Modeling Biological Sequences Using HTK

William Noble Grundy
Department of Computer Science and Engineering
University of California, San Diego

Prepared for Entropic Research Laboratory, Inc.
March 27, 1997

1 Introduction

Entropic Research Laboratory's Hidden Markov Model Toolkit (HTK) [17, 18, 19, 20] is a software toolkit for designing and implementing state-of-the-art speech recognition systems. Recently, hidden Markov models (HMMs) have been applied by computational biologists to the task of characterizing families of related protein or DNA sequences. This report describes how HTK can be applied to this new problem domain. Using HTK, computational biologists can bring to bear many of the HMM techniques and algorithms developed by speech recognition researchers over the past two decades.

Hidden Markov models were first applied to problems in molecular biology in 1989 by Gary Churchill [5]. Krogh *et al.* [12] applied HMMs to protein modeling and brought widespread recognition to the approach. We refer to the linear HMMs described in that paper as "standard HMMs". Both of the most commonly used, publicly available software systems for hidden Markov modeling of biological sequences use these standard HMMs: the Sequence and Alignment Modeling System (SAM) [16, 11] developed by the Haussler group at UC Santa Cruz, and Sean Eddy's HMMER programs [10, 6].

Although the standard HMM topology is efficient and has proven to work well in general, the constraints it imposes are not appropriate for every problem. In particular, standard HMMs are acyclic: once a state has been traversed it can never be revisited. Many biological sequence families contain repeated domains for which an acyclic model is inappropriate. HTK allows the computational biologist to experiment with arbitrary HMM topologies, including models with cycles for repeated domains.

Furthermore, HTK allows computational biologists to exploit a long history of HMM research in speech recognition. HMMs were first applied to speech recognition in 1975 by J. K. Baker [3], and many important advances have been made in the ensuing 22 years. HTK might, for example, allow for feature-based models of biological sequences in which the relationships between features (e.g., domains or motifs) would be specified via a high-level grammar or other language model [13]. Word-spotting techniques, used in speech recognition to extract isolated words from a stream of background noise, might be applied to gene recognition or motif recognition tasks. And protein family recognition tasks, in which a single unknown protein is compared to large set of protein family HMMs, could be straightforwardly accomplished in HTK by combining all candidate models in parallel using a simple network specification.

Finally, HTK is a state-of-the-art software system with guaranteed technical support. In the ARPA tests evaluating all major speech recognition systems, HTK was ranked the best performing system overall in 1993 and 1994. The programs are available with source code, giving the researcher complete flexibility. HTK is currently used by speech researchers throughout the world. Entropic Research Laboratory provides these researchers with technical support.

This report describes how to build and use HTK models of protein families. The description is aimed at the computational biologist who is familiar with the use of HMMs. The following section describes how standard sequence formats can be converted to a format usable by HTK. The process of designing prototype models and of training them using known family members is described next. Finally, the procedure for comparing sequences to models and generating sequence scores is explained.

2 Sequence Input

As anyone who has worked with DNA or protein sequences knows, sequence data can be recorded in many different formats. Some of the more common formats include Fasta, GenBank, NBRF, EMBL, PIR, and ASN.1. Unfortunately, because HTK is primarily a speech recognition toolkit, it does not directly support any of these biological sequence formats. The easiest way to convert sequences into HTK format is to first convert them to a simple, standard format such as Fasta format, and then to write a conversion program to translate from Fasta to HTK format. The first step of this conversion process can be accomplished easily using the publicly available `readseq` sequence conversion program [8]. The second step is more complicated.

A program which translates from Fasta to HTK format must accomplish three separate tasks. First, whereas biological sequence data is typically stored in flat files containing many sequences, HTK expects each sequence to appear in a separate file. Thus, the conversion program must split a Fasta file containing n sequences into n separate files. Within each sequence, the program must convert the letters of the DNA or amino acid alphabet into numbers, since HTK expects numeric input. Finally, the program must store this numeric representation in a binary format readable by the computer. The details of how to effect this storage are given in Appendix A.

Note that the conversion program requires that a decision be made concerning

```
~ o <DISCRETE> <StreamInfo> 1 1
<BeginHMM>
  <NumStates> 4
  <State> 2
    <NumMixes> 20
    <DProb> 7105 7105 7105 7105 7105 7105 7105 7105 7105 7105
            7105 7105 7105 7105 7105 7105 7105 7105 7105 7105
  <State> 3
    <NumMixes> 20
    <DProb>  8361 14027 15473 13904  8472 13919 14743 4108 13547 1345
            10693 11262 14320 13488 13374 12351 11579 5123  9094 12865
  <State> 4
    <NumMixes> 20
    <DProb> 10436 12142 15140 14148  9870 14035 15094 2760 14021  7731
            12305 14317 14583 14520 12845 12649 11310 1330 16684 14137
  <State> 5
    <NumMixes> 20
    <DProb> 11737 14596 14152 14516 15130 14630 15585 9244 13276 13305
            13911 11918 14705 13580 13581  6972   347 9661 17696 16123
  <TransP> 5
  0.0 1.0 0.0 0.0 0.0
  0.0 0.9 0.1 0.0 0.0
  0.0 0.0 0.0 1.0 0.0
  0.0 0.0 0.2 0.0 0.8
  0.0 0.0 0.0 0.0 0.0
<ENDHMM>
```

Figure 1: A sample HTK HMM in text format. This particular HMM contains five states. State 2 has a self-loop, and states 3 through 5 are connected in a cycle.

ambiguous characters (e.g., for proteins, `B, U, X,` and `Z`); i.e., whether to code each ambiguous character separately or to combine one or more of these characters. Whatever convention is selected must be maintained in the later, modeling phase.

3 Designing Models

In order to train the parameters of a hidden Markov model, HTK needs a prototype model to use as a starting point. HTK stores hidden Markov models as text files in a simple, human readable format. A sample, four-state model specification is shown in Figure 1. The keyword "<DISCRETE>" indicates that this model (like all models of biological sequences) uses discrete emission probability distributions.

The first section of the HMM specification contains an emission probability distribution for each state. The "<NumMixes>" parameter at each state tells the number of characters in the alphabet and hence the number of bins in the discrete distribution. The HMM in Figure 1 models protein data, so it uses a 20-character alphabet. The discrete probability distribution appears after the keyword "<DProb>". Probabilities are scaled to integer values via the following equation:

$$P_{js}[v] = exp(-d_{js}[v]/2371.8)$$

where $d_{js}[v]$ is the stored discrete log probability for symbol v in stream s of state j. This storage format improves the efficiency of HTK, while allowing the storage of probabilities between 0.000001 and 1.0. For example, the emission distribution of state 1 in Figure 1 is uniform; hence, each value in the distribution is 7105. This corresponds to an actual probability of $exp(-7105/2371.8) = 0.05$.

The model specification ends with a description of the transition probabilities between states of the HMM. The transition matrix, labeled "<TransP>," specifies these probabilities directly. Each value in the matrix specifies the probability associated with a particular arc in the HMM. The row index of the value corresponds to the source of the arc, and the column index corresponds to the arc's target. Thus, for example, in Figure 1, state 2 has two arcs emanating from it, one leading back to state 2 (with probability 0.9) and one leading to state 3 (with probability 0.1). The sum of probabilities in each row of the transition matrix must be 1.0, with the exception of the last row, which contains all zeroes. Similarly, because state 1 is always the start state, the first column of the matrix must contain all zeroes.

4 Initializing Models

Although the structure of the initial model may be designed by hand, the initial emission and transition probability distributions should be generated in some principled fashion. HTK provides a straightforward means of estimating initial probability distributions. The tool `HInit` uniformly segments the training data and computes emission distributions using the occurrence count for each symbol. `HInit` then uses the Viterbi algorithm to re-segment the data and re-compute the parameters. Transition probabilities are computed by counting transition occurrences in the Viterbi alignments. This re-estimation procedure is repeated until convergence. Note that `HInit` takes a prototype HMM as input, but that `HInit` ignores the values of the given HMM's emission and transition probabilities. Thus, the user need only specify by hand the form of the HMM.

As an example, the following command line would initialize an HMM:

```
HInit -w 1.0 -S training.list -M initial an_HMM
```

Here, the prototype HMM is called `an_HMM`, and the list of files containing training sequences is `training.list`. The `-w 1.0` option sets a floor for emission log probabilities to prevent them from reaching zero. The re-estimated model will be called `an_HMM` and will be written to the the directory `initial`. For more information about model initialization see pages 158–159 and 227–229 of *The HTK Book*.

`HInit` provides the simplest means of initializing HTK models, but other methods are available. For example, acyclic HMMs generated by a software package such as SAM or HMMER could be translated into HTK format and used to initialize more complex, HTK models. Alternatively, non-HMM models of motifs or domains may be translated into HTK format and used in the initialization phase. Candidates for such initialization software include MEME [1, 15], the Gibbs sampler [14] and BLOCKMAKER [9, 4].

5 Training Models

Once the model has been designed and initialized, it is ready to be fully trained. The HTK tool `HRest` uses the Baum-Welch version of the expectation-maximization algorithm, rather than the Viterbi estimation algorithm employed by `HInit`. Otherwise, the usage of `HRest` is exactly similar to `HInit`. The following command line would continue the training of the previously initialized model, `initial/an_HMM`:

```
HRest -w 1.0 -S training.list -M trained \
      initial/an_HMM
```

The re-estimated HMM `an_HMM` will be stored in the directory `trained`.

Note that, because HTK is typically used for training small models, the `HRest` program contains an error check for model states which receive no training data. For biological modeling, it is often the case that a few states of the model receive no training data. Consequently, before using `HRest` to train a large, discrete model, it may be necessary to remove this error check. Appendix B explains how to make this change.

6 Scoring Sequences

The final step in using HTK is to apply the trained model to the task of recognizing related sequences. For a model of a protein family, for example, this would involve comparing the model to each member of a database such as GenBank [7] or SWISS-PROT [2]. The HTK tool `HVite` provides a Viterbi recognizer which will score each sequence. Because HTK typically recognizes speech utterances, the interface to HVite involves specifying some elements, such as a dictionary and a grammar, which are not relevant to biological sequence comparisons. However, most of these details are easy to automate via a shell script.

A sample `HVite` command line looks like this:

```
HVite -T 1 -i label_file -o TWM -w lattice_file \
      -S sequence_list dictionary HMM_list
```

The elements of the above command line have the following meanings:

- The first option, `-T 1`, tells `HVite` to set the trace level to 1. This causes the program to print diagnostic information, including the length of the given sequence and the Viterbi score.
- The `-i label_file` option tells `HVite` to write the Viterbi scores to a file called `label_file`. Note that these scores are recorded with a higher precision than the score printed in the trace output.
- The option `-o TWM` simplifies the label file output by telling `HVite` not to print times, words or model names.
- The `lattice_file` specifies the relationship between multiple HMMs being used at once. Since most biological applications involve the use of a single model, the lattice file is simple:

```
N=3 L=2
I=0 W=!NULL
I=1 W=an_HMM
I=2 W=!NULL
J=0 S=2 E=1
J=1 S=0 E=2
```

The above file may be used as a template for any single-model recognition, where `an_HMM` is the name of the model. For more details on how this lattice file was generated, see Appendix C.

- `sequence_list` is a single file containing the filenames of all of the sequences in the database. Note that these files must have been previously converted to HTK format.
- The `dictionary` normally maps from models to their pronunciations. For single-model recognition, `dictionary` is a file containing the name of the model, twice:

```
an_HMM an_HMM
```

- Finally, the `HMM_list` normally contains the list of all HMMs being used during the recognition run. For single-model recognition, this list contains one entry:

```
an_HMM
```

Once the Viterbi scores have been computed, it is necessary to normalize them to account for varying sequence lengths. Simply dividing the scores by length is ineffective. Instead, the normalization may be computed empirically using, for example, the empirical Z-score estimation procedure of Krogh *et al.* [12], which is available as part of the SAM distribution [16]. Alternatively, log-odds scores may be computed. This second method requires the specification of a background model and the computation, using `HVite`, of a background score for each sequence.

7 Conclusion

HTK provides computational biologists with a powerful new tool for the analysis of biological sequence data. By developing more flexible hidden Markov models and by applying the techniques and algorithms developed in the field of speech recognition, users of HTK will be able to explore and improve the use of hidden Markov modeling in computational biology.

References

[1] T. L. Bailey and C. P. Elkan. Fitting a mixture model by expectation-maximization to discover motifs in biopolymers. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, 1994.

[2] A. Bairoch. The SWISS-PROT protein sequence data bank: current status. *Nucleic Acids Research*, 22(17):3578–3580, September 1994.

[3] J. K. Baker. The Dragon system — an overview. *IEEE Trans. Acoust. Speech Signal Processing*, ASSP-23(1):24–29, February 1975.

[4] Blocks WWW server. `http://www.blocks.fhcrc.org`.

[5] G. A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989.

[6] S. R. Eddy. Multiple alignment using hidden Markov models. In C. Rawlings et al., editor, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 114–120. AAAI Press, 1995.

[7] GenBank overview. `http://www.ncbi.nlm.nih.gov/Web/Gen-bank/index.html`.

[8] D. G. Gilbert. readseq. `gopher://ftp.bio.indiana.edu:70/-11/Molecular-Biology/Molbio%20archive/readseq`, 1990.

[9] S. Henikoff, J. G. Henikoff, W. J. Alford, and S. Pietrokovski. Automated construction and graphical presentation of protein blocks from unaligned sequences. *Gene-COMBIS, Gene*, 163(GC):17–26, 1995.

[10] S. R. Eddy group, Dept. of Genetics, Washington University. `http://-genome.wustl.edu/eddy/hmm.html`.

[11] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996.

[12] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[13] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized hidden Markov model for the recognition of human genes in DNA. In *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology*, 1996.

[14] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

[15] MEME – multiple EM for motif elicitation. `http://www.sdsc.edu/-MEME`.

[16] SAM: sequence alignment and modeling system. `http://www.cse.-ucsc.edu/research/compbio/sam.html`.

[17] P. C. Woodland, C. J. Leggetter, J. J. Odell, V. Valtchev, and S. J. Young. The 1994 HTK large vocabulary speech recognition system. *International Conference on Acoustics Speech, and Signal Processing*, 1:73–76, 1995.

[18] P. C. Woodland, J. J. Odell, V. Valtchev, and S. J. Young. Large vocabulary continuous speech recognition using HTK. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 125–128. IEEE, 1994.

[19] S. J. Young, P. C. Woodland, and W. J. Byrne. Spontaneous speech recognition for the credit card corpus using the HTK toolkit. *IEEE Transactions on speech and audio processing*, 2(4):615–621, October 1994.

[20] Steve Young, Joop Jansen, Julian Odell, Dave Ollasen, and Phil Woodland. *The HTK Book*. Cambridge University, 1995.

A Storing sequence data in HTK format

HTK data files are binary files which begin with a 12-byte header. The header contains four parameters: the length of the current sequence, the sample period in 100ns units (irrelevant for biological data), the number of bytes per sample, and a code representing the type of sample (discrete samples have code 10). For more information, see *The HTK Book*, pp. 72-73.

The following fragment of C code shows how to write binary data to `outfile`. The code uses the function `WriteHTKHeader` from the HTK module HWave. The sequence data is assumed to be stored in an array called `sequence_data` of length `num_samples`.

```
/* Print the header. */
WriteHTKHeader(outfile, num_samples, 10, 2, 10);

/* Write the data to a file. */
for (i_sample = 0; i_sample < num_samples; i_sample++)
{
  fwrite(&(sequence_data[i_sample]), sizeof(short),
         1, outfile);
}
```

B Removing the error check for untrained states

In the source file `HRest.c`, replace the following line (In HTK version 2.0, this is line 977.):

```
HError(2222,"RestTransP: Zero state %d occupation count",i);
```

with this:

```
continue;
```

C Generating HTK lattice files

Lattice files are generated from grammar files using the HTK tool `HParse`. A single-model grammar looks like this:

```
(an_HMM)
```

where `an_HMM` is the name of the model. Assuming that the above grammar is stored in the file `grammar`, an appropriate lattice file would be generated as follows:

```
HParse an_HMM lattice_file
```

More complex grammars might be useful for feature-based computational biology applications, such as the modeling of multiple domains. See pages 168–172 of *The HTK Book* for more information on grammars and lattices.