedure, in which the initial guesses are used to find the most likely

state-level segmentation of the data and then the model parameters are re-estimated for this alignment of data frames to model states.

- Vector quantization is one method for reducing the set of possible feature vectors to a number for which robust training is possible. A parametric model, such as the normal (Gaussian) distribution or, more generally, a weighted sum (mixture) of Gaussian distributions, can also be used to describe the feature statistics. The re-estimation is then applied to the distribution parameters.
- HMMs can be extended to deal with word sequences, in which each state of the model represents one word, and the transition probabilities are determined by word sequence statistics of the language.
- One way of overcoming scaling problems because of very small numbers in the probability calculations is to represent all the numbers by their logarithms, and to use a special technique for finding the logarithm of the sum of two numbers.
- The dynamic programming recognition method described in Chapter 8 can be shown to be equivalent to using a very simplified form of HMM.
- The durational characteristic of an HMM state is determined only by the self-loop transition probability, and is such that the most likely duration is always only one frame and probabilities for longer durations decrease exponentially, so forming a geometric progression.

## CHAPTER 9 EXERCISES

**E9.1** What is the significance of the word 'hidden' in hidden Markov models?

**E9.2** Why is it not necessary to explicitly consider all possible state sequences when calculating the probability of an HMM generating observed data?

**E9.3** What is the essential difference between the Viterbi algorithm and the total likelihood method when calculating the probability of a word model generating observed data? What practical advantages can be gained by using the Viterbi algorithm for recognition?

**E9.4** How can the form of an HMM be constrained by choice of initial parameters provided for re-estimation?

**E9.5** What is the purpose of the 'vector quantization' sometimes used in HMMs?

**E9.6** What are the benefits of using normal distributions to model feature statistics for HMMs? What are the limitations of simple normal distributions and how can these be overcome?

**E9.7** How do the calculations required for Viterbi training differ from those for Baum–Welch re-estimation?

**E9.8** What are the practical difficulties associated with implementing forward and backward probability calculations? What solutions are usually adopted?

**E9.9** How can a simple HMM be interpreted as equivalent to a DTW recognizer?

**E9.10** Why are the state durational characteristics of HMMs not very appropriate for modelling speech? What are the effects on duration characteristics if a single state is replaced by a sequence of several identical states?

# CHAPTER 10

# Introduction to Front-end Analysis for Automatic Speech Recognition

## 10.1 INTRODUCTION

The term "front-end analysis" refers to the first stage of ASR, whereby the input acoustic signal is converted to a sequence of acoustic **feature vectors**. As explained in Section 8.3, the short-term spectrum provides a convenient way of capturing the acoustic consequences of phonetic events. Ideally the method of front-end analysis should preserve all the perceptually important information for making phonetic distinctions, while not being sensitive to acoustic variations that are irrelevant phonetically. As a general policy for ASR, it seems desirable not to use features of the acoustic signal that are not used by human listeners, even if they are reliably present in human productions, because they may be distorted by the acoustic environment or electrical transmission path without causing the perceived speech quality to be impaired. Over the years many different front-ends have been tried, for use first with DTW recognizers and, more recently, with HMM systems. These front-ends vary in the extent to which they incorporate knowledge about human auditory perception, but currently the most successful analysis methods include at least some of the known properties of perception. These successful methods are, however, also characterized by a compatibility with the mathematical techniques that are generally used in HMM recognizers (as will be explained later). In this chapter we will introduce various aspects of front-end analysis for ASR.

## 10.2 PRE-EMPHASIS

The spectrum of voiced speech is characterized by a downward trend, whereby frequencies in the upper part of the spectrum are attenuated at about 6 dB/octave. This downward trend is due to a combination of the typical –12 dB/octave slope of the glottal source spectrum with the +6 dB/octave lift given by the radiation effect due to the lips (see Chapter 2). For the purpose of front-end analysis, it is common to compensate by applying a **pre-emphasis** of 6 dB/octave so that the analysed signal has a roughly flat spectral trend. This pre-emphasis is easily applied to the speech signal as the first processing stage. Although the above argument for pre-emphasis only applies to voiced regions, in practice it is usually applied throughout without causing any obvious problems for the analysis of voiceless regions.

## 10.3 FRAMES AND WINDOWING

Due to physical constraints, the vocal tract shape generally changes fairly slowly with time and tends to be fairly constant over short intervals (around 10–20 ms). A

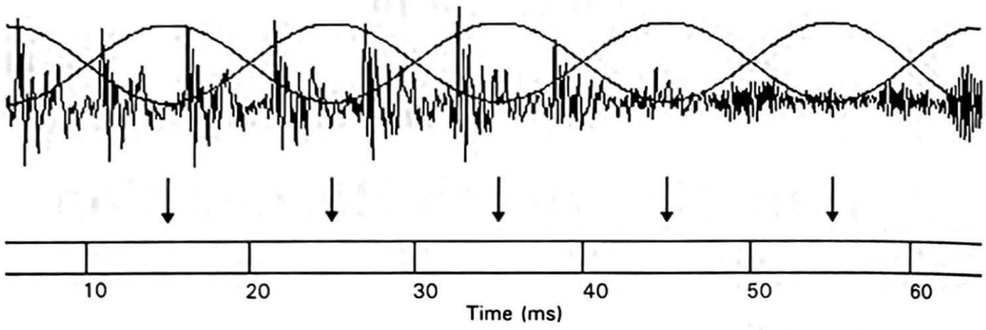



**Figure 10.1** Analysis of a speech signal into a sequence of frames. This example shows a 20 ms Hanning window applied at 10 ms intervals to give a frame rate of 100 frames/s.

reasonable approximation is therefore to analyse the speech signal into a sequence of **frames**, where each frame is represented by a single feature vector describing the average spectrum for a short time interval.

Prior to any frequency analysis, each section of signal is multiplied by a tapered **window** (usually a **Hamming** or **Hanning** window). This type of windowing is necessary to reduce any discontinuities at the edges of the selected region, which would otherwise cause problems for the subsequent frequency analysis by introducing spurious high-frequency components into the spectrum. The length of each analysis window must be short enough to give the required time resolution, but on the other hand it cannot be too short if it is to provide adequate frequency resolution. In addition, because the analysis is normally performed at a fixed time interval, during voiced speech the window must be long enough so that it is not sensitive to exact position relative to the glottal cycle (i.e. there needs to always be at least one complete glottal cycle in the main part of the window). Long windows also have the advantage of smoothing out some of the random temporal variation that occurs in unvoiced sounds such as fricatives, but at the expense of blurring rapid events such as the releases of stop consonants. A common compromise is to use a 20–25 ms window applied at 10 ms intervals (giving a **frame rate** of 100 frames/s and an overlap between adjacent windows of about 50%), as shown in Figure 10.1.

## 10.4 FILTER BANKS, FOURIER ANALYSIS AND THE MEL SCALE

In Section 8.3 we introduced a speech signal representation using a filter bank with channels whose bandwidth and spacing increase with frequency (motivated by psychophysical studies of the frequency resolving power of the human ear). A convenient implementation of filter-bank analysis involves applying a Fourier transform. The output of the Fourier analysis will usually be at a finer frequency resolution than is required, especially at high frequencies. Thus the Fourier magnitudes are summed into a smaller number of channels, whose bandwidth and spacing conform to a perceptual scale such as the Bark or mel scale (see Section 3.5). Typically no more than 20 such channels are used for speech with a 4 kHz bandwidth, with a few additional channels being needed for higher-bandwidth signals. As already explained in Section 8.3, it is advantageous for the filter-bank output to represent power logarithmically, which reflects the phonetic

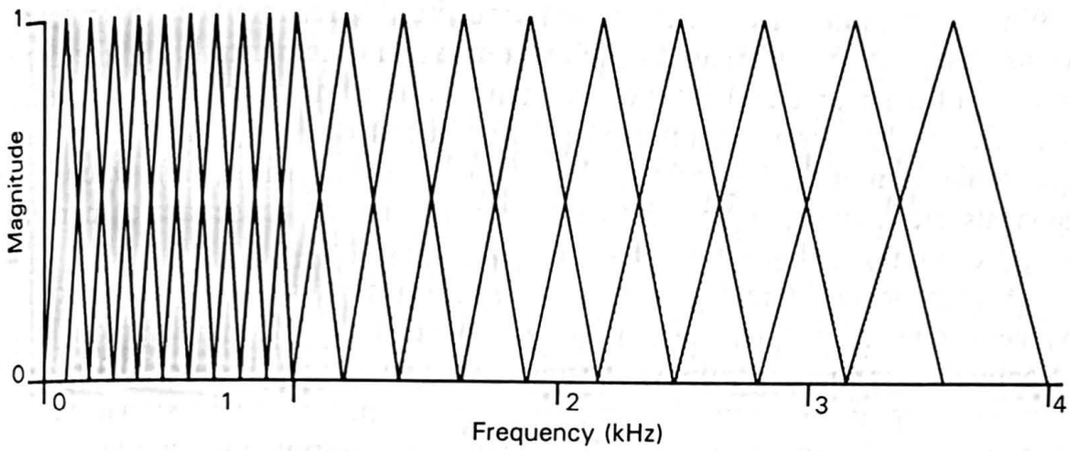



**Figure 10.2** Triangular filters of the type suggested by Davis and Mermelstein (1980) for transforming the output of a Fourier transform onto a mel scale in both bandwidth and spacing.

significance of level variations and accords with evidence of a similar compressive non-linearity in auditory systems (see Chapter 3). A consequence of the logarithmic compression is that, when sampled from representative speech over a long period of time, the distribution of the energy in each of the channels tends to follow a Gaussian distribution, and is therefore compatible with any Gaussian assumptions that are made in the modelling.

Figure 10.2 shows a set of triangular 'filters' that can be used to compute a weighted sum of Fourier spectral components, so that the output of the process approximates to a mel scale. Here the centre frequencies of the filters are spaced equally (at intervals of 100 Hz) on a linear scale from 100 Hz to 1 kHz, and equally on a logarithmic scale above 1 kHz. (Other slightly different spacings are also often used.) Each filter's magnitude frequency response is triangular in shape, and is equal to unity at the centre frequency and decreases linearly to zero at the centre frequencies of the two adjacent filters. This configuration of mel filters, which is now very widely used in ASR, was suggested by Davis and Mermelstein (1980).

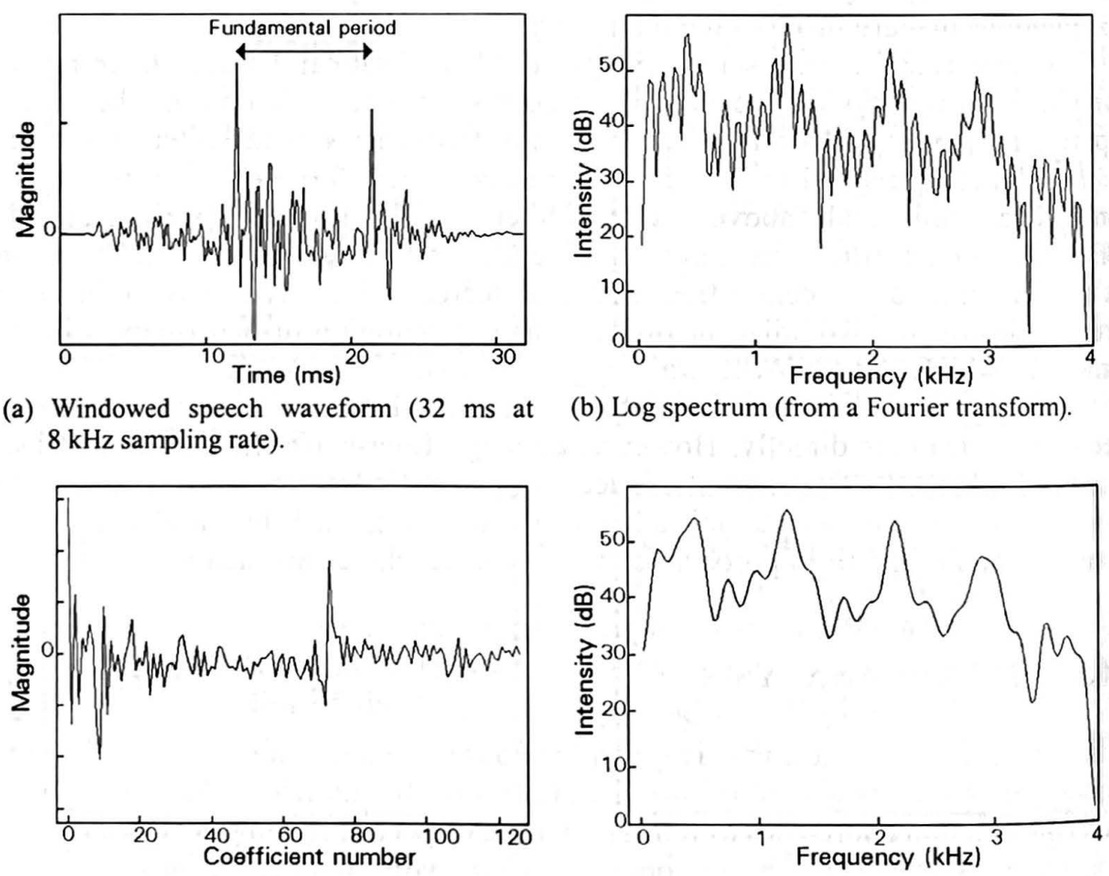
One option is to use the output of a filter-bank analysis to provide the recognition features directly. However, although filter-bank energies were widely used and achieved a fair amount of success as acoustic features in early recognition systems, there are substantial advantages to be gained by applying further transformations and this approach is the more usual choice nowadays.

## 10.5 CEPSTRAL ANALYSIS

The frequency resolution that is given by Fourier analysis applied to a 20–25 ms window of speech is generally sufficient to resolve the individual harmonics of the voiced excitation source, as well as showing the spectral shaping that is due to the vocal tract. Because the filtering operation of the vocal tract is the most influential factor in determining phonetic properties of speech sounds, it is desirable to separate out the excitation component from the filter component. The vocoders described in Chapter 4 are based on this principle. **Cepstral analysis** is another technique for estimating a separation of the source and filter components. Here the starting point is the observation that passing an excitation signal through a vocal-

tract filter to generate a speech signal can be represented as a process of **convolution** in the time domain, which is equivalent to multiplying the spectral magnitudes of the source and filter components. When the spectrum is represented logarithmically, these components are *additive*, because the logarithm of a product is equal to the sum of the logarithms ( $\log(A \times B) = \log(A) + \log(B)$ ). Once the two components are additive, it is relatively straightforward to separate them using filtering techniques.

A typical logarithmic spectrum cross-section shows the rapidly oscillating component due to the excitation superimposed on a more gradual trend representing the influence of the vocal tract resonances (see Figure 10.3(b)). If we now imagine that this combined shape represents a time-domain signal, the rapid oscillations would correspond to high-frequency components, while the more gradual changes would be due to low-frequency components. If a Fourier transform were applied, the two components would therefore appear at opposite ends of the resulting spectrum. Thus by starting with the log magnitude spectrum and computing a Fourier transform, to obtain the so-called **cepstrum** (an anagram of "spectrum"), the excitation is effectively separated from the vocal-tract filtering, as shown in Figure 10.3(c). In fact, because the log magnitude spectrum is a symmetric function, the Fourier transform can be conveniently simplified to a




(a) Windowed speech waveform (32 ms at 8 kHz sampling rate).

(b) Log spectrum (from a Fourier transform).

(c) Cepstrum computed from the log spectrum shown in (b).

(d) Log spectrum reconstructed from the first 40 cepstral coefficients in (c).

**Figure 10.3** Analysing a section of speech waveform to obtain the cepstrum and then to reconstruct a cepstrally smoothed spectrum.

**discrete cosine transform (DCT)**. For a spectral representation comprising $N$ channels with log magnitudes $A_1$ to $A_N$, the DCT can be computed as follows:

$$c_j = \sqrt{\frac{2}{N}} \sum_{i=1}^{N} A_i \cos\left(\frac{\pi\, j(i-0.5)}{N}\right) \quad \text{for } 0 \le j < N, \tag{10.1}$$

where $c_j$ is the $j^{\text{th}}$ **cepstral coefficient**. When $j = 0$, Equation (10.1) simplifies so that $c_0$ is proportional to the mean of the individual log channel signals $A_1$ to $A_N$. The $c_1$ term reflects the balance between energy at low frequencies and energy at high frequencies. As $j$ increases, $c_j$ captures increasingly fine spectral detail: first overall spectrum shape, then general formant structure, then more detailed spectral structure between the formants and, at high values of $j$, the excitation structure. There is no simple relationship between the $c_j$ terms and the formants. However, for periodic speech the effect of the excitation source tends to be seen as a clear 'spike' at the pitch period duration (see Figure 10.3(c)). Cepstral analysis is therefore one method that can be used to estimate fundamental frequency. For example, in Figure 10.3(c) the spike occurs at $c_{73}$ which, for the sampling frequency of 8 kHz (i.e. a sample duration of 0.125 ms), corresponds to a fundamental period of $73 \times 0.125 = 9.125$ ms. This value can be seen to be roughly equal to the interval between the pitch pulses in Figure 10.3(a).

Although the cosine transformation given by Equation (10.1) ensures that the Euclidean distance in transformed space is exactly equal to the distance between the sets of untransformed channel signals, the information that is of phonetic significance becomes concentrated in the lower-order terms. Filtering the cepstrum (a process usually referred to as **liftering**) can be applied to remove certain components or alter the relative influence of the different components. A simple lifter is one which simply truncates the cepstral sequence, by giving a weight of one to the low coefficients (up to some specified index) and a weight of zero to all the higher coefficients. By setting the cut-off point to just below the coefficient corresponding to the pitch period, most of the influence of the fundamental period is effectively removed from the spectrum. This process is shown in Figure 10.3(d), in which the spectrum has been re-constructed (by a Fourier transform) from just the low-order cepstral coefficients. The resulting spectrum can be seen to be much smoother than the original and show the formant peaks more clearly. The lower the cut-off point is set, the more detail will be removed and the smoother the spectrum will be. The process of smoothing the spectrum by truncating the sequence of cepstral coefficients is often referred to as **cepstral smoothing**.

The effectiveness of cepstral analysis for separating out the fundamental-frequency component of a speech signal depends on the frequency of the fundamental relative to the frequencies of the formants. The method generally works best for adult male speech (as shown in the example in Figure 10.3). For typical female and children's speech both the pitch and formant frequencies are higher, but the pitch increases more relative to the formant frequencies and so the cepstrum gives a less clear separation of the excitation component. It is therefore more difficult to set a cut-off point for cepstral smoothing that removes the pitch influence without also removing useful information about the formant structure.

In addition to the beneficial effect of concentrating on the information that is of greatest phonetic significance, discarding the high-order cepstral coefficients

reduces the number of features, so less computation is needed for the pattern-matching process.

The cepstrum has another desirable property for use in speech recognition. For typical speech signals it is found that, in contrast with the original channel signals, the variation of the separate coefficients tends to be uncorrelated. As a consequence, when using HMMs with continuous-density probability distributions, full covariance matrices can be replaced by much simpler diagonal covariance matrices (see Section 9.8.1) without any great loss in performance. Using diagonal covariance matrices substantially reduces both the computational requirements and the number of parameters needed to represent each distribution.

Cepstral coefficients have the property that (ignoring coefficients that are associated with pitch) both the variance and average numerical values decrease as the coefficient index increases (see Figure 10.3(c)). A consequence for a DTW recognizer using a simple Euclidean distance metric is that the distance calculation is affected most by the lowest-order coefficients and the coefficients that are more related to formant structure tend to be given insufficient weight. A solution that has often been adopted is to apply a lifter with a weighting for each coefficient that acts to roughly equalize the variances for the different coefficients. The problem does not arise when using probability distributions in HMM systems, because the variance is accommodated in the probability calculations. The liftering is often still applied, however, because the effect of making the variances of all the features cover a similar range makes it easier to study model parameters and to place restrictions on variances as part of re-estimation (see Section 11.4.1).

As explained above, the $c_0$ coefficient is proportional to the mean of the log channel signals and therefore provides an indication of overall level for the speech frame. Sometimes $c_0$ is included in the feature set, but often it is discarded and replaced by a different energy measure that is derived from the true signal energy. The energy in each frame will depend on overall speaking level, but for identifying sounds the most relevant factor is the *relative* level for different frames in an utterance. Therefore, for those applications for which the whole utterance becomes available before recognition needs to start, the measured energy is often normalized with respect to the maximum energy found over all frames in the utterance. (See Section 8.3.2 for further discussion about measures of speech level.)

In order to retain the advantages of a perceptually motivated filter-bank analysis, for ASR the cosine transform is usually applied to the output of non-linearly spaced filter-bank channels (see Section 10.4 above). A popular choice is to use **mel-frequency cepstral coefficients** (**MFCCs**), which are obtained by applying a DCT to the output of mel filters such as the ones shown in Figure 10.2. An acoustic representation using MFCCs is often simply referred to as a mel **cepstrum**. As explained above, it is generally advantageous to discard the higher-order coefficients. For example, with 8 kHz bandwidth speech, there might be 24 mel channels but only the first 12 MFCCs are generally used in the final feature set. Although the use of the non-linear filter-bank means that the cosine transform no longer gives a simple separation of the excitation from the vocal-tract filtering (and much of the excitation effect will usually have already been smoothed out by the mel averaging), the truncation of the cepstral sequence has a general spectral smoothing effect that is normally desirable because it tends to remove phonetically irrelevant detail.

## 10.6 ANALYSIS BASED ON LINEAR PREDICTION

An alternative to filter-bank methods for representing the short-term spectrum is to derive linear prediction (LP) coefficients (usually called LPC analysis because of its origin in linear predictive coding, see Chapter 4). In the past, mainly during the 1970s, many recognizers were built using LPC-derived features and these systems generally gave performance comparable with that obtained from recognizers using filter-bank methods. During the 1980s it became more popular to use LPC-derived cepstral coefficients rather than the LP coefficients themselves because, as in the case of the filter-bank representation, the addition of the cepstral transformation was found to improve recognition performance. A convenient method exists for computing cepstral coefficients directly from the LP coefficients. LP analysis has the advantage that it produces an estimate of the smoothed spectrum, with much of the influence of the excitation removed. However, there is less freedom to apply non-linear processing to combat noise than there is with a filter-bank front-end. In addition, LPC inherently gives uniform weighting to low- and high-frequency regions of the spectrum. A non-linear frequency scale can be incorporated, but complicates the analysis to a greater extent than when using filter-bank methods.

**Perceptual linear prediction (PLP)** (Hermansky, 1990) is one LP-based analysis method that successfully incorporates a non-linear frequency scale and other known properties from the psychophysics of hearing. In PLP analysis, a Fourier transform is first applied to compute the short-term power spectrum, and the perceptual properties are applied while the signal is represented in this filter-bank form. The spectrum is transformed to a Bark scale, and this spectrum is pre-emphasized by a function that approximates the sensitivity of human hearing at different frequencies (see Figure 3.5). The output is compressed to approximate the non-linear relationship between the intensity of a sound and its perceived loudness. The all-pole model of LPC is then used to give a smooth, compact approximation to the simulated auditory spectrum, and finally the LP parameters are usually transformed to cepstral coefficients for use as recognition features. Apart from the use of LPC to achieve spectral smoothing, PLP analysis is very similar to MFCC analysis, but with perceptual properties incorporated in a way that is more directly related to psychophysical results (see Table 10.1 for a comparison of the two methods). In recent years a number of recognition systems have used PLP-based cepstral coefficients as acoustic features, and experimental evidence suggests that overall they give performance that is comparable with that obtained using MFCCs.

**Table 10.1** Comparison between the properties of PLP cepstral coefficients and typical MFCCs.

| MFCCs | PLP cepstral coefficients |
|---|---|
| Cepstrum-based spectral smoothing | LPC-based spectral smoothing |
| 6 dB/octave pre-emphasis applied to speech waveform | equal-loudness pre-emphasis applied to spectrum |
| triangular mel filters | critical-band filters |
| logarithmic amplitude compression | cube root amplitude compression |

## 10.7 DYNAMIC FEATURES

In the HMM probability calculations (see Section 9.3), the probability of a given acoustic vector corresponding to a given state depends only on the current vector and the current state, and is otherwise independent of the sequence of acoustic vectors preceding and following the current vector and state. It is thus assumed that there is no dependency between the observations, other than through the underlying state sequence. In reality, however, an acoustic feature vector representing part of a speech signal is highly correlated with its neighbours. In fact, it is often the dynamic characteristics of the features that provide most information about phonetic properties of speech sounds (related to, for example, formant transitions or the closures and releases of stop consonants). These correlations can be captured to some extent by augmenting the original set of (**'static'**) acoustic features (such as MFCCs) with dynamic features that are a measure of the change in the static features. These dynamic features are often referred to as **time derivatives** or **deltas**. One way of computing the delta features is by simple differencing between the feature values for two frames either side of the current frame:

$$\Delta y_t = y_{t+D} - y_{t-D}, \tag{10.2}$$

where $D$ represents the number of frames to offset either side of the current frame and thus controls the width of the window over which the differencing operation is carried out. Typically $D$ is set to a value of 1 or 2.

Although time-difference features have been used successfully in many systems, they are sensitive to random fluctuations in the original static features and therefore tend to be 'noisy'. A more robust measure of local change is obtained by applying linear regression over a sequence of frames:

$$\Delta y_t = \frac{\sum_{\tau=1}^{D} \tau \left( y_{t+\tau} - y_{t-\tau} \right)}{2 \sum_{\tau=1}^{D} \tau^2} \tag{10.3}$$

With linear regression, a value of $D = 2$ is the usual choice for an analysis frame rate of 100 frames/s. This regression window of five frames (50 ms) is long enough to smooth out random fluctuations, yet short enough to capture local dynamics.

The delta features described above are first-order time derivatives, which can in turn be used to calculate second-order time derivatives (sometimes referred to as **delta-deltas**). Including first-order time derivative features usually gives a large gain in recognition performance, and adding second-order derivatives (which capture changes in the first-order dynamics) tends to give an additional but smaller improvement. The majority of current HMM systems incorporate first-order derivative features, most often applied to a basic feature set of MFCCs and an energy feature, and many also include second-order derivatives. Most of the benefit from derivative features is due to their ability to capture dynamic information. However, these features also have the useful property that they are not affected by any constant or slowly changing disturbances to the signal (such as linear filtering in microphone pre-amplifiers and on telephone channels, for example), provided that these distortions are additive in the feature domain (see Section 11.2).

## 10.8 CAPTURING THE PERCEPTUALLY RELEVANT INFORMATION

Both in Chapter 3 and at the beginning of the current chapter we explained the desirability of capturing properties of human phonetic perception in the front-end analysis for ASR. Analysis methods such as PLP take into account several known facts about the lower levels of human auditory processing. However, there is no attempt to model higher-level auditory processing or more specific properties of speech perception in any of the analysis methods that have been described above.

It is now well established that the frequencies of the speech formants, particularly the first and second, are vitally important phonetically. Relative formant amplitudes are much less important, and the detailed structure of the lower-level spectral regions between formants is of almost no consequence. There would therefore seem to be potential for better performance in ASR if these factors could be taken into account when designing acoustic analysis methods and distance metrics. Although auditory models have shown considerable promise for incorporating into systems for ASR (see Section 3.7), these types of features have not yet replaced more general spectral features such as MFCCs or PLP-cepstra as the preferred choice in HMM-based systems. It is possible that substantial changes in the design of the recognizers themselves will be required before it will be possible to gain the full benefit from incorporating auditory models. We will return to this issue in Chapter 16, when we will also discuss the prospects and issues for extracting and using formant information more explicitly in ASR.

## 10.9 GENERAL FEATURE TRANSFORMATIONS

The DCT is one orthogonal transformation that reduces the dimensionality of a filter-bank output by concentrating the most useful information into a small number of features. Other orthogonal transformations for data reduction include **principal components analysis (PCA)** and **linear discriminant analysis (LDA)**. PCA performs a linear transformation on an input feature set, to produce a different feature set of lower dimensionality in a way that maximizes the proportion of the total variance that is accounted for. LDA also applies a linear transformation on the input feature set, but here the transformation is chosen to maximize a measure of class separability, and hence to improve discrimination. In order to determine the transformation, this procedure requires each input feature vector to have first been associated with a single class. PCA and LDA are both general data-reduction techniques that can usefully be applied to reduce the dimensionality of any diverse feature set, including for example static spectral or cepstral features with first- and second-order time derivatives, or even the output of auditory models. Both PCA and LDA generate new feature sets that are uncorrelated, thus allowing diagonal covariance matrices to be used for HMM state emission p.d.f.s.

## 10.10 VARIABLE-FRAME-RATE ANALYSIS

It has been assumed so far that all the frames in an utterance are of equal importance when making a comparison with stored templates or models. However,

a slight difference of vowel quality, for example, may not affect the identity of a word, whereas formant transitions at vowel–consonant boundaries may be crucial in identifying the consonant. Because, for many consonants, such transitions are very rapid, they do not occupy many frames. Although the addition of time-derivative features increases the importance of matching the transition characteristics, still rapid transitions may make only a small contribution to the cumulative distance or probability, even when they are matched very badly. The vowels and steady-state parts of long consonants can, in contrast, make a large contribution overall even when they match fairly well on each frame.

To overcome this difficulty it is necessary to give more weight to parts of the signal that are changing rapidly, and less weight to long steady regions. One way that is sometimes used to achieve this effect is to perform the original acoustic analysis at a fairly high frame rate (e.g. 100–200 frames/s), but then to discard a variable proportion of the frames depending on the distance between consecutive pairs of frames. Thus all frames are retained in rapid transitions, but perhaps only one in five is kept in very steady long vowels. This **variable-frame-rate** analysis method is similar to the scheme described in Section 4.3.5 for efficient speech coding. In the case of speech analysis for ASR, not only is there a computational saving, but also the overall match of an input utterance to stored templates or models shows much greater relative sensitivity to mismatch in transition regions.

## CHAPTER 10 SUMMARY

- When deriving features for speech recognition, input speech is often first pre-emphasized by 6 dB/octave, so that the signal for subsequent analysis has a roughly flat spectral trend. Speech is analysed into a sequence of frames: most usually a 20–25 ms tapered window is applied at 10 ms intervals.
- One popular method of representing the speech spectrum is to use a filter bank with triangular filters whose width and spacing follow a mel scale. To obtain features for ASR, the output of such a filter bank is often subjected to a cosine transform (so deriving mel-frequency cepstral coefficients: MFCCs). An alternative is to derive cepstral coefficients from perceptual linear prediction.
- The cosine transform causes the features to become largely decorrelated so that diagonal covariance matrices can be used in the HMMs. In addition, information of phonetic significance is concentrated in the lower-order terms, so a more efficient representation can be obtained with fewer features.
- ASR performance is often greatly improved by adding 'delta' (first-order time-derivative) features, which are usually computed for each frame by applying linear regression over a window of five frames centred on the current frame.

## CHAPTER 10 EXERCISES

**E10.1** Why is cepstral analysis a useful tool in speech processing?

**E10.2** Explain the stages that are typically used to analyse a speech signal into MFCCs and their first- and second-order time derivatives.

**E10.3** How are properties of auditory perception simulated in front-ends for ASR?

# CHAPTER 12

# Automatic Speech Recognition for Large Vocabularies

## 12.1 INTRODUCTION

The previous four chapters have concentrated on introducing underlying theory and algorithms for ASR, together with some of the techniques for using the algorithms successfully in real situations. The discussion so far has deliberately concentrated either specifically on distinguishing between a small number of different words or on more general methods irrespective of the particular recognition task. In this chapter, we consider issues relevant to systems for recognizing continuously spoken utterances using large vocabularies, which may be anywhere from a few thousand up to around 100,000 different words.

## 12.2 HISTORICAL PERSPECTIVE

One of the earliest major efforts aimed at large-vocabulary ASR was initiated during 1971 in the United States by the Advanced Research Projects Agency (ARPA), with funding for a five-year programme of research and development. The overall objective was to make significant progress in the field of speech understanding by developing several alternative systems. The specific goal was to achieve a level of performance that was expressed in terms of semantic errors (less than 10%) on a continuous speech recognition task with a total vocabulary size of at least 1,000 words but using constrained-language input.

Although isolated-word recognition using pattern-matching techniques had achieved some initial success by the time of this ARPA programme, it was not generally obvious then how to extend the approach to accommodate the contextual effects that were known to occur in continuous speech. Therefore most systems adopted what at that time was the more traditional approach, using two separate stages. The first stage began by detecting **phonetic features** (e.g. formant frequencies, energy in different frequency bands, etc.) that were known to be important for distinguishing different speech sounds. Rules were used to convert from the measured features to a hypothesized phonetic transcription, which usually included some alternatives. The second stage then converted this transcription to a recognized word sequence. Inevitably there would be errors in the initial phonetic transcription, but the hope was that these errors would be corrected by the higher-level post-processing. However, in practice the first stage was so error-prone that information was lost which could not be recovered later. As a consequence, all the systems using this **knowledge-based** approach gave disappointing performance. In fact, the only system to achieve the required level of performance used a completely different method, based on a systematic search of a large network of

states with strong syntactic constraints, and it was one of the early large-vocabulary speech recognition systems using HMMs. The system was developed (somewhat separately from the main ARPA projects) at CMU by Lowerre (1976) as a Ph.D. project, extending the earlier pioneering work on HMMs by Baker (1975).

The results of the 1970s ARPA programme, while disappointing in terms of achievements for the money invested, provide a convincing demonstration of the benefits of **data-driven** statistical pattern matching over knowledge-based methods. In particular, the principle of delayed decision making is crucial, as it allows the overall best solution to be found incorporating all constraints, including those on construction of individual words and on allowed word sequences. This principle is fundamental to the design of all modern large-vocabulary speech recognizers.

Concurrent with the ARPA projects, research was in progress at IBM on the use of statistical methods for ASR. Early work was published by Jelinek (1976), independently of the work being carried out at CMU during the same period by Baker (1975). Work at IBM continued with an emphasis on applying HMMs to large-vocabulary speech recognition, and in the early 1980s the group focused on developing a system for dictation of office correspondence. The resulting system, "Tangora", as described by Jelinek (1985), was a speaker-dependent, isolated-word, near-real-time recognizer with a 5,000-word vocabulary. Although this system required users to leave pauses between words, it established the principles underlying the use of HMMs for a large-vocabulary task. Since the mid-1980s, further developments in many laboratories have led to significant further progress, and systems are now able to recognize fluent, naturally spoken continuous speech with very large vocabularies. There are a variety of systems for **large-vocabulary continuous speech recognition** (**LVCSR**) in existence, both as commercial products and as research systems in laboratories. At present, the successful systems are all based on HMMs, usually incorporating many of the refinements described in Chapter 11, but also with components that are specific to demands imposed by the need to cope with large vocabularies.

## 12.3 SPEECH TRANSCRIPTION AND SPEECH UNDERSTANDING

Large-vocabulary speech recognition tasks fall into two quite distinct categories:

1. *Speech transcription*: The user wishes to know exactly what the speaker said, in the form that it would be transcribed by an audio typist to produce orthographic text. Such a system may be used for dictation, and for tasks such as producing transcripts of broadcast news programmes.
2. *Speech understanding*: The semantic content of the message is required, and any recognition errors do not matter provided that the meaning is not changed. In fact often the real requirement is for the system to perform the correct action, irrespective of what words are recognized. Speech understanding systems may involve an interactive dialogue between a person and a machine to retrieve information from some computerized database. Other uses include automatic information extraction, for example to summarize spoken reports or broadcasts.

The interactive nature of many speech-understanding tasks, together with the fact that the subject area is often restricted, means that the relevant vocabulary at

any one point can be much smaller than the total vocabulary that is needed for more general transcription tasks. However, in order to interpret meaning of utterances, more detailed syntactic and semantic analyses are necessary than are required when just transcribing the words that were spoken. The principles of large-vocabulary recognition using HMMs apply both to transcription and to understanding, but the way in which the recognizer output is used is rather different. The first, main part of this chapter concentrates on transcription, while the latter part of the chapter briefly describes the use of large-vocabulary ASR in speech understanding systems.

## 12.4 SPEECH TRANSCRIPTION

The input speech waveform (typically sampled at 16 kHz) is first analysed into a sequence of acoustic feature vectors such as MFCCs (see Chapter 10). A popular choice is the first 12 cepstral coefficients and an overall energy feature together with first and second time derivatives of these features, giving a 39-element vector.

Once the input speech has been analysed into a sequence of feature vectors, the recognition task is to find the most probable word sequence $\hat{W}$ given the observed vector sequence $\boldsymbol{Y}$. Revisiting Bayes' theorem (see Section 9.2), but applying it to the task of finding a word *sequence*, the most probable sequence can be derived from the probability $P(W|\boldsymbol{Y})$ of any one sequence $W$ as follows:

$$\hat{W} = \arg\max_{W} P(W \mid \boldsymbol{Y}) = \arg\max_{W} \frac{P(\boldsymbol{Y} \mid W)P(W)}{P(\boldsymbol{Y})} = \arg\max_{W} P(\boldsymbol{Y} \mid W)P(W) \,. \quad (12.1)$$

Equation (12.1) states that the most likely word sequence is the one which maximizes the product of $P(\boldsymbol{Y}|W)$ and $P(W)$. The first term denotes the probability of observing vector sequence $\boldsymbol{Y}$ given the word sequence $W$, and is determined by an **acoustic model**. The second term represents the probability of observing word sequence $W$ independently from the acoustic signal, and is determined by a **language model**. Chapter 9 focused on the task of calculating acoustic-model probabilities, which is fundamental to any speech recognition system based on statistical models. However, for all but the most simple of applications, the language-model probability is also a major factor in obtaining good performance: restrictions imposed by the language model can greatly reduce the number of different alternatives to be distinguished by the acoustic model. As with the acoustic model, the language model for LVCSR is usually a statistical model that is automatically trained on data. In the case of the language model, these data usually take the form of *text* material chosen to be representative of the recognition task.

Assuming that models have been trained, Figure 12.1 illustrates a framework for classifying an unknown utterance by computing $P(\boldsymbol{Y}|W)P(W)$. The language model postulates a word sequence ("ten pots" in this example[1]) and determines its probability $P(W)$. In order to calculate the acoustic-model probability $P(\boldsymbol{Y}|W)$, a

[1] The phrase "ten pots" will be used in this chapter to illustrate a variety of different points. This phrase was chosen to provide a simple example for which the phonetic and orthographic transcriptions are very similar. For convenience of notation, we will represent the vowel in "pots" with its orthographic transcription /o/ in place of the correct phonetic notation for southern British English /ɒ/.

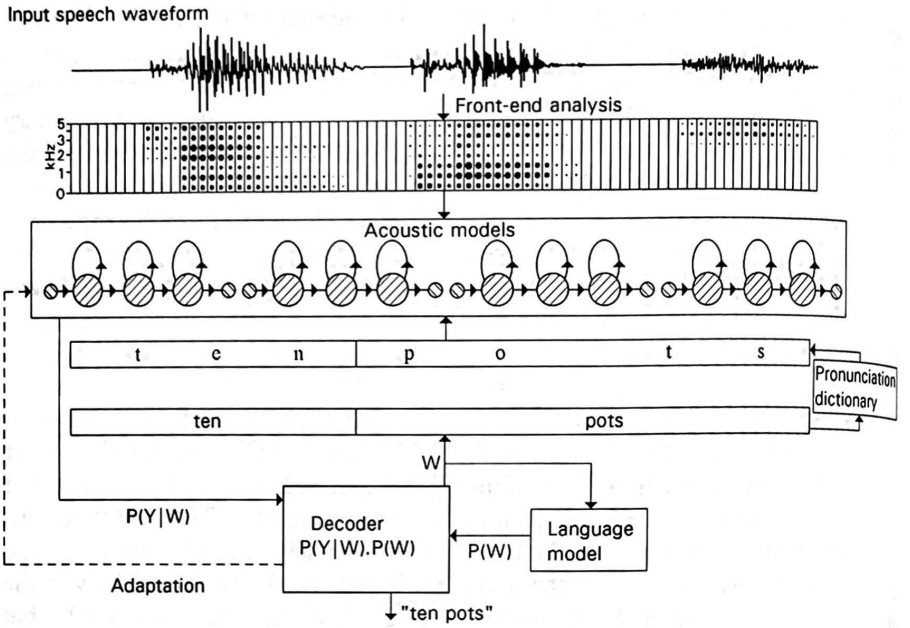



**Figure 12.1** Framework for decoding a speech signal by computing the probability of a word sequence in terms of language-model and acoustic-model probabilities, shown for recognition of the phrase "ten pots". A simple filter-bank analysis is shown here for clarity of illustration, although in practice other features such as MFCCs would be used. Due to space limitations, only four of the seven models needed to represent the phone sequence in "ten pots" are shown.

composite model for the word sequence is generated. Rather than having a separate HMM for each word, the component models represent phone-size units and a pronunciation dictionary is used to specify the sequence of models for each word in the vocabulary. For any word sequence, the dictionary is used to look up the required sequence of phone models for each word, and these phone models are concatenated together to form the model for the word sequence. The probability of that model generating the observed acoustic sequence is calculated, and this probability is multiplied together with the language-model probability. In principle, this process can be repeated for all possible word sequences allowed by the language model, with the most likely sequence being selected as the recognizer output. In practice, decoding for LVCSR requires a very efficient search strategy for evaluating the vast number of different possibilities, as will be explained later.

## 12.5 CHALLENGES POSED BY LARGE VOCABULARIES

Issues for the design of large-vocabulary recognition systems include the following:

1. In continuous fluent speech, there are many instances when words cannot be distinguished based on acoustic information alone and it is necessary to rely on a language model for discrimination. Difficulties in making acoustic distinctions arise for two main reasons. Firstly, due to co-articulation between adjacent words, word boundaries are not usually apparent in the acoustic signal. In some cases, two utterances may be linguistically different but acoustically very similar

or even identical (as in the "grey day" versus "grade A" example given in Chapter 1). Secondly, the pronunciation of many words, particularly function words, can be so reduced that there is very little acoustic information at all.

2. The memory and computation requirements can become excessive. In recent years, advances in computer technology have greatly reduced the impact of this limitation. However, memory and computation are still influential, especially in determining the choice of search mechanism for use in decoding.
3. As the vocabulary size increases, it becomes increasingly difficult to provide enough representative examples of all the words, both as text to train the language model and as spoken examples to train the acoustic model.

Many of the design features of modern LVCSR systems are determined by the need to deal with these issues. The design of the acoustic model, the language model and the decoding operation are all crucial factors for the success of an LVCSR system. The following three sections describe each of these three components in turn.

## 12.6 ACOUSTIC MODELLING

Although some early systems used HMMs with discrete distributions for their emission p.d.f.s (e.g. Lee (1989)), current systems generally use fully continuous distributions or tied-mixture distributions, usually with diagonal covariance matrices. These latter types will be the focus of the explanation given here, which is based mainly on descriptions of the research system developed at Cambridge University (e.g. Young (1996)). This system is one of the most successful systems to date, but there are many other systems that have fairly similar structure and give broadly comparable performance, although they differ in various details.

The need to make the best use of any available acoustic training data has important consequences for the design of the acoustic-model component. With a large vocabulary, it is impractical to expect any one person to provide enough examples to train models for all the words from scratch, even if the system is intended for speaker-dependent operation. Therefore, a speaker-independent model set is used, at least to provide a starting point. Speaker-adaptation techniques are often used to improve performance for any one individual. Unsupervised adaptation may be performed using the recognizer output, as shown by the dotted data path in Figure 12.1. In addition, for a system to be used by one known person, that person can be required to speak some specific utterances, which can be used for supervised model adaptation before the person uses the system to perform any real task.

Even with several speakers to provide the data, it is not practical to train a separate model for each word in a large-vocabulary system. Even if it were practical, this approach would not make the best use of the data, as it does not take account of the fact that different words can share sub-components. Therefore large-vocabulary systems are based on **sub-word** models. The usual method, as shown in Figure 12.1, is to use models of phone-size units, with the sequence of phones for each word being specified in a pronunciation dictionary. Thus, the requirement for the training is to provide sufficient examples of all the phone-size units, and all the words in the vocabulary will not necessarily have occurred in the training data. In fact, provided suitable models are available, words can be added to the vocabulary at any time simply by extending the pronunciation dictionary.

### 12.6.1 Context-dependent phone modelling

As approximately 44 phonemes are needed to represent all English words, this number of models would be the minimum needed to build word models for English. However, the effects of co-articulation are such that the acoustic realization of any one phoneme can vary greatly with acoustic context. Therefore **context-dependent** HMMs are generally used, with different models for different phonetic contexts. Additional variation tends to arise because many speakers will, either consistently or occasionally, use word pronunciations that are different from those given in the dictionary. Although alternative pronunciations can be included in the dictionary, it is difficult to include every possible pronunciation and any that are not covered will need somehow to be accommodated in the chosen set of context-dependent HMMs.

The simplest and most popular approach is to use **triphones**, whereby every phone has a distinct HMM for every unique pair of left and right neighbours. For example, consider the word "ten". When spoken in isolation, this word could be represented by the sequence `sil t e n sil`, with the `sil` model being used for silence at the start and end. Using triphones, with the notation $_{x}y_{z}$ to denote phone `y` preceded by phone `x` and followed by phone `z`, the word would be modelled as

$$\texttt{sil}\ {}_{\texttt{sil}}\texttt{t}_{\texttt{e}}\ \ {}_{\texttt{t}}\texttt{e}_{\texttt{n}}\ \ {}_{\texttt{e}}\texttt{n}_{\texttt{sil}}\ \ \texttt{sil}.$$

Now consider the phrase "ten pots", for which the triphone sequence would be

$$\texttt{sil}\ {}_{\texttt{sil}}\texttt{t}_{\texttt{e}}\ \ {}_{\texttt{t}}\texttt{e}_{\texttt{n}}\ \ {}_{\texttt{e}}\texttt{n}_{\texttt{p}}\ \ {}_{\texttt{n}}\texttt{p}_{\texttt{o}}\ \ {}_{\texttt{p}}\texttt{o}_{\texttt{t}}\ \ {}_{\texttt{o}}\texttt{t}_{\texttt{s}}\ \ {}_{\texttt{t}}\texttt{s}_{\texttt{sil}}\ \ \texttt{sil}.$$

The two instances of the phone [t] are represented by different models because their contexts are different. Note that the triphone contexts span word boundaries, so that the first and last triphones used to represent a word depend on the preceding and following words respectively. For example, if the phrase were "ten dogs", the last triphone used to model "ten" would be ${}_{\texttt{e}}\texttt{n}_{\texttt{d}}$ rather than ${}_{\texttt{e}}\texttt{n}_{\texttt{p}}$. This use of **cross-word triphones** enables co-articulation effects across word boundaries to be accommodated, but creates complications for the decoding process as the sequence of HMMs used to represent any one word will depend on the following word.

The decoding task can be greatly simplified by using only **word-internal triphones**, whereby 'word boundary' acts as a context and so the sequence of HMMs is fixed for each word. Thus, in the above example the triphones ${}_{\texttt{e}}\texttt{n}_{\texttt{p}}$ and ${}_{\texttt{n}}\texttt{p}_{\texttt{o}}$ would be replaced by ${}_{\texttt{e}}\texttt{n}_{-}$ and ${}_{-}\texttt{p}_{\texttt{o}}$ respectively, with – being used to represent a word boundary. Early triphone systems were restricted to word-internal triphones, but the inability to model contextual effects across word boundaries is a serious disadvantage and current systems generally include cross-word context-dependent models. The consequences for decoding are explained in Section 12.8.

### 12.6.2 Training issues for context-dependent models

For a language with 44 different phones, the number of possible triphones is $44^3$ = 85,184. In fact, phonotactic constraints are such that not all of these triphones can occur. However, an LVCSR system which includes cross-word triphones will still typically need around 60,000 triphones. This large number of possible triphones poses problems for training the models:

- The total number of parameters needed for the models is very large: it is usual to use three-state models with somewhere in the region of 10 mixture components to represent the output distribution for each state. This number of mixture components tends to be needed to represent the wide range of speakers (including a range of different accent types) who must be accommodated within a single model. Assuming that diagonal covariance matrices are used with 39-element acoustic feature vectors and 10 mixture components, each state would require around 790 parameters ($39 \times 10$ means, $39 \times 10$ variances, and 10 mixture weights). Thus 60,000 three-state models would have a total of over 142 million parameters. Any feasible quantity of training data would not be large enough to train this number of parameters adequately.
- In any given set of training data, many triphones will inevitably not occur at all, especially if cross-word triphones are allowed (as it is very difficult to include all the phone combinations that might occur in all possible sequences of words). Thus some effective method is required for generating models for these **unseen triphones** that do not occur in the training data.

Similar issues have already been mentioned as difficulties with using whole-word models for large vocabularies. However, when using smaller model units that are meaningful in phonetic terms, it is easier to see how the problems can be reduced. The challenge is to balance the need for detail and specificity in the models against a requirement to have enough training data to obtain robust parameter estimates. To tackle the problem various different **smoothing** techniques have been used:

1. *Backing off*: When there are insufficient data to train a context-specific model, it is possible to **back off** and use a less-specific model for which more data are available. For example, one approach is to replace a triphone by the relevant **biphone**[2], representing the phone dependent on only one adjacent context, which may be either to the left or to the right. Given a choice between the left or the right biphone context, it is generally better to choose the right context as articulation tends to be anticipatory, such that following context has a greater influence than preceding context. If there are insufficient examples to train a biphone, it is possible to resort to the context-independent phone model, or **monophone**. The backing-off mechanism ensures that every model in the final system is adequately trained, but can result in only a relatively small number of full triphone models, so that several contexts are not modelled very accurately.
2. *Interpolation*: A greater degree of context dependency can be retained by **interpolating** the parameters of a context-specific (triphone) model with those of a less-specific model (such as a biphone or monophone), to give model parameters which represent some compromise between the two sets. Thus some of the context dependency of the original triphone models is preserved, while increasing their robustness by taking into account additional (less specific) data.
3. *Parameter sharing*: An alternative is to take all the triphones representing any one phone, apply some form of **clustering** procedure to group the individual

[2] Note that the term **biphone** is used to refer to a phone that is dependent upon a single context (either left or right), and that this unit is different from the **diphone** unit discussed in Chapter 5, which represents the region from the middle of one phone to the middle of the next.

models (or parts of models) into clusters with similar characteristics, and share the parameters between them. This sharing of parameters, often referred to as **parameter tying**, allows the data associated with similar states to be pooled to improve the robustness of the parameter estimates. This approach can retain a higher degree of context specificity than is possible with the first two methods.

Although both backing off and parameter interpolation have been used with some success, the greater power of more general parameter sharing to obtain a better compromise between accuracy and robustness is such that this method is now widely used in LVCSR. The method is described in more detail below.

### 12.6.3 Parameter tying

The technique of tying provides a general mechanism for sharing parameters between models or parts of models. One example is provided by the tied-mixture distributions introduced in Chapter 9, where the means and variances of each mixture component are tied across all model states. Smoothing the parameters of context-dependent models represents another application of tying. Here tying is usually applied to all model parameters for a subset of the triphones representing a phone. The aim is to tie together those models or states for which any differences in the acoustic realizations are not significant for phonetic discrimination.

Initial developments in parameter sharing between context-dependent models concentrated on clustering together triphone *models*, to give **generalized triphones**. However, this approach assumes that the degree of similarity between any two models is the same for all the states in the models. In fact, the different effects of left and right context are such that this assumption is rarely justified. For example, consider three triphones of /e/: $_{t}e_{n}$, $_{t}e_{p}$ and $_{k}e_{n}$. The first state of the $_{t}e_{n}$ and $_{t}e_{p}$ triphones can be expected to be very similar, while the last state of the $_{t}e_{n}$ and $_{k}e_{n}$ triphones will be similar. Thus tying at the state level offers much more flexibility to make the most effective use of the available data for training a set of models. We will now consider two important issues associated with the use of state tying: firstly the general procedure used to train the tied-state multiple-component mixture models (assuming that it is known which states to tie together), and secondly the choice of clustering method used to decide on the state groupings. The discussion focuses on state tying, but the principles apply in the same way when the tying is applied to complete models.

### 12.6.4 Training procedure

Careful design of the training procedure is essential to maximize the robustness and accuracy of the final set of tied-state context-dependent HMMs. When training sub-word models, it is not usual for the individual speech segments to have been identified and labelled in the available training data. In fact it is most likely that the data will have been transcribed as a sequence of words but not segmented at all. Rather than attempting to segment these data, they can be used directly for parameter estimation by adopting the **embedded training** approach described in Section 9.11. When using sub-word models, it is necessary first to construct a

model for each word from the sub-word units, before then constructing a model for the whole utterance from the individual words. Similarly to the procedure outlined in Section 12.4 for recognition, the pronunciation dictionary is used to look up the phone sequence required to represent each training utterance. A composite HMM is constructed by concatenating the appropriate sub-word models, and the relevant statistics for re-estimation are accumulated over all occurrences of each model.

Phone sequence constraints across different utterances are such that the embedded training method should generally be effective in associating appropriate speech frames with each model state, provided that in the early stages of training each model is used in a sufficient range of different contexts. For this situation it is even adequate to use the very simple 'flat' initialization of all the model parameters to identical values (see Section 9.9). It is usual to start with single-Gaussian distributions and train simple monophone models. Because there are very many examples of each one, these monophones can be trained very robustly, and provide a good basis for initializing the more specific context-dependent models. A typical procedure for training context-dependent models is illustrated in Figure 12.2, and summarized below:

1. A set of monophone HMMs, using single-Gaussian output distributions with diagonal covariance matrices, is created and trained.
2. All the training utterances are transcribed in terms of the complete set of possible triphones. For each triphone, an initial model is created by cloning the appropriate monophone. The transition probability matrix is typically not cloned, but is tied across all triphones of a phone. The triphone model parameters are re-estimated and the state occupancies, which represent the expected number of observations used to estimate the parameters of each triphone (see Section 9.5.2), are retained for later use.
3. For the set of triphones representing each phone, similar states are clustered together to create tied states (see Sections 12.6.5 and 12.6.6 for more explanation). As part of the state tying process, it is important to check that there are sufficient data associated with each tied state. This situation can be achieved by only allowing clusters for which the total state occupancy (i.e. the estimated 'count' of number of frames for which the state is occupied) exceeds a suitable threshold value (typically around 100). The parameters of the tied-state single-Gaussian models can then be re-estimated. The use of tying does not alter the form of the re-estimation formulae and can be made transparent to the re-estimation process if the data structures used to store the information are set up appropriately. Storage can be set up for accumulating the numerator and denominator for re-estimating each parameter of each tied state, with individual states simply pointing to the relevant storage.
4. Finally, multiple-component mixture models are trained using the iterative mixture splitting procedure explained in Section 9.10.4.

Delaying the introduction of the multiple-component Gaussians until the final stage has a number of advantages:

- The difficulties associated with training untied triphone mixture distributions are avoided, as multiple mixture components are only introduced once the model inventory has been set up to ensure adequate training data for each state.
- The state tying procedure is simplified because, by using single-Gaussian

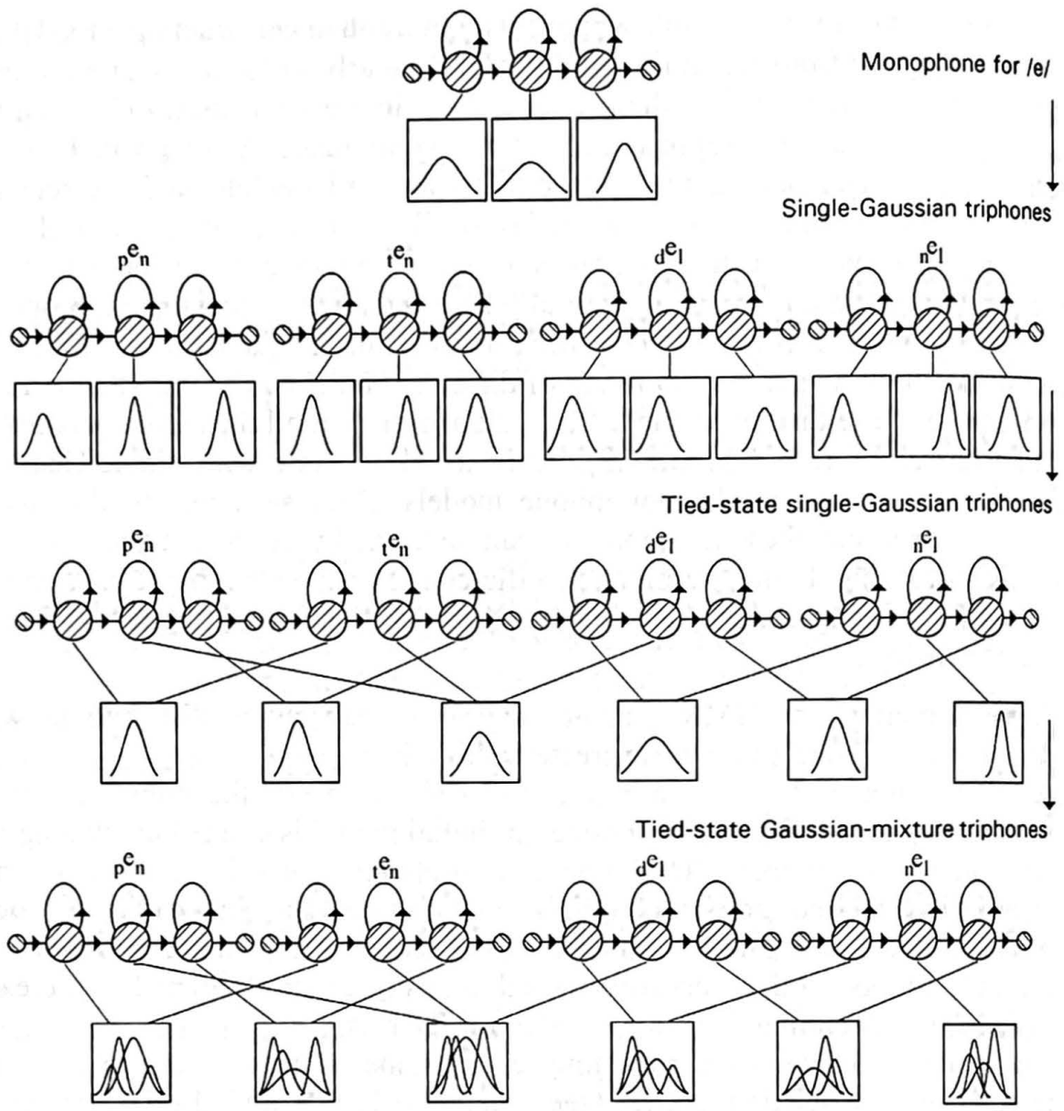



**Figure 12.2** Sequence of stages for training tied-state Gaussian-mixture triphones, illustrated for a group of triphones representing the /e/ phoneme.

distributions, it is much easier to compute a similarity measure and to calculate the parameters of the combined states (see Section 12.6.6).

- By not introducing the mixture distributions at the monophone stage, the process avoids potential complications that could arise if the mixture components were used to accommodate contextual variation which would at a later stage be covered by the context-dependent models. It is generally better to accommodate contextual influences explicitly so that the predictable nature of these effects can be exploited as far as possible. The multiple mixture components are then needed mainly to allow for the fact that any one model represents data from a wide variety of different speakers.

In addition to the benefits in terms of robustness, computation and storage, state tying has the potential to lead to models with better discrimination. The potential advantages of sub-word over whole-word models in terms of discrimination power have already been mentioned. These arguments extend to the use of state tying. If the differences between the acoustic realizations associated with two different model states are simply a consequence of random variation, it is

detrimental for these differences to be included in the models. By combining them into a single model state, discrimination will be more focused on those regions of words containing the most useful acoustic cues. This benefit is dependent upon finding an appropriate method for determining which states to tie together.

### 12.6.5 Methods for clustering model parameters

Clustering methods for grouping together similar states can be divided into two general types. These methods can be used to cluster triphone states as follows:

1. *Bottom-up clustering*: Starting with a separate model for each individual triphone, similar states are merged together to form a single new model state. The merging process is continued until some threshold is reached which ensures that there are adequate data to train each new clustered state. This data-driven approach is often referred to as **agglomerative clustering**. The method should ensure that there are sufficient data to train every state in the final set, while keeping the models as context-specific as possible given the available training data. However, for any triphones that do not occur at all in the training data, it is still necessary to back off to more general models such as biphones or monophones.
2. *Top-down clustering*: Initially all triphones for a phoneme are grouped together and a hierarchical splitting procedure is used to progressively divide up the group according to the answers to binary yes/no questions about either the left or the right immediately adjacent phonetic context. The questions are arranged as a **phonetic decision tree**, and the division process starts at the root node of the tree and continues until all the leaf nodes have been reached. All the states clustered at each leaf node are then tied together. A tree is generated for each state of each phone. An example showing the use of a decision tree to cluster the centre state of some /e/ triphones is shown in Figure 12.3. The context questions in the tree may relate to specific phones (e.g. "Is the phone to the right /l/?"), or to broad phonetic classes (e.g. "Is the phone to the left a nasal?"). Using the tree, the correct tied state to use for any given context can be found by tracing the path down the tree until a leaf node is reached (see Figure 12.3).

The main advantage of the top-down approach to clustering is that a context-dependent model will be specified for *any* triphone context, even if that context did not occur in the training data. It is thus possible to build more accurate models for unseen triphones than can be achieved with the simple backing-off strategy, assuming that the questions in the tree are such that contexts are grouped appropriately. Although the tree could be constructed by hand based on phonetic knowledge, this approach does not work very well in practice, as it does not take into account the acoustic similarity of the triphones in the data. It is, however, possible to construct trees automatically by combining the use of phonetic questions with tests of acoustic similarity and a test for sufficient data to represent any new division. This automatic construction provides generalization to unseen contexts while maintaining accuracy and robustness in the acoustic models. A popular version of this effective technique for constructing phonetic decision trees is explained in more detail in the next section.

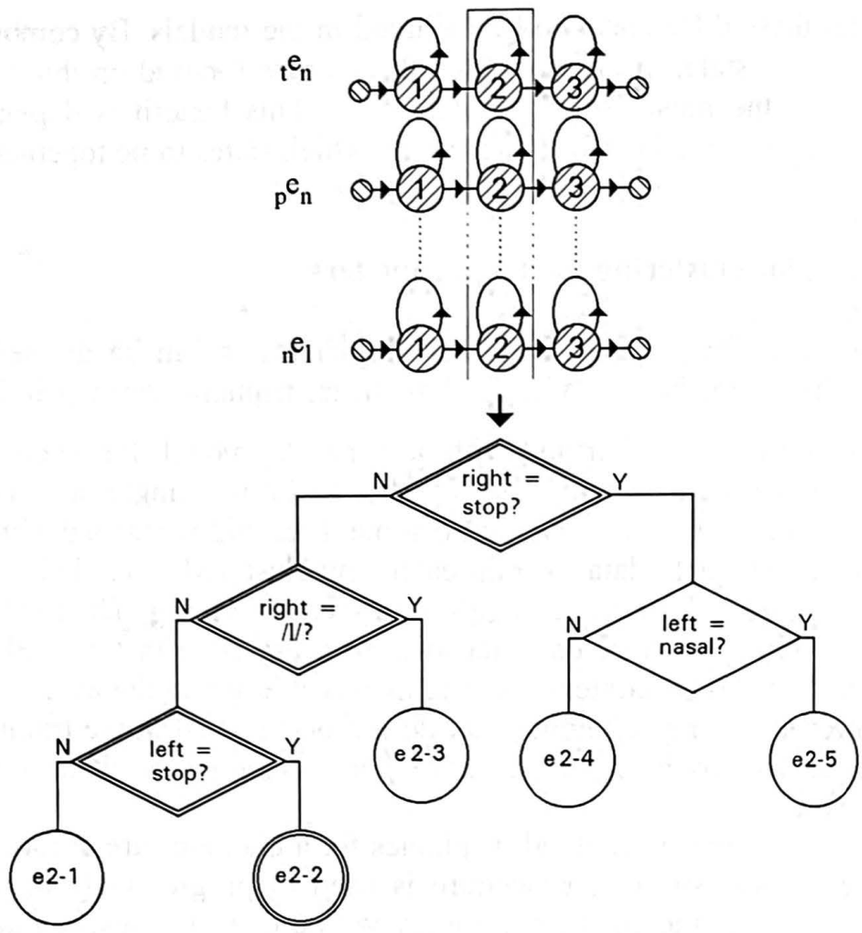



**Figure 12.3** Example illustrating the use of a phonetic decision tree to cluster the centre state for a group of triphones of the /e/ phoneme. Each triphone is moved down the tree until a terminal 'leaf' node is reached (shown as circles), and a new model state is formed from the members of the cluster. The tree shown here is a very simple one intended simply as an illustration of the principles of clustering, and in any real application there will be many more questions before a terminal node is reached. The tree can then be used to find the appropriate clustered state for any given context. As an example, the route down this simple tree is shown (using double lines round the chosen decision boxes and the final leaf node) for the context of a preceding /t/ and a following /n/. In this context the clustered state e2-2 will be used.

### 12.6.6 Constructing phonetic decision trees

First, linguistic knowledge is used to choose a set of possible context questions that might be used to divide the data. This question set will usually include tests for each specific phone, tests for phonetic classes (e.g. stop, vowel), tests for more restricted classes (e.g. voiced stop, front vowel) and tests for more general classes (e.g. voiced consonant, continuant). Typically, there will be over 100 questions for the left context and a similar number for the right context. For each state of each phone, the aim is to build a tree where the question asked at each node is chosen to maximize the likelihood of the training data given the final set of tied states. A condition is imposed to ensure that there are sufficient data associated with each tied state (i.e. that the total occupancy of the tied state exceeds some threshold).

It would in principle be possible to build all possible tree architectures (for all states), train a set of models for each architecture, and choose the set for which the likelihood of the training data is the highest while satisfying the occupancy

condition for each of the final tied states. This strategy would, however, be computationally intractable. Fortunately, by making the assumption that the assignment of acoustic observations to states is unchanged from the assignment for the original triphone models, it is possible to build a tree in a computationally efficient manner using just the state occupancies and the triphone model parameters. When using single-Gaussian distributions, this information is sufficient to calculate new model parameters for any putative combination of the individual triphone states.

The tree-building process starts by placing all the states to be clustered together at the root node of the tree. The mean and variance vectors are calculated assuming that all the states $\boldsymbol{S}$ are tied. Using these values of the model parameters it is then possible to estimate $L(\boldsymbol{S})$, the likelihood of the data associated with the pool of states $\boldsymbol{S}$. The next step is to find the best question for splitting $\boldsymbol{S}$ into two groups. For each question $q$, the states are divided up according to the answer to the question and new model parameters are computed. The likelihoods $L(\boldsymbol{S}_y(q))$ and $L(\boldsymbol{S}_n(q))$ can then be calculated for the sets of data corresponding to the answers "yes" and "no" respectively. For question $q$ the total likelihood of the data associated with the pool of states will increase by:

$$\Delta L_q = L(\boldsymbol{S}_y(q)) + L(\boldsymbol{S}_n(q)) - L(\boldsymbol{S}) . \tag{12.2}$$

Thus by computing $\Delta L_q$ for all possible questions, the question for which this quantity is the maximum can be selected. Two new nodes are created and the splitting process is then repeated for each of the new nodes, and so on. The splitting procedure is terminated when, for all of the current leaf nodes, the total occupancy of the new tied state which could be created at that node falls below the designated occupancy threshold. An additional termination condition is also used, whereby the splitting is halted when the increase in likelihood which would result from a split falls below a specified likelihood threshold. This second termination condition avoids the unnecessary use of different models for contexts which are acoustically similar (even if sufficient data are available for separate models to be used).

Once the tree has been constructed, this tree can be used to accomplish the state tying required for step 3 of the training process described in Section 12.6.4.

### 12.6.7 Extensions beyond triphone modelling

A useful feature of the phonetic decision tree approach is that it can be extended beyond simple triphone contexts. For example, decision trees can be built using questions relating to the next-but-one left and right contexts as well as the immediately adjacent contexts. The resulting models are often referred to as **quinphones**, as they can incorporate information over a sequence involving up to five phones. Questions relating to the presence of word boundaries can also be included. When building these complex decision trees with such a large number of possible contexts, it is not usually practical to start by training a fully context-dependent system because such a system would typically require a vast number of models and state occupancies to be stored. It is preferable to begin with some more manageable model set and use Viterbi alignment to provide a state-level segmentation of the data. If a tied-state triphone system has been built first, this

system can be used to provide a good alignment upon which to base the derivation of a decision tree for a word-boundary-dependent quinphone system.

As well as incorporating context dependency in the acoustic models, it can be beneficial to deal with male and female speech separately, because the differences between male and female speech tend to be much greater than the differences between talkers of the same sex. Models built in this way are commonly described as **gender dependent**. One way of training gender-dependent models robustly is to introduce gender dependency at the final training stage by cloning the trained multiple-component mixture models and then re-estimating the state means and mixture weights for each model set using the data for the male and female speech separately. State variances are usually kept gender independent in order to avoid the robustness issues that would otherwise arise due to the more limited data available to train variances for each gender separately. Once the gender-dependent models have been trained, a straightforward method for using them in recognition is to run two recognizers in parallel, one using the 'male' models and the other using the 'female' models. The system output is then taken from the recognizer which gives the highest probability for an utterance (where the utterance represents a sequence of words known to have been spoken by the same person).

## 12.7 LANGUAGE MODELLING

In any language, there are syntactic, semantic and pragmatic constraints that have the effect of making some sequences of words more likely than others. For an ASR system that is intended for a particular application domain, the language that the system can recognize may be quite limited, in terms both of vocabulary size and of utterance syntax. However, for more general recognition of large vocabularies, *any* word sequence must be allowed, but different probabilities need to be assigned to different sequences. The purpose of the language model is to make effective use of linguistic constraints when computing the probability of the different possible word sequences. Assuming a sequence of $K$ words, $W = w_1, w_2, \ldots, w_K$, the sequence probability $P(W)$ can be expanded in terms of conditional probabilities as follows:

$$P(W) = P(w_1, w_2, ..., w_K) = \prod_{k=1}^{K} P(w_k \mid w_1, ..., w_{k-1}) . \tag{12.3}$$

This expression simply states that the probability of the word sequence can be decomposed into the probability of the first word, multiplied by the probability of the second word given the first word, multiplied by the probability of the third word given the first and the second words, and so on for all the words in the sequence. Initially, the probabilities of the words will be influenced mostly by general constraints on the types of words that are most likely to start utterances. As the utterance continues, an increasing number of words will have already been spoken and so it becomes easier to predict the next word.

For any natural language, there is of course a vast number of possible word combinations and hence a huge number of conditional probabilities that might be required. It is not feasible to specify all of these probabilities individually, and so some modelling assumptions are needed to make the task of specifying the conditional probabilities more manageable.

### 12.7.1 *N*-grams

A simple solution to the problem of estimating word-sequence probabilities is to use ***N*-grams** (where $N$ is a small number). Here it is assumed that the probability of observing any word $w_k$ depends only on the identity of the previous $N-1$ words (so that each conditional probability depends on a total of $N$ words, including the current word). Using $N$-grams Equation (12.3) can be approximated as follows:

$$P(W) = \prod_{k=1}^{K} P(w_k \mid w_1, \ldots, w_{k-1}) \approx \prod_{k=1}^{K} P(w_k \mid w_{k-N+1}, \ldots, w_{k-1}). \tag{12.4}$$

If $N=1$ the model is a **unigram** and just represents the probability of the word. A **bigram** ($N=2$) models the probability of a word given the immediately preceding word, while a **trigram** ($N=3$) takes into account the previous two words.

To illustrate the use of $N$-grams to estimate word-sequence probabilities, consider the phrase "ten pots fell over". For a unigram model, the probability of the sequence is obtained simply by multiplying together the word probabilities:

$$P(\text{ten pots fell over}) \approx P(\text{ten})\,P(\text{pots})\,P(\text{fell})\,P(\text{over}). \tag{12.5}$$

In the case of a bigram model the probability of the sequence is estimated as:

$$P(\text{ten pots fell over}) \approx P(\text{ten} \mid \text{START})\,P(\text{pots} \mid \text{ten})\,P(\text{fell} \mid \text{pots})\,P(\text{over} \mid \text{fell}), \tag{12.6}$$

where START is used to indicate the beginning of the sequence. For a trigram model, the probability estimate becomes:

$$P(\text{ten pots fell over}) \approx P(\text{ten} \mid \text{START})\,P(\text{pots} \mid \text{START ten})\,P(\text{fell} \mid \text{ten pots})\,P(\text{over} \mid \text{pots fell}). \tag{12.7}$$

In principle, $N$-grams can be estimated using simple frequency counts from training data to obtain maximum-likelihood estimates for the required probabilities. Considering the bigram $(w_{k-1}, w_k)$ and the trigram $(w_{k-2}, w_{k-1}, w_k)$, the conditional probabilities $P(w_k \mid w_{k-1})$ and $P(w_k \mid w_{k-2}, w_{k-1})$ could be estimated thus:

$$\hat{P}(w_k \mid w_{k-1}) = \frac{C(w_{k-1}, w_k)}{C(w_{k-1})}, \quad \hat{P}(w_k \mid w_{k-2}, w_{k-1}) = \frac{C(w_{k-2}, w_{k-1}, w_k)}{C(w_{k-2}, w_{k-1})}, \tag{12.8}$$

where the notation $C(x)$ is used to represent the count of number of examples of $x$. For the examples of the bigram (fell over) and the trigram (pots fell over) we have:

$$\hat{P}(\text{over} \mid \text{fell}) = \frac{C(\text{fell over})}{C(\text{fell})}, \quad \hat{P}(\text{over} \mid \text{pots fell}) = \frac{C(\text{pots fell over})}{C(\text{fell over})}. \tag{12.9}$$

### 12.7.2 Perplexity and evaluating language models

The task of the language model can be viewed as one of predicting words in a sequence, and a good language model will be one that provides a good predictor of

the word in any position based on the words observed so far. Given a particular sequence of $K$ words in some test database, the value of $P(W)$ for that sequence provides a measure of how well the language model can predict the sequence: the higher the value of $P(W)$, the better the language model is at predicting the word sequence. Word sequences differ in length, and an average measure of the probability per word is obtained by taking the $K^{\text{th}}$ root of the probability of the sequence. The inverse of this probability defines the **perplexity**, $PP(W)$, thus:

$$PP(W) = \left[P(w_1, w_2, ..., w_K)\right]^{-1/K} = \left[\prod_{k=1}^{K} P(w_k \mid w_1, ..., w_{k-1})\right]^{-1/K} . \qquad (12.10)$$

For any given language model, it is possible to calculate the perplexity of some corpus to be recognized. For artificially constrained tasks with a rigid syntax, perplexity can be interpreted as being equivalent to the average number of different words that would need to be distinguished at any point in the word sequence, if all words at any particular point were equiprobable. Perplexity is therefore sometimes referred to as representing an **average branching factor**. The lower the value of perplexity, the fewer the number of alternatives that must be considered. The lowest possible value of perplexity is 1, but this value would only be obtained if all the individual word probabilities were equal to 1, such that only one word sequence could be recognized by the system. At the other extreme, if any word in a sequence is assigned a probability of zero by the language model, then the probability of the complete sequence will be equal to zero and the perplexity will be infinitely large.

As we have already seen, a major challenge for any model of natural language is to avoid probabilities of zero by not excluding any of the vocabulary words, while making the prediction of the next word as good as possible by having only a few high-probability alternatives at any one point. A good language model should give a low value of perplexity on a large corpus of representative text material (with no part of that material having previously been used to train the model).

The perplexity measure provides a means for evaluating alternative possible language models on some test corpus without needing to run a complete recognition experiment. A perplexity evaluation allows the language-model component to be assessed independently from the acoustic-model component, but it cannot take into account any interactions between the two models: good discrimination by the language model may not have much effect on recognition performance if the words concerned are acoustically very distinct, but could have a large effect for acoustically confusable words. Furthermore, any effects of the search algorithm cannot be allowed for in the perplexity calculation. Thus perplexity on a test data set is helpful for comparing alternative language models and also provides a useful indicator of the difficulty of the recognition task to be performed by the acoustic models, but the final test must be in terms of the recognition accuracy of the complete system.

### 12.7.3 Data sparsity in language modelling

Maximum-likelihood estimates obtained from frequency counts using expressions such as those in Equation (12.8) are a good approximation to the true probabilities,

provided that the sample size is large in relation to the number of possible outcomes. Unfortunately, in the case of $N$-gram modelling there are a vast number of possible outcomes. A vocabulary of $V$ words provides $V^2$ potential bigrams and $V^3$ potential trigrams. For a 20,000-word vocabulary there are thus 400 million possible word bigrams and eight million million possible trigrams! It would be completely impracticable to provide sufficient speech data to determine the language-model probabilities, and instead very large text corpora are used. Even so, while typical text corpora may contain over 100 million words, most of the possible bigrams and the vast majority of possible trigrams will not occur at all in the training data and many others will only appear once or twice.

Data sparsity is a much greater problem for the language model than for the acoustic model, due to the much larger size of the inventory of basic units (words rather than phones). As a consequence of this severe data-sparsity problem, it is not possible to rely only on simple frequency counts for estimating language-model probabilities, as many of the bigram and trigram probabilities would be set to zero (so that it would then be impossible to recognize any word sequence containing one of these combinations) and many other probabilities would be poorly estimated. The successful use of $N$-grams for LVCSR is dependent upon the use of **smoothing** techniques for obtaining accurate, robust (non-zero) probability estimates for *all* the possible $N$-grams that can occur for a given vocabulary. Zero probabilities and low non-zero probabilities are adjusted upwards, and high probabilities are reduced. Thus the overall effect is to make the probability distributions more uniform, and hence 'smoother'. Some different aspects of smoothing algorithms are described briefly in the following sections.

### 12.7.4 Discounting

For any set of possible 'events', such as bigrams or trigrams, the sum of the probabilities for all the possibilities must be equal to one. When only a subset of the possible events occurs in the training data, the sum of the probabilities of all the observed events must therefore be less than one. This effect can be achieved by reducing the observed frequency counts. The process is generally known as **discounting**, and is often described in terms of 'freeing' **probability mass** from the observed events which can then be redistributed among the unseen events.

Several different methods have been used to achieve discounting, and these methods differ in the way in which the reduced probabilities are calculated. Full coverage of discounting methods is outside the scope of this book but, for example, one simple but effective technique is **absolute discounting**. Here some small fixed amount (between zero and one) is subtracted from each frequency count (the numerators in Equation (12.8) or in the examples shown in Equation (12.9)). Thus the probability of every observed event is reduced, but the effect decreases as the observed frequency count increases (when the maximum-likelihood estimate should be more reliable). However, the same discount value may not be optimum for the full range of frequency counts. In particular, there is some evidence that absolute discounting imposes too great a reduction in the probability of events that occur only once or twice. A variant of the technique overcomes this problem by having separate discount values for these rare events.

### 12.7.5 Backing off in language modelling

Probability mass which has been made available as a result of discounting can be allocated to the unseen events. In estimating the probabilities of the unseen events, it is desirable to make use of any information that is available about the relative probabilities of the different events. In the case of $N$-grams, useful information may be available in the form of the probability according to a more general distribution. Thus it is possible to formulate a recursive procedure of '**backing off**': if a trigram is not observed, the model backs off to the relevant bigram probability, and if the bigram is not available, it backs off further to the unigram probability. For any words which do not occur in the training texts at all, it is possible to back off to a uniform distribution whereby all words are assumed to be equally likely.

Backing off can be illustrated by considering the task of estimating the conditional trigram probabilities $P(w_k | w_{k-2}, w_{k-1})$ for word $w_k$ in the context of the sequence of preceding words $(w_{k-2}, w_{k-1})$. We will assume that some appropriate discounting method has been used to assign probabilities to all observed trigrams, and these probability estimates will be denoted $\widetilde{P}(w_k | w_{k-2}, w_{k-1})$. Thus, using $P_s(w_k | w_{k-2}, w_{k-1})$ to denote an estimate for the probability of word $w_k$ given the sequence of preceding words $(w_{k-2}, w_{k-1})$, a backing-off scheme for obtaining this probability estimate is as follows:

$$P_s(w_k | w_{k-2}, w_{k-1}) = \begin{cases} \widetilde{P}(w_k | w_{k-2}, w_{k-1}) & \text{if } C(w_{k-2}, w_{k-1}, w_k) > 0 \\ B(w_{k-2}, w_{k-1})\, P_s(w_k | w_{k-1}) & \text{otherwise.} \end{cases} \quad (12.11)$$

$P_s(w_k | w_{k-1})$ is the estimated bigram probability of $w_k$ given preceding word $w_{k-1}$. $B(w_{k-2}, w_{k-1})$ is a normalizing constant for trigram context $(w_{k-2}, w_{k-1})$, and scales the bigram probabilities so that the sum of the $P_s(w_k | w_{k-2}, w_{k-1})$ terms is equal to 1 when computed over all possible words $w_k$. The sum of all the probabilities that are calculated by backing off must be equal to the total probability mass that has been freed from discounting the relevant trigram context. The normalizing constant is therefore chosen to be equal to the appropriate fraction of this freed probability.

Backing off from bigram to unigram and from unigram to uniform distributions can be accomplished in an analogous manner.

By setting the count threshold to zero, backing off only occurs for $N$-grams with no examples at all. However, as observing just one or two examples is unlikely to provide a reliable probability estimate, it can be beneficial to apply a higher threshold and so disregard those $N$-grams that occur just a few times in the training data. In this way only robustly estimated $N$-grams should be retained, which has the added benefit of a substantial saving in the memory needed for the language model.

### 12.7.6 Interpolation of language models

Backing off involves choosing between a specific and a more general distribution. An alternative is to compute a weighted average of different probability estimates, obtained for contexts that can range from very specific to very general. The idea is to improve the accuracy and robustness of a context-specific probability estimate by combining it with more general estimates for which more data are available. One option involves taking a linear combination of different probability estimates. For

example, a trigram probability could be estimated by linear interpolation between relative frequencies of the relevant trigrams, bigrams and unigrams, thus:

$$P_s(w_k \mid w_{k-2}, w_{k-1}) = \lambda_3 \frac{C(w_{k-2}, w_{k-1}, w_k)}{C(w_{k-2}, w_{k-1})} + \lambda_2 \frac{C(w_{k-1}, w_k)}{C(w_{k-1})} + \lambda_1 \frac{C(w_k)}{K}, \quad (12.12)$$

where $K$ is the number of different words, and the sum of the non-negative weights $\lambda_1 + \lambda_2 + \lambda_3 = 1$. The values for the weights need to be chosen to make the best compromise between specificity and ability to generalize to new data. The training data are therefore divided into two parts. The first (larger) part of the data is used to derive the frequency counts, which are then used when finding the optimum values of the weights in order to maximize the probability for the second part of the data. Because the parameters that are estimated will tend to depend on how the data are partitioned, often alternative sets of parameters are estimated for several different ways of splitting the data, and then the individual estimates are combined. This smoothing method is often called **deleted interpolation**.

For simplicity, interpolation has been introduced using probability estimates obtained from simple frequency counts. However, smoothing by interpolation can also be applied to other probability estimates, including $N$-gram probabilities that have been obtained by, for example, the absolute discounting technique mentioned in Section 12.7.4. Interpolation schemes can also be used with probabilities from other types of language model, such as those discussed in Section 12.7.8 below.

### 12.7.7 Choice of more general distribution for smoothing

In the previous sections we have described ways of smoothing $N$-grams using lower-order $N$-grams. The lower-order distribution is usually defined in a way which is exactly analagous to the definition for the higher-order distribution that is being smoothed. There is however an alternative method which may give more reliable estimates of the lower-order probabilities. This method is best explained by describing an example. Consider a word, such as "Francisco", that almost always occurs following just one other word ("San"). If the training text contains several examples of "San Francisco", both the bigram probability $P$(Francisco|San) and the unigram probability $P$(Francisco) will be high. Thus if we then use a discounting method such as absolute discounting to derive the probability of "Francisco" occurring after some other bigram history, this probability will also tend to be fairly high. However, intuitively it seems more plausible that, if the only information we have implies that "Francisco" always follows "San", the probability of "Francisco" following any other word should be low.

It is possible to define the unigram probability used for smoothing not in terms of number of examples of a word but rather in terms of the number of *different* contexts in which the word occurs. Chen and Goodman (1999) have proposed using this approach together with smoothing by interpolation, and incorporating a variant of absolute discounting that uses three separate discounts (one for $N$-grams occurring just once, one for $N$-grams occurring twice, and a third for all other $N$-gram counts). In comparative experiments, they demonstrated that this method performed consistently better (in terms of both perplexity and recognition performance) than a wide range of other smoothing techniques.

### 12.7.8 Improving on simple *N*-grams

*N*-grams have proved to be a very popular approach to language modelling. Bigrams and trigrams are the most widely used but 4-grams and even 5-grams are sometimes also included. *N*-grams provide a simple representation of language structure by focusing on local dependencies based only on word identity. Effects due to syntax, semantics and pragmatics are captured simultaneously but with no distinction between them. Although the method is really very crude, in practice it is very effective for languages such as English for which word order is important and the strongest contextual effects tend to be from immediately adjacent words.

While *N*-grams are good at modelling local context, an obvious deficiency is their inability to capture longer-term effects: syntactic constraints (e.g. subject–verb agreement) and semantic influences (certain words tending to occur together) may both operate over a span of several words. Various methods have been suggested to incorporate these effects, usually to provide additional information that can be interpolated with *N*-gram probabilities. A few of the developments are briefly mentioned in the following paragraphs.

*N*-grams are simple to compute directly from text data, without any need for explicit linguistic knowledge about the individual words. However, if information about syntactic classes (or other word groupings) is available, it is possible to estimate class-based *N*-gram probabilities and to back off or interpolate using these rather than needing to go to a shorter context (e.g. using a class-based trigram rather than resorting to a bigram). Taking this idea further, decision-tree methods can be applied to language models to find the best way of partitioning the data into different clusters based on questions about syntactic and/or semantic context. Other methods have involved attempting some grammatical analysis to determine the syntactic structure of the hypothesized sentence so far.

For generality, a language model needs to be trained on a large body of text from diverse sources. However, language is actually very dynamic in character, with the probability of many words being very different depending on the subject matter. One way of taking account of short-term patterns of word usage is to introduce a cache component (by analogy with "cache memory" in hardware terminology). The cache is simply a buffer which stores the frequency of occurrence of some number of the most recent words (typically around 200 different words). Word probabilities can be estimated by interpolating conventional *N*-gram probabilities with the probabilities as given by frequency of occurrence in the cache. A related concept is the idea of word 'triggers', whereby certain words tend to be very strong indicators, or triggers, for other words to occur in the general vicinity, but not necessarily within the span covered by a trigram. Triggers may be related to subject matter (e.g. "airline" associated with "flights") or to linguistic constructs (e.g. "neither" tends to be followed fairly soon afterwards by "nor"). Trigger models are used in conjunction with a cache component to keep a record of the recent words from which to adjust the probabilities for likely 'triggered' words.

The use of a cache and the incorporation of triggers are two ways of capturing dynamics in language usage. It is also possible to apply language-model adaptation techniques (analogous to the acoustic-model adaptation methods described in Chapter 11) to adapt *N*-gram probabilities based on adaptation data, which might for example relate to a new topic or reflect talker-specific language patterns.

## 12.8 DECODING

The recognition task is to find the most probable sequence of words $\hat{W}$, as given by Equation (12.1). As the vocabulary size becomes larger, the number of different possible word sequences soon becomes prohibitive and any practical system needs an efficient means of searching to find the most probable sequence. The component of the system that performs this search operation is often referred to as the **decoder**.

Although recognition using whole-word HMMs is not practicable with a large vocabulary, the same principles can be applied to units of any size. In the previous sections we have explained how statistical models can be used at different levels. One level captures the relationship between phone-size units and their acoustic realization, then there are the different phone sequences that can make up a word, and finally there are the probabilities of different word sequences. Thus a language can be modelled as a network of states representing linguistic structure at many levels. At the lowest level a small network of states represents a triphone or similar unit. At the next level a network of triphones forms a state to represent a word. A complete sentence can then be generated by a network of word states, where the connections between the states correspond to the language-model probabilities. The decoding task is to find the best path through this multiple-level network, and the recognized word sequence is given by the best path at the highest level.

In order to apply the principles of DP using the Viterbi algorithm to a multiple-level network, probabilities need to be evaluated for all valid partial paths at all levels, with no decisions being reached until all earlier parts of a path enter the same highest-level state at the same time, thus delaying decisions until all relevant evidence has been used. The use of delayed decisions is a fundamental principle of this HMM approach, as it enables the knowledge and constraints at all levels to be employed simultaneously to find the best explanation of the data. In practice, even at any one point in time, in a large-vocabulary system there are so many possibilities (both at the word and at the sub-word level) that it is impossible to evaluate all of them. However, the language model provides strong constraints that act to make many of the possibilities extremely unlikely, and it is necessary to find an efficient way of applying these constraints within the decoding.

In a first-order HMM the probability of a transition from any one state to any other state depends only on the identities of the source and destination states. This model cannot accommodate a trigram (or higher-order) language model, because here the word transition probability depends on more than just the immediately preceding state. In order to use such a language model, some way of dealing with the higher-order dependencies needs to be found. An additional complication arises when using cross-word triphones because the identity of a word-final triphone, and hence the word probability, depends on the identity of the following word.

The issues associated with large-vocabulary decoding have been addressed in various different ways. Three main types of method are briefly described below.

### 12.8.1 Efficient one-pass Viterbi decoding for large vocabularies

In order to accommodate cross-word triphone models, the state network for a Viterbi search needs to include multiple entries for each word to cover all possible

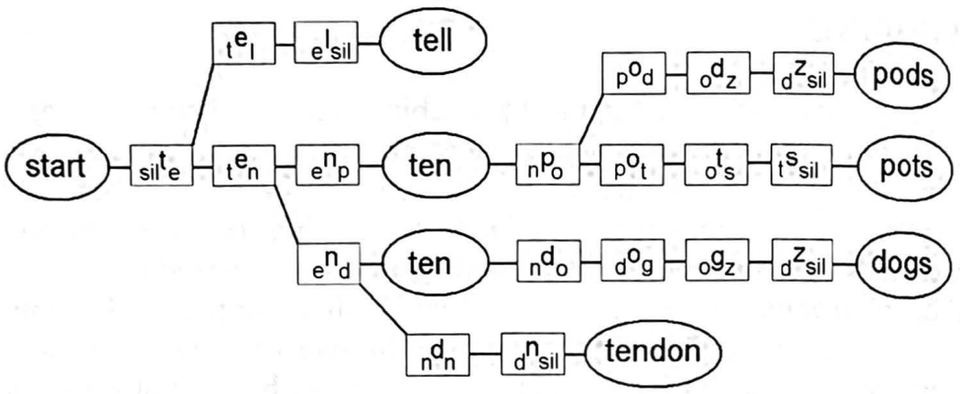



**Figure 12.4** A very small fragment of a network for decoding using cross-word triphone models. This fragment shows the sequence "ten pots" and a few possible alternatives that branch at different positions in the network. Note that the word "ten" is represented by different nodes in the network, depending on whether the following word is "pots" or "dogs".

different triphones that may end any one word. Similarly, a trigram language model can be used by expanding the network to keep multiple copies of each word so that each transition between words has a unique two-word history. To make the network manageable, it is usually represented as a tree structure, as shown in Figure 12.4. In the tree, different hypotheses that start with the same sequence of sub-word models share those models. This tree network is built dynamically as required.

In Section 8.8 we introduced the concept of score pruning to reduce the number of hypotheses to be evaluated in a DP search. Efficient pruning is essential for LVCSR systems. The usual scheme employs a **beam search**, whereby at each time frame all paths whose likelihood score is not within some specified threshold of the best-scoring path are pruned out. In practice, it is generally the case that the likelihoods for all except just the few most likely states tend to be very small, so it is possible to concentrate the search on a narrow beam of possible alternatives.

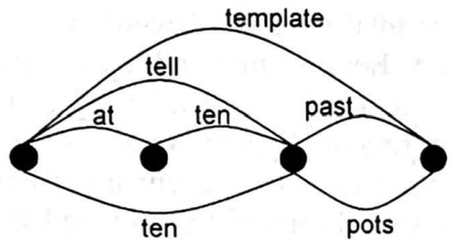
Language constraints act to restrict the set of words that are likely at any given point in an utterance. It is therefore advantageous to use the language model to prune out unlikely possibilities as soon as possible when decoding an utterance, but in conventional Viterbi decoding the language-model probability is not known until the end of a word is reached. However, if a record is kept of the current possible words associated with each sub-word model, it is possible to use the language-model probabilities to prune out unlikely hypotheses at an earlier stage.

### 12.8.2 Multiple-pass Viterbi decoding

An alternative to attempting complete and accurate recognition in a single pass is to use a multiple-pass approach. The idea here is to start by using a simple recognition system to identify a small number of likely hypotheses, and then to use a more elaborate system to choose between these possibilities. For example, the first recognition pass could use only word-internal triphones and a bigram language model. A second recognition pass might then incorporate cross-word triphones and other more specific acoustic models as well as trigram and cache language models.

The output of the first recognition pass is usually expressed either as a rank-ordered ***N*-best** list of possible word sequences, or as a **word graph** or **lattice**

1 ten pots
2 tell pots
3 template
4 at ten past
5 ten past
6 tell past




**(a)** *N*-best list of hypotheses. **(b)** Lattice representation of the alternatives shown in (a).

**Figure 12.5** Possible alternative utterances that might be generated for a short utterance.

describing the possibilities as a network. Figure 12.5 shows an example of an *N*-best list and a word lattice that could be generated for an utterance of "ten pots". In order to output more than one possible sequence, the first-pass DP search needs to be extended to retain several hypotheses at each stage.

### 12.8.3 Depth-first decoding

The Viterbi approach is known as a **breadth-first** search, because all possibilities are considered in parallel, with all paths to time $t$ being evaluated before proceeding to time $t + 1$. An alternative is to adopt a **depth-first** search, for which the principle is to pursue the most promising hypothesis until the end of the utterance. This strategy is often referred to as **stack decoding**: the idea is to keep an ordered 'stack' of possible hypotheses and to iteratively take the best hypothesis from the stack, choose the most likely next word and then add this hypothesis as a new entry to the stack, reordering the hypotheses as necessary. Once the end of the utterance is reached, the best-scoring hypothesis is output as the recognized word sequence. At any point in the search it may be necessary to compare hypotheses of different lengths. Because the score for any one path is a product of probabilities, it decreases with time and so a simple comparison of different scores will be biased towards shorter hypotheses. This problem can be overcome by normalizing the score for each path based on the number of frames that it accounts for.

With depth-first decoding a new hypothesis is generated each time a word is added to one of the existing hypotheses, so it is possible to evaluate the different options that are required for long-range language models. The method can, however, be expensive in terms of both memory and processing requirements.

## 12.9 EVALUATING LVCSR PERFORMANCE

### 12.9.1 Measuring errors

When recognizing connected speech, there are three types of recognition error: **substitution** (the wrong word is recognized), **deletion** (a word is omitted) and **insertion** (an extra word is recognized). The **word error rate (WER)** is given by:

$$\text{WER} = 100 \times \frac{C(\text{substitutions}) + C(\text{deletions}) + C(\text{insertions})}{N}\%, \qquad (12.13)$$

where $N$ is the total number of words in the test speech and $C(x)$ is the count of errors of type $x$. Because there will not in general be a one-to-one correspondence between the recognized and actual word sequences, a DP alignment process is used to find the best way of aligning the two before the WER can be calculated.

The percentage **word accuracy** is equal to 100 – WER. Sometimes the percentage of words correctly recognized is quoted, but this measure may not be such a good indicator of true performance as it does not include insertion errors.

### 12.9.2 Controlling word insertion errors

The probability that is assigned to any word sequence will depend on the relative contributions from the language and acoustic models. However, both models involve approximations and assumptions, and hence only provide estimates of the relevant probabilities. In particular, the probability given by the acoustic model will depend on the number and choice of acoustic features that are used and typically has a disproportionately large influence relative to the language-model probability. When the language model is given insufficient weight, the consequence is often a large number of errors due to insertion of many short function words. The short duration and high variability that are typical of these words mean that a sequence of their models may provide the best acoustic match to short regions of speech, even though the word sequence is given a low probability by the language model.

There are two practical solutions that are often adopted to tackle the problem of word insertion errors. One approach is to impose a **word insertion penalty**, whereby the probability of making a transition from one word to another is explicitly penalized by multiplying the word probability by some number less than 1. Alternatively or in addition, the influence of the language model on the combined probability can be increased by raising its probability to some power greater than 1. Values, both for this power and for any word insertion penalty, are usually chosen experimentally to maximize performance on some evaluation data.

### 12.9.3 Performance evaluations

By the mid-1980s HMMs were showing promise for practical application to large-vocabulary recognition tasks. It was, however, difficult to compare competing systems from different laboratories because individual systems tended to be trained and tested on different sets of data, due to the absence of any common databases for training or for testing. This widely recognized problem was addressed when ARPA (also known as DARPA, the *Defense* Advanced Research Projects Agency) began funding a second major research programme in the area of ASR. As part of this programme, a recognition task was defined and speech data were recorded and made publicly available. The data were divided into a **training set** for training a set of models, a **development test set** for testing a recognition system during its development, and an **evaluation test set** on which the final system could be tested and scored just once. The advent of this type of database enabled direct comparisons to be made between different algorithms and systems, with controlled assessments of performance that could also be used to measure progress over time.

In 1989 the first formal competitive evaluation to measure the performance of different systems was organized. This evaluation marked the start of a series of tests that is still continuing today. Any organization participating in an evaluation has to test its recognition system on the designated evaluation test data (following the pre-specified rules of the evaluation) and then submit the recognition results for scoring by the National Institute of Standards and Technology (NIST), a U.S. government organization. This scoring of the results by an independent body helps to ensure the applicability of direct and fair comparisons between different systems.

All the recognition tasks that have been chosen by ARPA have involved speaker-independent recognition, but over the years the tasks have progressively become more challenging both by increasing the vocabulary size and by changing the nature of the speech material. The first recognition task was called Resource Management (RM). The data for this task consisted of read queries on the status of U.S. naval resources, using a vocabulary of about 1,000 words and a fairly constrained syntax (giving a test-set perplexity of about 60). Recognition tests were first carried out on this task in 1987. Formal evaluations began in 1989 and ended in 1992. New testing data were provided for each of the evaluations, and each year the recognition performance improved as shown in Figure 12.6.

The next ARPA evaluations focused on recognition of spoken newspaper texts. This application allowed much larger-vocabulary tasks to be studied and also ensured that there was easy availability of large quantities of text data for training statistical language models. Initially the texts were all obtained from the *Wall Street*

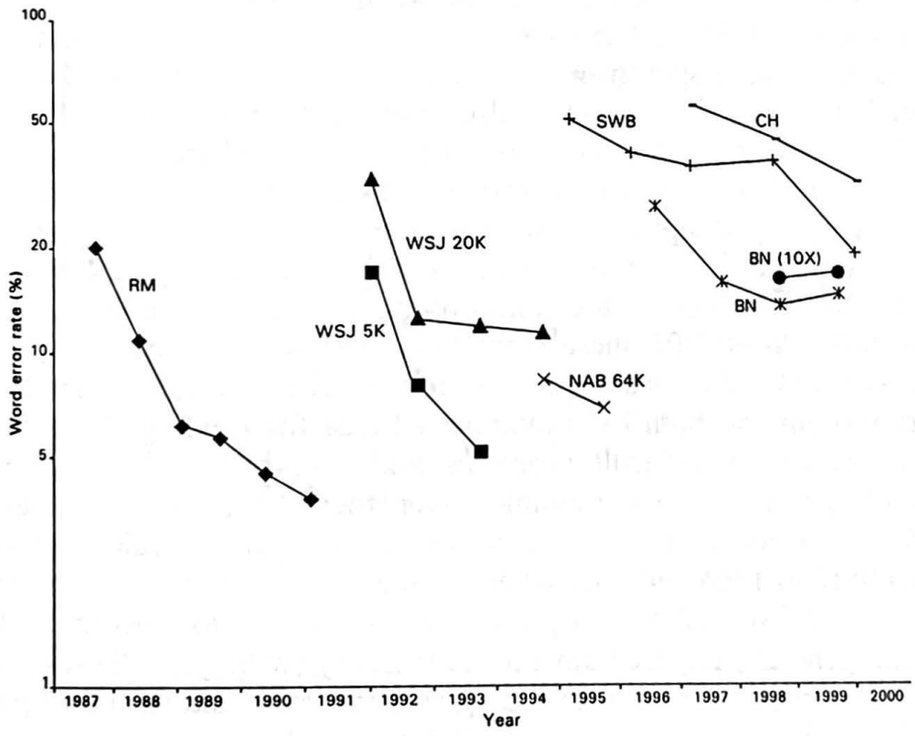



**Figure 12.6** Graph showing recognition performance (in terms of word error rate, plotted on a logarithmic scale) for various transcription tasks used in ARPA evaluations. Each data point has been taken from published results for the relevant evaluation, with the aim of showing the performance of the system that was the best on that particular test. Note that, even within any one task, different evaluation data have been used each year, and in some cases there were differences in the task details. Thus, while the lines in the graph give an indication of general trends in performance, strict comparisons of performance in successive years are more difficult.

*Journal* (WSJ), and included both a 5,000-word recognition test and a 20,000-word test. More recently, the test set was expanded to include a range of North American business (NAB) news and restrictions on training for the acoustic and language models were removed. The recognition vocabulary became essentially unlimited and performance was generally evaluated using systems designed for recognizing about 64,000 words. WSJ and NAB evaluations were conducted in the period between 1992 and 1995. Over the years the complexity and variety of the tests increased, but recognition performance still improved each year (see Figure 12.6).

Later ARPA evaluations moved away from read speech, and concentrated on transcribing 'found' speech from broadcast news (BN) shows on radio and television. Such data provided a lot of variation in speech quality, microphones and background noise, as well as non-native speech and periods containing only non-speech data such as sound effects and music. For the first BN evaluation in 1996, performance dropped considerably from that which had been obtained in the previous evaluations using read speech. However, researchers soon found ways of coping with the greater variety of material, and by the final BN evaluations in 1998 and 1999 typical error rates had roughly halved from those obtained in 1996.

ARPA have also sponsored a research programme on the recognition of conversational speech, including the collection of two different corpora. Data for one corpus, known as Switchboard (SWB), first became available in 1992. The data comprised telephone conversations between two strangers about some topic that had been specified in advance. A few years later, in 1995, collection of the Callhome (CH) corpus was initiated to provide more natural conversational speech by recording telephone conversations between family members. The CH set includes a number of different languages. Conversational speech is rather different in nature to either read speech or television and radio broadcasts. Conversations tend to include many disfluencies, false starts and so on, as well as a lot of phonological reduction and special use of prosody. There is also turn-taking behaviour and reliance on shared knowledge to assist in the communication, especially when the participants know each other very well (as in the CH data). As a result, errors are much higher than for the other tasks (see Figure 12.6), but again substantial progress has been made in improving performance on the conversational recognition tasks. As of 2001, these evaluations are still continuing.

The ARPA initiative has been very influential in the development of large-vocabulary recognizers, both by providing a focus for refining the techniques to cope with increasingly difficult problems and also by making available huge quantities of speech data for training. Over the years, the result has been a progressive improvement in performance on increasingly difficult tasks, as can be seen in Figure 12.6. There are now several research systems which give impressive performance on substantial transcription tasks. However, to perform at their best these systems generally require both a lot of memory (with typically several million parameters in both the acoustic and language models), and a lot of processing power (often operating at a few hundred times real time). In the last two BN evaluations, the effect of processing demands was tested by introducing a category for systems that operated at no slower than 10 times real time (shown as BN(10×) in Figure 12.6). Enforcing this constraint led to around 15% more errors than if there were no processing restrictions. We will consider performance in relation to the requirements of different applications in Chapter 15.

## 12.10 SPEECH UNDERSTANDING

In order to understand spoken language, it is necessary not only to recognize the words, but also to determine the linguistic structure, including both syntactic and semantic aspects. Given a word sequence, the field of computational linguistics provides techniques for deriving a representation of the linguistic structure. Traditionally these techniques involve a complex linguistic analysis based on qualitative models that are usually stated in the form of a grammar that is specified manually by human experts, although in recent years data-driven statistical methods have also been used. Even so, any detailed linguistic analysis will involve modelling long-range effects that may operate over the entire span of an utterance. It is very difficult to incorporate such a model into a speech recognition search.

Due to the difficulties involved in attempting to incorporate detailed linguistic analysis into the recognition search, speech understanding systems generally treat these tasks as separate components: an ASR system of the type described earlier in this chapter, and a second component that uses artificial-intelligence techniques to perform linguistic analysis in order to 'understand' the recognizer output. To reduce the problems caused by errors made at the recognition stage, the output of the recognizer should include alternatives, which can be provided in the form of a word lattice or an $N$-best list. The final recognized output is typically taken to be the highest-scoring alternative that is also allowed by the linguistic model.

The process of determining the linguistic structure of a sentence is known as **parsing**. Syntactic structure is usually expressed in terms of a formal grammar, and there are a variety of grammar formalisms and associated methods for **syntactic parsing**. However, traditional parsers aim to recover complete, exact parses. This goal will often not be achievable for spoken language, which tends to contain grammatical errors as well as hesitations, false starts and so on. The problem is made even worse when dealing with the output of a speech recognizer, which may misrecognize short function words even when overall recognition performance is very good. It can therefore be useful to adopt **partial parsing** techniques, whereby only segments of a complete word sequence are parsed (for example, noun phrases might be identified). Other useful tools include **part-of-speech taggers**, which aim to disambiguate parts of speech (e.g. the word "green" acts as an adjective in the phrase "green coat" but as a noun in "village green"), but without performing a parsing operation. Some taggers are rule-based, but there are also some very successful taggers that are based on HMMs, with the HMM states representing tags (or sequences of tags). Transition probabilities are probabilities of tag(s) given previous tag(s) and emission probabilities are probabilities of words given tags. Partial parsing and part-of-speech tagging enable useful linguistic information to be extracted from spoken input without requiring a comprehensive linguistic analysis.

General **semantic analysis** for the derivation of meaning representations is a complex aspect of computational linguistics. However, many speech-understanding systems have used a simpler approach that is more application dependent. A popular technique uses templates, or 'frames', where each frame is associated with an action to be taken by the system. A frame has 'slots' for different items of relevant information. In a flight enquiry system for example, one frame could represent a request for flight information. This frame could include slots for the type of information (e.g. flights, fares) and for any constraints (e.g. date, price).

The task is to find the frame that gives the best match to the input, and to obtain values for the slots in that frame. Once a frame has been recognized and its slots have been filled, the system's response can be determined.

Many applications of speech understanding involve the user asking questions in order to access information from a computer system. Often a query may be insufficiently specified or ambiguous. However, by entering into an interactive dialogue with the user, the system should be able to obtain the additional information or resolve any ambiguities. A **dialogue manager** can greatly assist in the usability of these **spoken dialogue systems**, by keeping track of the interaction, guiding the user and asking questions where necessary.

### 12.10.1 Measuring and evaluating speech understanding performance

When evaluating speech understanding systems, word and sentence recognition errors are obviously of interest. However, some measure of a system's ability to recognize meaning is also required. This measure is usually the percentage of utterances for which the complete system gives an acceptable output, as judged by a trained human operator. For a spoken dialogue system that is intended for a particular application, it may be most informative to evaluate performance using task-based measures such as the percentage of tasks successfully completed and time taken to accomplish the tasks, together with measures of user satisfaction.

In parallel with its speech transcription evaluations, ARPA has also been conducting a series of evaluations of spoken language understanding systems. In the late 1980s, work began on an Air Travel Information System (ATIS) task, and formal evaluations were carried out in the period between 1990 and 1994. The ATIS task involved recognizing and responding to spontaneous queries about airline reservations (e.g. "List the flights from Boston to Dallas." or "Is lunch served on that flight?"), using a vocabulary of about 2,500 words. In 1994, the best speech recognition performance was a word error rate of 2.3% (Moore *et al.*, 1995), which improved to 1.9% when the output of the speech recognizer was subjected to post-processing by a natural-language system that was tuned to the application domain. Although this task involves spontaneous speech, the constraints of the task domain enable high recognition accuracy to be achieved. The best understanding performance, taken over all the answerable spoken queries, was an error rate of 8.6% (Pallett *et al.*, 1995). Although there were more understanding errors than word recognition errors, this level of performance is probably adequate for many database query applications. However, achieving such performance requires the system to be tuned to its particular application domain, with a lot of 'knowledge' that is specific to the one domain (flights in the case of the ATIS task).

More recent ARPA evaluations have focused on other types of tasks, which involve processing large quantities of speech material in a way that requires some understanding capability. Past and ongoing tasks include retrieval of spoken documents, detection and tracking of topics in broadcasts, and extraction of the content (meaning) from transcripts of spoken news material. Automatic speech understanding is a pre-requisite for all of the most advanced applications of spoken language processing, of which one of the hardest is automatic speech translation (see Chapters 15 and 16 for further discussion).

## CHAPTER 12 SUMMARY

- Some large-vocabulary recognition tasks may require accurate transcription of the words that have been said, while others will need understanding of the semantic content (but not necessarily accurate recognition of every word).
- For speech transcription the task is to find the most likely sequence of words, where the probability for any one sequence is given by the product of acoustic-model and language-model probabilities.
- The principles of continuous speech recognition using HMMs can be applied to large vocabularies, but with special techniques to deal with the large number of different words that need to be recognized.
- It is not practical or useful to train a separate model for every word, and instead sub-word models are used. Typically phone-size units are chosen, with the pronunciation of each word being provided by a dictionary.
- Triphone models represent each phone in the context of its left and right neighbours. The large number of possible triphones is such that many will not occur in any given set of training data. Probabilities for these triphones can be estimated by 'backing off' or interpolating with biphones (dependent on only the left or the right context) or even context-independent monophones.
- Another option, which allows greater context specificity to be achieved, is to group ('cluster') similar triphones together and share ('tie') their parameters. A phonetic decision tree can be used to find the best way to cluster the triphones based on questions about phonetic context. The idea is to optimize the fit to the data while also having sufficient data available to train each tied state.
- An embedded training procedure is used, typically starting by estimating parameters for very general monophone models for which a lot of data are available. These models are used to initialize triphone models. The triphones are trained and similar states are then tied together. Multiple-component mixture distributions are introduced at the final stage.
- The purpose of the language model is to incorporate language constraints, expressed as probabilities for different word sequences. The perplexity, or average branching factor, provides a measure of how good the language model is at predicting the next word given the words that have been seen so far.
- $N$-grams model the probability of a word depending on just the immediately preceding $N-1$ words, where typically $N=2$ ('bigrams') or $N=3$ ('trigrams').
- The large number of different possible words is such that data sparsity is a massive problem for language modelling, and special techniques are needed to estimate probabilities for $N$-grams that do not occur in the training data.
- Probabilities for $N$-grams that occur in the training text can be estimated from frequency counts, but some probability must be 'freed' and made available for those $N$-grams that do not occur. Probabilities for these unseen $N$-grams can then be estimated by backing off or interpolating with more general models.
- The principles of HMM recognition extend to large vocabularies, with a multiple-level structure in which phones are represented as networks of states, words as networks of phones, and sentences as networks of words. In practice the decoding task is not straightforward due to the very large size of the search space, especially if cross-word triphones are used. Special treatment is also required for language models whose probabilities depend on more than the

immediately preceding word (i.e. for models more complex than bigrams). The one-pass Viterbi search can be extended to operate with cross-word triphones and with trigram language models, but the search space becomes very large and is usually organized as a tree. Efficient pruning is essential.

- An alternative search strategy uses multiple passes. The first pass identifies a restricted set of possibilities, which are typically organized as an *N*-best list, a word lattice or a word graph. Later passes select between these possibilities. Another option is to use a depth-first search.
- Automatic speech understanding needs further processing of the speech recognizer output to analyse the meaning, which may involve syntactic and semantic analysis. To reduce the impact of recognition errors, it is usual to start with an *N*-best list or word lattice. Partial parsing techniques can be used for syntactic analysis to deal with the fact that the spoken input may be impossible to parse completely because parts do not fit the grammar, due to grammatical errors, hesitations and so on.
- Meaning is often represented using templates, which will be specific to the application and have 'slots' that are filled by means of a linguistic analysis.
- In spoken dialogue systems, a dialogue manager is used to control the interaction with the user and ensure that all necessary information is obtained.
- ARPA has been influential in promoting progress in large vocabulary recognition and understanding, by sponsoring the collection of large databases and running series of competitive evaluations. Error rates of less than 10% have been achieved for transcribing unlimited-vocabulary read speech and for understanding spoken dialogue queries. Recognition of more casually spoken spontaneous speech is still problematic.

## CHAPTER 12 EXERCISES

**E12.1** Explain the different requirements and problems in speech transcription and speech understanding.

**E12.2** What are the special issues for the design of the acoustic model, the language model and the search component when a recognizer needs to cope with a large vocabulary size?

**E12.3** Give examples of how acoustic/phonetic knowledge can be used when choosing an acoustic model set for a large-vocabulary recognition system.

**E12.4** When estimating trigram language-model probabilities based on counts in some training text, explain how probabilities can be estimated for trigrams that do not occur in the training data. How does this operation affect the probabilities that need to be assigned to trigrams that do occur?

**E12.5** What are the similarities and differences between the concept of 'backing off' in language modelling and in acoustic modelling?

**E12.6** Explain the relative merits of performing large-vocabulary recognition using a one-pass Viterbi search versus using a multiple-pass search strategy.

**E12.7** What measures are required to evaluate the performance of a speech understanding system?

With the growing impact of information technology on daily life, speech is becoming increasingly important for providing a natural means of communication between humans and machines. This extensively reworked and updated new edition of **SPEECH SYNTHESIS AND RECOGNITION** is an easy-to-read introduction to current speech technology.

Aimed at advanced undergraduates and graduates in electronic engineering, computer science and information technology, the book is also relevant to professional engineers who need to understand enough about speech technology to be able to apply it successfully and to work effectively with speech experts. No advanced mathematical ability is required and no specialist knowledge prior of phonetics or of the properties of speech signals is assumed.

**SPEECH SYNTHESIS AND RECOGNITION:**

- **explains the complexity of speech communication**
- **describes mechanisms and models of human speech production and perception**
- **covers concatenative synthesis techniques and formant synthesis by rule, as well as the processing required for synthesis from text**
- **introduces methods for automatic speech recognition by whole-word template matching and by statistical pattern matching using hidden Markov models**
- **describes practical techniques that contribute to the successful implementation of speech recognition systems, including those for recognizing very large vocabularies**
- **includes chapters covering the related technologies of digital speech coding and automatic recognition of speaker characteristics**
- **discusses applications and performance of current speech technology**

Throughout the book the emphasis is on explaining underlying principles with sufficient but not unnecessary detail, so as to provide the reader with a thorough grounding in the problems and techniques involved in speech synthesis and recognition. This book is therefore ideal as an introduction before tackling more advanced texts.

**Wendy Holmes** is a research scientist with 20/20 Speech Ltd in Malvern, England. The late **John Holmes** was an internationally regarded expert in the field of speech research, who spent several years as head of the UK government's Joint Speech Research Unit.

**Electrical engineering**
**Design: www.egelnickandwebb.com**

ISBN 0-7484-0857-6

9 780748 408573

11 New Fetter Lane
London EC4P 4EE
29 West 35th Street
New York NY10001
www.tandf.co.uk
Printed in Great Britain

TAYLOR & FRANCIS
Founded 1798