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

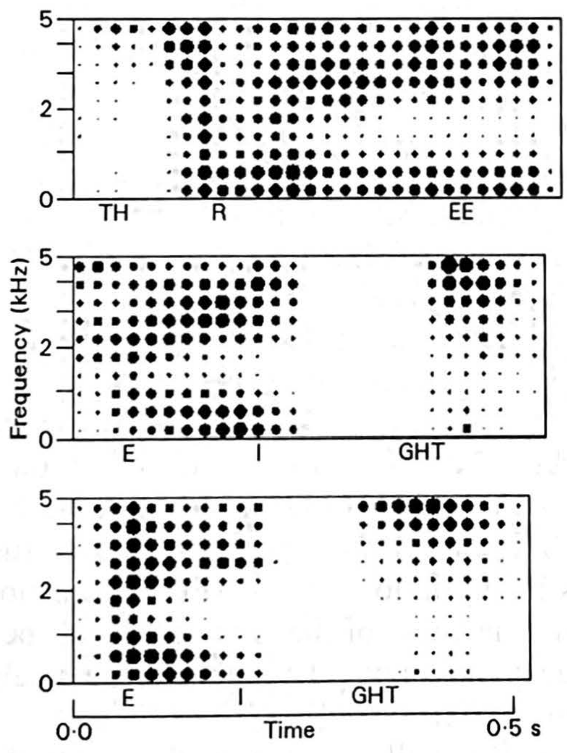

Figure 8.1 Spectrographic displays of a 10-channel filter-bank analysis (with a non-linear frequency spacing of the channels), shown for one example of the word "three" and two examples of the word "eight". It can be seen that the examples of "eight" are generally similar, although the lower one has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

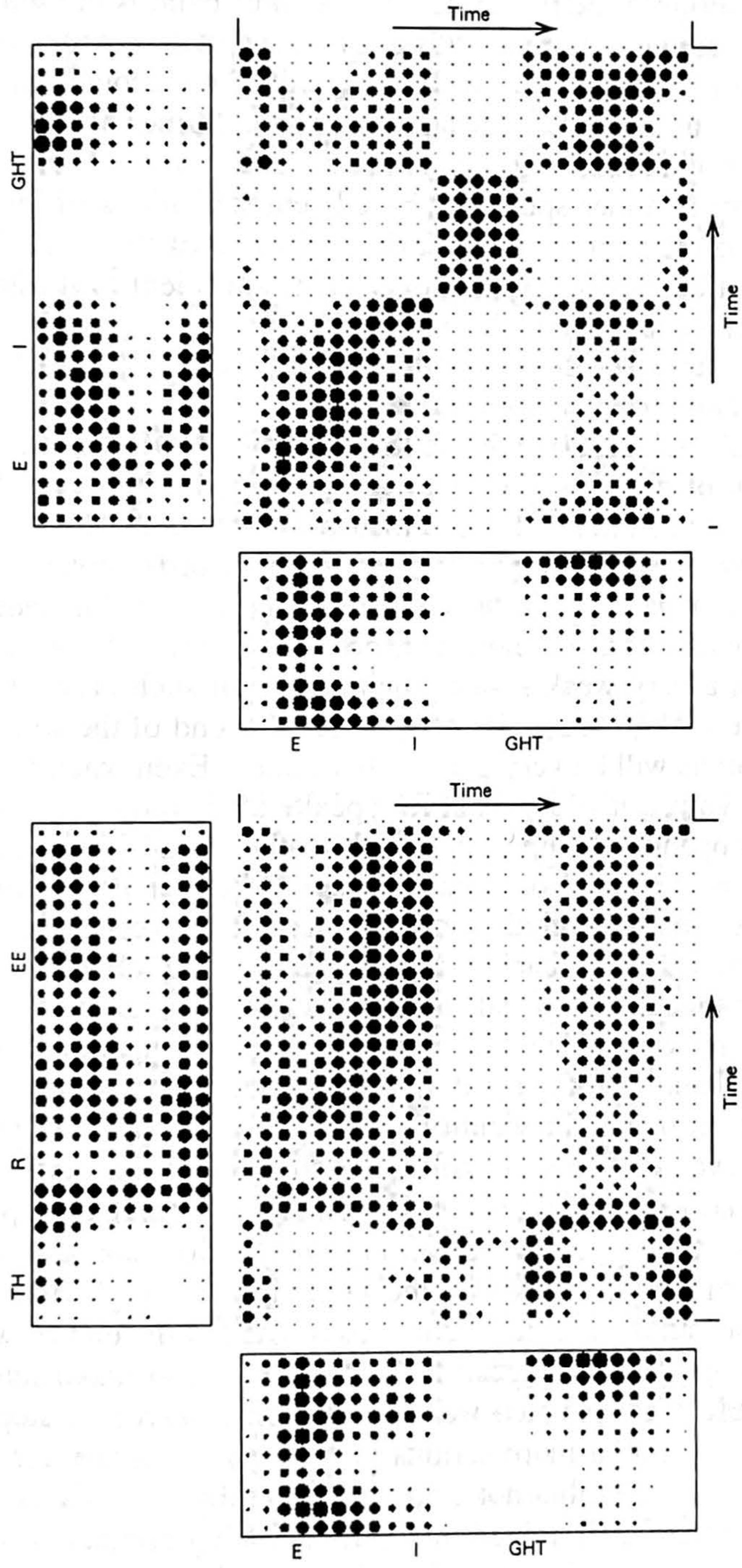

Figure 8.2 Graphical representation of the distance between frames of the spectrograms shown in Figure 8.1. The larger the blob the smaller the distance. It can be seen that there is a continuous path of fairly small distances between the bottom left and top right when the two examples of "eight" are 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

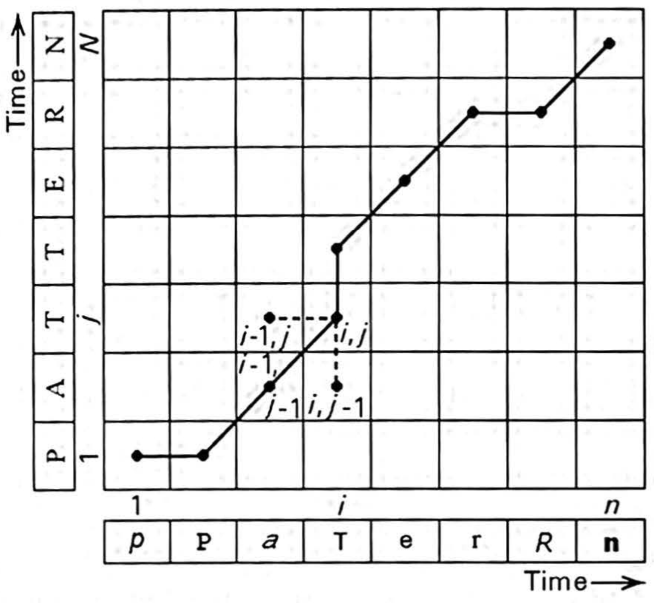

Figure 8.3 Illustration of a time-alignment path between two words that differ in their timescale. Any point i,j can have three predecessors as shown.

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:

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

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

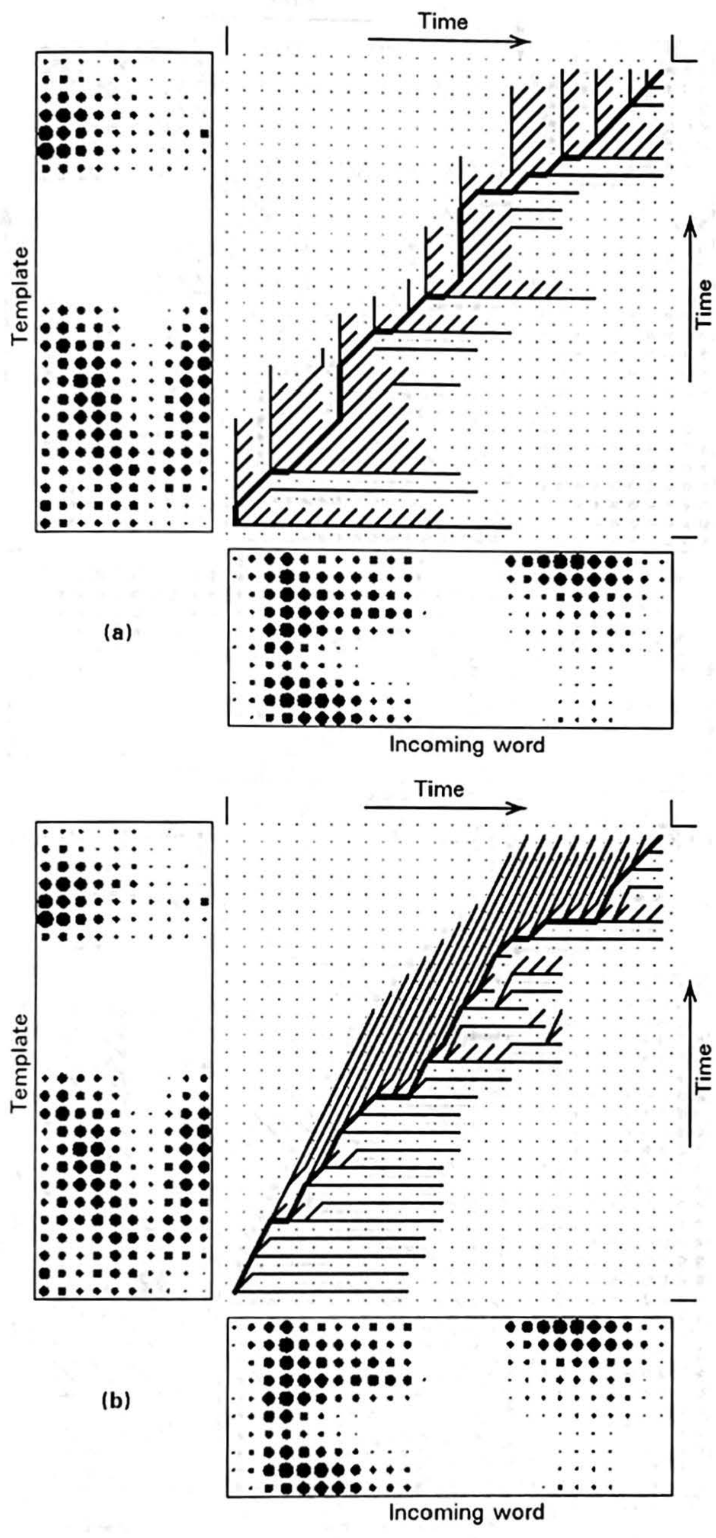

Figure 8.4 **(a)** DP alignment between two examples of the word "eight", with no timescale distortion penalty but with score pruning. The optimum path, obtained by tracing back from the top right-hand corner, is shown by the thick line. **(b)** Match between the same words as in (a), but using an asymmetric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

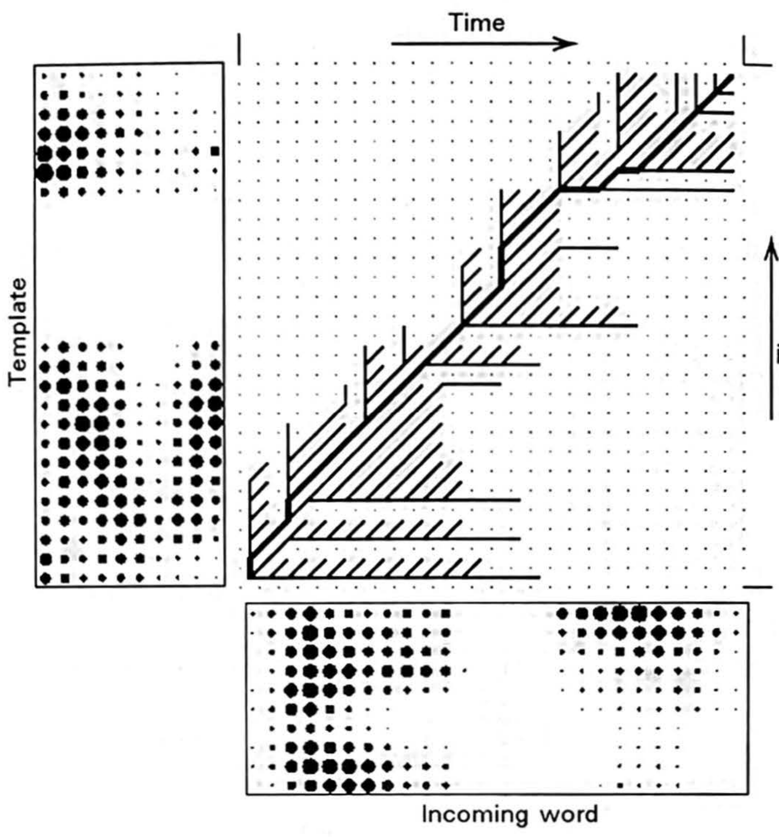

Figure 8.5 As for Figure 8.4(a), but with a small timescale distortion penalty.

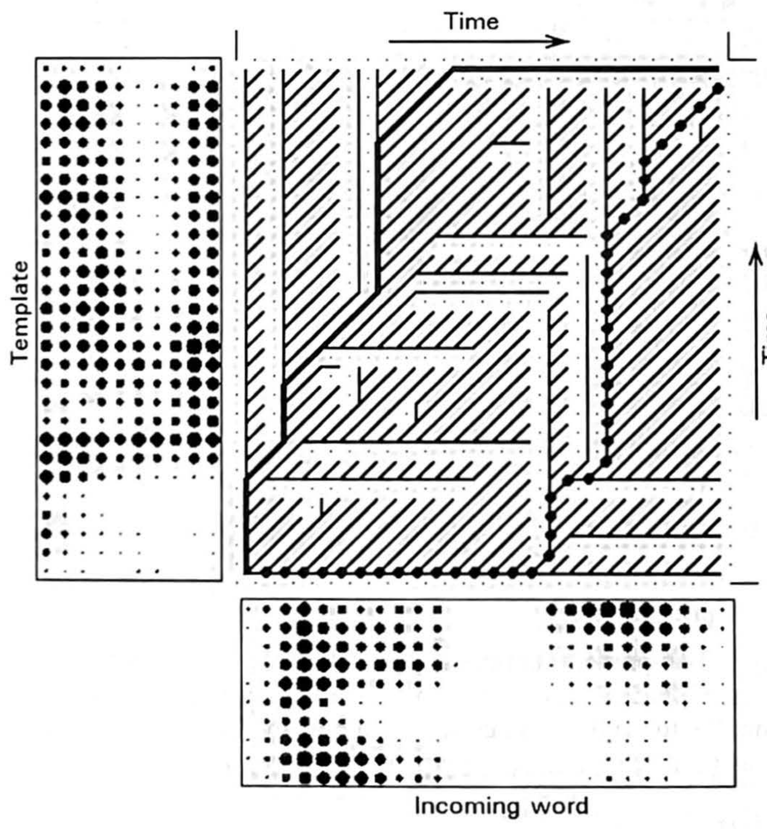

Figure 8.6 The result of trying to align two dissimilar words ("three" and "eight") within the same DP algorithm as was used for Figure 8.5. The score pruning was removed from this illustration, because any path to the end of the word would then have been seriously sub-optimal. It can be seen that if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

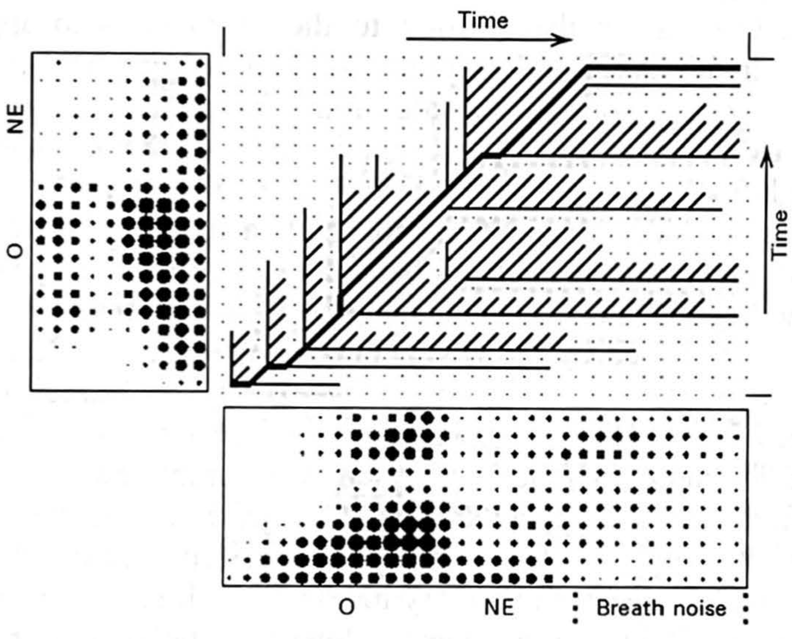

Figure 8.7 An example of the word "one" followed by breath noise, being aligned with a "one" template. A timescale distortion penalty was used except for the beginning and end frames.

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

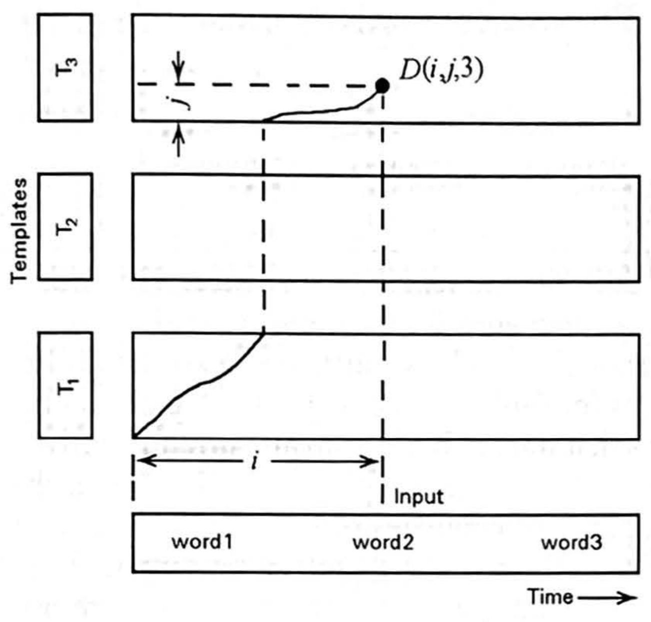

Figure 8.8 Diagram indicating the best-matching path from the beginning of an utterance to the j^{th} frame of template T_3 and the i^{th} frame of the input. In the example shown i is in the middle of the second word of the input, so the best path includes one complete template (T_1) and a part of T_3. 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

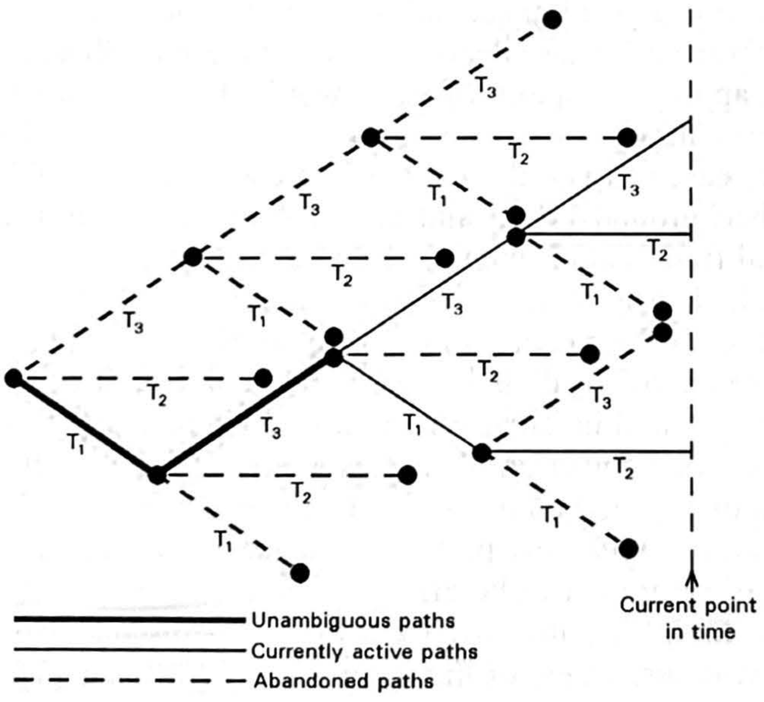

Figure 8.9 Trace-back through a word decision tree to identify unambiguous paths for a three-word vocabulary continuous recognizer. Paths are abandoned when the cumulative distances of all routes to the ends of the corresponding templates are greater than for paths to the ends of different template sequences 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

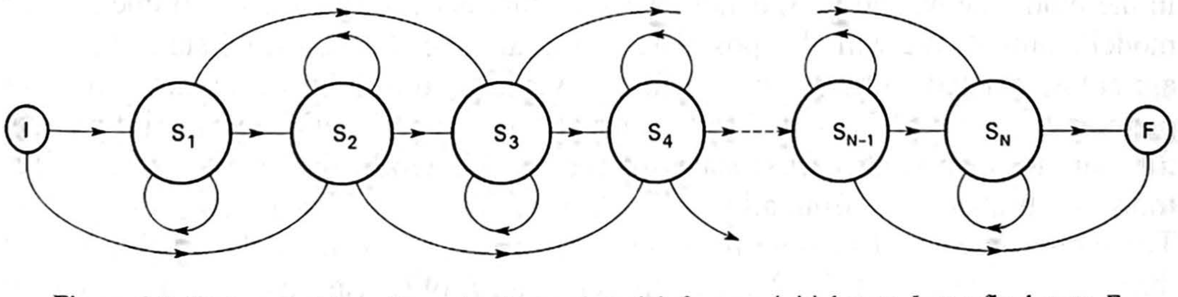

Figure 9.1. State transitions for a simple word model, from an initial state, I, to a final state, F.

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 nonemitting 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:

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

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

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:

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

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:

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

[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:

$$P(y_1, y_2, \cdots, y_T) = \sum_{\substack{\text{over all possible} \\ \text{state sequences} \\ \text{of length } T}} a_{Is_1} \left(\prod_{t=1}^{T-1} b_{s_t}(y_t) a_{s_t s_{t+1}} \right) b_{s_T}(y_T) a_{s_T F} \,. \qquad (9.5)$$

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.

$$\alpha_j(t) = P(y_1, y_2, \cdots, y_t, s_t = j) \qquad (9.6)$$

$$= \left(\sum_{i=1}^{N} \alpha_i(t-1) a_{ij} \right) b_j(y_t) \quad \text{for } 1 < t \le T \qquad (9.7)$$

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 $b_j(y_1)$.

$$\alpha_j(1) = a_{Ij} b_j(y_1) \qquad (9.8)$$

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:

$$P(y_1, y_2, \cdots, y_T) = \alpha_F(T) = \sum_{i=1}^{N} \alpha_i(T) a_{iF} \,. \qquad (9.9)$$

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 α. However, there is some variation in the way in which the α 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 β, γ and ξ, 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:

$$\hat{P}(y_1, y_2, \cdots, y_T) = \max_{\substack{\text{over all possible}\\ \text{state sequences}\\ \text{of length } T}} \left(P(y_1, y_2, \cdots, y_T, s_1, s_2, \cdots, s_T)\right). \tag{9.10}$$

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^{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):

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

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

$$\hat{\alpha}_j(1) = \alpha_j(1) = a_{Ij}b_j(y_1)\,. \tag{9.12}$$

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:

$$\hat{P}(y_1, y_2, \cdots, y_T) = \hat{\alpha}_F(T) = \max_{\text{over } i}\left(\hat{\alpha}_i(T)a_{iF}\right). \tag{9.13}$$

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^{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^{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 jth state for the tth 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.

$$\gamma_j(t) = P(s_t = j \mid y_1, y_2, \cdots, y_T) = \frac{P(y_1, y_2, \cdots, y_T \mid s_t = j)P(s_t = j)}{P(y_1, y_2, \cdots, y_T)}$$

$$= \frac{P(y_1, y_2, \cdots, y_T, s_t = j)}{P(y_1, y_2, \cdots, y_T)} = \frac{\alpha_j(t)\beta_j(t)}{\alpha_F(T)} \qquad (9.19)$$

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:

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

In Equation (9.20), quantities for the e^{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^{th} frame of the example, with $\gamma_j(t, e)$ being used for the value of $\gamma_j(t)$ for the e^{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:

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

[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:

$$\xi_{Ij}(0) = \frac{a_{Ij}b_j(y_1)\beta_j(1)}{\alpha_F(T)} . \tag{9.23}$$

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 γ_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:

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

where $\xi_{ij}(t, e)$ denotes the value of $\xi_{ij}(t)$ for the e^{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:

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

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:

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

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**, μ, and the **standard deviation**, σ. For a quantity x, the probability density, $\phi(x)$, is given by:

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

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^{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**