The HTK Hidden Markov Model Toolkit: Design and Philosophy

SJ Young

CUED/F-INFENG/TR.152
September 6, 1994

Cambridge University Engineering Department
Trumpington Street, Cambridge, CB2 1PZ

(sjy@eng.cam.ac.uk)

1 Introduction

HTK is an integrated suite of software tools for building and manipulating continuous density Hidden Markov Models (HMMs). It consists of a set of library modules and a set of more than 20 tools (programs). It is written in ANSI C and runs mainly on UNIX-based systems although it can run under any modern OS. It is currently used in over 100 speech laboratories around the world and it is also used for teaching in a number of Universities. A HTK-based recogniser was included in both the ARPA September 1992 Resource Management Evaluation and the November 1993 Wall Street Journal CSR Evaluation, where in both cases performance was comparable with the systems developed by the main ARPA contractors (see section 9).

HTK was designed to be a general-purpose platform for research, benchmark testing and product development[Young *et al* 1993]. Its libraries provide an effective programming environment for implementing new algorithms and its tools allow a variety of recognisers to be built quickly and efficiently.

Although HTK is general-purpose, it also encourages a particular approach to building speech recognition systems. Firstly, HTK is restricted to continuous density systems in preference to discrete systems because, for the purposes of research, they have a number of mathematically desirable properties and, for the purposes of practical application, they are believed to be more robust. Secondly, parameter tying is regarded as being an essential requirement and hence HTK provides a generalised mechanism which allows tying at all levels. Thirdly, HTK fosters an incremental approach to model building whereby a system of HMMs is refined through a number of stages involving interleaved model manipulation and model re-estimation. Finally, HTK acknowledges that building a complex HMM-based system involves manipulating a diverse range of data including speech, transcriptions, and dictionaries. It therefore provides a rich set of integrated tools to facilitate these activities.

HTK is fully described by its user, reference and programmer manuals. However, these do not address the underlying concepts of its design nor do they make any direct attempt to explain its philosophy. The aim of this paper is to address just these issues.

The remainder of the paper is organised as follows. It begins in section 2 with a general overview of HTK and this is followed in section 3 by a discussion of its core, that is, the methods that are used to represent HMMs both internally and externally. Section 4 discusses HTK's incremental model building philosophy and section 5 describes the design of the recognition tools. The user interface design is then described in section 6. HTK is designed to support large-scale system building and as such, optimisation is an important concern. Section 7 describes the optimisations built in to HTK for both training and recognition. Section 8 discusses some of the relevant software design aspects of HTK and finally section 9 describes a number of the applications and systems which have been built using HTK. This includes a summary of the performance results obtained on the three major recognition benchmarks: phoneme recognition on the TIMIT database; 1000 word recognition on the Resource Management Database; and 5,000 and 20,000 word recognition on the Wall Street Journal Database.

2 Overview of the HTK Toolkit

The HTK toolkit is designed to facilitate the construction of systems using continuous density Gaussian mixture HMMs[Liporace1982, Juang 1985, Juang *et al* 1986, Bahl *et al* 1987]. It consists of a number of tools (programs) and a comprehensive set of library interface modules. The library modules ensure that all tools behave in a uniform way and they also simplify the development of new tools.

2.1 The HTK Library

The HTK library consists of ten modules that provide a fixed interface between tools and the outside world, and they also provide a variety of useful support functions. The library modules are listed in Table 1.

Name	Module Function
`HDbase`	Training token database
`HGraf`	Interactive graphics interface
`HLabel`	Label file i/o
`HMath`	Additional math support
`HMem`	Memory management
`HModel`	HMM definitions & i/o
`HParse`	Grammar support
`HShell`	Operating system interface
`HSigP`	Signal processing routines
`HSpIO`	Speech data file i/o

Table 1: HTK library modules

Figure 1 illustrates the way in which these modules are used by a typical training tool which reads a set of existing HMM definitions and some training data, and then produces new HMM files. The input and output of HMM definitions is performed by `HModel` which converts between the external textual representation of a HMM and the internal memory representation. Speech data is input via `HSpIO`. This module can read both waveform and parameterised data in most standard data formats. It can also perform automatic parameter conversions. For example, first and second difference coefficients can be automatically appended *on-the-fly* to reduce external data storage requirements with minimal computational overhead. The module `HDBase` gives efficient access to speech segments for tools that need to cyclically process the training data examples, and handling of the data transcription files is performed by `HLabel`. `HShell` provides command line argument handling and other facilities that enable HTK tools to work in a uniform manner across a range of operating systems, `HMem` provides facilities for efficient memory allocation and de-allocation, and the module `HGraf` supplies a simple event-driven graphics interface. Finally, `HParse` converts a set of syntax rules (including word-pair grammars, simple loops, optional terms etc.) to a network representation and is used primarily by the recognition tools.

2.2 HTK Tools

There are over 25 tools in the current HTK distribution and table 2 lists a selection of them. The way that these tools are used to build HMM-based systems is discussed in some detail in section 4 and this varies, of course, depending on the type of HMMs required. However, as a brief example, the construction of a continuous speech sub-word recogniser would involve the following steps.

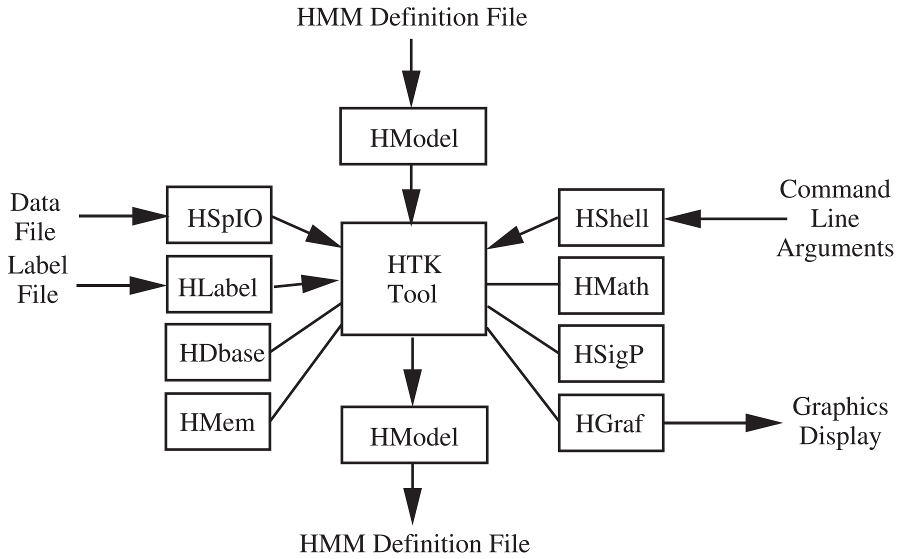

Figure 1: Typical HTK Tool and Library Interactions

Name	Tool Function
`HAlign`	Perform forced alignments
`HCode`	Speech analysis (LPC, MFCC etc.)
`HDEd`	Batch mode dictionary editor
`HERest`	Embedded Baum-Welch re-estimation
`HHEd`	Batch mode HMM editor
`HInit`	Isolated unit segmental k-means for model initialisation
`HLEd`	Batch mode label file editor
`HList`	List the contents of a data file
`HRest`	Isolated unit Baum-Welch re-estimation
`HResults`	Results analysis
`HSLab`	Simple interactive label file editor
`HSnor2Lab`	Convert SNOR orthography to label files
`HSnor2Net`	Convert SNOR dictionary to HParse network
`HSource`	generate data using a HMM as statistical source
`HVite`	Viterbi decoder (isolated & connected)

Table 2: The main HTK tools

Firstly, the speech data is parameterised using `HCode` and phone level transcription files are prepared. The latter may be derived from a dictionary and a set of SNOR[1] format orthographic transcriptions using `HSnor2Lab` and subsequently modified using the label editor `HLEd`. If necessary, the dictionary itself may need editing using `HDEd`.

Initial monophone (i.e. context independent) models are created from a small amount of transcribed bootstrap data using the tools `HInit` and `HRest`. Alternatively, if no transcribed data is available, it is possible to use a *flat-start* procedure whereby all models are given the same initial parameters and embedded training is used straightaway.

HTK has an *incremental build* philosophy at the core of which is the HMM Editor `HHEd` and the embedded re-estimation tool `HERest`. Starting with the initial monophone models, the HMMs are repeatedly *edited* and *re-estimated* until the required level of model complexity and performance is reached. Typical edit operations include cloning, tying/untying, adding/removing transitions, modifying the input data vector sizes, and mixture component incrementing.

`HERest` is the main training tool. It provides an embedded training facility that does not need a time aligned transcription—only the sequence of models in each training utterance is required. `HERest` also has a facility for training optional units by allowing models to have a direct transition from the non-emitting entry state to the non-emitting exit state. This can be used to allow optional silence between words. `HERest` is designed to support large training databases and includes pruning on the forward and backward passes, and parallel operation over a number of machines. It also allows many different types of tied structures to be trained.

If the dictionary used has multiple pronunciations, then new label transcriptions should be generated once reasonable models have been obtained and then all the models should be re-estimated using the new more accurate transcriptions. A suitable `HParse` word pronunciation network can be generated automatically from a SNOR format dictionary using `HSnor2Net` and then a realignment can be performed efficiently using this network by `HAlign` to yield the mostly likely pronunciation for each training file.

Once trained, the models are tested using `HVite`. This is a general purpose Viterbi recognition tool that recognises word sequences according to a given `HParse` syntax. `HVite` features include a beam search (pruning) mechanism; support for word-recognition using sub-word units (pronunciation graphs defined by any `HParse` network); bigram support at both the phone and word level; control of insertions via either a fixed penalty or a grammar scale factor; and also a number of computational optimisations for shared parameters. The label files produced by `HVite` can be scored by `HResults` which uses a dynamic programming string matching procedure which is compatible with the standard NIST scoring software. `HResults` provides a number of output formats and options including overall statistics, confusion matrices, and per speaker analyses. It will also generate Receiver Operating Characteristic (ROC) information and Figure of Merit (FOM) scores for word-spotting applications.

3 The Representation of Hidden Markov Models in HTK

This section discusses the way that Hidden Markov Models are represented within HTK, both externally and internally. Intrinsic to these representations is the mechanism for

[1] Standard normal orthographic representation used by NIST.

parameter tying. Hence this will be described in some detail including the associated re-estimation process.

3.1 The HMM Parameter Set

A HMM in HTK consists of an arbitrary number of states N where the entry state 1 and the exit state N are non-emitting confluent states. The output distribution associated with each state may depend on one or more statistically independent streams and the probability of each independent stream can be exponentially weighted by a set of stream weights $\{\gamma_s\}$. If $\boldsymbol{o}_{st}$ is the input observation vector of dimensionality V in stream s at time t and $b_{js}(\boldsymbol{o}_{st})$ is the probability of that vector in state j, then the total probability $b_j(\boldsymbol{o}_t)$ of the composite input vector $\boldsymbol{o}_t$ at time t in state j is

$$b_j(\boldsymbol{o}_t) = \prod_{s=1}^{S} b_{js}(\boldsymbol{o}_{st})^{\gamma_s}$$

Each individual stream probability $b_{js}(\boldsymbol{o}_{st})$ is represented by a mixture Gaussian so that, if there are M_s mixture components in stream s then

$$b_{js}(\boldsymbol{o}_{st}) = \sum_{m=1}^{M_s} c_{jsm}\mathcal{N}(\boldsymbol{o}_{st}; \boldsymbol{\mu}_{jsm}, \boldsymbol{\Sigma}_{jsm})$$

where c_{jsm} is the weight of mixture m in stream s of state j and $\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes a multivariate Gaussian of mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, that is

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

In addition to the state output distribution parameters, there are a number of ancillary parameters. Firstly, the transition probabilities between states are represented by a single $N \times N$ matrix $\{a_{ij}\}$. Secondly, each state may have a duration probability distribution associated with it represented by the parameter set $\{d_k\}$. Also, the complete model may have a duration probability distribution associated with it represented by the parameter set $\{D_l\}$. Note however that HTK does not prescribe the form of these parameters beyond allowing them to be named (e.g. Poisson, Gamma, etc.) and none of the standard HMM tools support them.

There are very few limits in HTK beyond those imposed by available memory. Thus, for example, any model can have any number of states and any number of components per mixture. The very general formulation of the output probability distributions is designed to support a number of different modelling paradigms and be flexible enough to satisfy future needs. HTK fully supports standard single-stream and multiple-stream mixture Gaussians, and multiple stream tied mixture systems. However, the HMM representation also encompasses features which are not directly supported by the supplied tools but are included for those who wish to use HTK as a basis for research. For example, multiple streams can be used to model disparate sources of data and the state-based stream weights can be used to combine these sources discriminatively. As alluded to earlier, various forms of durational model can be incorporated both at the state level and the model level. Covariance matrices can be full, diagonal or non-square. The latter form is provided to allow arbitrary sub-space transformations to be used.

3.2 The External Representation of HMMs

The full set of parameters which constitute a HMM definition can be arranged in a hierarchy as shown in Fig 2 in which the top of the hierarchy is on the left descending to the leaves on the right.

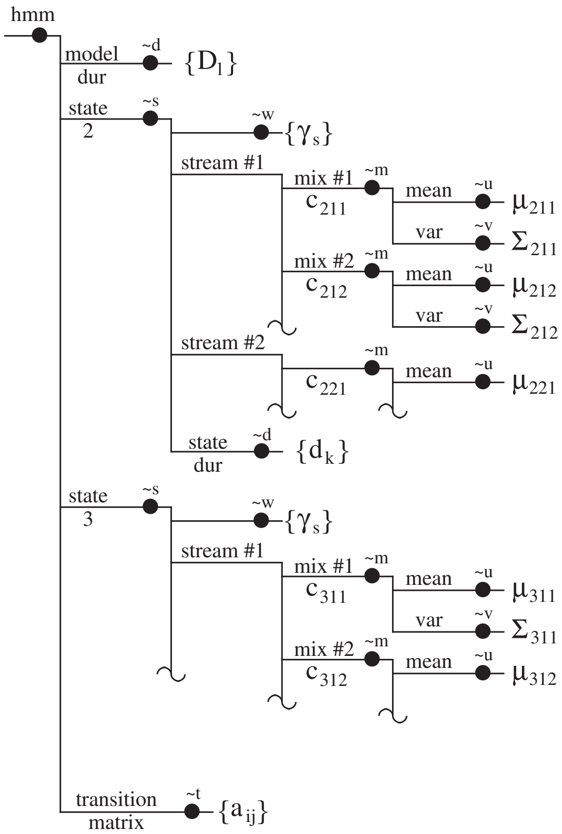

Figure 2: The Hierarchy of HMM Parameters

Externally, HMM definitions are stored in text files using a simple formal language whose structure mirrors this hierarchy. The full syntax and semantics of this language are defined in the HTK Reference Manual, here it will be sufficient to just give a flavour of it by means of a few examples. Figure 3 shows a HMM definition file which represents a single-stream single-mixture diagonal covariance left-right HMM with 5 states (3 emitting). Keywords are indicated by surrounding angle brackets and the whole definition is bracketted by the keywords `<BeginHMM>` and `<EndHMM>`. The notation is meant to be easily readable by humans whilst being efficient to parse by machine. The general structure is that global parameters are given first followed by the parameters for each state. Thus, in the example, the HMM has 5 states (`<NumStates>`), a vector size of 8 (`<VecSize>`), diagonal covariance (`<DIAGC>`), no duration parameters (`<NULLD>`) and data parameterisation consisting of Mel-Frequency Cepstral Coefficients (`<MFCC>`). The data for each state consists of an 8 dimensional mean vector followed by an 8 dimensional variance vector. Finally, the keyword (`<TransP>`) introduces the 5×5 transition matrix.

Referring back to Fig 2, it can be seen that the layout of the above definition file

```
<BeginHMM>
<NumStates> 5   <VecSize> 8
<DIAGC> <NULLD> <MFCC>
<State> 2
  <Mean> 8
    -5.949  4.748  2.095  3.653  0.587 -2.360  1.752 -1.277
  <Variance> 8
     31.68  1.131  0.696  0.606  1.321  0.366  1.642  2.285
<State> 3
  <Mean> 8
    -0.032  5.125  1.868  3.430  1.017 -3.460  1.622 -0.881
  <Variance> 8
     3.261  9.365  1.603  2.043  1.336  3.887  2.380  0.215
<State> 4
  <Mean> 8
     1.357  1.428 -2.536  3.976  1.035 -5.003 -0.651  0.152
  <Variance> 8
     6.035  12.07  2.617  4.613  0.973  3.095  0.991  1.097
<TransP> 5
  0.000 1.000 0.000 0.000 0.000
  0.000 0.628 0.372 0.000 0.000
  0.000 0.000 0.699 0.301 0.000
  0.000 0.000 0.000 0.526 0.474
  0.000 0.000 0.000 0.000 0.000
<EndHMM>
```

Figure 3: External Definition of a Single Gaussian 5 state HMM

follows the parameter hierarchy diagram except that the middle stream and mixture layers have been omitted and the defaults of a single stream and single Gaussian distributions have been assumed. Figure 4 shows a fragment of a more complex HMM definition in which most of the options are used. Notice that it is not necessary to have the same number of mixture components in each distribution. For each state, the number of mixture components in each stream is given by the `<NumMixes>` keyword. Thus, in the example, state 2 has 2 mixture components in stream 1 and 3 components in stream 2. Notice also that stream weights may be set individually for each stream but the number of streams is fixed across all states of all models.

The above two examples give the general flavour of the HMM definition language and they also demonstrate its flexibility. There is, however, still one major feature of the HTK HMM definition language left to explain. Referring back to Fig 2, each large black dot on this diagram represents a parameter tie-point. Any pair of similar dots can be joined so that the entire sub-structure to the right of the dots is shared. This mechanism is implemented within external HMM definitions by using a simple *macro definition* mechanism whereby the shared sub-structure is given a macro name and defined separately in a macro definition file. Each time that the shared sub-structure is used in the actual definition file, the macro name is given instead. Every macro name consists of a type identifier followed by a user-defined name. Type identifiers consist of a tilde followed by a single mnemonic character, the various kinds are shown in Fig 2. As an example of this mechanism, Fig 5 shows the same HMM definition as in Fig 3 except that all of the variance vectors are shared, the single shared variance vector being defined in the separate macro definition file called `macros`. When this HMM definition is used (e.g. in a recognition tool), it behaves as if each variance had been replicated in the definition file. However, when the HMM is re-estimated by a training tool, the behaviour is not

```
<BeginHMM>
 <NumStates> 5 <VecSize> 12 <StreamInfo> 6 6
 <DIAGC> <NULLD> <MFCC_D>
 <State> 2 <NumMixes> 2 3 <SWeights> 1.0 0.8
  <Stream> 1
    <Mixture> 1 0.43
      <Mean> 6
         4.748  2.095  4.663  0.587 1.752 -1.277
      <Variance> 6
         1.131  0.696  1.606  1.351 1.632  2.285
    <Mixture> 2 0.57
      <Mean> 6
        -5.989  3.728  1.195  4.653  1.333 -2.260
      <Variance> 6
        21.684  1.131  0.696  0.606  1.321  0.366
  <Stream> 2
    <Mixture> 1 0.32
      <Mean> ..... <Variance> .....
    <Mixture> 2 0.16
      <Mean> ..... <Variance> .....
    <Mixture> 3 0.52
      <Mean> ..... <Variance> .....
 <State> 3 <NumMixes> 3 1 <SWeights> 1.0 0.8
  <Stream> 1
    <Mixture> 1 0.25
      <Mean> .....
    etc
<EndHMM>
```

Figure 4: External Definition of a Mixture-Gaussian, Multiple-Stream HMM

the same. In this latter case, the data which would have been used to re-estimate each individual variance is pooled and the variances therefore remain identical. If the same values had been simply replicated in the definition file, then they would most likely be different after re-estimation. This is the parameter tying mechanism that underlies much of HTK's operation and it is discussed further below.

Note that macro definitions can contain other macro definitions provided that the nested definitions are defined before they are used in the macro file. In practice, it is not often that either HMM definition files or macro definition files are written by hand. They are usually generated by HTK tools as part of the incremental build process outlined in section 4. Thus, there is rarely any need in practice to understand the detailed rules for the construction of HMM and macro definition files.

As well as tying sub-structures of HMMs, HTK also allows whole HMMs to be tied together, however, the mechanism used for this is different. Every HMM in HTK has both a *logical* and a *physical* name and these are stored in a *HMM List*. The logical and physical names are commonly the same, however, they need not be. When several logical HMMs share the same physical name they are effectively tied and during re-estimation the training data for each logical HMM in a tied set is pooled to yield just one set of (physical) HMM parameters.

3.3 Internal Representation of HMMs and Parameter Estimation

Internally, HMMs are stored in memory with the same hierarchical organisation as is used for the external definitions with the one important difference that the parameter tying

```
<BeginHMM>
<Use> macros
<NumStates> 5   <VecSize> 8
<DIAGC> <NULLD> <MFCC>
<State> 2
  <Mean> 8
    -5.949  4.748  2.095  3.653  0.587 -2.360  1.752 -1.277
  ~v gvar
<State> 3
  <Mean> 8
    -0.032  5.125  1.868  3.430  1.017 -3.460  1.622 -0.881
  ~v gvar
<State> 4
  <Mean> 8
     1.357  1.428 -2.536  3.976  1.035 -5.003 -0.651  0.152
  ~v gvar
<TransP> 5
  0.000 1.000 0.000 0.000 0.000
  0.000 0.628 0.372 0.000 0.000
  0.000 0.000 0.699 0.301 0.000
  0.000 0.000 0.000 0.526 0.474
  0.000 0.000 0.000 0.000 0.000
<EndHMM>
```

File: macros

```
~v gvar
  <Variance> 8
     31.68  1.131  0.696  0.606  1.321  0.366  1.642  2.285
```

Figure 5: External Definition Showing the Use of a Macro to Create a Tied Variance Vector

achieved by using macros is explicit in the internal representation. Each parameter set in the hierarchy is referenced by its parent using an address pointer. When a parameter is tied, then it has multiple parents referencing it.

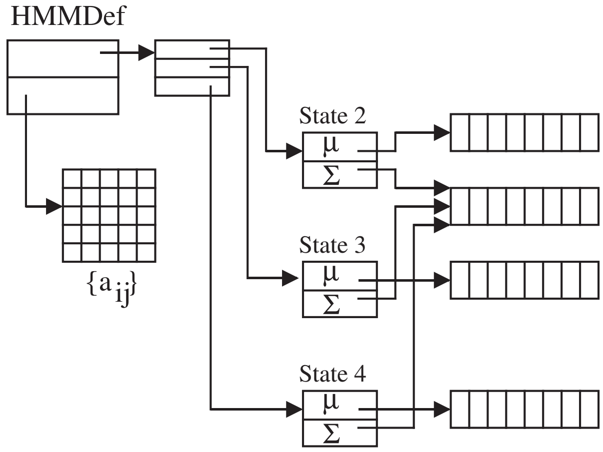

Figure 6: Internal Representation Showing a Tied Variance Vector

Figure 6 illustrates in a greatly simplified form how the example in Fig 5 in the previous section would be stored internally[2]. As can be seen, each state points to a unique mean vector but points to the same shared variance vector. Note that viewed from the top down, there is no difference between a tied and a non-tied system. Thus, for most operations tying is transparent and this includes parameter estimation.

HTK uses Baum-Welch re-estimation for estimating HMM parameters from training data. Parameter estimates using this procedure are based on weighted averages. For the simple case of a single-stream single-Gaussian HMM and a parameter belonging to state j, the weight for an observation $\mathbf{o}_t$ at time t is the likelihood $L_j(t)$ of being in state j at time t. For example, the re-estimation formula for the variance is[3]

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

$L_j(t)$ depends on the current HMM parameters and hence this formula is applied iteratively until the required convergence is achieved.

From the above it is clear that to re-estimate a parameter, storage is needed to accumulate the numerator of the re-estimation formula and to separately accumulate the denominator. The numerator has the same dimensions as the parameter itself and the denominator is effectively just a counter. In HTK, these accumulators are attached directly to the parameter being re-estimated and in this way, the Baum-Welch procedure works properly independently of whether or not the parameter is tied[Young 1992].

Fig 7 illustrates this process. Every shareable vector in HTK has space allocated at the head of the vector to store a pointer to an accumulator. During re-estimation, the

[2] The HTK Reference Manual describes the actual internal representation in terms of C data structures.

[3] The mean vector in this formula should in fact be the new estimate and not the existing estimate. However, it is simpler to use the existing estimate and empirically no significant difference has been found to arise from this practice.

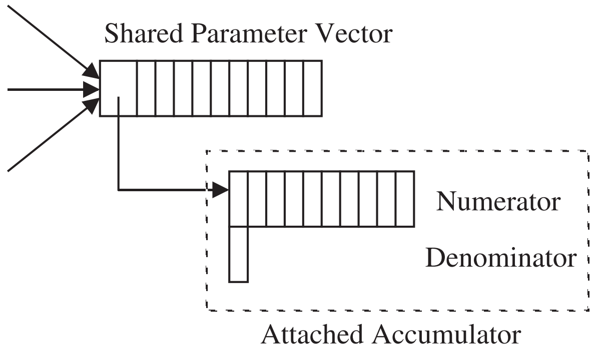

Figure 7: Parameter Re-estimation Involving a Tied Parameter Vector

shared parameter will be accessed via several different parents. Each time it is accessed, increments for the numerator and denominator will be added to the accumulators. The differing parents will not know or care that they are contributing to the same parameter. When all of the data has been processed, the numerators and denominators are used to update the parameter vectors to which they are attached.

By means of this simple storage organisation in which the accumulators needed for Baum-Welch re-estimation are attached directly to the parameter vectors themselves, the generalised tying facilities provided by HTK are effectively transparent to the re-estimation tools. Furthermore, it is simple to show that this tying process does not affect the convergence properties of the Baum-Welch algorithm[Bellegarda & Nahamoo 1990].

4 The HTK Model Build Philosophy

As noted previously, HTK adopts an incremental philosophy to the construction of HMM-based systems and the representations used for the models themselves have all been selected to give the flexibility needed to support this. In the introduction it was noted that HTK is based on continuous rather than discrete density distributions because of their improved robustness and desirable mathematical properties. However, the use of parametric mixture distributions has the further advantage that the precision of any individual distribution can easily be changed simply by adding or removing mixture components. In HTK, a component is added to a mixture simply by making a copy of the component with the largest weight, perturbing the means by ± 0.2 standard deviations and halving the weights. This mechanism is referred to as *mixture splitting* and although crude, when followed immediately by Baum-Welch re-estimation, it works well. Mixture splitting coupled with the generalised tying mechanism enables the complexity of a HMM-based system to be adjusted so that it can be carefully balanced against the amount of available training data.

The general approach to model building in HTK is illustrated by Fig 8. The input is a prototype HMM definition whose function is to specify the topology and global characteristics of each HMM. Using the prototype, a set of initial HMMs is created using `HInit`, `HRest` and a small amount of *bootstrap* data for which the model unit boundaries have been marked by hand. The set of initial models is then retrained on the whole training data set using `HERest` to perform embedded training for which only the sequence of units

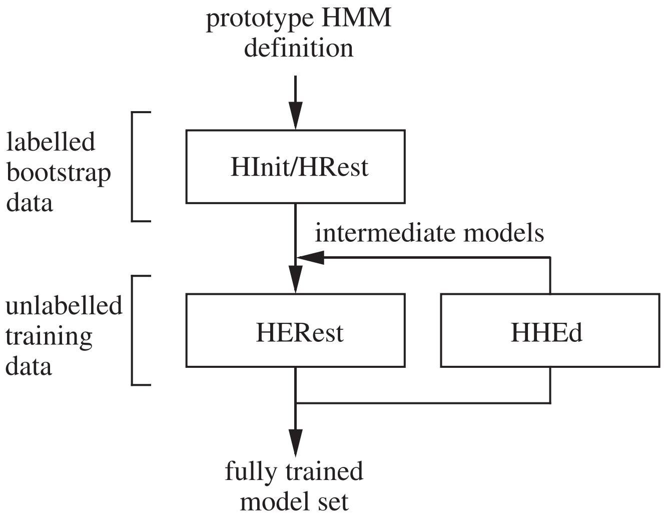

Figure 8: The HMM Construction Procedure

in each training utterance need be known. Once a set of fully trained models have been produced, they can be refined using the HMM editor `HHEd`.

For example, in small vocabulary whole word recognition with very limited training data, single-Gaussian HMMs can be built. If there is insufficient data to estimate individual state variances, then the variances can be tied either within models or globally. If more data becomes available for some units, then further mixture components can be added and the variances untied for those units.

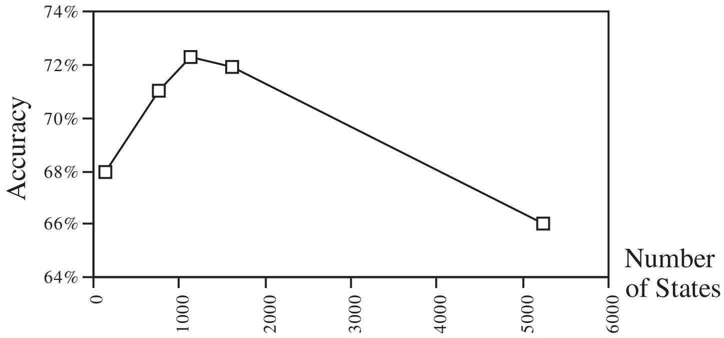

Figure 9: Recognition Accuracy vs. Number of Tied States in a Phoneme Recognition Task using the TIMIT Database

In large vocabulary continuous speech recognition, it is necessary to model contextual effects by using context-dependent sub-word units. For example, the commonest such unit is the triphone which is a phone model used only with specific left and right neighbours. For a typical 40 phone set, there are 40^3 potential triphones and there will not be sufficient data to robustly estimate the parameters of all of them. Furthermore, experimental evidence shows that mixture Gaussian distributions must be used to achieve adequate modelling accuracy creating a need for even more training data. These problems can again be solved by tying and mixture component incrementing. In particular, state tying has been found to be very effective[Woodland & Young 1992, Hwang & Huang 1992, Young & Woodland 1993, Hwang & Huang 1993]. The procedure for this is that firstly a set of single Gaussian triphones is trained and then the corresponding states within each allophone set are clustered and tied. The clustering process, ensures that the resulting tied states each have sufficient data to estimate more accurate mixture Gaussian distributions. The aim in this process is to find the optimal balance between modelling contextual effects and accurately modelling the underlying distributions. Figure 9 shows how recognition performance varies as a function of the number of tied states for a standard phoneme recognition task using the TIMIT database and right-context dependent biphone HMMs[Lee & Hon 1989]. For each system with a differing number of tied states, the number of mixture components was incremented until maximum performance was achieved. The far left of this graph corresponds to the case of context independent monophones and the far right corresponds to conventional context dependent biphones. As can be seen, there is a clear peak in performance when state tying is used.

In the TIMIT experiments, a data-driven agglomerative clustering procedure was used. It is also possible to use a phonetically-driven tree-based clustering procedure in which the decision at each tree node is based on a question about the left and right contexts[Bahl *et al* 1991, Odell 1992, Kannan *et al* 1994, Odell *et al* 1994]. Though more complex, this procedure has the advantage that, once built, the decision trees can be used to find appropriate state distributions to use for *unseen* triphones i.e. those triphones for which no examples occur in the training data.

From the above, it is clear that `HHEd` is a key component of the HTK toolkit. It operates by loading in a complete set of HMMs, applying a sequence of edit commands to them and then writing out a new set of definitions. Edit commands consist of a two character command name plus one or more arguments, one of which is typically an *item list*. The latter defines a set of similar items in the parameter hierarchy which are the target for some operation. For example, the following `HHEd` command will tie all variance vectors for mixture component 1 of states 2 to 6 of the HMM called `one`. The resulting macro is called `var01`.

```
TI var01 {one.state[2-6].mix[1].cov}
```

In general, the `TI` command simply ties all of the items in the item list together using the first item in the list as the exemplar. When mean vectors are tied, the exemplar is formed from the average and when covariances are tied, the exemplar is formed from the maximum over all the items.

As in HMM definitions, the syntax of an item list mirrors the HMM parameter hierarchy with an explicit stream indicator being optional since it is very often unity. Within the item list specification, the metacharacter `*` may be used in model names to match zero or more characters and `?` may be used to match a single character. For example, context dependent models in HTK use the naming convention `L-ph+R` where `ph` is the phone name, `L` is the left context and `R` is the right context. The following command would apply the state clustering procedure described above to state 2 of all allophones of the phone `ih`

```
TC 0.5 ih_2 {(*-ih,ih+*,*-ih+*).state[2]}
```

The first argument is the clustering threshold and the second is the root name of the macro to use such that for this example, the actual macros for each tied state would then be called ih_2.1,ih_2.2,ih_2.3, etc. A tied-state HMM system is therefore created simply by executing a command like this for each state of each allophone.

One special case, which should be mentioned is that an item list ending in the word `mix` without any component indices denotes a set of pdfs. When pdfs are specified in a `TI` command, the pdfs are *joined* rather than tied and the result is a *tied mixture* or *semi-continuous* system[Huang & Jack 1989, Bellegarda & Nahamoo 1990, Paul 1990]. For example, the commands

```
JO 128 2.0
TI mix {*.state[2-4].mix}
```

indicate that the pdfs of states 2 to 4 of all models are to be joined. In this case, all of the mixture components in all of the items specified by the list are sorted in order of mixture weight. The `JO` command sets the total number J of tied mixtures required. Mixture components are either discarded or split until the required number J is obtained and all of the mixture weights are normalised subject to the floor specified by the second argument of the `JO` command. Macros called `mix1`, `mix2`, etc. are then created and all pdfs share all 128 components. Since tied mixture systems are common, special notation is provided to represent them externally and various computations internal to HTK are optimised for them (see section 7).

This section has outlined the philosophy of model building in HTK. It is based on the notion of incremental building and successive refinement, primarily via interleaved executions of the HMM editor `HHEd` and the embedded re-estimation tool `HERest`. This allows model complexity to be balanced against the amount of available training data and the experience gained so far with this approach suggests that it is very effective.

5 Speech Recognition

HTK provides two recognition tools. Firstly, `HVite` is a Viterbi-based tool which can be used for connected whole-word recognition, continuous speech sub-word based recognition and word-spotting. Secondly, `HAlign` is a derivative of `HVite` which is specially configured to perform *forced alignments* down to the state level. Both of these tools gain their flexibility from the syntax definition language provided by `HParse`.

5.1 Syntax Definition

The function of a Viterbi decoder is to find the sequence of HMMs for which the likelihood of the unknown speech is maximum given that sequence. In practice, it is extremely useful to be able to prescribe the allowable sequence of models in order to constrain the recogniser to operate in some desired way. For example, in a voice-based control interface, only a few very limited commands may be allowed. These commands may be conveniently defined using a grammar. Similarly, in sub-word recognition, several different pronunciations for each word might be allowed and again a grammar can be used to specify these.

HTK uses an extended Backus Naur Form (EBNF) of grammar notation to specify recognition constraint networks[Wirth 1976]. EBNF is actually a context-free grammar notation, but in HTK all variables must be defined before they are used thus constraining the grammar to be regular. A full definition of the `HParse` network definition language is given in the Reference Manual. In brief, a network definition consists of an optional set of variable definitions followed by an expression which defines the actual network. Any name

Expression	Meaning (E,F,G are expressions)
a	the model a
$v	the variable v
E F G	sequence
$E \mid F \mid G$	alternatives, E or F or G in parallel
$(E \mid F)\ G$	factoring, same as $E\ G \mid F\ G$
{ E }	zero or more repetitions of E
< E >	one or more repetitions of E
<< E >>	context sensitive loop (see text)
$v = E;	define variable v as equivalent to E

Table 3: Network Definition Constructs

beginning with a '$' is a variable otherwise it is the name of a HMM. The right hand side of variable definitions and the network expression itself is built using the constructs listed in Table 3.

```
$cmd2 = turn (left | right);
$cmd2 = stop;
( sil <  ($cmd1 | $cmd2) sil > )
```

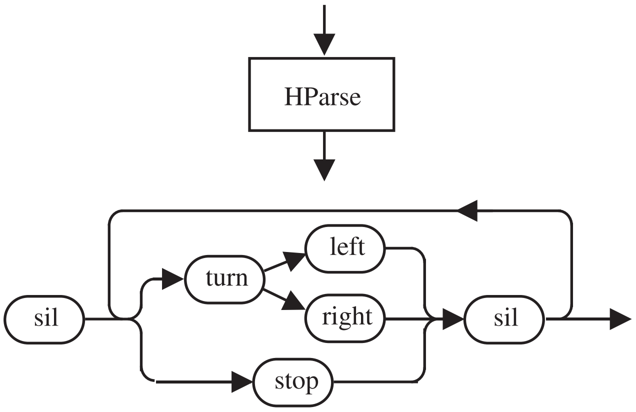

Figure 10: Network Construction using `HParse`

Each of the EBNF constructs maps into an equivalent network construct in a fairly obvious way. As a simple example, Fig 10 shows a simple network definition file which allows sequences of the form "turn left, turn right, stop" with a silence between each command. The figure also shows how the network definition would be translated into an actual network by `HParse`.

For sub-word recognition, the special names `WD_BEGIN` and `WD_END` are used to mark the beginning and end of a pronunciation. For example, Fig 11 shows a syntax definition which defines a simple word recogniser in which any word can follow any other word. Notice that the name of the word is attached to the `WD_BEGIN` and `WD_END` symbols and

```
$A       = WD_BEGIN%A (ax|ae) WD_END%A;
$ABDOMEN = WD_BEGIN%ABDOMEN ae b d ax m ax n WD_END%ABDOMEN;
$ABIDES  = WD_BEGIN%ABIDES ax b ay d z WD_END%ABIDES;
$ABOLISH = WD_BEGIN%ABOLISH (ax|ae) b aa l ih sh WD_END%ABOLISH;
 ... etc


( <
   $A | $ABDOMEN | $ABIDES | $ABOLISH | ... etc
> )
```

Figure 11: Defining Word Pronunciations

this is output by the recogniser instead of the constituent phone model names.

```
$V = iy | ih | eh | ae | ix | ax | ah | ax-h |  ... etc
$C = ch | j | dh | b | d | dx | g | p |  ... etc
$N = m | n | en | ng | em | nx | eng;
$L = l | el | r | y | w | er | axr;
$S = sil | pau;

( << V-iy+V | N-iy+V | L-iy+V | ... |
     L-vcl+C | C-vcl+C | sil >> )
```

Figure 12: Defining a Context-Sensitive Phone Loop

The philsophy in designing the HTK network definition language was to try to make it straightforward to define networks for the most common tasks. An alternative would have been to provide a language to define a finite state network directly but this was rejected on the basis that for many cases it would have been insufferably tedious. However, the lack of direct control over the form of the network constructed does have limitations. For example, it is not possible to define a triphone loop in such a way that only consistent triphone sequences are allowed. Similarly, at the word level, it is not possible to define word-pair grammars. To solve these specific problems, syntax network definitions may contain context sensitive loops. For example, the syntax in Fig 12 defines a network consisting of a set of context dependent phone models in parallel. Each context is a broad class and it is defined using a variable whose value is a list of phones. The resulting network will be such that each model `V-iy+V`, `N-iy+V`, `L-iy+V`, etc. is linked back to all and only those models whose contexts match according to the context definitions. For example, `V-p+N` would link back to all vowels which had a `C` right context such as `C-iy+C`.

5.2 Viterbi Decoding using Token Passing

Both `HVite` and `HAlign` are time-synchronous Viterbi decoders which find the sequence of HMMs which has the maximum likelihood of generating the unknown speech sequence and which conforms with the syntax constraints specified by the recognition network described above. In `HVite`, the recognition network is defined explicitly by the user and it is used for all input speech files. In `HAlign`, the network is constructed *on-the-fly* for each input speech file using a supplied transcription. Thus, `HVite` finds the most likely interpretation of each input file subject to the constraints of a recognition network, whereas `HAlign` does a *forced* recognition to find the optimal alignment between the HMM states and the speech data.

In HTK, Viterbi decoding is implemented using the *Token Passing Model* in which the concept of a state alignment path is made explicit[Young *et al* 1989, Young *et al* 1991]. In this model, each node of the recognition network corresponding to a HMM contains a single *HMM instance*. Each HMM instance contains storage sufficient to hold one *token* for each HMM state. At time t, the token in state j of HMM instance h represents the best partial match between the observation sequence $\boldsymbol{o}_1$ to $\boldsymbol{o}_t$ and some sequence of HMMs ending in state j of model h.

Initially, all possible HMM entry states contain a token with partial path probability of $log(1)$ and all other states hold a token with partial path probability of $log(0)$. At each time frame, every token is copied into all possible successor states and probabilities are updated according to the transition and output probabilities. Whenever this process would leave multiple tokens in the same state, all but the best token is discarded. When all input frames have been consumed, the exit states of all HMMs which can end the utterance are examined and the token with the highest log probability identifies the optimal sequence. In order to find this sequence, the propagation of tokens must be accompanied by some house-keeping operations.

The history of a token's route through the network is recorded efficiently as follows. In addition to the partial path probability, every token carries a single pointer called a *link*. When a token is propagated from the exit state of one HMM to the entry state of another, that transition represents a potential HMM boundary. Hence a record called a *Link Record* is generated which stores the identity of the HMM 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 Link Record. Figure 13 illustrates this process. Once all of the unknown speech has been processed, the Link Records 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 models. At the same time the positions of the word boundaries can also be extracted if required. During the recognition process, many Link Records will be generated which cannot lie on the optimal path. Hence, all active tokens are periodically traced-back to find all useful Link Records so that the remainder can be garbage-collected.

The above describes the essence of the Token Passing paradigm. It is a simple yet powerful conceptual model which can be extended in a number of ways. For example, `HVite` allows an additional bigram language model to be super-imposed on top of a syntax constraint network so that a model-to-model or word-to-word transition probability is added to the overall path probabilities. When word pronunciations are defined using the `WD_BEGIN` and `WD_END` constructs, Link Records are only generated on word transitions since there is no need to know the positions or identity of word-internal transitions. In `HAlign`, each token also holds a record of how many input speech frames it stayed in each state. This allows the trace back of a token's history to be recorded down to the state level.

The combination of the very general syntax definition mechanism, the facilities for

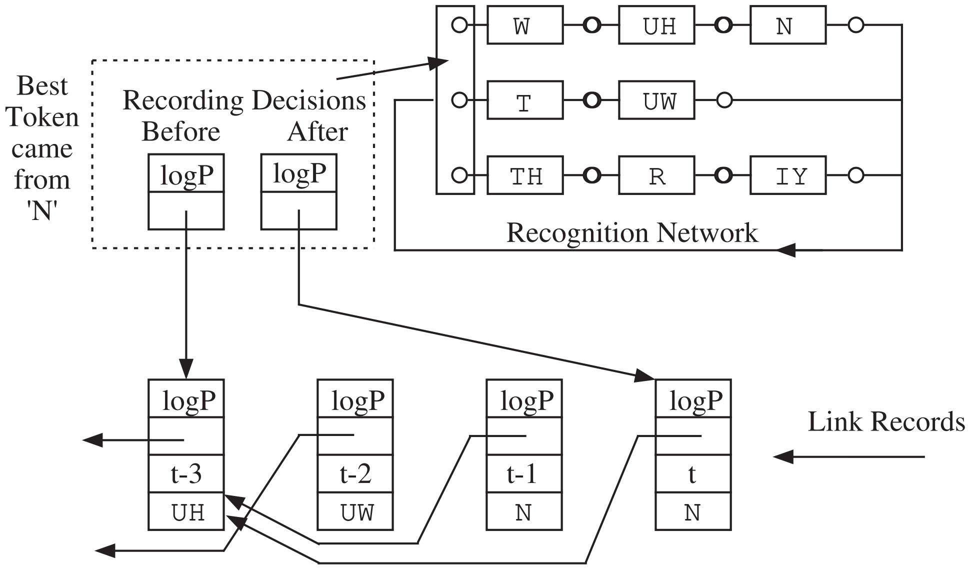

Figure 13: Link Record Generation during Token Propagation

word pronunciations and the ability to specify a bigram language model make the standard HTK recognition tools powerful, flexible and easy to use. Typical recognition tasks that can be handled include isolated and connected digits, phoneme recognition, simple task-oriented commands, phone-based word recognition upto 5000 words and various forms of word-spotting. Similarly, the flexibility of the syntax definition mechanism makes `HAlign` multi-purpose. When no syntax definition is given, `HAlign` aligns the sequence of models corresponding exactly to the transcription. This is useful for research purposes where the underlying state-sequence is required. However, when a syntax definition is given which contains word pronunciations and the corresponding transcriptions are orthographic, then `HAlign` will find the optimal pronunciation for each word in each utterance. This is extremely useful for building a phone-based word recogniser for which only othographic transcriptions of the training data are available.

6 User Interface Design

HTK tools are typically invoked by typing a command with the form

```
HTool -a arg -b arg .... hmmlist sp1.mfc sp2.mfc ....
```

that is the name of the tool is followed by zero or more optional arguments followed by one or more file names. This form of interface was chosen in preference to a more interactive WIMP-style of interface to make HTK portable across operating systems and to ensure that it was simple to execute HTK tools from within scripts. For user convenience, typing the name of a HTK tool alone without any arguments prints a summary of how to invoke the tool. All option names consist of a single character as shown in the example. To improve consistency, all upper-case options have the same meaning across all tools. For example, the option `-T N` sets the trace level to N for all tools.

HTK tools typically process three kinds of file: HMM definitions, speech files and transcriptions (label files). Transcriptions consist of a sequence of labels with optional start and end boundary times. However, even if present, the boundary times are ignored during embedded training which is the commonest case. In general, every speech file has a corresponding transcription.

In the example above, the first argument to the generic HTK tool `HTool` is a *HMM list*. This is a file which contains a list of HMM definitions to load. The files `sp1.mfc`, `sp2.mfc`, etc will typically be the names of speech files to use to train or test this set of HMMs. For each such speech file `spN.mfc`, HTK will expect to find a corresponding label file with the name `spN.lab`. Options exist to change the various extensions and the directories in which the searches for HMM definitions and label files are carried out. However, in many cases, the training or testing of a HMM system will involve many thousands of HMM definitions, and many thousands of speech files. To accommodate these situations, HTK has a number of additional mechanisms for avoiding the many problems of scale that would otherwise arise. Firstly, most operating systems have an upper limit on the number of files that can be specified on the command line. Hence, HTK allows some or all of the file name arguments of a command to be written in a separate file. Thus, for example, when running the embedded re-estimation tool `HERest`, the most common approach would be to list all the training files in a file called `train.scp` and then execute

```
HERest  .... -S train.scp hmmlist
```

this command then behaves as if all the contents of `train.scp` were appended to the command line after `hmmlist`. This simple facility is also useful for keeping a permanent record of which files were used to train any given system.

The need for a separate transcription for every speech file also causes efficiency problems. Firstly, most transcriptions will contain only a few bytes of information and storing this in a separate file is very inefficient. Secondly, it will commonly be the case that several speech files may require identical transcriptions. For example, in training an isolated digit recogniser, all examples of "one", say, might require a transcription containing just the labels "silence one silence". To increase efficiency, HTK provides the concept of a *Master Label File (MLF)*. This is a single file which contains a number of transcriptions stored sequentially within it. Each such transcription is preceded by a pattern and terminated by a period. The pattern can be a complete file name, in which case the behaviour would be identical to the case where each transcription was stored in a separate file. However, if the pattern contains the *wildcard* metacharacters `*` or `?`, then many speech files can reference the same transcription by simple pattern matching. For example, if an MLF contained

```
....

one*.lab
silence
one
silence
.

....
```

then any speech file whose name had the form `one.NN.mfc` would expect a corresponding transcription with a name of the form `one.NN.lab`. Since the latter matches the pattern preceding the transcription "silence one silence", all such speech files would reference the same transcription.

Master label files can also be used to redirect the search for a transcription to another directory thereby allowing a HTK tool to access label files dispersed throughout a large database. Similar facilities also exist for storing a set of HMM definitions in a single *Master Model File*.

7 Optimisations

HTK is designed to support the construction of large-scale HMM-based systems and for this reason it is important to reduce the computational burden wherever possible. To this end, HTK includes a number of optimisations in its implementation of the standard Baum-Welch and Viterbi decoding algorithms.

7.1 Parameter Tying

The major load in both training and recognition using continuous density HMMs is in the computation of output probabilities. Parameter tying not only allows more robust estimation from sparse data, it also allows, in certain cases, more efficient computation of output probabilities. The basic mechanism used in HTK to exploit parameter tying is very simple. When a shared single Gaussian, a shared stream distribution or a shared complete state distribution is evaluated, its value is stored so that if some other *owner* attempts to re-evaluate the same quantity, the pre-computed value can be used.

One special but common case of the above is where all mixture component Gaussians are tied across all states of all models in order to create a so-called *Tied Mixture System*. The key point about this form of tying is that all Gaussian mixture components are computed once globally and then each individual output distribution is formed from a weighted sum over all these Gaussians. However, since many of the individual Gaussians will have a very low probability for any given observation vector, it is only worth summing those Gaussian probabilities which fall within a beam relative to the most likely. To do this, the set of Gaussians is sorted into rank order and only those at the top of the list within the beam are used in the output probability calculations.

7.2 Pruning

The implementation of the Baum-Welch re-estimation formulae requires that the forward probabilities $\alpha_j(t)$ and the backward probabilities $\beta_j(t)$ be computed for all states j and all time frames t. In the case of the embedded training performed by `HERest`, a composite model consisting of the sequence of HMMs corresponding to the speech file transcription is matched against an utterance which may be several hundred frames long. In this case, it is extremely wasteful to compute the α and β probabilities for all j and t, so instead a forward–backward beam search is used as illustrated in Fig 14.

The backward probabilities are computed first. Starting at the last speech frame, the last HMM in the sequence is activated and β values are computed backwards in time. At each time t, the beam of active models is extended by one nearer the start and $\beta_j(t)$ is computed for all states j of all active models. During this process the maximum value of β, $\tilde{\beta}$ say, is recorded. When all active models have been processed at time t, the beam is

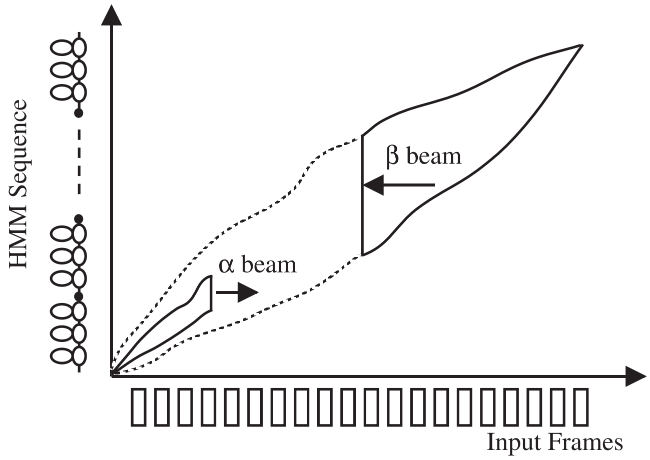

Figure 14: Forward–Backward Pruning

shrunk until all models in the beam have at least one state j for which

$$log(\tilde{\beta}) - log(\beta_j(t)) < c_1$$

where c_1 is a user-defined threshold.

Once all of the β values are known, the α values are calculated in a similar way. However, in this case, since the β values are already known, it is possible to calculate the actual state occupation likelihoods from the $\alpha\beta$ product so that all models in the beam have at least one state j for which

$$logP(O|M) - log(\alpha_j(t)\beta_j(t)) < c_2$$

where $logP(O|M)$ is the total log likelihood calculated from $\beta(0)$ and c_2 is a fixed threshold. Since the α beam depends on actual state occupation likelihoods, the threshold c_2 can be set quite precisely and the α beam itself is much narrower and lies within the β beam.

In order to minimise memory and computation, storage for the β values is created on demand and output probabilities are calculated on demand. On the forward pass, only $\alpha(t)$ and $\alpha(t-1)$ are stored and the actual re-estimation formulae numerator and denominator sums are updated at each α step.

The Viterbi decoder implemented by `HVite` also uses a beam search similar in principle to that used by the β pass described above. In this case however, there are a number of extra complications. Firstly, the simple sequence of HMMs assumed in re-estimation is replaced by an arbitrary network of HMMs in recognition. Secondly, in phone-based word recognition, pruning can be applied at both the word and phone level.

These are solved in `HVite` by keeping a list of all active HMM instances in the beam in a global array. Inter-model token propagation is then modified such that only the output tokens from active models whose scores lie within the beam are passed to successor models. If the receiving model is inactive then it is activated. After each cycle, all active models are checked and any model containing no tokens within the beam is deactivated. As in the above, the beam is calculated relative to the maximum partial path probability in any state at that time. In the case of recognition networks using the `WD_BEGIN` and `WD_END` constructs, a second list is maintained of active word ends to which a second independent beam can be applied.

8 Software Engineering

The general structure of HTK was outlined in section 2, and as noted there, HTK is intended to serve not just as a ready-to-use toolkit but also as a development environment for users to create new tools. As a consequence of this, HTK is distributed in source form and some care has been taken in engineering it.

HTK is written in ANSI standard C and most of the main data structures are represented by explicit data types. For example, HMM definitions are represented by the type `HMMDef` and an associated set of ancillary types such as `StateInfo`, `StateElem`, etc. defining the lower level structures in the HMM parameter hierarchy. In order to support these types, a set of routines is provided for accessing and manipulating them.

All of the types and routines associated with a particular logical function are bundled into a separate library module. The header file for that module defines the interface to the rest of the system. For example, the definitions and support routines for the `HMMDef` type are defined within `HModel.h` and `HModel.c`. Modules in HTK are thus designed using the modern principle of *Data Abstraction*. However, they are *not* actually abstract data types. Far from it, all HTK data types are very concrete. The full definition of each type is visible outside of the module that defines it and the program which uses that type is free to manipulate its innards. Thus, from a software engineering perspective, the construction of HTK is unsafe since it is all too easy for an external agent to corrupt the internal operation of a module. Furthermore, it is necessary for an external agent to have a detailed understanding of each library module data type in order to use it effectively. Again, the `HMMDef` type provides a good example since this type represents a large hierarchical structure which HTK tools need to traverse and manipulate. To do this they have to access and manipulate the structure directly and since it is complex, this kind of operation will be prone to error.

There are several reasons why HTK has been constructed like this. Firstly, and perhaps most importantly, it is very hard to isolate a complex data structure such as a `HMMDef` to make it secure, whilst at the same time allowing tools to use it efficiently. Since all of the main HTK tools are invariably *compute-bound*, the additional overheads incurred in accessing truly abstract data types were thought to be unacceptable. Secondly, one of the functions of HTK is to provide a programming environment to support research in speech recognition, and it is expected that individual researchers will wish to have direct access to HTK's internals at any level of detail. Thirdly, ANSI C was selected as the implementation language because it is widely available and portable. A more principled use of abstact data types would have demanded the use of a language which gave explicit support for data abstraction. At the time of HTK's inception, there was no such language which was also stable, portable, efficient and widely available.

HTK uses floating-point arithmetic throughout and all re-estimation and decoding algorithms use log arithmetic representations in preference to explicit scaling. All parameter storage and accumulators used in the re-estimation tools use single precision in order to conserve memory. However, double precision is used in the calculation of the forward (α) and backward (β) probabilites in the re-estimation tools, and in the accumulation of log probabilities in the recognition tool in order to ensure that there is no significant loss of numerical accuracy when processing very long observation sequences.

HTK's generalised parameter tying mechanism requires the ability to share both structures and arrays amongst a number of *owners* and to attach additional storage structures to them. To accommodate this, all shareable objects have a field called `hook` to store a pointer and a field called nUse to record the current number of owners. Nearly all HTK data structures are allocated dynamically and the nUse field is necessary to know when it is safe to dispose of a shared structure.

9 Applications and Systems

As noted in the introduction, HTK is now licensed for use at over 100 speech laboratories around the world and it is used for a wide variety of applications. In this final section, some examples of its use in the Cambridge Speech Group will be described to illustrate its flexibility and the performance that can be achieved using it.

One of the simplest applications of HTK is in whole word recognition. The basic approach is to define a prototype model for each word and then train it using the isolated word training tools HInit and HRest. The training data can consist either of isolated examples, or continuous speech for which the word end-points are known. Alternatively, if the training data is continuous speech and its orthography is known, HERest can be used for embedded training. In this type of application, the flexibility of the syntactic constraint network allows various recognition modes to be used and silence and noise models can be inserted into the network wherever they are required. This type of set-up has been used extensively in our study of noise compensation using Parallel Model Combination (PMC). In this work, speech models trained on clean speech are combined with a HMM trained on examples of the prevailing background noise to form a compensated set of HMMs whose parameters approximate those that would be obtained if the speech models had been trained directly on the noisy speech. Tested on both isolated and connected digit recognition tasks performed against a variety of background noises, typical performance at 0dB SNR improves from 30% or worse to over 97%[Gales & Young 1992, Gales & Young 1993a, Gales & Young 1993b].

The development of the TIMIT acoustic phonetic database has made it possible to use phoneme recognition as a standard test for the acoustic accuracy of a speech recogniser [Lee & Hon 1989]. To use HTK on this test, a training corpus of 3694 phonetically transcribed utterances was used to estimate the parameters of 48 left-right 3 state phoneme HMMs. To take account of co-articulation effects, the 48 context independent models were cloned to form 1748 right-context dependent biphone models. State tying, as described in section 4, was then used to reduce the total number of states from 5244 to 1142 and the single Gaussian distributions were incremented to 8 component mixture Gaussians. Using a context-sensitive phone loop and a context independent bigram computed using HLStats, a phone recognition rate of 76.7% correct and 72.3% accuracy was achieved[Young & Woodland 1993, Young & Woodland 1994].

HTK can be used to build speech recognition systems with vocabularies of several thousand words and a variety of grammatical constraints including word-pair and bigram language models. For example, tied-state word-internal triphone-based recognition systems have been built for the 1000 word Resource Management Task[Woodland & Young 1992, Woodland & Young 1993], the 2000 word Credit Card Corpus[Young *et al* 1994] and the 5000 word Wall Street Journal Task[Woodland *et al* 1994]. In each case, the recognition accuracy achieved compared favourably with systems developed specifically for those tasks.

Using a specially designed decoder, HTK has also been used to build 5000 word and 20000 word recognisers for the Wall Street Journal task which include full cross-word triphones and both bigram and trigram language models. In the ARPA November 1993 WSJ Evaluation, this HTK recogniser produced the lowest error rate on the 5000 word bigram (8.8%), 5000 word trigram (5.0%), and 20000 word bigram (14.5%); and the second lowest error rate on the 20000 word trigram (12.7%) [Woodland *et al* 1994].

In addition to building speech recognition systems, HTK has been used to build word spotting systems [James & Young 1994], speaker separation systems [Wang & Young 1992] and even face recognition systems [Samaria 1993]. It has also been used as a research environment for studies in discriminative training [Woodland 1992, Kapadia *et al* 1993], hybrid HMM/Neural Net systems [Young 1991, Valtchev *et al* 1993], prosody [Jones & Woodland 1993] and many others.

10 Acknowledgements

Through the years of developing HTK a number of people have made substantial contributions to its design and implementation. First and foremost, Phil Woodland has played a major role in improving and extending all aspects of HTK, and he is now co-developer with the author. Several members of the CUED Speech Group, both past and present, have assisted in debugging HTK, suggesting improvements and contributing code.

There are too many to acknowledge every contribution but of particular note are Valerie Beattie who co-authored early versions of the isolated word re-estimation tools `HInit` and `HRest`; Sadik Kapadia who helped tune the embedded re-estimator `HERest` and who wrote the source generator `HSource`; Julian Odell who developed the tree-based clustering software in `HHEd`; and Valtcho Valtchev who wrote the label editor `HSLab`. Others who have contributed to HTK over the years include David Cole, Mark Gales, Matthew Jones, Chris Leggetter, and Gordon Ramsay.

Many of the above contributions arose as a side-effect of developing research software which used HTK as a base. Much of this software is substantial and innovative; and one day some of it may find its way into the public version of HTK. In the mean time, its very existence is evidence that the author has achieved at least one of his goals in producing the HTK system.

References

[Browning *et al* 1993] Browning S, Russell M, Downey S. *Phoneme Decision Tree Construction for Automatic Speech Recognition.* DRA Memorandum No 4666, Defence Research Agency, Malvern, Worc.

[Bahl *et al* 1987] Bahl LR, Brown PF, de Souza PV, Mercer RL. *Speech recognition with continuous parameters hidden Markov models.* Computer Speech and Language, Vol 2, No 3/4, pp219-234.

[Bahl *et al* 1991] Bahl LR, de Souza PV, Gopalakrishnan PS, Nahamoo D, Picheny MA. *Context Dependent Modeling of Phones in Continuous Speech Using Decision Trees.* Proc DARPA Speech and Natural Language Processing Workshop, pp264-270, Pacific Grove, Calif, Feb.

[Bellegarda & Nahamoo 1990] Bellegarda JR, Nahamoo D. *Tied Mixture Continuous Parameter Modeling for Speech Recognition.* IEEE Trans ASSP Vol 38, No 12, pp2033-2045.

[Gales & Young 1992] Gales MJF, Young SJ. *An Improved Approach to the Hidden Markov Model Decomposition of Speeech and Noise.* Proc ICASSP, S35.1, pp233-236, San Francisco, March.

[Gales & Young 1993a] Gales MJF, Young SJ. *Cepstral Parameter Compensation for HMM Recognition in Noise.* Speech Communication, Vol 12, No 3, pp231-239.

[Gales & Young 1993b] Gales MJF, Young SJ. *HMM Recognition in Noise using Parallel Model Combination.* Proc Eurospeech '93, pp837-840, Berlin, Sept.

[Huang & Jack 1989] Huang XD, Jack MA. *Semi-continuous hidden Markov models for Speech Signals.* Computer Speech and Language, Vol 3, No 3, pp239-252.

[Hwang & Huang 1992] Hwang M-Y, Huang X. *Subphonetic Modeling with Markov States - Senone.* Proc ICASSP, Vol 1, pp33-36, San Francisco.

[Hwang & Huang 1993] Hwang M-Y, Huang X. *Shared Distribution Hidden Markov Models for Speech Recognition.* IEEE Trans Speech and Audio Processing, Vol 1, No 4, pp414-420.

[James & Young 1994] James D, Young SJ. *A Fast Lattice-Based Approach to Vocabulary Independent Wordspotting.* ICASSP 94, Adelaide.

[Jones & Woodland 1993] Jones M, Woodland PC. *Exploiting Variable Width Features in Large Vocabulary Speech Recognition.* Proc ICASSP, Vol 2, pp323-326, Minneapolis.

[Juang *et al* 1986] Juang B-H, Levinson SE, Sondhi MM. *Maximum Likelihood Estimation for Multivariate Mixture Observations of Markov Chains.* IEEE Trans on Information Theory, Vol 32, No 2, pp307-309.

[Juang 1985] Juang B-H. *Maximum-Likelihood Estimation for Mixture Multivariate Stochastic Observations of Markov Chains.* ATT&T Technical J, Vol 64, No 6, pp1235-1249.

[Kapadia *et al* 1993] Kapadia S, Valtchev V, Young SJ. *MMI Training for Continuous Parameter Recognition of the TIMIT Database.* Proc ICASSP, Minneapolis.

[Kannan *et al* 1994] Kannan A, Ostendorf M, Rohlicek JR. *Maximum Likelihood Clustering of Gaussians for Speech Recognition.* IEEE Trans on Speech and Audio Processing, July.

[Lee & Hon 1989] Lee K-F, Hon H-W. *Speaker Independent Phone Recognition Using Hidden Markov Models.* IEEE Trans ASSP, Vol 37, No 11, pp1641-1648.

[Liporace1982] Liporace LA. *Maximum-Likelihood Estimation for Multivariate Observations of Markov Sources.* IEEE Trans Information Th, Vol IT-28, No 5, pp729-734.

[Maxwell & Woodland 1993] Maxwell BA, Woodland PC. *Hidden Markov Models Using Shared Vector Linear Predictors.* Proc Eurospeech, p819-822, Berlin.

[Odell 1992] Odell JJ. *The Use of Decision Trees with Context Sensitive Phoneme Modelling.* MPhil Thesis, Cambridge University Engineering Department, Sept.

[Odell *et al* 1994] Odell JJ, Young SJ, Woodland PCW. *Tree-Based State Clustering for Large Vocabulary Speech Recognition.* IEEE Conf on Image, Speech and Neural Nets, Hong Kong.

[Paul 1990] Paul DB. *The Lincoln Tied Mixture HMM Continuous Speech Recogniser.* Proc DARPA Speech and Natural Language Workshop, pp332-336, Hidden Valley, Pennsylvania, June.

[Samaria 1993] Samaria F. *Face Segmentation for Identification Using Hidden Markov Models.* Proc 4th British Machine Vision Conference, Springer-Verlag.

[Valtchev *et al* 1993] Valtchev V, Kapadia S, Young SJ. *Recurrent Input Transformations for Hidden Markov Models.* Proc ICASSP, Minneapolis.

[Wang & Young 1992] Wang MQ, Young SJ. *Speech Recognition Using Hidden Markov Model Decomposition and a General Background Speech Model.* Proc ICASSP, S35.6, San Francisco, March.

[Wirth 1976] Wirth N. *Algorithms+Data Structures = Programs.* Prentice-Hall, Series in Automatic Computation, Englewood Cliffs, New Jersey.

[Woodland 1992] Woodland PC. *Using Vector Linear Prediction in Hidden Markov Models.* Proc ICASSP, San Francisco, March.

[Woodland & Young 1992] Woodland PC, Young SJ. *Benchmark DARPA RM Results with the HTK Portable HMM Toolkit.* Proc DARPA Continuous Speech Recognition Workshop, Stanford, Sept.

[Woodland & Young 1993] Woodland PC, Young SJ. *The HTK Continuous Speech Recogniser.* Proc Eurospeech '93, pp2207-2219, Berlin, Sept.

[Woodland *et al* 1994] Woodland PC, Odell JJ, Valtchev V, Young SJ. *Large Vocabulary Continuous Speech Recognition using HTK.* ICASSP 94, Adelaide.

[Young 1992] Young SJ. *The General Use of Tying in Phoneme-Based HMM Speech Recognisers.* Proc ICASSP, S66.5, San Francisco, March.

[Young *et al* 1989] Young SJ, Russell NH, Thornton JHS. *Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems.* Technical Report CUED/F-INFENG/TR38, Cambridge University Engineering Dept.

[Young *et al* 1991] Young SJ, Russell NH, Thornton JHS. *The Use of Syntax and Multiple Alternatives in the VODIS Voice Operated Database Inquiry System.* Computer Speech and Language, Vol 5, No 1, pp65-80.

[Young 1991] Young SJ. *Competitive Training: a Connectionist Approach to the Discriminative Training of Hidden Markov Models.* Proc IEE, Part I, Vol 138, No 1, pp61-68.

[Young & Woodland 1993] Young SJ, Woodland PC. *The Use of State Tying in Continuous Speech Recognition.* Proc Eurospeech '93, pp2203-2206, Berlin, Sept.

[Young & Woodland 1994] Young SJ, Woodland PC. *The Use of State Clustering in Continuous Speech Recognition.* Submitted to Computer Speech and Language.

[Young *et al* 1993] Young SJ, Woodland PC, Byrne WJ. *HTK Version 1.5: User, Reference and Programmer Manual.* Publ. Entropic Research Laboratories, Washington DC.

[Young *et al* 1994] Young SJ, Woodland PC, Byrne WJ. *Spontaneous Speech Recognition for the Credit Card Corpus using the HTK Toolkit.* Submitted to IEEE Trans Audio and Speech Processing, special issue on Robust Processing.