UTEP Technical Report UTEP-CS-08-29

Testing the Value of a Time-based Language Model for Speech Recognition

Nisha Kiran[1], **Nigel G. Ward**

Department of Computer Science
University of Texas at El Paso
500 West University Avenue
El Paso, TX 79968-0518

email: nisha.iitkgp@gmail.com, nigel@cs.utep.edu

July 14, 2008

Speech recognition relies on the language model in order to decode an utterance, and in general a better language model improves the performance of a speech recognizer. We have recently found that a time-based language model can improve on a standard trigram language model in terms of perplexity [5]. This technical report presents the evaluation of this new language model in the context of speech recognition. First, a basic speech recognizer was built using the HTK [6] tool. Then the recognizer was run using the standard language model and using the time-based one. On a testset of 39,147 words from the Switchboard corpus, there was a slight improvement, with the percentage of words correctly recognized going from 11.31% to 11.40%.

Index Terms: speech recognition, Switchboard corpus, time-based language model, HTK

1 Introduction

Speaking is a cognitive process in time and also a communicative process in time [1], but these processes are not directly modeled by today's language models, which generally treat speech as consisting merely of sequences of words. This reports aims to test an initial implementation of a model which considers the time-into-utterance when estimating the probabilities of words [5]. The work consists of building an automatic speech recognition system and then testing the performance of the time-based language model by using it to rescore the lattices produced during the initial decoding stage. The aim is to indicate whether this model can actually be useful for speech recognition, and to produce a testbed that can be used in the future to test refined time-based models.

[1] visiting from the Indian Institute of Technology at Kharagpur

Section 2 describes how the Hidden Markov Models were trained on the Switchboard corpus. Section 3 describes the modifications done to the HTK code to test the time-based language model and how to use the changed code to test the time based model. Section 4 presents results.

2 Acoustic Model Training

Acoustic models were trained on the data available from the Switchboard Corpus. This did not require any new code; the HTK Tools provide all the needed functions. However preparation of the data involved several steps, as did the training itself.

2.1 Data Preparation

2.1.1 Training Corpus

The Switchboard corpus, a collection of short telephone conversations on light topics between mostly unacquainted adults [2, 3] was licensed from the LDC and used for training as well as testing. The audio files as well as their transcription files were first broken down into single utterances. The transcriptions were then converted to the HTK label format, also known as the MLF format. Two sets of word-level MLF files were generated. One set for training and one for testing.

A total of 518 Switchboard A tracks were used for training the HMM models as listed in the file *trainingFiles*. Each track was broken down into single-utterance label files. These were found by looking for the regions labeled *silence* in the B track, that is, the track for the other speaker. While there are other possible ways to determine utterance regions, this was done to ensure that the training used only audio regions containing the voice of only one speaker. Utterances which contained regions labeled *[laughter], [vocalized-noise], [noise], -[o]kay* etc. were excluded from the training data. This produced a total of 11960 utterances. The transcriptions were converted into the HTK label format. The corresponding audio files for each utterance were then clipped out. Then the labels and the audio were used together for training the HMM acoustic models.

2.1.2 Test Corpus

A total of 96 Switchboard tracks were used for testing the HMM models, as listed in the file *test_files*. As above, each audio track and its corresponding transrciption was split into single-utterance audio and single-utterance label files respectively. This produced a test set of 2557 utterances.

2.1.3 Creating the transcription files

After the creation of the training as well as the test corpus, the next task was to acquire a dictionary with the phonetic transcriptions of words. For this the online dictionary *cmudict.0.6d* was used. For the creation of the final dictionary, first a wordlist named *wlist* was generated which

contained the list of all the words which were in the training as well as the test data. Then the HTK Tool *HDMan* was used to construct the final dictionary:

HDMan -m -w wlist -n phone1.list -l dlog dict.list cmudict.0.6d

This command generated the dictionary *dict.list* by searching the source dictionary *cmudict.0.6d* to find the pronunciations for each word in *wlist*. The option '-l' instructs *HDMan* to output a log file *dlog* which contained various statistics about the constructed dictionary. The phone list *phone1.list* contained the 40 phones for which the training was done.

To train a set of HMMs, every file of the training data must have an associated phone level transcription. The phone level MLF was generated from the word-level transcriptions and *dict.list* using the label editor *HLEd*:

HLEd -d dict.list -i phones0.mlf mkPhones0.led word0.mlf

Where the HLEd edit script *mkPhones0.led* contained the following commands:

EX

IS sil sil

DE sp

where the expand command *EX* replaces each word in *word0.mlf* by the corresponding pronunciation in the dictionary, the *IS* command inserts a silence model *sil* at the start and end of every utterance, and , the delete *DE* command deletes all short pause *sp* labels, which are not wanted in the transcription labels for the initial training stage.

2.1.4 Coding the data

The final stage of data preparation was to parameterize the raw speech waveforms into sequences of feature vectors, namely Mel Frequency Cepstral Coefficients (MFCCs).

Coding was performed using the tool *HCopy*, configured to automatically convert its input into MFCC vectors. To do this the following configuration parameters were set:

1. $SOURCEKIND = WAVEFORM$
2. $SOURCEFORMAT = WAVE$
3. $SOURCERATE = 625.0$
4. $TARGETKIND = MFCC\_0$
5. $TARGETRATE = 100000.0$
6. $WINDOWSIZE = 250000.0$
7. $USEHAMMING = T$
8. $NUMCHANS = 20$
9. $CEPLIFTER = 22$

10. $NUMCEPS = 12$

The following command was executed to create the MFCCs for the audio files:

HCopy -C copy.cfg -S copy.scp

where *copy.cfg* contains the above configuration parameters and *copy.scp* contains the names of the single-utterance audio files whose MFCC is to be created.

The files generated were stored in the *mfcc/* directory.

2.2 Training

This section explains the actual training of the acoustic models HMMs.

2.2.1 Model Initialization

A simple left-to-right topology with three emitting states was used as the HMM topology. A general prototype *proto* was constructed, just as a definition of the model topology. This prototype has zero means and the diagonal elements of the covariance matrix are all ones.

To initialize the HMM models the tool *HCompV* was used. *HCompV* scans the training data and computes the global means and variances for the whole training corpus and outputs that; thus at this stage all states have the same output probabilies. *HCompV* requires a configuration file. The following configuration parameters were used:

1. $TARGETKIND = MFCC\_0\_D\_A$
2. $TARGETRATE = 100000.0$
3. $SAVECOMPRESSED = T$
4. $SAVEWITHCRC = T$
5. $WINDOWSIZE = 250000.0$
6. $USEHAMMING = T$
7. $PREEMCOEF = 0.97$
8. $NUMCHANS = 20$
9. $CEPLIFTER = 22$
10. $NUMCEPS = 12$
11. $ENORMALISE = F$

The following command was used to initialize the HMM model:
HCompV -C comp.cfg -f 0.01 -S train.scp -M hmm0/ proto
where *comp.cfg* contains the above configuration parameters, *train.scp* contains the names of the files which were used for training, *hmm0* is the directory in which the resulting model is output and *proto* is the prototype model.

2.2.2 Re-estimation

The next task was to re-estimate the flat start monophone models. Before re-estimating two files were generated. The first file *hmm0/macros* was generated by adding the following line in the beginning of the *hmm0/vFloors* file, which was generated in the previous stage: ~o <MFCC_0_D_A> <VecSize> 39

The second file *hmm0/hmmdefs* was generated by using a small script, *init*.

./init phone0.list >hmm0/hmmdefs

This script takes as an input the phone list which lacks *sp*. It uses the *hmm0/vFloors* and *phone0.list* to generate the *hmm0/hmmdefs* file by copying the HMM model in the *hmm0/vFloors* for each phone in *phone0.list*.

The last point before re-estimation is that new label files need to be generated for *HERest*, since it expects the phone-labeled data without *sp* in the same directory as the mfcc files. To convert the phone level MLF file *phone0.mlf* to the corresponding label files, a small script *phone2label* was used. The script simply generates the label files in the *mfcc/* directory.

./phone2label phones0.mlf

Next, the re-estimation was done using *HERest* which requires the configuration file *comp.cfg* (used with *HCompV* to generate the flat start monophones), a set of pruning thresholds, a training file, *macros*, *hmmdefs*, and *sp*-less monophones file *phone0.list*.

HERest -C comp.cfg -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1/ phone0.list

10 rounds of re-estimation were performed.

2.2.3 Fixing the silence models

The next task was to fix the silence models to introduce the short-pause model. The short-pause model is used to model the pauses of the speaker between words. To build the short-pause model the phone level transcription files were modified and then the HMM models were trained using these transcriptions.

sp model was created in three steps. The first step was to copy the middle state from the HMM built for *sil* in the previous stage and use it to build a model for *sp*. The script *sil2sp* extracted the middle state from the *sil* model and generated this HMM definition file with the *sp* model.

./sil2sp hmm10/hmmdefs >hmm11/hmmdefs
The *macros* file was copied from the hmm10/ directory to the hmm11/ directory.

The next task was to create states 2 and 4 for the *sp* model and tie the middle state of the *sil* model to the state 2 of the *sp* model. This is performed by the tool *HHEd*:

HHEd -H hmm11/macros -H hmm11/hmmdefs -M hmm12/ sil.hed phone1.list
where *phone1.list* also contains the *sp* phone and *sil.hed* contains the following commands:

AT 2 4 0.2 sil.transP

AT 4 2 0.2 sil.transP

AT 1 3 0.3 sp.transP

TI silst sil.state[3],sp.state[2]

After the HMMs for the silence model were generated, the phone level MLF file *phone1.mlf*, which has *sp* inserted after every word, was created using the tool *HLEd*.

HLEd -d dict.list -i phone1.mlf mkPhones1.led word0.mlf

After the phone level MLF with *sp* inserted has been generated, the corresponding label files were generated in the mfcc/ directory before re-estimation.

./phone2label phone1.mlf
Then 8 more rounds of re-estimation was done using *HERest*

2.2.4 Realigning the data

The dictionary contained multiple pronunciations for some words, particularly the function words. The phone models created so far were used to realign the training data and create new transcriptions. For this the tool *HVite* was run in the forced alignment mode. One of the main uses of forced alignment is to determine the actual pronunciations used in the utterances used to train the HMM models. The transcriptions thus generated were then used to train the HMM models.
This was done with a single invocation of the HTK recognition tool *HVite*, viz.

HVite -o SWT -b sil -C comp.cfg -a -H hmm20/macros -H hmm20/hmmdefs -i phone2.mlf -m -t 250.0 -y lab -I word0.mlf -S train.scp new_dict.list phone1.list

This command uses the HMMs stored in *hmm20/* to transform the input word level transcriptions *word0.mlf* to the new phone level transcriptions *phone2.mlf* using the pronunciations stored in the dictionary. The key difference between this operation and the original word-to-phone mapping performed by *HLEd* is that the recogniser considers all pronunciations for each word and outputs the pronunciation that best matches the acoustic data.

2.2.5 Making Triphones from Monophones

Given a set of monophone HMMs, the final stage of model building was to create context-dependent triphone HMMs. This was done in two steps. Firstly, the monophone transcriptions were converted to triphone transcriptions and a set of triphone models were created by copying the monophones and re-estimating. Secondly, similar acoustic states of these triphones were tied to ensure that all the state distributions could be robustly estimated.

The tool *HLEd* was used to generate a list of all triphones for which there was at least one example in the training data.

HLEd -n triphones1 -i wintri.mlf mktri.led phone2.mlf
Executing the above command converts the monophone transcriptions in *phone1.mlf* to an equivalent set of triphone transcriptions in *wintri.mlf*. At the same time, a list of triphones is written to the file *triphones1*. The script *mktri.led* contains the following commands:

WB sp

WB sil

TC
The cloning of the HMM models was done using the tool *HHEd*.

HHEd -H hmm20/macros -H hmm20/hmmdefs -M hmm21/ mktri.hed phone1.list
The file *mktri.hed* was generated using the Perl script *maketrihed* which was included in the *HTKTutorial* directory.

After the clones were generated, first the label files corresponding to the triphone transcriptions were created in the mfcc/ directory and then 9 more rounds of re-estimation of the HMM models were done.

./phone2label_1 wintri.mlf
The final re-estimation using *HERest* was performed by setting the -s option as the output would be used in performing the state-tying in the next stage.

HERest -C comp.cfg -I wintri.mlf -t 250.0 150.0 1000.0 -s stats -S train.scp -H hmm20/macros -H hmm20/hmmdefs -M hmm21/ triphones1

2.2.6 Making Tied-State Triphones

The outcome of the previous stage was a set of triphone HMMs with all the triphones in a phone set sharing a transition matrix. The last step in the model building process was to tie states within triphone sets in order to share data and thus be able to make robust parameter estimates.

Before the decision tree tying could be preformed first a file named *fulllist* was generated using the tool *HDMan*. The file *fulllist* contained the list of all possible triphones that can be generated using the dictionary.

HDMan -b sp -n fulllist -g global1.ded -l flog beep-tri dict.list
where *global1.ded* is an edit script which contained the following commands:

HTK Results
SENT: %Correct = 0.86 [H=22, S=2522, N=2544]
WORD: %Corr=9.35, Acc.=-3.33 [H=3659, D=4602, S=30855, I=4962, N=39116]

Table 1: The recognition results obtained using the bigram language model. The line starting with SENT: indicates that out of 2544 utterances, 22 were recognised. The following line starting with WORD: gives the word level statistics and indicates that of the 39116 words in total, 3659 were recognised correctly. There were 4602 deletion errors (D), 30855 substitution errors (S) and 4962 insertion errors (I). The accuracy figure (Acc) of -3.33% is lower than the percentage correct (Cor) because it takes account of the insertion errors which the latter ignores.

AS sp

RS cmu

MP sil sil sp

TC

After this, decision tree tying was performed by running *HHEd*:

HHEd -H hmm30/macros -H hmm30/hmmdefs -M hmm31/ tree.hed triphones1 >log
where *tree.hed* contained the contents to be examined for possible clustering. This was generated using a script *mkclscript* included in the *RMHTK/scripts* directory.

Finally, few more rounds of training using *HERest* were performed and the final HMM, hmm40/ was generated, which was used in the recognition.

2.3 Recognition results using bigram language model

The bigram language model which was used in the basic recognition stage was built using SRILM [4]. The list of words used for training the HMM model was used as a vocabulary to generate the bigram backoff language model using SRILM.

After the bigram language model was built, the bigram word network was constructed using the HTK Tool *HBuild*. The following command was used to build the word network:

HBuild -n BigramLM.lm -z wlist bilatfile
where *BigramLM.lm* is the language model which was generated using SRILM, and *wlist* is the list of all the words in the language model including the start of sentence marker and the end of sentence marker. The final output, which is a word network, is written to the file *bilatfile*.

Using the generated bigram word network, the basic recognition task was done using the HTK Tool *HVite*. This created a lattice for each utterance that could later be rescored using the HTK Tool *HLRescore*. The generated lattices were stored in the *lattice/* directory. The basic recognition was done by executing the following command:

HVite -T 1 -C test.cfg -H hmm40/macros -H hmm40/hmmdefs -S test.scp -i recout.mlf -w bilatfile -z lat -n 5 20 -l lattice/ new_dict.list tiedlist

where *test.cfg* is a configuration file containing the following parameter:

$TARGETKIND = MFCC\_0\_D\_A$

and *test.scp* contained the list of test files and *new_dict.list* was same as *dict.list* with the following three additional pronunciations added to it:

sil sil

<s> [ ] sil

</s> [ ] sil

where <s> is the start of sentence marker and </s> is the end of sentence marker.

The *-n* option here specifies the number of N-best tokens to store per state and the number of N-best hypotheses to generate. In this case top 5 tokens would be stored in each state and the top 20 hypotheses would be output. The lattice are generated using the *-z lat*. The lattice files are stored in the *lattice/* directory.

The statistics on the performance of this basic recognition were obtained with the following command:

HResults tiedlist recout.mlf

The result are given in Table 1.

3 Modifying HTK to use the Time-Based Language Model

This section discusses the modification that was done to the HTK code *HLRescore.c* to test the time-based language model. It also explains how the code can be invoked from the command line to test the time-based language model.

3.1 Lattice Expansion and Lattice Rescoring

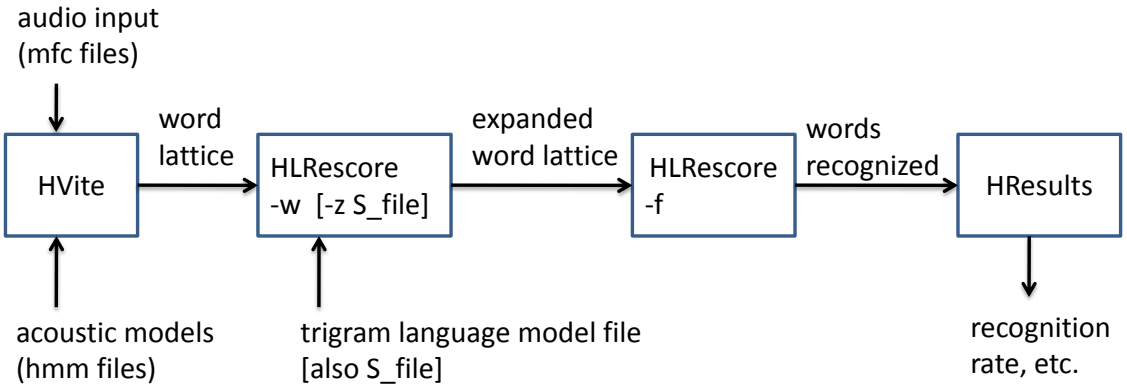

Figure 1: Workflow, with time-based additions in brackets

HTK Results Analysis
SENT: %Correct = 1.30 [H=33, S=2513, N=2546]
WORD: %Corr=11.31, Acc.=2.25 [H=4426, D=6603, S=28088, I=3547, N=39117]

Table 2: The recognition results obtained after rescoring lattices using trigram language model. The language model scaling factor s was set to 5.0 and the fixed penalty p was set to 0.0

Word lattices are composed of the word hypothesis. A word lattice consists of a list of nodes and a list of arcs. Each node in the lattice represents a point in time measured in seconds and each arc represents a word spanning the segment of the input starting at the time of its start node and ending at the time of its end node. For each such span there is an acoustic model score and a language model score.

In order to perform a second pass over the generated lattices with the higher order language models, the lattices must be expanded to the same order. This must be done since each word in the lattice must have a unique history of the same order.

This expansion is done by *HLRescore* with the *-w* option. To apply the trigram model, the lattice generated using the *BigramLM.lm* was rescored using the trigram language model *TrigramLM.lm* which was built using SRILM [4]. The following command did the lattice expansion using the trigram language model. Each lattice file in the *lattice/* directory was expanded and overwritten.

HLRescore -T 1 -D -C rescore_config.cfg -w -n TrigramLM.lm -S test_rescore.scp new_dict.list

The *rescore_config.cfg* file contained the following lines:

STARTWORD=<s>

ENDWORD=</s>

STARTLMWORD=<s>

ENDLMWORD=</s>

After the lattice has been expanded using the new language model, the expanded lattice is rescored to find the best path through the lattice. This can be done by setting the -f option while invoking *HLRescore*. The acoustic model scores and the new language model scores are used to find the best path through the lattice. The following command finds the best path through the expanded lattice:

HLRescore -T 1 -D -C rescore_config.cfg -f -i recout_tri.mlf -S test_rescore.scp new_dict.list

The overall work flow is shown in Figure 1.

3.2 Modifications done to the HTK Code

This section discusses the modifications done in the HTK code *HLRescore.c* to test the time based language model.

HTK Results Analysis
SENT: %Correct = 1.26 [H=32, S=2514, N=2546]
WORD: %Corr=11.39, Acc.=2.28 [H=4456, D=6605, S=28056, I=3563, N=39117]

Table 3: The recognition results obtained after rescoring lattices using trigram and the June 13 S values [1], with the parameters set as before.

HTK Results Analysis
SENT: %Correct=1.26 [H=32, S=2514, N=2546]
WORD: %Corr=11.40, Acc=2.30 [H=4460, D=6607, S=28050, I=3562, N=39117]

Table 4: The recognition results obtained after rescoring lattices using trigram and the July 3 S values, with the parameters set as before.

First it is necessary to understand how lattice expansion is done by *HLRescore.c*. There are several stages. First, *HLRescore.c* does the initialization of various data structures. Then it calls the function *LatExpand()* in file *HLat.c* which does the basic lattice expansion. The function *LatExpand()* splits each node and arc in the lattice into multiple sub-nodes and sub-arcs as required by the new language model. While splitting the nodes and arcs into sub-nodes and sub-arcs respectively, the function *LatExpand()* calls the function *LatLMTrans()* in file *HLat.c* to get the language model probabilities. The function *LatLMTrans()* in turn calls the function *LMTrans()* in the file *HLM.c* to get the new language model probabilities. Given the current word and its history, the function *LMTrans()* returns the log probability using the new language model.

In order to perform the lattice expansion using the time based language model, the tool *HLRescore* was modified to accept the options -w and '-z f' where *f* is the name of the file which contains the value of scaling factor [5] for the various words. Each line in the file *f* starts with a word and then is followed by 24 float values.

Our modified version of *HLRescore.c* first calls the function *InitSValue()* which initializes an array with the scaling factor values for all the words in the file. For the time-based expansion, *HLRescore.c* calls function *TLatExpand()* in *HLat.c* which does the time-based lattice expansion. This function in turn calls the function *TLMTrans()* in *HLM.c* to get the time-based language model probabilities. The function *TLMTrans()* takes the start time of the current word as one of its argument. The function *bucketForTimeInf()* in *HLM.c* finds the bucket to which a word belongs using this start time of the word. The log of the S-Value for a given word and a given bucket is returned by the function *S_Value()* which looks in the array to find the S-Value and returns the log of it. The function *TLMTrans()* finds the log probability of the current word given the history (the local context). These two values are then added and returned, thereby performing the tweaking described in [5]. The function *TLMTrans()* thus returns the tweaked language model probabilities.

After the lattice has been expanded using the time-based language model, the lattice is rescored to find the best path through the lattice using the new language model scores. The *HLRescore* is invoked with the -f option as before to find the best path through the lattice.

	% Correct	Correctly Identified	Deletions	Substitutions	Insertions	Total Words
Bigram	9.35	3659	4602	30855	4962	39116
Trigram	11.31	4426	6603	28088	3547	39117
Time-based 1	11.39	4456	6605	28056	3563	39117
Time-based 2	11.40	4460	6607	28050	3562	39117

Table 5: Summary of Results

The commands which do the time-based lattice expansion and rescoring are shown below:

HLRescore -T 1 -D -C rescore_config.cfg -w -n TrigramLM.lm -z ~/TModel_data/S_file -S test_rescore.scp new_dict.list

HLRescore -T 1 -D -C rescore_config.cfg -f -i recout_tri2.mlf -S test_rescore.scp new_dict.list

The overall working of the modified code is shown in Figure 1, with the modifications noted in brackets.

4 Results

The lattices were generated using the tool *HVite* considering the top 20 hypotheses.

The recognition result using the trigram language model is shown in Table 2. The recognition result using two time-based language models are shown in Tables 3 and 4. The first is the model documented in [5] (the June 13 model). The second is a slightly refined model (the July 3 model) where the chi-squared-statistic is multiplied by .5 before computing p, the value of k is .35 instead of .3, and 1-second buckets were used instead of .5 second buckets in the range from 4 seconds to 8 seconds, and a 1.5 second bucket is used for 8 seconds to 9.5 seconds.

As can be seen from the table, the time-based language model was able to recognize 30 more words as compared to the standard trigram language model.

The overall results are summarized in Table 5.

Acknowledgments

We thank Alejandro Vega for help with Switchboard and SRILM, and David Novick for comments. This work was supported in part by the Defense Advance Projects Research Agency and the National Science Foundation under award IIS-0415150.

Appendix A: Evaluation Process Summary

This section summarizes the process of evaluating a time-based language model in terms of its ability to improve the recognition results obtained by HTK. Also the process of generating the forced alignments is given.

In the typical case, this will involve the desire to test the utility of a refined time-based model; specifically the desire to test on a new S-ratio file (aka a dump file). These files are typically generated using the -tb-dump option to the locally modified version of SRILM's ngram program.

The data files mentioned below are in *~nkiran/HMM_training*, with copies in *~nigel/rescore*. The executables, in particular *HLRescore* and *HResults*, are in *~nkiran/Desktop/htk/bin* .

1. Start with the word lattices generated by HTK using the simple bigram language model and the test set. These are the files in *~nkiran/Desktop/backup/lattice/* directory, which should only change if *HVite* is run on a different test set, so they are read-only. *HLRescore* needs a copy of these that it can overwrite, so our first step is to copy all of these to a working directory.

rm -r lattices-tmp

cp -r lattice lattices-tmp

*chmod 644 lattices-tmp/**

This has to be done afresh every time, to avoid inadvertently rescoring lattices which have already been rescored.

2. It is necessary to specify for *HLRescore* where to find the lattice files. *test_rescore.scp* is the master list of all the files in lattice, but the files to operate on are the copies in *lattice-tmp*. If the *lattice-list.scp* file is not already up-to-date, it is necessary to copy *test_rescore.scp* to that location and then edit *lattice-list.scp* to change the directory parts of the paths to the correct location, here *lattices-tmp*.

cp test_rescore.scp lattice-list.scp

emacs lattice-list.scp

3. The next step is to actually rescore the lattices.

HLRescore -T 1 -D -C rescore_config.cfg -w -n TrigramLM.lm -z dump-july3 -S lattice-list.scp new_dict.list

The -z option specifies the file of S-ratios. This rescoring takes a couple of minutes. The result is that the files in *lattices-tmp* are overwritten.

4. The next step is to run a search to find the best paths through these lattices:

HLRescore -T 1 -D -C rescore_config.cfg -f -i recout.mlf -S lattice-list.scp -p 0.0 -s 5.0 new_dict.list

Note that there is both a -S option and a -s option. After a minute or so this creates the file *recout.mlf*, which contains the recognition results.

5. Now the recognition results can be compared to those in the transcripts. Since the tran-

scripts are in the directory *mfcc*, it is necessary to first edit *recout.mlf* to replace all references to *lattice-list* with *mfcc*.

emacs recout.mlf

Finally HResults gives the statistics:

HResults tiedlist recout.mlf

This gives the %correct words and the other statistics.

Note: if running this command results in a *Fatal Error: 6550 HTK Format error*, then a file needs to be fixed. To find out where, run the command with the -T 10 option to see the name of the mfc file where the error occurs. In the corresponding transcription file there may be arabic numerals; if so, replace them with the corresponding English word. For example, the number *8* should appear as *EIGHT* in the transcription file.

Appendix B: Forced Alignment

This appendix summarizes how HVite can be made to compute the forced alignments for the utterances, perhaps as a preliminary step in the generation of a time-based model for new corpus.

1. First the MFCC files need to be generated for the utterances whose time alignments are desired. For this first we need to generate the list of audio files whose time alignments are desired. An example of this kind of file is given in *∼nkiran/HMM_Training/test_copy.scp*.

2. After the list of the files has been generated, the following command generates the MFCC files:

HCopy -T 1 -C copy.cfg -S test_copy.scp

This command will generate the MFCC files. The label files corresponding to the audio files should also be in the same directory as the MFCC files and should have the same file name as the corresponding MFCC file but the extension should be .lab.

3. Running the following command will generate the time aligned files with the extension .rec and they will be written to the same directory as the MFCC and label files.

HVite -a -b sil -o SW -C comp.cfg -H hmm40/macros -H hmm40/hmmdefs -S abc.scp new_dict.list tiedlist

where *abc.scp* is the list of the MFCC files whose time alignments are to be generated. An example of this kind of file is given in *∼nkiran/HMM_Training/test.scp*.

The output transcriptions can be written to an MLF using the -i option.

References

[1] Clark, Herbert H., Speaking in Time. *Speech Communication*, 36, pp 5–13, 2002.

[2] Manually Corrected Switchboard Word Alignments. January 29, 2003. ISIP, Mississippi State University. retrieved from http://www. ece.msstate.edu/research/isip/projects/ switchboard/

[3] Godfrey, J. J., Holliman, E. C., & McDaniel J. Switchboard: Telephone speech corpus for research and development. Proceedings of ICASSP, pp. 517-520, 1992.

[4] Stolcke, Andreas. SRILM - An Extensible Language Modeling Toolkit, in Proc. Intl. Conf. Spoken Language Processing, 2002.

[5] Ward, Nigel G. and Vega, Alejandro
Modeling the Effects on Time-into-Utterance on Word Probabilities. Interspeech 2008, to appear.

[6] The HTK Book, from http://htk.eng.cam.ac.uk/docs/docs.shtml