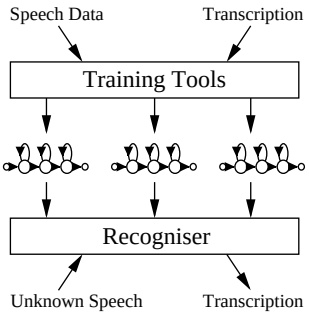
Chapter 1

The Fundamentals of HTK



HTK is a toolkit for building Hidden Markov Models (HMMs). HMMs can be used to model any time series and the core of HTK is similarly general-purpose. However, HTK is primarily designed for building HMM-based speech processing tools, in particular recognisers. Thus, much of the infrastructure support in HTK is dedicated to this task. As shown in the picture above, there are two major processing stages involved. Firstly, the HTK training tools are used to estimate the parameters of a set of HMMs using training utterances and their associated transcriptions. Secondly, unknown utterances are transcribed using the HTK recognition tools.

The main body of this book is mostly concerned with the mechanics of these two processes. However, before launching into detail it is necessary to understand some of the basic principles of HMMs. It is also helpful to have an overview of the toolkit and to have some appreciation of how training and recognition in HTK is organised.

This first part of the book attempts to provide this information. In this chapter, the basic ideas of HMMs and their use in speech recognition are introduced. The following chapter then presents a brief overview of HTK and, for users of older versions, it highlights the main differences in version 2.0 and later. Finally in this tutorial part of the book, chapter 3 describes how a HMM-based speech recogniser can be built using HTK. It does this by describing the construction of a simple small vocabulary continuous speech recogniser.

The second part of the book then revisits the topics skimmed over here and discusses each in detail. This can be read in conjunction with the third and final part of the book which provides a reference manual for HTK. This includes a description of each tool, summaries of the various parameters used to configure HTK and a list of the error messages that it generates when things go wrong.

Finally, note that this book is concerned only with HTK as a tool-kit. It does not provide information for using the HTK libraries as a programming environment.

1.1 General Principles of HMMs

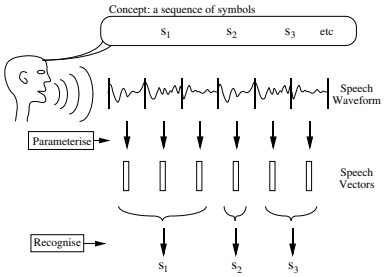


Fig. 1.1 Message Encoding/Decoding

Speech recognition systems generally assume that the speech signal is a realisation of some message encoded as a sequence of one or more symbols (see Fig. 1.1). To effect the reverse operation of recognising the underlying symbol sequence given a spoken utterance, the continuous speech waveform is first converted to a sequence of equally spaced discrete parameter vectors. This sequence of parameter vectors is assumed to form an exact representation of the speech waveform on the basis that for the duration covered by a single vector (typically 10ms or so), the speech waveform can be regarded as being stationary. Although this is not strictly true, it is a reasonable approximation. Typical parametric representations in common use are smoothed spectra or linear prediction coefficients plus various other representations derived from these.

The rôle of the recogniser is to effect a mapping between sequences of speech vectors and the wanted underlying symbol sequences. Two problems make this very difficult. Firstly, the mapping from symbols to speech is not one-to-one since different underlying symbols can give rise to similar speech sounds. Furthermore, there are large variations in the realised speech waveform due to speaker variability, mood, environment, etc. Secondly, the boundaries between symbols cannot be identified explicitly from the speech waveform. Hence, it is not possible to treat the speech waveform as a sequence of concatenated static patterns.

The second problem of not knowing the word boundary locations can be avoided by restricting the task to isolated word recognition. As shown in Fig. 1.2, this implies that the speech waveform corresponds to a single underlying symbol (e.g. word) chosen from a fixed vocabulary. Despite the fact that this simpler problem is somewhat artificial, it nevertheless has a wide range of practical applications. Furthermore, it serves as a good basis for introducing the basic ideas of HMM-based recognition before dealing with the more complex continuous speech case. Hence, isolated word recognition using HMMs will be dealt with first.

1.2 Isolated Word Recognition

Let each spoken word be represented by a sequence of speech vectors or *observations* $\boldsymbol{O}$, defined as

$$\boldsymbol{O} = \boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T \tag{1.1}$$

where $\boldsymbol{o}_t$ is the speech vector observed at time t. The isolated word recognition problem can then be regarded as that of computing

$$\arg\max_i \{P(w_i|\boldsymbol{O})\} \tag{1.2}$$

where w_i is the i'th vocabulary word. This probability is not computable directly but using Bayes' Rule gives

$$P(w_i|\boldsymbol{O}) = \frac{P(\boldsymbol{O}|w_i)P(w_i)}{P(\boldsymbol{O})} \tag{1.3}$$

Thus, for a given set of prior probabilities $P(w_i)$, the most probable spoken word depends only on the likelihood $P(\boldsymbol{O}|w_i)$. Given the dimensionality of the observation sequence $\boldsymbol{O}$, the direct estimation of the joint conditional probability $P(\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots|w_i)$ from examples of spoken words is not practicable. However, if a parametric model of word production such as a Markov model is

assumed, then estimation from data is possible since the problem of estimating the class conditional observation densities $P(\boldsymbol{O}|w_i)$ is replaced by the much simpler problem of estimating the Markov model parameters.

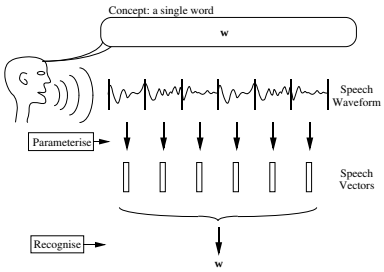


Fig. 1.2 Isolated Word Problem

I n HMM based speech recognition, it is assumed that the sequence of observed speech vectors corresponding to each word is generated by a Markov model as shown in Fig. 1.3. A Markov model is a finite state machine which changes state once every time unit and each time t that a state j is entered, a speech vector $\boldsymbol{o}_t$ is generated from the probability density $b_j(\boldsymbol{o}_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}. Fig. 1.3 shows an example of this process where the six state model moves through the state sequence $X = 1, 2, 2, 3, 4, 4, 5, 6$ in order to generate the sequence $\boldsymbol{o}_1$ to $\boldsymbol{o}_6$. Notice that in HTK, the entry and exit states of a HMM are non-emitting. This is to facilitate the construction of composite models as explained in more detail later.

The joint probability that $\boldsymbol{O}$ is generated by the model M moving through the state sequence X is calculated simply as the product of the transition probabilities and the output probabilities. So for the state sequence X in Fig. 1.3

$$P(\boldsymbol{O}, X|M) = a_{12}b_2(\boldsymbol{o}_1)a_{22}b_2(\boldsymbol{o}_2)a_{23}b_3(\boldsymbol{o}_3)\ldots \tag{1.4}$$

However, in practice, only the observation sequence $\boldsymbol{O}$ is known and the underlying state sequence X is hidden. This is why it is called a *Hidden Markov Model*.

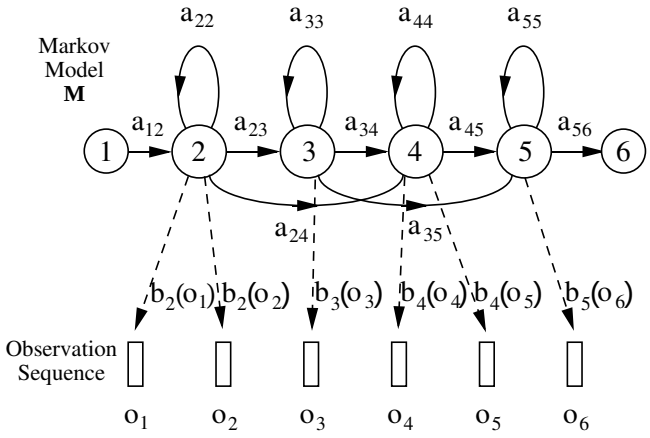


Fig. 1.3 The Markov Generation Model

Given that X is unknown, the required likelihood is computed by summing over all possible state sequences $X = x(1), x(2), x(3), \ldots, x(T)$, that is

$$P(\boldsymbol{O}|M) = \sum_X a_{x(0)x(1)} \prod_{t=1}^{T} b_{x(t)}(\boldsymbol{o}_t)a_{x(t)x(t+1)} \tag{1.5}$$

where $x(0)$ is constrained to be the model entry state and $x(T+1)$ is constrained to be the model exit state.

As an alternative to equation 1.5, the likelihood can be approximated by only considering the most likely state sequence, that is

$$\hat{P}(\boldsymbol{O}|M) = \max_{X}\left\{a_{x(0)x(1)}\prod_{t=1}^{T} b_{x(t)}(\boldsymbol{o}_t)a_{x(t)x(t+1)}\right\} \tag{1.6}$$

Although the direct computation of equations 1.5 and 1.6 is not tractable, simple recursive procedures exist which allow both quantities to be calculated very efficiently. Before going any further, however, notice that if equation 1.2 is computable then the recognition problem is solved. Given a set of models M_i corresponding to words w_i, equation 1.2 is solved by using 1.3 and assuming that

$$P(\boldsymbol{O}|w_i) = P(\boldsymbol{O}|M_i). \tag{1.7}$$

All this, of course, assumes that the parameters $\{a_{ij}\}$ and $\{b_j(\boldsymbol{o}_t)\}$ are known for each model M_i. Herein lies the elegance and power of the HMM framework. Given a set of training examples corresponding to a particular model, the parameters of that model can be determined automatically by a robust and efficient re-estimation procedure. Thus, provided that a sufficient number of representative examples of each word can be collected then a HMM can be constructed which implicitly models all of the many sources of variability inherent in real speech. Fig. 1.4 summarises the use of HMMs for isolated word recognition. Firstly, a HMM is trained for each vocabulary word using a number of examples of that word. In this case, the vocabulary consists of just three words: "one", "two" and "three". Secondly, to recognise some unknown word, the likelihood of each model generating that word is calculated and the most likely model identifies the word.

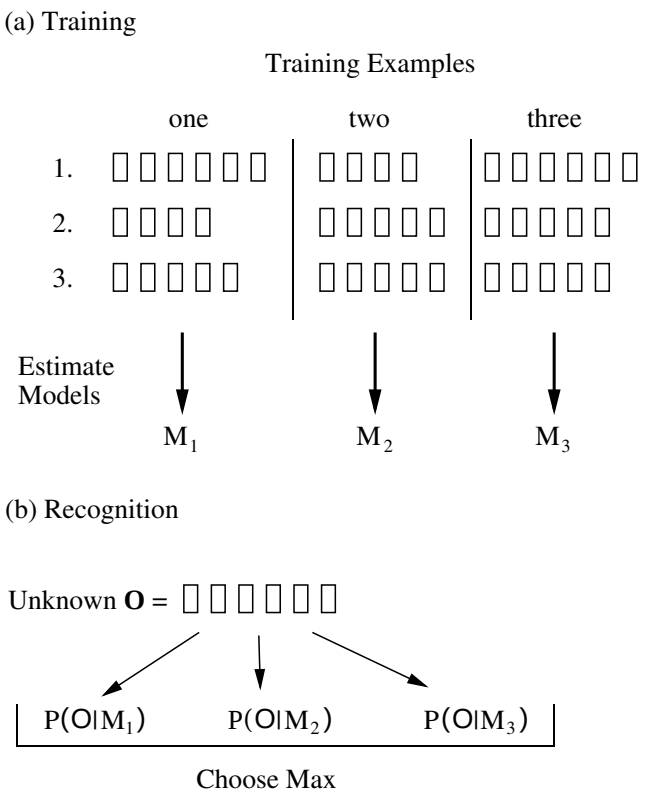


Fig. 1.4 Using HMMs for Isolated Word Recognition

1.3 Output Probability Specification

Before the problem of parameter estimation can be discussed in more detail, the form of the output distributions $\{b_j(\boldsymbol{o}_t)\}$ needs to be made explicit. HTK is designed primarily for modelling continuous parameters using continuous density multivariate output distributions. It can also handle observation sequences consisting of discrete symbols in which case, the output distributions are discrete probabilities. For simplicity, however, the presentation in this chapter will assume that continuous density distributions are being used. The minor differences that the use of discrete probabilities entail are noted in chapter 7 and discussed in more detail in chapter 11.

In common with most other continuous density HMM systems, HTK represents output distributions by Gaussian Mixture Densities. In HTK, however, a further generalisation is made. HTK allows each observation vector at time t to be split into a number of S independent data streams o_{st}. The formula for computing $b_j(\boldsymbol{o}_t)$ is then

$$b_j(\boldsymbol{o}_t) = \prod_{s=1}^{S} \left[\sum_{m=1}^{M_s} c_{jsm} \mathcal{N}(\boldsymbol{o}_{st}; \boldsymbol{\mu}_{jsm}, \boldsymbol{\Sigma}_{jsm}) \right]^{\gamma_s} \tag{1.8}$$

where M_s is the number of mixture components in stream s, c_{jsm} is the weight of the m'th component and $\mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a multivariate Gaussian with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, that is

$$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\boldsymbol{o}-\boldsymbol{\mu})} \tag{1.9}$$

where n is the dimensionality of $\boldsymbol{o}$.

The exponent γ_s is a stream weight[1]. It can be used to give a particular stream more emphasis, however, it can only be set manually. No current HTK training tools can estimate values for it.

Multiple data streams are used to enable separate modelling of multiple information sources. In HTK, the processing of streams is completely general. However, the speech input modules assume that the source data is split into at most 4 streams. Chapter 5 discusses this in more detail but for now it is sufficient to remark that the default streams are the basic parameter vector, first (delta) and second (acceleration) difference coefficients and log energy.

1.4 Baum-Welch Re-Estimation

To determine the parameters of a HMM it is first necessary to make a rough guess at what they might be. Once this is done, more accurate (in the maximum likelihood sense) parameters can be found by applying the so-called Baum-Welch re-estimation formulae.

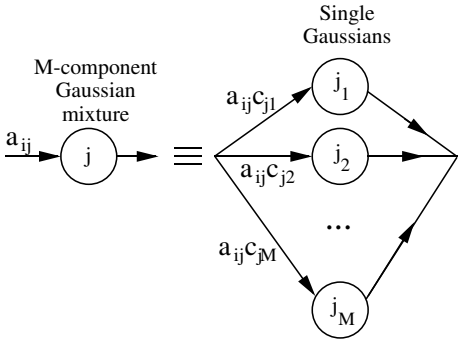


Fig. 1.5 Representing a Mixture

Chapter 8 gives the formulae used in HTK in full detail. Here the basis of the formulae will be presented in a very informal way. Firstly, it should be noted that the inclusion of multiple data streams does not alter matters significantly since each stream is considered to be statistically

[1] often referred to as a codebook exponent.

independent. Furthermore, mixture components can be considered to be a special form of sub-state in which the transition probabilities are the mixture weights (see Fig. 1.5).

Thus, the essential problem is to estimate the means and variances of a HMM in which each state output distribution is a single component Gaussian, that is

$$b_j(\boldsymbol{o}_t) = \frac{1}{\sqrt{(2\pi)^n|\boldsymbol{\Sigma_j}|}} e^{-\frac{1}{2}(\boldsymbol{o}_t-\boldsymbol{\mu}_j)'\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{o}_t-\boldsymbol{\mu}_j)} \tag{1.10}$$

If there was just one state j in the HMM, this parameter estimation would be easy. The maximum likelihood estimates of $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ would be just the simple averages, that is

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{o}_t \tag{1.11}$$

and

$$\hat{\boldsymbol{\Sigma}}_j = \frac{1}{T}\sum_{t=1}^{T} (\boldsymbol{o}_t - \boldsymbol{\mu}_j)(\boldsymbol{o}_t - \boldsymbol{\mu}_j)' \tag{1.12}$$

In practice, of course, there are multiple states and there is no direct assignment of observation vectors to individual states because the underlying state sequence is unknown. Note, however, that if some approximate assignment of vectors to states could be made then equations 1.11 and 1.12 could be used to give the required initial values for the parameters. Indeed, this is exactly what is done in the HTK tool called HInit. HInit first divides the training observation vectors equally amongst the model states and then uses equations 1.11 and 1.12 to give initial values for the mean and variance of each state. It then finds the maximum likelihood state sequence using the Viterbi algorithm described below, reassigns the observation vectors to states and then uses equations 1.11 and 1.12 again to get better initial values. This process is repeated until the estimates do not change.

Since the full likelihood of each observation sequence is based on the summation of all possible state sequences, each observation vector $\boldsymbol{o}_t$ contributes to the computation of the maximum likelihood parameter values for each state j. In other words, instead of assigning each observation vector to a specific state as in the above approximation, each observation is assigned to every state in proportion to the probability of the model being in that state when the vector was observed. Thus, if $L_j(t)$ denotes the probability of being in state j at time t then the equations 1.11 and 1.12 given above become the following weighted averages

$$\hat{\boldsymbol{\mu}}_j = \frac{\sum_{t=1}^{T} L_j(t)\boldsymbol{o}_t}{\sum_{t=1}^{T} L_j(t)} \tag{1.13}$$

and

$$\hat{\boldsymbol{\Sigma}}_j = \frac{\sum_{t=1}^{T} L_j(t)(\boldsymbol{o}_t - \boldsymbol{\mu}_j)(\boldsymbol{o}_t - \boldsymbol{\mu}_j)'}{\sum_{t=1}^{T} L_j(t)} \tag{1.14}$$

where the summations in the denominators are included to give the required normalisation.

Equations 1.13 and 1.14 are the Baum-Welch re-estimation formulae for the means and covariances of a HMM. A similar but slightly more complex formula can be derived for the transition probabilities (see chapter 8).

Of course, to apply equations 1.13 and 1.14, the probability of state occupation $L_j(t)$ must be calculated. This is done efficiently using the so-called *Forward-Backward* algorithm. Let the forward probability[2] $\alpha_j(t)$ for some model M with N states be defined as

$$\alpha_j(t) = P(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_t, x(t) = j|M). \tag{1.15}$$

That is, $\alpha_j(t)$ is the joint probability of observing the first t speech vectors and being in state j at time t. This forward probability can be efficiently calculated by the following recursion

$$\alpha_j(t) = \left[\sum_{i=2}^{N-1} \alpha_i(t-1)a_{ij}\right] b_j(\boldsymbol{o}_t). \tag{1.16}$$

[2] Since the output distributions are densities, these are not really probabilities but it is a convenient fiction.

This recursion depends on the fact that the probability of being in state j at time t and seeing observation $\boldsymbol{o}_t$ can be deduced by summing the forward probabilities for all possible predecessor states i weighted by the transition probability a_{ij}. The slightly odd limits are caused by the fact that states 1 and N are non-emitting[3]. The initial conditions for the above recursion are

$$\alpha_1(1) = 1 \tag{1.17}$$

$$\alpha_j(1) = a_{1j} b_j(\boldsymbol{o}_1) \tag{1.18}$$

for $1 < j < N$ and the final condition is given by

$$\alpha_N(T) = \sum_{i=2}^{N-1} \alpha_i(T) a_{iN}. \tag{1.19}$$

Notice here that from the definition of $\alpha_j(t)$,

$$P(\boldsymbol{O}|M) = \alpha_N(T). \tag{1.20}$$

Hence, the calculation of the forward probability also yields the total likelihood $P(\boldsymbol{O}|M)$.

The backward probability $\beta_j(t)$ is defined as

$$\beta_j(t) = P(\boldsymbol{o}_{t+1}, \ldots, \boldsymbol{o}_T | x(t) = j, M). \tag{1.21}$$

As in the forward case, this backward probability can be computed efficiently using the following recursion

$$\beta_i(t) = \sum_{j=2}^{N-1} a_{ij} b_j(\boldsymbol{o}_{t+1}) \beta_j(t+1) \tag{1.22}$$

with initial condition given by

$$\beta_i(T) = a_{iN} \tag{1.23}$$

for $1 < i < N$ and final condition given by

$$\beta_1(1) = \sum_{j=2}^{N-1} a_{1j} b_j(\boldsymbol{o}_1) \beta_j(1). \tag{1.24}$$

Notice that in the definitions above, the forward probability is a joint probability whereas the backward probability is a conditional probability. This somewhat asymmetric definition is deliberate since it allows the probability of state occupation to be determined by taking the product of the two probabilities. From the definitions,

$$\alpha_j(t)\beta_j(t) = P(\boldsymbol{O}, x(t) = j | M). \tag{1.25}$$

Hence,

$$\begin{aligned} L_j(t) &= P(x(t) = j | \boldsymbol{O}, M) \\ &= \frac{P(\boldsymbol{O}, x(t) = j | M)}{P(\boldsymbol{O}|M)} \\ &= \frac{1}{P} \alpha_j(t) \beta_j(t) \end{aligned} \tag{1.26}$$

where $P = P(\boldsymbol{O}|M)$.

All of the information needed to perform HMM parameter re-estimation using the Baum-Welch algorithm is now in place. The steps in this algorithm may be summarised as follows

1. For every parameter vector/matrix requiring re-estimation, allocate storage for the numerator and denominator summations of the form illustrated by equations 1.13 and 1.14. These storage locations are referred to as *accumulators*[4].

[3] To understand equations involving a non-emitting state at time t, the time should be thought of as being $t - \delta t$ if it is an entry state, and $t + \delta t$ if it is an exit state. This becomes important when HMMs are connected together in sequence so that transitions across non-emitting states take place *between frames*.

[4] Note that normally the summations in the denominators of the re-estimation formulae are identical across the parameter sets of a given state and therefore only a single common storage location for the denominators is required and it need only be calculated once. However, HTK supports a generalised parameter tying mechanism which can result in the denominator summations being different. Hence, in HTK the denominator summations are always stored and calculated individually for each distinct parameter vector or matrix.

2. Calculate the forward and backward probabilities for all states j and times t.
3. For each state j and time t, use the probability $L_j(t)$ and the current observation vector $\boldsymbol{o}_t$ to update the accumulators for that state.
4. Use the final accumulator values to calculate new parameter values.
5. If the value of $P = P(\boldsymbol{O}|M)$ for this iteration is not higher than the value at the previous iteration then stop, otherwise repeat the above steps using the new re-estimated parameter values.

All of the above assumes that the parameters for a HMM are re-estimated from a single observation sequence, that is a single example of the spoken word. In practice, many examples are needed to get good parameter estimates. However, the use of multiple observation sequences adds no additional complexity to the algorithm. Steps 2 and 3 above are simply repeated for each distinct training sequence.

One final point that should be mentioned is that the computation of the forward and backward probabilities involves taking the product of a large number of probabilities. In practice, this means that the actual numbers involved become very small. Hence, to avoid numerical problems, the forward-backward computation is computed in HTK using log arithmetic.

The HTK program which implements the above algorithm is called HREST. In combination with the tool HINIT for estimating initial values mentioned earlier, HREST allows isolated word HMMs to be constructed from a set of training examples using Baum-Welch re-estimation.

1.5 Recognition and Viterbi Decoding

The previous section has described the basic ideas underlying HMM parameter re-estimation using the Baum-Welch algorithm. In passing, it was noted that the efficient recursive algorithm for computing the forward probability also yielded as a by-product the total likelihood $P(\boldsymbol{O}|M)$. Thus, this algorithm could also be used to find the model which yields the maximum value of $P(\boldsymbol{O}|M_i)$, and hence, it could be used for recognition.

In practice, however, it is preferable to base recognition on the maximum likelihood state sequence since this generalises easily to the continuous speech case whereas the use of the total probability does not. This likelihood is computed using essentially the same algorithm as the forward probability calculation except that the summation is replaced by a maximum operation. For a given model M, let $\phi_j(t)$ represent the maximum likelihood of observing speech vectors $\boldsymbol{o}_1$ to $\boldsymbol{o}_t$ and being in state j at time t. This partial likelihood can be computed efficiently using the following recursion (cf. equation 1.16)

$$\phi_j(t) = \max_i \{\phi_i(t-1)a_{ij}\}\, b_j(\boldsymbol{o}_t). \tag{1.27}$$

where

$$\phi_1(1) = 1 \tag{1.28}$$

$$\phi_j(1) = a_{1j} b_j(\boldsymbol{o}_1) \tag{1.29}$$

for $1 < j < N$. The maximum likelihood $\hat{P}(O|M)$ is then given by

$$\phi_N(T) = \max_i \{\phi_i(T)a_{iN}\} \tag{1.30}$$

As for the re-estimation case, the direct computation of likelihoods leads to underflow, hence, log likelihoods are used instead. The recursion of equation 1.27 then becomes

$$\psi_j(t) = \max_i \{\psi_i(t-1) + log(a_{ij})\} + log(b_j(\boldsymbol{o}_t)). \tag{1.31}$$

This recursion forms the basis of the so-called Viterbi algorithm. As shown in Fig. 1.6, this algorithm can be visualised as finding the best path through a matrix where the vertical dimension represents the states of the HMM and the horizontal dimension represents the frames of speech (i.e. time). Each large dot in the picture represents the log probability of observing that frame at that time and each arc between dots corresponds to a log transition probability. The log probability of any path

is computed simply by summing the log transition probabilities and the log output probabilities along that path. The paths are grown from left-to-right column-by-column. At time t, each partial path $\psi_i(t-1)$ is known for all states i, hence equation 1.31 can be used to compute $\psi_j(t)$ thereby extending the partial paths by one time frame.

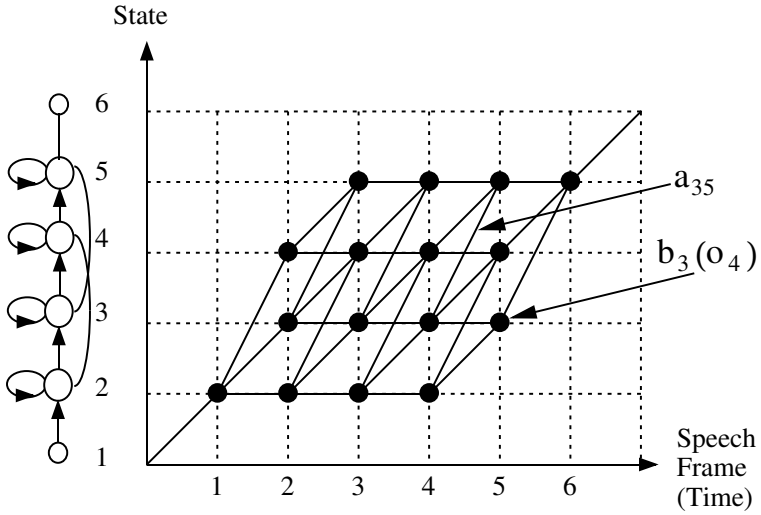


Fig. 1.6 The Viterbi Algorithm for Isolated Word Recognition

This concept of a path is extremely important and it is generalised below to deal with the continuous speech case.

This completes the discussion of isolated word recognition using HMMs. There is no HTK tool which implements the above Viterbi algorithm directly. Instead, a tool called HVite is provided which along with its supporting libraries, HNet and HRec, is designed to handle continuous speech. Since this recogniser is syntax directed, it can also perform isolated word recognition as a special case. This is discussed in more detail below.

1.6 Continuous Speech Recognition

Returning now to the conceptual model of speech production and recognition exemplified by Fig. 1.1, it should be clear that the extension to continuous speech simply involves connecting HMMs together in sequence. Each model in the sequence corresponds directly to the assumed underlying symbol. These could be either whole words for so-called *connected speech recognition* or sub-words such as phonemes for *continuous speech recognition*. The reason for including the non-emitting entry and exit states should now be evident, these states provide the *glue* needed to join models together.

There are, however, some practical difficulties to overcome. The training data for continuous speech must consist of continuous utterances and, in general, the boundaries dividing the segments of speech corresponding to each underlying sub-word model in the sequence will not be known. In practice, it is usually feasible to mark the boundaries of a small amount of data by hand. All of the segments corresponding to a given model can then be extracted and the *isolated word* style of training described above can be used. However, the amount of data obtainable in this way is usually very limited and the resultant models will be poor estimates. Furthermore, even if there was a large amount of data, the boundaries imposed by hand-marking may not be optimal as far as the HMMs are concerned. Hence, in HTK the use of HInit and HRest for initialising sub-word models is regarded as a *bootstrap* operation[5]. The main training phase involves the use of a tool called HERest which does *embedded training*.

Embedded training uses the same Baum-Welch procedure as for the isolated case but rather than training each model individually all models are trained in parallel. It works in the following steps:

[5] They can even be avoided altogether by using a *flat start* as described in section 8.3.

1. Allocate and zero accumulators for all parameters of all HMMs.
2. Get the next training utterance.
3. Construct a composite HMM by joining in sequence the HMMs corresponding to the symbol transcription of the training utterance.
4. Calculate the forward and backward probabilities for the composite HMM. The inclusion of intermediate non-emitting states in the composite model requires some changes to the computation of the forward and backward probabilities but these are only minor. The details are given in chapter 8.
5. Use the forward and backward probabilities to compute the probabilities of state occupation at each time frame and update the accumulators in the usual way.
6. Repeat from 2 until all training utterances have been processed.
7. Use the accumulators to calculate new parameter estimates for all of the HMMs.

These steps can then all be repeated as many times as is necessary to achieve the required convergence. Notice that although the location of symbol boundaries in the training data is not required (or wanted) for this procedure, the symbolic transcription of each training utterance is needed.

Whereas the extensions needed to the Baum-Welch procedure for training sub-word models are relatively minor[6], the corresponding extensions to the Viterbi algorithm are more substantial.

In HTK, an alternative formulation of the Viterbi algorithm is used called the *Token Passing Model* [7]. In brief, the token passing model makes the concept of a state alignment path explicit. Imagine each state j of a HMM at time t holds a single moveable token which contains, amongst other information, the partial log probability $\psi_j(t)$. This token then represents a partial match between the observation sequence $\boldsymbol{o}_1$ to $\boldsymbol{o}_t$ and the model subject to the constraint that the model is in state j at time t. The path extension algorithm represented by the recursion of equation 1.31 is then replaced by the equivalent *token passing algorithm* which is executed at each time frame t. The key steps in this algorithm are as follows

1. Pass a copy of every token in state i to all connecting states j, incrementing the log probability of the copy by $log[a_{ij}] + log[b_j(\boldsymbol{o}(t)]$.
2. Examine the tokens in every state and discard all but the token with the highest probability.

In practice, some modifications are needed to deal with the non-emitting states but these are straightforward if the tokens in entry states are assumed to represent paths extended to time $t - \delta t$ and tokens in exit states are assumed to represent paths extended to time $t + \delta t$.

The point of using the Token Passing Model is that it extends very simply to the continuous speech case. Suppose that the allowed sequence of HMMs is defined by a finite state network. For example, Fig. 1.7 shows a simple network in which each word is defined as a sequence of phoneme-based HMMs and all of the words are placed in a loop. In this network, the oval boxes denote HMM instances and the square boxes denote *word-end* nodes. This composite network is essentially just a single large HMM and the above Token Passing algorithm applies. The only difference now is that more information is needed beyond the log probability of the best token. When the best token reaches the end of the speech, the route it took through the network must be known in order to recover the recognised sequence of models.

[6] In practice, a good deal of extra work is needed to achieve efficient operation on large training databases. For example, the HERest tool includes facilities for pruning on both the forward and backward passes and parallel operation on a network of machines.

[7] See "Token Passing: a Conceptual Model for Connected Speech Recognition Systems", SJ Young, NH Russell and JHS Thornton, CUED Technical Report F_INFENG/TR38, Cambridge University, 1989. Available by anonymous ftp from `svr-ftp.eng.cam.ac.uk`.

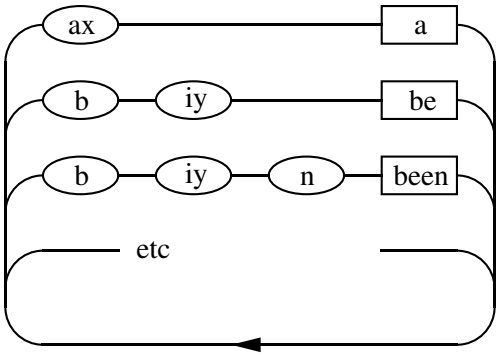


Fig. 1.7 Recognition Network for Continuously Spoken Word Recognition

The history of a token's route through the network may be recorded efficiently as follows. Every token carries a pointer called a *word end link*. When a token is propagated from the exit state of a word (indicated by passing through a word-end node) to the entry state of another, that transition represents a potential word boundary. Hence a record called a *Word Link Record* is generated in which is stored the identity of the word from which the token has just emerged and the current value of the token's link. The token's actual link is then replaced by a pointer to the newly created WLR. Fig. 1.8 illustrates this process.

Once all of the unknown speech has been processed, the WLRs attached to the link of the best matching token (i.e. the token with the highest log probability) can be traced back to give the best matching sequence of words. At the same time the positions of the word boundaries can also be extracted if required.

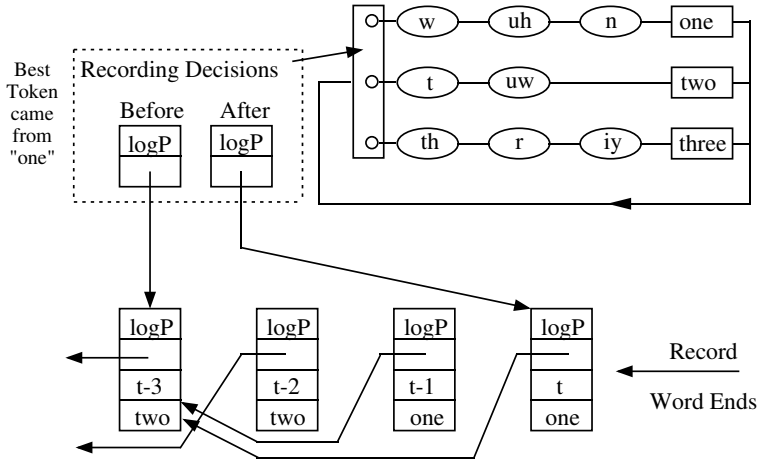


Fig. 1.8 Recording Word Boundary Decisions

The token passing algorithm for continuous speech has been described in terms of recording the word sequence only. If required, the same principle can be used to record decisions at the model and state level. Also, more than just the best token at each word boundary can be saved. This gives the potential for generating a lattice of hypotheses rather than just the single best hypothesis. Algorithms based on this idea are called *lattice N-best*. They are suboptimal because the use of a single token per state limits the number of different token histories that can be maintained. This limitation can be avoided by allowing each model state to hold multiple-tokens and regarding tokens as distinct if they come from different preceding words. This gives a class of algorithm called *word*

N-best which has been shown empirically to be comparable in performance to an optimal N-best algorithm.

The above outlines the main idea of Token Passing as it is implemented within HTK. The algorithms are embedded in the library modules HNet and HRec and they may be invoked using the recogniser tool called HVite. They provide single and multiple-token passing recognition, single-best output, lattice output, N-best lists, support for cross-word context-dependency, lattice rescoring and forced alignment.

1.7 Speaker Adaptation

Although the training and recognition techniques described previously can produce high performance recognition systems, these systems can be improved upon by customising the HMMs to the characteristics of a particular speaker. HTK provides the tools HERest and HVite to perform adaptation using a small amount of enrollment or adaptation data. The two tools differ in that HERest performs offline supervised adaptation while HVite recognises the adaptation data and uses the generated transcriptions to perform the adaptation. Generally, more robust adaptation is performed in a supervised mode, as provided by HERest, but given an initial well trained model set, HVite can still achieve noticeable improvements in performance. Full details of adaptation and how it is used in HTK can be found in Chapter 9.