TODE User Manual

Darren C. Moore
Dalle Molle Institute for Perceptual Artificial Intelligence (IDIAP)
CP 592, rue du Simplon 4,
1920, Martigny, Switzerland,
moore@idiap.ch, http://www.idiap.ch/~moore

January 31, 2003

Contents

Chapter 1

Introduction

TODE (TOrch DEcoder) is a continuous speech recogniser based on a time-synchronous beam-search algorithm that is compatible with the Torch machine learning library. It's purpose is to satisfy the general speech decoding needs of researchers at IDIAP and in the wider speech community. TODE has been designed to be a flexible recogniser with a straightforward implementation, that overcomes some of the limitations of other popular decoders while maintaining an acceptable level of efficiency.

The major features of TODE are :

- Efficient beam search decoder.
- Can be used with both ANN and GMM-based acoustic models.
- Accepts features or emission probabilities as input.
- N-gram language modelling with full back-off and caching.
- Supports many commonly used file formats (model definition, ANN weights, features, language model, etc).
- Uses a linear lexicon
- Implementation is straightforward, and can be readily modified/upgraded to meet the needs of researchers.
- Easily adapted for use in non-speech decoding applications.
- Fully supported with development ongoing.

This document describes how to use the stand-alone TODE executable for speech recognition tasks.

Chapter 2

Installation

TODE is distributed as part of the Torch machine learning library (`http://www.torch.ch`), which means that you must download and install Torch first, in order to compile and use TODE. The steps for installation are as follows:

1. Download and follow the Torch installation instructions.
   `http://www.torch.ch/matos/install.pdf`
2. The following Torch packages are required to build TODE:
   - decoder
   - core
   - datasets
   - distributions
   - gradients
   - speech
   - examples
3. You might want to use the `FLOATING = DOUBLE` option in your `Makefile_options_<os>` file. TODE will be slower, but the extra floating point precision may be required (depending on your application).
4. The "main" TODE source file (`tode.cc`) is located in your Torch directory under `examples/decoder`. Follow the steps in section 5 of the Torch installation instructions to compile this file. TODE is now ready for use.

Chapter 3

How to use

The TODE command line is of the form

```
tode <option> <option> ...
```

An option consists of one or two command line arguments: a keyword (eg. `-input_file`) followed by a value (eg. `<string>`). The value field is not required for boolean options. Some options are mandatory (eg. a dictionary file must be defined).

All TODE options are described in detail in the folowing sections.

A summary of all options can be obtained by typing

```
tode -help
```

3.1 General Options

3.2 -input_fname

Required	Yes
Format	`-input_fname <string>`
Summary	Describes where (feature or emission probability) input file/s are located.
Details	If the input format (see `-input_format` below) is an archive format (ie. `lna_archive` or `online_ftrs_archive`), then the string value denotes the actual archive file. Otherwise, the string value specifies the file that contains the filenames of the individual input files.
Default	undefined

3.2.1 -input_format

Required Yes

Format `-input_format <string>`

Summary Describes the format of the input files.

Details Valid file formats are :

- `htk` : HTK feature file readable by Torch IOHTK class with 1 utterance per file.
- `lna` : LNA 8-bit emission probabilities (see Appendix E) with 1 utterance per file.
- `lna_archive` : LNA 8-bit emission probabilities with all utterances in a single (big) archive file.
- `online_ftrs` : Online features format (see Appendix D) with 1 utterance per file.
- `online_ftrs_archive` : Online features format with all utterances in a single (big) archive file.

The format of input files must be compatible with the acoustic model settings.

Default undefined

3.2.2 -output_fname

Required	No
Format	`-output_fname <string>`
Summary	Specifies where decoder output will be written.
Details	
Default	stdout

3.2.3 -output_ctm

Required	No
Format	`-output_ctm`
Summary	Specifies that the output is to be written in CTM format (see Appendix F).
Details	
Default	false

3.2.4 -wrdtrns_fname

Required	No
Format	`-wrdtrns_fname <string>`
Summary	Specifies a file containing reference transcipts for all input utterances.
Details	If a reference transcription file is specified, then a verbose output is provided by the decoder, showing the input file as well as expected and actual results for each utterance. In addition, after all input files have been decoded, recognition statistics are computed and output (accuracy, insertions, substitutions, deletions). If this option is not specified, then only the recognition output words are output (1 utterance per line). If the input file format is non-archive (ie. `htk`, `lna` or `online_ftrs` then the reference transcription file can be in HTK MLF format (see Appendix J) or "raw" format (1 utterance per line). The ordering of utterance transcriptions in the HTK MLF file does not need to match the order of the input files. The ordering of utterances in the "raw" format transcription files must match the ordering of the input files. For archive input formats (ie. `lna_archive` or `online_ftrs_archive`), the transcription file must be in "raw" format.
Default	undefined

3.2.5 -msec_step_size

Required	No
Format	`-msec_step_size <real>`
Summary	Specifies the step size of input frames in millieseconds.
Details	Used only to compute durations when `-output_ctm` is specified.
Default	10.0ms

3.3 Acoustic Model Options

3.3.1 -am_models_fname

Required	Yes
Format	`-am_models_fname <string>`
Summary	Specifies the file containing the HMM definitions for the phone models.
Details	If HMM/GMM decoding is required then the models file must be in (simple) HTK model definition format (see Appendix I). If HMM/ANN decoding is required then the file must be in Noway model definition format (see Appendix G). All phones mentioned in the dictionary file must have a model defined in this file. There can be additional phone models defined (eg. a short pause model).
Default	undefined

3.3.2 -am_sil_phone

Required	No
Format	`-am_sil_phone <string>`
Summary	Specifies a "silence" phone.
Details	If defined, there must be a corresponding model defined in the phone models file. Specifying a silence phone has no effect unless a pause phone is also defined.
Default	undefined

3.3.3 -am_pause_phone

Required	No
Format	`-am_pause_phone <string>`
Summary	Specifies a "pause" phone.
Details	If defined, there must be a corresponding model defined in the phone models file. When word HMM's are created by contenating individual phone models, the pause model is added to the end of each word model. If the phone transcription for a word (as defined in the dictionary file) ends with a pause phone, then an additional pause is *not* added. If a silence phone is specified and the phone transcription for a word ends with a silence phone, then the pause phone is *not* added. A pause model with an initial-final state transition is valid.
Default	undefined

3.3.4 -am_phone_del_pen

Required	No
Format	`-am_phone_del_pen <real>`
Summary	Specifies the non-log phone-level deletion penalty.
Details	This value is used to scale the (non-log) transition probabilities for transitions originating from the initial state of each phone model. When phone models are concatenated to form word-level HMM's, this scaling serves as a phone deletion penalty.
Default	1.0

3.3.5 -am_apply_pause_del_pen

Required	No
Format	`-am_apply_pause_del_pen`
Summary	Indicates that the phone deletion penalty is to be applied to the model for the "pause" phone.
Details	This option is used only if a pause phone is defined.
Default	false

3.3.6 -am_priors_fname

Required	No
Format	`-am_priors_fname <string>`
Summary	Specifies the file containing the phone prior probabilities.
Details	The phone priors are required for HMM/ANN decoding, but are not used for HMM/GMM decoding. The format of the file must be in ICSI `priors` format (see Appendix B). The ordering of the prior probabilities must match the order in which phone models are defined in the models file. Any emission probability used for decoding, whether it originates from an LNA file or is computed on-the-fly by an MLP, is divided by its corresponding prior probability before being used in decoding calculations.
Default	undefined

3.3.7 -am_mlp_fname

Required	No
Format	`-am_mlp_fname <string>`
Summary	Specifies the file containing MLP weights.
Details	The file must be in MLPW binary format (see Appendix A). The file is required for HMM/ANN decoding, when using features as input (ie. input format is `htk`, `online_ftrs` or `online_ftrs_archive`, and the models file is in Noway format).
Default	undefined

3.3.8 -am_mlp_cw_size

Required	
Format	`-am_mlp_cw_size <integer>`
Summary	Specifies the context window size to use with an MLP.
Details	Required when performing HMM/ANN decoding with features as input. The feature vector size multiplied by this number must equal the number of input units in the MLP. Note that timing output information (eg. when using `-output_ctm` option) will be affected. The timings will correspond to the input feature file with the first and last $\frac{N}{2} - 1$ vectors stripped (where N is the context window size).
Default	undefined

3.3.9 -am_norms_fname

Required	No
Format	-am_norms_fname <string>
Summary	Specifies the file containing means and inverse standard deviations used to normalise features.
Details	The norms file is only used during HMM/ANN decoding with features as input. If specified, each input feature vector is normalised before it is input to the MLP. This file must be in ICSI norms format (see Appendix C). The number of means (and inverse stddevs) in the file must be equal to the number of input feature vector elements. If a norms file is not specified, features are read from file and input to the MLP without modification.
Default	undefined

3.3.10 -am_online_norm_ftrs

Required	No
Format	-am_online_norm_ftrs
Summary	Activates online normalisation of input features.
Details	This feature is only used during HMM/ANN decoding with features as input and when a norms file is defined. If specified, a simple, first-order online mean and variance normalisation is applied to each feature dimension. The feature means and variances are updated at each time step (see -am_online_norm_alpha_m and -am_online_norm_alpha_v below).
Default	false

3.3.11 -am_online_norm_alpha_m

Required	No
Format	-am_online_norm_alpha_m <real>
Summary	The update constant for feature means during online normalisation.
Details	This option is only used during HMM/ANN decoding with online normalisation of features. At each time step, and for each feature dimension, the existing mean value is scaled by $(1 - \alpha_m)$, and α_m times the current feature value is added to obtain the new mean.
Default	0.005

3.3.12 `-am_online_norm_alpha_v`

Required	No
Format	`-am_online_norm_alpha_v <real>`
Summary	The update constant for feature variances during online normalisation.
Details	This option is only used during HMM/ANN decoding with online normalisation of features. At each time step, and for each feature dimension, the existing variance value is scaled by $(1 - \alpha_v)$, and α_v times the square of the current feature value is added to obtain the new variance.
Default	0.005

3.4 Lexicon Options

3.4.1 `-lex_dict_fname`

Required	Yes
Format	`-lex_dict_fname <string>`
Summary	Specifies the file containing the dictionary used for recognition.
Details	The dictionary file contains entries for all pronunciations that can be recognised. The format of each entry is : `word(prior) ph1 ph2 ... phn` The `(prior)` field denotes the prior probability of a pronunciation, and is optional (defaults to 1.0 if omitted). Multiple pronunciations of the same word are permitted. All phones in each entry must be present in the phone models file (see `-am_models_fname`).
Default	undefined

3.4.2 `-lex_sent_start_word`

Required	No
Format	`-lex_sent_start_word <string>`
Summary	Specifies the word that starts every result sentence.
Details	If specified, TODE constrains all output word sequences to begin with this word. The sentence start word can be the same as the silence word and the sentence end word (most commonly defined as silence). The presence of the sentence start word in the language model is optional. TODE removes the sentence start word before writing the decoding result to the output file.
Default	undefined

3.4.3 -lex_sent_end_word

Required	No
Format	`-lex_sent_end_word <string>`
Summary	Specifies the word that ends every result sentence.
Details	If specified, TODE constrains all output word sequences to end with this word. The sentence end word can be the same as the silence word and the sentence start word (most commonly defined as silence). The presence of the sentence end word in the language model is optional. TODE removes the sentence end word before writing the decoding result to the output file.
Default	undefined

3.4.4 -lex_sil_word

Required	No
Format	`-lex_sil_word <string>`
Summary	Specifies the silence word.
Details	Specifies a silence word. This word is treated like any other word during decoding, but all instances in the final output word sequence are removed before the decoding result is written to file. The silence word can be the same as the sentence start word and the sentence end word. The silence word is ignored during language model calculations.
Default	undefined

3.5 Language Model Options

3.5.1 -lm_fname

Required	No
Format	`-lm_fname <string>`
Summary	Specifies the file containing the N-gram language model
Details	The file must be in ARPA format (see Appendix H)
Default	undefined

3.5.2 `-lm_ngram_order`

Required	No
Format	`-lm_ngram_order <integer>`
Summary	Specifies order of N-gram to use for the language model.
Details	The value specified must be $\leq$ the order of the language model file. A value of 0 results in no language model being used during decoding. Note that for N-grams with $N > 2$, the language model is incorporated in an approximate way. In the tri-gram LM case (N=3), when evaluating a transition from w_i to w_j, the predecessor word of w_i, say $w_i^{'}$ (as determined by the Viterbi search), is used to retrieve the LM prob that gets associated with the transition between w_i and w_j.
Default	0

3.5.3 `-lm_scaling_factor`

Required	No
Format	`-lm_scaling_factor <real>`
Summary	Scales language model probabilities during decoding.
Details	Whenever a language model probability is retrieved (in log domain), it is multiplied by this factor before being incorporated in the decoding.
Default	1.0

3.6 Beam Search Decoding Options

3.6.1 `-dec_int_prune_window`

Required	No
Format	`-dec_int_prune_window <real>`
Summary	Specifies the (log) window used for pruning hypotheses in word-interior states.
Details	Needs to be a positive log value. At each time step during decoding, a threshold is calculated by subtracting this constant from the score of the best word-interior hypothesis. Any interior-state hypotheses that have scores below this threshold are deactivated and removed from further consideration. A 0 or negitive value results in no pruning of interior-state hypotheses.
Default	0.0

3.6.2 -dec_end_prune_window

Required	No
Format	`-dec_end_prune_window <real>`
Summary	Specifies the (log) window used for pruning hypotheses in word-end states.
Details	Needs to be a positive log value. At each time step during decoding, a threshold is calculated by subtracting this constant from the score of the best word-end hypothesis. Any word-end state hypotheses that have scores below this threshold are deactivated and removed from further consideration. The pruning occurs before language model probabilities are applied. A 0 or negitive value results in no pruning of end-state hypotheses.
Default	0.0

3.6.3 -dec_word_entr_pen

Required	No
Format	`-dec_word_entr_pen <real>`
Summary	Specifies the (log) word insertion penalty used during decoding.
Details	The word insertion penalty value (most commonly a negative log value) gets added to word-end hypothesis scores during evaluation of word transitions.
Default	0.0

3.6.4 -dec_delayed_lm

Required	No
Format	`-dec_delayed_lm`
Summary	Specifies that the application of language model probabilities is to be delayed.
Details	Usually a language model probability $P(w_2 \vert w_1)$ (assuming a bigram LM) is applied when a hypothesis makes a transition from the final state of w_1 to the initial state of w_2. If this option is used, the application of language model probabilities is delayed and $P(w_2 \vert w_x)$ is applied to hypotheses that reach the final state of w_2 (w_x is the predecessor word for the hypothesis). This approximation can result in significant computational savings (less LM lookups).
Default	false

3.6.5 `-dec_verbose`

Required	No
Format	`-dec_verbose`
Summary	Specifies that frame-by-frame decoding information is to be output.
Details	
Default	false

Appendix A

MLPW File Format

Reproduction of ICSI man page.

NAME

mlpw - Family of binary-encoded neural-net weights file formats used by QuickNet

DESCRIPTION

The *mlpw* file format is used to store neural net weights in a more compact and more quickly-accessed format than the traditional ASCII RAP3 weights(5) format. The same information is stored in the same order, but the values are coded, typically as 32 bit floats or 16 bit fixed-point ints, less often as 8 or 32 bit ints, or 64 bit doubles. Each section (e.g. weights or biases of a particular layer) may be coded in a different format.

mlpw files are usually created with qnstrn(1) (or will be when it is modified to support them) and converted to and from other formats with qncopywts(1). They will be read directly by future versions of qnsfwd(1) and ffwd(1).

The header

The header as currently defined consists of 5 4-byte integers in big-endian order:

magic	magic number = 0x4D4C5057 ("MLPW")
version	version code = 20010313 (today)
nettype	nettype/version (e.g. softmax)
nlayers	count of unit layers (3 for MLP3)
nsections	count of sections (4 for MLP3)

Then follow *nlayers* 4-byte ints specifying the number of units in each layer (starting at the input), followed by the sections.

Each section also has a small header, consisting of 3 4-byte integers:

sectiontype	QN_SectionSelector tag for this section
numvalues	how many weights in this section
datatype	data type flag (bytes/wt + 32 for float)

For fixed-point data formats (only), this is followed by a 4-byte int giving the fixed-point 'exponent' for this section. After this come the actual coded weight values.

In an MLP3, there are 4 sections: (0) input-to-hidden weights, (1) hidden-to-output weights, (2) hidden layer bias weights, and (3) output layer bias weights. Since bias values occupy a slightly different range (they are typically distributed around -log(n_units)), they are often stored with a larger exponent and/or more bits per weight. The MLPW file format supports this without difficulty.

NOTES/BUGS

Short-format (MLPWS) files are typically 1/4 the size of ASCII RAP3 files, or 1/2 the size of gzipped ASCII files, and load 5–10x faster. Since the weights are calculated on the SPERT boards using 16 bit fixed-point arithmetic, there is usually no accuracy loss in storing them this way.

You shouldn't ever have to access these files directly. Instead, use the QuickNet class QN_MLPWeightFile_MLPW(3).

Little tested at present.

AUTHOR

Dan Ellis <dpwe@ee.columbia.edu>

SEE ALSO

qncopywts(1).

Appendix B

Priors File Format

Reproduction of ICSI man page.

NAME

priors − file format for list of prior probabilities

DESCRIPTION

priors is a y0 −compatible file format for prior probabilities. These are a by-product of training and are used to compensate for inequities in the amount of training data for each target.

The file has the following format:

```
<0's prior>
<1's prior>
<2's prior>
 ...
<n's prior>
```

Where

<n's prior>
: is the prior probability of neural network output number n.

EXAMPLE

Here is a simple example of a prior file

```
0.85
0.01
0.04
0.02
0.05
0.01
0.01
0.01
```

In this example, the file contains prior probabilities for eight neural network outputs.

FILES

An example file can be found in ~drspeech/data/TIMIT/timit61.PHONE.uniform.p

AUTHOR

Dr. Speech <drspeech@icsi.berkeley.edu>
This manpage was written by Su-Lin Wu <sulin@icsi.berkeley.edu>

SEE ALSO

isr_train(1),

NOTES

Note that for y0 compatibility it is necessary for the priors file to contain only numbers. Any extraneous words or lines will cause errors. Also, y0 does not currently check for the number of priors matching the number of neural network outputs.

Appendix C

Norms File Format

Reproduction of ICSI man page.

NAME

norms - RAP style speech feature normalization file

DESCRIPTION

The norms file format is used to store speech feature file normalization data. A norms file is typically associated with a specific pfile. norms files are used by mlp training and feed forward programs such as bob(1), CLONES, qntrain(1) and qnforward(1)

The norms file consists of two vectors of information - a vector of means for each feature in the feature file and a vector of the reciprocal of the standard deviation of each feature in the feature file. The format of the vectors is tagged ASCII as produced by the RAP matrix/vector library.

FORMAT

The norms file format is:

```
vec <# of features>
<mean of each feature, one per line>
vec <# of features>
<1/(standard deviation) of each feature, one per line>
```

EXAMPLE

```
vec 18
-4.638622e-01
-3.881508e-01
-3.207185e-01
-2.973742e-01
-2.367414e-01
-1.349086e-01
-1.126812e-01
-3.952942e-02
1.188954e-02
-1.105962e+00
5.939942e-03
2.394057e-01
2.015354e-01
2.305043e-01
5.061634e-02
5.421233e-02
5.521029e-03
-3.096025e-02
vec 18
1.127764e+00
3.574011e+00
3.911481e+00
4.302862e+00
4.556445e+00
```

```
6.444429e+00
9.395655e-01
4.333607e+00
3.928129e+00
1.324948e-01
6.590791e-01
5.079605e-01
6.311077e-01
5.412703e-01
8.254844e-01
7.489987e-01
9.016648e-01
1.070975e+00
```

AUTHOR

David Johnson <davidj@ICSI.Berkeley.EDU>

SEE ALSO

bob(1), qnnorm(1), qntrain(1), qnforward(1), pfile(5)

Appendix D

Online Features File Format

Reproduction of ICSI man page.

NAME

online_ftrs - format for feature streams for online use

DESCRIPTION

The _online ftrs_ file format is used when passing speech feature files around during online recognition. In this context, "online" means real-time - i.e. there is someone waiting for the results of the processing and data must be operated on before a complete sentence is available. This situation has different requirements from e.g. the storage of features for MLP training, and consequently the data format is different.

FORMAT

The format consists of a continuous stream of frames from one or more sentences. Each frame starts with a single flag byte, followed by a fixed number of big-endian IEEE single precision floating point values. For most frames, the flag byte is zero. For the last frame in each sentence, the flag byte is 0x80

Note that _online ftrs_ streams contain no speech label information, unlike the pfile(5) file format.

EXAMPLE

An example of a trivial _online ftrs_ file with three features in each frame and two sentences might be:

```
0x00 1.20   5.40  -5.43
0x00 0.03   5.41   0.76
0x80 0.04   2.31   0.03
0x00 0.34   0.02   1.23
0x00 3.34   4.56   3.23
0x00 4.34   3.43   2.56
0x80 1.02   1.03   0.01
```

AUTHOR

David Johnson <davidj@ICSI.Berkeley.EDU>

SEE ALSO

pfile(5), qnforward(1), berpdemo(1)

Appendix E

LNA File Format

Reproduction of ICSI man page.

NAME

lna - compressed format for MLP output probablility files

SYNOPSIS

*.lna

DESCRIPTION

lna is a compression format for speech developed by Tony Robinson, used by y0(1) and noway(1). There are really two _lna_ formats (8 bit and 16 bit) supported by the software, but everybody just uses 8 bit.

Basically, each floating point probability is quantized to an 8 or 16 bit integer by the following formula:

intval = floor(-LNPROB_FLOAT2INT * log(_x_ + VERY_SMALL))

where LNPROB_FLOAT2INT is 24 for 8 bit, and 5120 for 16 bit. The int is then pinned to between 0 and 255 (or 65535). VERY_SMALL prevents ugliness if the probability is 0.0.

As for the actual file format, it is a binary stream of frames, where each frame consistes of a fixed number of 8 or 16 bit values.

EOS _Val0_ _Val1_ _Val2_ ... _Valn_

EOS is 0x80 if the frame is the last frame in a sentence, 0 otherwise. _Val0_ ... _Valn_ are the quantized integers corresponding to the probabilities.

SEE ALSO

lna2y0new(1), rap2lna(1)

AUTHOR

This man page was written by:

Jonathan Segal <jsegal@ICSI.Berkeley.EDU>
Eric Fosler <fosler@ICSI.Berkeley.EDU>.

updated by: Alfred Hauenstein <alfredh@icsi.berkeley.edu>

Appendix F

CTM File Format

NAME

ctm - Definition of time marked conversation scoring input

DESCRIPTION

This describes the time marked conversation input files to be used for scoring the output of speech recognizers via the NIST sclite() program. Both the reference and hypothesis input files can share this format.

The ctm file format is a concatenation of time mark records for each word in each channel of a waveform. The records are separated with a newline. Each word token must have a waveform id, channel identifier [A | B], start time, duration, and word text. Optionally a confidence score can be appended for each word. Each record follows this BNF format:

```
 CTM :== <F> <C> <BT> <DUR> word [ <CONF> ]

     Where :
      <F>  ->
           The waveform filename.  NOTE: no pathnames or
           extensions are expected.
      <C>  ->
           The waveform channel.  Either "A" or "B".
      <BT> ->
           The begin time (seconds) of the word, measured
           from the start time of the file.
      <DUR>  ->
           The duration (seconds) of the word.
      <CONF>  ->
           Optional confidence score.  It is proposed that
           this score will be used in the future.
```

The file must be sorted by the first three columns: the first and the second in ASCII order, and the third by a numeric order. The UNIX sort command: "sort +0 -1 +1 -2 +2nb -3" will sort the words into appropriate order.

Lines beginning with ';;' are considered comments and are ignored. Blank lines are also ignored.

Included below is an example:

```
;;
;;  Comments follow ';;'
;;
;;  The Blank lines are ignored

;;
7654 A 11.34 0.2  YES -6.763
7654 A 12.00 0.34 YOU -12.384530
```

```
7654 A 13.30 0.5  CAN 2.806418
7654 A 17.50 0.2  AS 0.537922
      :
7654 B 1.34 0.2  I -6.763
7654 B 2.00 0.34 CAN -12.384530
7654 B 3.40 0.5  ADD 2.806418
7654 B 7.00 0.2  AS 0.537922
      :
```

For CTM reference files, a format extension exists to permit marking alternate transcripts. The alternation uses the same file format as described above, except three word strings, "<ALT_BEGIN>", "<ALT>" and "<ALT_END>", are used to delimit the alternation. Each tag is treated as a word, with a conversation id, channel and "*"'s for the begin and duration time.

The alternation is begun using the word "<ALT_BEGIN>", and terminated using the word "<ALT_END>". In between the start and end, are at least 2 alternative time-marked word sequences separated by the word "<ALT>". Each word sequence can contain any number of words. An empty alternative signifies a null word.

Below is and example alternate reference transcript for the words "uh" and "um".

```
;;
7654 A   *    *   <ALT_BEGIN>
7654 A 12.00 0.34 UM
7654 A   *    *   <ALT>
7654 A 12.00 0.34 UH
7654 A   *    *   <ALT_END>
```

SEE ALSO
sclite(1)

BUGS/COMMENTS
Please contact Jon Fiscus at NIST with any bug reports or comments at the email address jfiscus@nist.gov or by phone, (301)-975-3182. Please include the version number of sclite, and any other relevant information.

Appendix G

Noway Phone Models File Format

Extracted from the Noway LVCSR decoder manual page. Note that the 'interword_pause' phoneme discussed on the following page is *not* mandatory in TODE.

-phone_models file

This file defines the phone models. It specifies the number of states (including entry and exit null states), the model topology, the transition probabilities and the output probability distributions associated with each state (obtained using the acoustic input options). The format of the file is as follows. The first line consists of the string 'PHONE', and the second line contains an integer giving the number of phone models. The remainder of the file contains the descriptions of each phone model. Within a phone HMM 0 indexes the ENTRY null state, 1 indexes the EXIT null state and 2 onwards index the real emitting states. The format for a phone model is:

```
<id> <number of states> <label>
-1 -2  <probid-1> <probid-2> ...
<from_state> <#out-trans> <to_state> <prob>  ...
<from_state> <#out-trans> <to_state> <prob>  ...
 ...
```

Where -1 and -2 represent dummy phone numbers for the the entry and exit states, and <probid-n> represents the element of the acoustic probability vector corresponding to that state (1 for each state). The number of integers on this line equals the number of states. The remaining lines specify the transition probabilities giving the transitions out of each state; prob is a floating point number (not logprob). An example entry for the phone 'aa' is:

```
2 4 aa
-1 -2  1 1
0 1 2 1.00000
1 0
2 1 3 0.50000
3 2 3 0.50000 1 0.50000
```

Here 'aa' has 2 non-null states, making 4 states total and is a left-to-right 'Viterbi' model, with output probabilities corresponding to acoustic probability element 1. Note that an 'interword-pause' phone model is essential to the operation of noway. This between-word pause model will typically contain 1 non-null state that may be skipped, and will use the 'silence' distribution. The interword-pause model is placed at the root of the lexicon and corresponds to an optional pre-word pause; for edge effects it is also the acoustic realization of sentence_end. Note that the name 'interword-pause' is currently hardwired in, and such a model must appear in the phone models file.

Appendix H

ARPA Language Model File Format

Reproduction of man page downloaded from SRI website.
$\log_{10}$ N-gram probabilities in ARPA files that are < -90.0 are interpreted by TODE as $-\infty$.
$\log_{10}$ back-off weights in ARPA files that are < -90.0 are interpreted by TODE as 0.0.

ngram-format

NAME

ngram-format - File format for ARPA backoff N-gram models

SYNOPSIS

\data
ngram 1=*n1*
ngram 2=*n2*
...
ngram *N*=*nN*
\1-grams:
p w [*bow*]
...
\2-grams:
p w1 w2 [*bow*]
...
N-**grams:**
p w1 ... *wN*
...
\end

DESCRIPTION

The so-called ARPA (or Doug Paul) format for N-gram backoff models starts with a header, introduced by the keyword **\data**, listing the number of N-grams of each length. Following that, N-grams are listed one per line, grouped into sections by length, each section starting with the keyword *N*-**gram:**, where *N* is the length of the N-grams to follow. Each N-gram line starts with the logarithm (base 10) of conditional probability *p* of that N-gram, followed by the words *w1*...*wN* making up the N-gram. These are optionally followed by the logarithm (base 10) of the backoff weight for the N-gram. The keyword **\end** concludes the model representation.

Backoff weights are required only for those N-grams that form a prefix of longer N-grams in the model. The highest-order N-grams in particular will not need backoff weights (they would be useless).

Since log(0) (minus infinity) has no portable representation, such values are mapped to a large negative number. However, the designated dummy value (-99 in SRILM) is interpreted as log(0) when read back from file into memory.

The correctness of the N-gram counts *n1*, *n2*, ... in the header is not enforced by SRILM software when reading models (although a warning is printed when an inconsistency is encountered). This allows easy textual insertion or deletion of parameters in a model file. The proper format can be recovered by passsing the model through the command
ngram -order *N* -lm *input* -write-lm *output*

Note that the format is self-delimiting, allowing multiple models to be stored in one file, or to be surrounded by ancillary information. Some extensions of N-gram models in SRILM store additional parameters after a basic N-gram section in the standard format.

SEE ALSO

ngram(1), ngram-count(1), lm-scripts(1), pfsg-scripts(1).

BUGS

The ARPA format does not allow N-grams that have only a backoff weight associated with them, but no conditional probability. This makes the format less general than would otherwise be useful (e.g., to support pruned models, or ones containing a mix of words and classes). The ngram-count(1) tool satisfies this constraint by inserting dummy probabilities where necessary.

For simplicity, an N-gram model containing N-grams up to length *N* is referred to in the SRILM programs as an *N*-th order model, although techncally it represents a Markov model of order *N*-1.

AUTHOR

The ARPA backoff format was developed by Doug Paul at MIT Lincoln Labs for research sponsored by the U.S. Department of Defense Advanced Research Project Agency (ARPA).
Man page by Andreas Stolcke <stolcke@speech.sri.com>.

Appendix I

HTK HMM Model Definition File Format

Extracted from The HTK Book (for HTK version 3.2). TODE supports only the format shown in Figure 7.3 on the following page. The `<GCONST>` and `<STREAMINFO>` keywords are also permitted in the file but are ignored by TODE. Any other variation from the format of Figure 7.3 will cause TODE to return an error.

```
~h "hmm2"
<BeginHMM>
    <VecSize> 4 <MFCC>
    <NumStates> 4
    <State> 2 <NumMixes> 2
      <Mixture> 1 0.4
        <Mean> 4
          0.3 0.2 0.2 1.0
        <Variance> 4
          1.0 1.0 1.0 1.0
      <Mixture> 2 0.6
        <Mean> 4
          0.1 0.0 0.0 0.8
        <Variance> 4
          1.0 1.0 1.0 1.0
    <State> 3 <NumMixes> 2
      <Mixture> 1 0.7
        <Mean> 4
          0.1 0.2 0.6 1.4
        <Variance> 4
          1.0 1.0 1.0 1.0
      <Mixture> 2 0.3
        <Mean> 4
          2.1 0.0 1.0 1.8
        <Variance> 4
          1.0 1.0 1.0 1.0
    <TransP> 4
      0.0 1.0 0.0 0.0
      0.0 0.5 0.5 0.0
      0.0 0.0 0.6 0.4
      0.0 0.0 0.0 0.0
<EndHMM>
```

Fig. 7.3 Simple Mixture Gaussian HMM

Notice that only the second state has a full covariance Gaussian component. The first state has a mixture of two diagonal variance Gaussian components. Again, this illustrates the flexibility of HMM definition in HTK. If required the structure of every Gaussian can be individually configured.

Another possible way to store covariance information is in the form of the Choleski decomposition L of the inverse covariance matrix i.e. $\mathbf{\Sigma}^{-1} = LL'$. Again this is stored externally in upper triangular form so L' is actually stored. It is distinguished from the normal inverse covariance matrix by using the keyword <LLTCovar> in place of <InvCovar>[3].

The definition for hmm3 also illustrates another macro type, that is, ~o. This macro is used as an alternative way of specifying global options and, in fact, it is the format used by HTK tools when they write out a HMM definition. It is provided so that global options can be specifed ahead of any other HMM parameters. As will be seen later, this is useful when using many types of macro.

As noted earlier, the observation vectors used to represent the speech signal can be divided into two or more statistically independent data streams. This corresponds to the splitting-up of the input speech vectors as described in section 5.13. In HMM definitions, the use of multiple data streams must be indicated by specifying the number of streams and the width (i.e dimension) of each stream as a global option. This is done using the keyword <StreamInfo> followed by the number of streams, and then a sequence of numbers indicating the width of each stream. The sum of these stream widths must equal the original vector size as indicated by the <VecSize> keyword.

[3] The Choleski storage format is not used by default in HTK Version 2

Appendix J

HTK MLF File Format

Extracted from The HTK Book (for HTK version 3.2). TODE supports a restricted MLF format, similar to example 2 on the following page. The first line of the file must be #!MLF!#. This is followed by a number of transcription entries.

A transcription entry consists of a filename line, followed by the words in the transcription (on separate lines), and is ended with a line containing the '.' character.

The filename must be enclosed in double quotes. The filename can be relative or absolute. The filename should have an extension (eg. `.lab`). TODE prunes all path information and the file extension from each filename and attempts to match the result to an input filename. Therefore, wildcards are not permitted after the final '/' in the file name. After pruning of path and extension information, the resulting string should uniquely identify an input file.

6.3.4 MLF Examples

1. Suppose a data set consisted of two training data files with corresponding label files:
   `a.lab` contains

   ```
    000000  590000  sil
    600000 2090000  a
   2100000 4500000  sil
   ```

   `b.lab` contains

   ```
    000000  990000  sil
   1000000 3090000  b
   3100000 4200000  sil
   ```

   Then the above two individual label files could be replaced by a single MLF

   ```
   #!MLF!#
   "*/a.lab"
    000000  590000  sil
    600000 2090000  a
   2100000 4500000  sil
   .
   "*/b.lab"
    000000  990000  sil
   1000000 3090000  b
   3100000 4200000  sil
   .
   ```

2. A digit data base contains training tokens `one.1.wav, one.2.wav, one.3.wav, ..., two.1.wav, two.2.wav, two.3.wav, ...`, etc. Label files are required containing just the name of the model so that HTK tools such as HEREST can be used. If MLFs are not used, individual label files are needed. For example, the individual label files `one.1.lab, one.2.lab, one.3.lab, ....` would be needed to identifiy instances of "one" even though each file contains the same entry, just

   ```
   one
   ```

   Using an MLF containing

   ```
   #!MLF!#
   "*/one.*.lab"
   one
   .
   "*/two.*.lab"
   two
   .
   "*/three.*.lab"
   three
   .
   <etc.>
   ```

   avoids the need for many duplicate label files.

3. A training database `/db` contains directories `dr1, dr2, ..., dr8`. Each directory contains a subdirectory called `labs` holding the label files for the data files in that directory. The following MLF would allow them to be found

   ```
   #!MLF!#
   "*" -> "/db/dr1/labs"
   "*" -> "/db/dr2/labs"
   ...
   "*" -> "/db/dr7/labs"
   "*" -> "/db/dr8/labs"
   ```