Forced alignment tool package user manual

Overview:

This package is designed for finding the correct pronunciation transcriptions for a large dataset. Usually, a large database does not provide phone level transcriptions. The phone level transcriptions, however, are needed for training HMM models. To find the correct phone level transcriptions, the basic assumption is that a database of wave files with associated word level transcriptions for each sentence is already available. A pronunciation dictionary for each word in the database is also needed. In the dictionary, some words might have multiple pronunciations. So, these words should have multiple entries. If a long (several minute) speech passage, with transcription) is available, this long passage should first be segmented into short sentence level (typically around 5 to 10 seconds each) segments with associated text transcriptions for each sentence, in order for the forced alignment and training to work well. If phonetic level transcriptions are already available, (i.e., TIMIT), this step is not needed at all, since essentially this step is only needed to create the phonetic level transcriptions. When dealing with a large database, the training stage of an ASR system involves two steps: initialization and embedded training. In initialization, all the models will be initialized by the global mean and covariance of all the acoustic features in the same manner. In embedded training, HTK will connect the models embedded in each sentence according to its phone transcription, and accumulate the statistics such as the mean and covariance, using the Baum-Welch method. Once all the data in the database has been processed, the accumulated statistics will be used to re-estimate the model parameters for all the models simultaneously. Since the model connection in embedded training depends entirely on the phonetic transcription, and embedded training is the core step in the training stage, it is very important to insure correct phonetic transcriptions using forced alignment.

Generally speaking, forced alignment task includes three steps: feature extraction, training an initial set of models, and forced alignment. Before these three steps, the starting point is a set of wave files, along with a set of word level transcriptions, as stated above. After the features have been extracted for each wave file, the word level transcription of each sentence will be expanded to an initial phonetic transcription using the pronunciation dictionary. For words that have multiple pronunciations, this initial phonetic transcription will arbitrarily use the first pronunciation, regardless of the true pronunciation by the speaker. This is why forced alignment is needed. Then, an initial set of HMM models will be trained using this set of initial phonetic transcriptions. The training involves two steps: flat start initialization and embedded training as described above. Finally, this initial set of models will be used to find the correct phonetic transcription of each sentence based on the features, namely, what the speaker really pronounced. Meanwhile, the updated phonetic transcriptions are used to retrain the HMM models. This processing will be iterated for a couple of times. After the last iteration of forced alignment, the resulting phonetic transcription will be regarded as the "perfect" version, and will be used in all the other parts of an ASR system.

The forced alignment tool package includes three separate tools:

1. Tool\_Compute\_Feat: this tool is used to extract features from a list of wave files.

2. Tool\_trainFA: this tool is used to train an initial set of HMM models.

3. Tool\_FA: this tool is used to do forced alignment and retrain the HMM models.

Note that tools 1 and 2 can be used for tasks other than forced alignment. However, before tool 3 can be used, features files must be created and an initial set of HMM models are needed—meaning that either tool 1 and tool 2 must be used first, or the features and acoustic models must be created with some other similar tools, which use the same file formats as tools 1 and 2. Note that the forced alignment tool, in addition to creating the phonetic transcriptions, updates HMM acoustic models.

Each tool is a matlab m file, and each m name begins with “Tool\_...” A setup file is needed for each tool, and the default file name of the setup file is always the same as the matlab file name, except the file extension is “dcf” rather than “m.” For example, the default setup file for Tool\_trainFA.m is Tool\_trainFA.dcf. The setup file for a tool contains all the control options and parameters for that tool. When a tool is called, the setup file is the only argument that can be passed. For example, to call Tool\_trainFA.m, the format will be Tool\_trainFA(‘Tool\_trainFA.dcf’). The setup file and the matlab file should be placed in the same experiment folder. If no argument is provided for a tool, the default setup file is assumed (same name as the tool name, except extension “.dcf.”) Next, a detailed description of how to use each tool is given.

1. Tool\_Compute\_Feat

1.1. Data preparation

Before you run this tool, some data preparation needs to be done. First, create a folder called “data.” Inside this folder, create a subfolder called “train\_wave.” This is the folder where all the training wave files are to be placed. Copy all the training wave files into this folder. Next, go outside “data” folder, and create another folder called “exp.” This will be your experiment folder. Then, copy "Tool\_Compute\_Feat.m" and "Tool\_Compute\_Feat.dcf" from the folder "Tool\_FA" into the “exp” folder. In addition, also copy "readhtk.m" file from "files needed" folder into "exp."

In addition, you need to make a list of all the wave files to be processed. You can easily write a program to do this. A short program called “makelist.m” is provided to do this (copy this program from "files needed" folder into “exp” folder). In this program, you need to specify which folder the wave files are stored (“..\data\train\_wave” in this case). The path is with respect to the experiment path. After you run the program, a folder “lists” will be created, inside which a list file“wavefile.lst” is created ( These names can be changed easily in this program). This list will look like this with each file in one row:

..\data\train\_wave\F00C1041.WAV

..\data\train\_wave\F00C1042.WAV

..\data\train\_wave\F00C1043.WAV

..\data\train\_wave\F00C1044.WAV

..\data\train\_wave\F00C1045.WAV

..\data\train\_wave\F00C1046.WAV

..\data\train\_wave\F00C1047.WAV

At this point, we are ready to run the tool to extract features.

1.2. Run tool

Use a text editor to open the file "Tool\_Compute\_Feat.dcf" inside "exp" folder. First, you will see some control questions such as "Trace\_on," "Feat\_on," etc. These are the options to turn on/off these functionalities. The only two valid answers for such questions are "y" and "n" (case insensitive). There are some other files or folders that you need to specify for the tool in this setup file. We describe each of them later in this document. Also, you can indicate comment lines using "%" at the beginning of the line. A comment line does not have to end with another "%. For clarity, "%" can be placed at each end.

Make sure that the setup file does not contain any "tab" (\t). "\t" is not recognized by the tool code.

To begin with, in the dcf setup file, there is a section called "Global settings." It has three items:

Trace\_on: if you set "y," you will be able to see the progress of the feature extraction for each utterance on your screen. You can turn it off by setting "n."

Clean\_up : "y" means to clean up all the old feature files in your feature file folder (specified later) before feature extraction.

LogDir: you can specify a folder to store a progress report. The report is named as "progress\_feat.log." This report will keep track of whether an error happened, how many features for each wave file, and what kinds of features were extracted, etc.

Next, there are two questions:

Feat\_On: "y" means to turn on feature extraction. "n" means to turn it off.

FrtEnd\_opt: this is to select different feature extraction methods. There are three available options: HTK\_MFCC, HTK\_PLP and User. HTK\_MFCC and HTK\_PLP are provided by HTK, and "User" means the tfrontm frontend. These options are all case-insensitive. If you choose to use tfrontm frontend, there are some extra steps to follow before you use it.

a. Copy the folder "v7" in "files needed" to "exp".

b. Open a command window, compile "tfrontm.m" inside "v7" using "mcc" command, then create a folder called "tfront" inside "exp" folder, and copy the file "tfrontm.exe" from "v7" to "tfront." Note that you need to have a “C” compiler installed (such as Microsoft visual C ). The version of the C compiler (64 or 32 bit ) needs to match that of your matlab. The Matlab compiler toolbox is also needed.

c. Copy the file "cp\_42.ini" from "files needed" folder to "tfront" folder, and rename the file "cp\_fea13.ini".

d. Copy the file "SNR801.trn" from "files needed" folder to "tfront" folder, and rename it "tfrontm.dat."

Please refer to a detailed description of the tfrontm frontend if you want to use it.

Note that no matter which frontend you use, the wave files are always needed. In tfrontm frontend, we provide a function called "rd\_audio.m." The wave file format supported by this function includes: NIST ( which is used by TIMIT, also called SPHERE), WAVE ( also called RIFF, which is the Microsoft WAVE files used on PCs ), and RAW (which has no headers).

Wave\_List: you need to specify the wave file list that contains all the wave files to be processed. In the example, "lists\wavefile.lst" is provided. This was generated by the program "makelist.m." This list, and a procedure for making the list, was described above.

Next, note that no matter which frontend you choose, two things will be generated:

Feat\_folder: this is the folder that stores all the feature files generated. You do not need to create this folder manually. It will be automatically created. You only need to specify its path. In our example, the path is "..\data\Feat." Again, the path is with respect to the experiment folder, which is "exp."

Feat\_List: A list of all the feature files will also be generated for later use. This is where you specify the path of this list. In our example, as specified, a "featfile.lst" will be generated in the folder "lists."

Next, we come to the "HTK\_MFCC" settings. First, create a folder called "toolconfs" inside the "exp" folder, and copy the file "hcopy\_MFCC.conf" from "files\_needed" folder into "toolconfs." "hcopy\_MFCC.conf" is a configuration file for HTK\_MFCC frontend. It specifies a set of necessary parameters for this frontend, such as the frame length, frame space, wave file format, number of filterbank channels, etc. Please read Chapter 5 of the HTKbook for details. Then, specify the configuration file path in "Conf\_MFCC."

Note that in our example, the wave file format is WAVE, which needs to be known ahead. So, as you can see in "hcopy\_MFCC.conf," the "SOURCEFORMAT" is also set to "WAV" to match the wave file format. If you change to TIMIT database, you need to change "SOURCEFORMAT" to "NIST" because that is the wave file format in TIMIT. Chapter 5 of the HTKbook provides details of all the wave file format supported by HTK.

Conf\_MFCC: specify the configuration file for HTK\_MFCC frontend.

Similarly, if you want to use HTK\_PLP frontend, you need to copy the file "hcopy\_PLP.conf" from "files\_needed" folder into "toolconfs" and specify its location in "Conf\_PLP." Also, change "FrtEnd\_opt" to HTK\_PLP

Conf\_PLP: specify the configuration file for HTK\_PLP frontend.

Finally, if you want to use the user defined tfrontm frontend, you need to specify its configuration file as well in "Conf\_tfrontm." Please refer to our tfrontm manual for detailed description for this frontend.

Conf\_tfrontm: specify the configuration file for tfrontm. As described before, the location of this file is tfront\tfrontm.dat.

You do not need to specify the configuration files for all three frontends since you will only use one of them at a time. The tool will only check the file location according to which frontend you selected. However, do NOT leave any configuration file place blank. For example, if you choose "User" as your frontend, you can put a "\" or "N\A", or "None" (or anything meaningful to you) in Conf\_MFCC and Conf\_PLP, and only specify the configuration file in Conf\_tfrontm. But do NOT leave Conf\_MFCC and Conf\_PLP blank.

2. Tool\_trainFA

This tool is to train a set of initial acoustic models for forced alignment.

2.1. Data preparation

Before you run this tool, you need to prepare a set of word level transcriptions for the wave files. Usually, when you deal with a large database, you do not have the phone level transcriptions; you only have the word level transcriptions for the utterances. The "standard" word level transcriptions HTK can use have one word in each row, and one transcription file for each sentence. Note that the file name of a transcription file be identical to the file name of the feature file for that sentence (however, the extensions can be anything). Since there are many possible formats the original word transcriptions can have when you obtain the database, there is no "standard" code to convert it to the HTK acceptable format. You need to write your own code to do this step. In our example, you need to create a folder called "train\_word\_trs" inside the folder "data," and put all the "standard" word level transcriptions you made into this folder. To make it clear, you can set the extension of the transcription files to be ".WRD."

Also, you need a list of all the transcription files. You can make this list using "makelist.m". You need to change the "path\_wave" variable to "..\data\train\_word\_trs," and change "listfile" to "wordtrs.lst," and run the program. A "wordtrs.lst" will be generated in the "lists" folder.

Copy "Tool\_trainFA.m" and "Tool\_trainFA.dcf" from "Tool\_FA" folder into "exp" folder.

2.2. Run tool

Use a text editor to open "Tool\_trainFA.dcf". Again, you will see the same three global settings. "Trace\_on" has the same meaning as before. But "Clean\_up" has different meaning:

Clean\_up: if this is set to "y", each target HMM folder (specified in later steps) will be cleaned up before new models are generated, if there are any old models in those folders. By “clean up” we mean that existing models will be deleted.

LogDir: again, you can specify a folder to store a progress report for the training processing. The default report file name is "progress\_trainFA.log." In this report, some information will be provided, such as the number of mixtures, the number of states, the progress of mixture splitting, and the feature vector length, etc.

The next step is "transcription preparation." In the very beginning, you have created a set of word level transcriptions for all the utterances, and a list of all these transcription files as well. In HTK, what is used is the "MLF" format of these transcriptions. Basically, a MLF file puts all the transcriptions in one file, and prefixes each file by its path. "transcription preparation" step converts the original word level transcriptions to its MLF format, and also generates initial phone transcriptions based on the word MLF file.

Trans\_prep: "y" turns on transcription preparation. "n" turns it off.

Gen\_Word\_MLF: "y" converts the original word level transcriptions to a MLF file.

Word\_trs\_list: you need to provide a list of all the original word transcriptions to convert. In our example, it is lists\wordtrs.lst.

Conf\_wrdmlf: this is the configuration file to make the MLF file. Copy "wordmlf.led" file from "files needed folder" to "toolconfs" folder, and specify the file path here.

WordMLF: this is the output MLF file. You need to manually create a folder called "labs" inside "exp" folder, and a MLF file "word.mlf" will be generated inside "labs" as specified in WordMLF place.

After the word MLF file has been generated, the next step is to create two phone level MLF files. The first MLF file does not contain "sp," which is a short pause after each word; the second MLF file has a "sp" after each word. Later on, a set of low order HMM models (usually 1 or 2 mixtures ) that does not have "sp" model will be trained using the first phone level MLF file, and then, a "sp" will be introduced, and the second MLF will be used to train this expanded set of HMM models. So, here, we need two MLF files for later use.

Gen\_Phn\_MLF: "y" means to generate phone level transcriptions. "n" turns it off.

Four input files are needed to generate phone level transcriptions:

WordMLF: this is the word MLF generated in the last step. Put the same file path here as in WordMLF.

Dict: a dictionary file is needed to convert word MLF to phone MLF. Manually create a folder called "dicts" inside the "exp" folder, and copy the file "dict863\_tone\_sp1" from "files needed" folder into "dicts" folder. Then, specify the dictionary file path in Dict. If you open this file, you will see that all the words are placed on the left side, and their phone pronunciations are on the right. Many words have multiple pronunciations. Note that a "sp" is appended after each word. There is an entry "SENT\_Boundary," and its pronunciation is "sil." This entry is for the forced alignment step, where the "SENT\_Boundary" will be added before and after each sentence.

Conf\_wrd2phn\_nosp: this is the configuration file to convert the word MLF to the phone MLF without "sp." Copy the file "word2phn\_nosp.led" from "files needed" folder into "toolconfs" folder, and specify the path here.

Conf\_wrd2phn\_sp: this is the configuration file to convert the word MLF to the phone MLF with "sp." Copy the file "word2phn\_sp.led" from "files needed" folder into "toolconfs" folder, and specify the path.

If you open either "word2phn\_sp.led" or "word2phn\_nosp.led", you will see a couple of commands, one command in each row. For example, "EX" means to expand each word with its phone pronunciation; "IS sil sil" means to insert a "sil" before and after each sentence. For a complete set of command descriptions, please refer to HTKbook section 17.10.

Four output files will be generated:

PhoneMLF\_nosp: this is the phone MLF file without "sp." In this example, it is generated in "lab" folder, and its file name is phone\_nosp.mlf.

PhoneMLF\_sp: this is the phone MLF file with "sp" between words. In this example, its file path is labs\phone\_sp.mlf.

Note that when HTK expands each word, only the first pronunciation of each word in the dictionary will be used. So, both the phone MLF files contain only the first pronunciation of each word. These initial MLF's will be used to train an initial set of HMMs. Then, forced alignment will be used to find the correct pronunciations according to the acoustic information.

PhoneList\_nosp: a list of all the phones encountered will be generated. This list does not contain "sp." it will be placed in the "lists" folder, and its file name is "monophone\_nosp" as specified.

PhoneList\_sp: a list of all the phones encountered will be generated. This list contains "sp." Specify its location in a similar way.

At this point, the transcription preparation step is done. These four output files will be used in the training step. Note that if you have already done this step, you can simply turn off "Trans\_prep."

The next step is to train a set of HMM models for forced alignment. Similarly, there are some global settings to be specified at the beginning.

Train\_on: "y" turns on training. "n" turns it off.

Feat\_List: this is the feature file list generated by the feature extraction tool. In Tool\_Compute\_Feat.dcf, it is specified in "Feat\_List" place. Put the same file path here for training.

Feat\_List\_ini: HTK computes the global mean and variances of all feature files provided to initialize each HMM model. However, sometimes, this will be a poor initialization for the following step, which is called embedded training. Empirically, do not use short sentences to initialize the models. If many short sentences (1 or 2 seconds long) are used for initialization, the embedded training step will fail. You should use long sentences, at least 3 seconds long each. So, you can select a subset of all utterances, and make a list of them, then, use this list of feature files to initialize the models. In our example, this list is located as "lists\featfile\_ini.lst." After the models have been initialized, the full feature file list "Feat\_List" will be used to do the embedded training.

numState: the number of emitting states. Two non-emitting states will also be used (one before and one after the emitting states). For example, if numStates=3, then, there are actually 5 states.

numMixture: specify the mixture splitting sequence. The models usually start from a low order (such as 1 mixture), and then gradually split to the desired order, each time increasing by a small amount. Each splitting step is separated by a ";." For example, "1;2;4;6;8;12;16" means to start from 1 mixture models, and split to 2 mixture models, and then 4, 6, 8, 12, 16. You can directly start from a high order model, such as 16, but usually won't get robust training. It's strongly recommended that you go through this splitting processing. You can refer to HTKbook section 10.6 and section 17.8 for details of how the splitting works and why it is more robust than directly starting from a high order.

Iteration: this specifies how many iterations of training will be conducted after each splitting. Again, this sequence is separated by ";" and it must have the same length as the mixture splitting sequence. For example, "3;5;6;7;7;7;7" means 3 iterations of training for 1 mixture models, 5 for 2 mixture models, etc, if numMixture is set to be "1;2;4;6;8;12;16."

Final\_hmmfolder: this is for convenience. In our example, this folder path is set to be "hmms\fhmm." These folders (hmms and fhmm) will be created automatically inside "exp" folder. There are two steps in model training: initialization and embedded training. You will notice that for each step, there is a "target folder" in the setup file to store the HMMs after each step (hmm1 for initialization and hmm2 for embedded training in our example). However, you may forget which step you stop at in training. So, in the forced alignment stage, when loading the HMMs, you do not know where to load the HMMs. Hence, a final HMM folder needs to be specified to store the final HMMs, no matter which training step you stop at. That means: if you choose to do the embedded training, the models from hmm2 (which is the target folder for embedded training) will be copied to the final HMM folder; if you stop at initialization, the models from hmm1 (which is the target folder for initialization) will be copied to the final HMM folder. Hence, you can always load HMMs from the final HMM folder in the forced alignment step. In addition, if you want to modify the previously trained HMMs, such as adding more mixtures by splitting, or adding more training iterations, you can directly load HMMs from the final HMM folder without memorizing which step you were at when you stopped last time.

Next, we come to the initialization settings.

Init: "y" turns on initialization; "n" turns it off.

Then, 4 input files/folders are needed for initialization.

hmmList\_nosp: this is the list of HMMs generated by the transcription preparation step. This list does not include "sp" model. Later on, a "sp" model will be introduced after the first mixture model (1 mixture in our example) is trained.

SrcDir\_init: this is the source directory for the prototype HMM to be initialized. In HTK, before initialization, a prototype HMM needs to be defined. Basically, the prototype defines the topology of the HMM, including the number of states, the transition format, the covariance matrix type, etc. In our example, a directory "hmms\proto" will be automatically created, and a prototype HMM will be generated inside this folder, as specified by SrcDir\_init.

Conf\_proto: this is the configuration file to generate the prototype HMM. You only need to specify the file path here. In our example, the file name is "puser.pcf," and it will be placed in "toolconfs" folder. This file will be automatically generated. You need to copy "makeProto.m" file from "files needed" folder to "exp" folder first.

Conf\_init: this is the configuration file to initialize the models. You need to copy the file "hcompv.conf" from folder "files needed" into "toolconfs" folder, and specify the path here.

The output of the initialization step is a set of HMM models stored in the target folder.

TgtDir\_init: this is the target folder for the initialization step. A set of new HMM models will be saved in the directory specified here. This directory is automatically generated. It is set to hmms\hmm1 in our example. Note that this set of models do not have "sp."

At this point, a set of models have been initialized.

The next step is embedded training. There are four global settings for this step.

Embed\_train: "y" turns on embedded training; "n" turns it off

fix\_sil: "y" means to introduce a "sp" model after the first low order model (does not include "sp") has been trained. Specifically, the "sp" model is copied from the central state of the silence model, and is a one state model (there are two non-emitting states, one emitting state). There is a direct transition between the entering and exit states, namely, the two non-emitting states, because there are actually no short pauses between words sometimes. The silence model is also modified. A forward and a backward transitions are added between the first and the third emitting states. You can refer to HTKbook section 3.2.2 for more details.

The code of the tool works in this way: before embedded training starts, it will first check whether a "sp" model is already in the model set loaded (specified by SrcDir\_Embd). This is because sometimes, you have already trained a set of models including "sp," and you only want to modify this set of models, such as adding mixtures, or adding training iterations. In this case, you can simply turn off the initialization step, and only perform the embedded training step. If a "sp" model is found in the model set, "fix\_sil" will be forced to "n," because there is no need to re-introduce the "sp" model and modify the silence model.

After the "sp" model is introduced, a couple of iterations of training will be performed according to "fix\_iter" specification, then, the model mixtures will be split, starting from the second lowest mixture, according to the "numMixture" sequence.

fix\_iter: this is how many training iterations after the "sp" model has been introduced.

embdOptStr: this is a pruning threshold for embedded training. Normally, you can set it to "-t 250.0 150.0 1000.0" and no need to change. Do not omit the " " on both sides. You can refer to HTKbook section 17.7 for details.

Seven input files/folders are needed as the input of embedded training.

SrcDir\_embd: this is the source directory to load HMMs for embedded training. In our example, we have initialized a set of models. So, we need to load models from hmms\hmm1, where the initial models are stored.

You do not have to always load models from the initialization step. You can actually load models from anywhere. For example, if you already have a set of models in the "Final\_hmmfolder," and you want to perform embedded training for this set of models, such as adding more mixtures, or adding more iterations for the models, you can set "SrcDir\_embd" to "Final\_hmmfolder."

hmmList\_nosp & hmmList\_sp: these are the HMM list with and without "sp." They are generated in the "transcription preparation" step. As stated, before embedded training begins, the tool will first check if a "sp" model already exists; if yes, "fix\_sil" will be forced off, and "hmmList\_sp" will be used; if no, the first low order model will be trained using "hmmList\_nosp," and then, the tool will check "fix\_sil;" if "fix\_sil" is yes, then, the "sp" model will be introduced, and "hmmList\_sp" will be used from then; if "fix\_sil" is no, then, "hmmList\_nosp" will be used all the way till the end.

monoMLF\_nosp & monoMLF\_sp : these are the phone level transcription with and without "sp." They are also generated in the "transcription preparation" step. Their usage is the same as that of hmmList\_nosp & hmmList\_sp.

Conf\_embd: this is the configuration file for embedded training. You need to copy "herest.conf" from "files needed" folder to "toolconfs" folder, and specify the path here.

Conf\_sil: this is the configuration file for fixing the silence. You need to copy "sil.hed" from "files needed" to "toolconfs" folder, and specify the path here. If you open "sil.hed" file, you will see a couple of commands such as "AT," "TI." "AT" means to add a transition between two states. "TI" means tie the designated states of two models together, so that they will have the same mean and covariance matrix. You can refer to HTKbook section 17.8 for more details.

There are two outputs for embedded training.

TgtDir\_embd: this is the target folder where the output HMMs are stored. In our example, the directory is hmms\hmm2.

Stat\_embd: this is a statistic file generated by embedded training. This is only a byproduct. You won't need it in the forced alignment state.

At this point, the embedded training is done.

We have described how to train a set of models from the initialization step till the end. Yet, there are cases that you already have a set of models, and suppose you only want to create more mixtures based on this set of models. Suppose you have a set of 8 mixture models with "sp," and you want to create 16 mixture models. Here is how you do it with the tool:

a. turn off "Trans\_prep" and turn on "Train\_on," then turn off "Init."

b. turn on "Embed\_train."

c. you can leave "fix\_sil" either y or n. ( set it to n if your 8 mixture models do not have "sp" )

d. set SrcDir\_embd to "hmms\fhmm" or to "hmms\hmm2," and set TgtDir\_embd to "hmms\hmm2."

e. Set numMixture to 8;16. Do not set it to 16 only.

f. Set Iteration to 0;7. Do not set it to 7 only.

3. Tool\_FA

After a set of initial HMMs have been trained, we come to the forced alignment step. Please copy "Tool\_FA.m" and "Tool\_FA.dcf" from "Tool\_FA" folder to "exp." Then, open "Tool\_FA.dcf" file.

In simple words, the forced alignment is an iterative processing. The phone transcriptions will be corrected using the set of models obtained in the training stage, then, the new transcription will be used to refine the models. This processing will be repeated for a couple of iterations. The final transcription will be the "perfect" version.

Again, at the beginning, there are three global options. "Trace\_on" has the same meaning as in other tools.

Clean\_up: "y" means to clean up the refined model before forced alignment begins, and also clean up the old aligned transcriptions before forced alignment.

LogDir: a progress report will be generated in this folder. The file name is "progress\_FA.log." The report keeps track of how many iterations there are, and how much change was made in the aligned transcription after each iteration, etc.

After global settings, there are other settings for forced alignment.

FA\_on: "y" turns on forced alignment; "n" turns it off.

FA\_iteration: how many iterations are needed. One iteration includes: first, update the phone transcriptions using current models; second: re-train the models using the updated phone transcriptions.

Embd\_iteration: this is how many embedded training iterations are needed to retrain the models each time after the transcriptions are updated. Again, the numbers are separated by ";" and the length of this sequence must be equal to FA\_iteration. A zero means that the models will not be retrained after forced alignment.

Output\_level: this specifies what you want to see in the aligned phone level transcriptions. "-o SWT" means that you only want the aligned phones. Scores (S), words (W) and time boundaries (T) will be suppressed. Do not forget the " " on both sides. You can refer to HTKbook section 17.8 for more details.

Prune\_FA: this is a prune factor for the decoder in forced alignment mode. The smaller the first number is, the faster the decoder will be, but more sentences will not be successfully aligned. The bigger the last number is, the more sentences will survive, but the speed will also be slower. You can refer to HTKbook section 17.8 for more details.

Prune\_embd: this is the same prune factor for embedded training as that in embdOptStr in Too\_trainFA.dcf.

Init\_phoneMLF: specify the initial phone level transcription file path (before any forced alignment iterations). Actually, the forced alignment processing itself does not need it. This is for statistical purpose. Each time after the transcription is updated, the tool will compute how much change between the new transcription and the old one, as you will see in the report. The first updated transcription will be compared with this initial one.

There are seven input files/folders needed by forced alignment and model refinement.

WordMLF: this is the word level MLF file generated in the "transcription preparation" step in Tool\_trainFA.dcf.

Feat\_List: this is the list of all the feature files to be aligned. This list is generated in the feature extraction step.

Dict: this is the dictionary file. Note that a "SENT\_Boundary" should be in the dictionary. In our example, the dictionary has a "sp" after each word. If your model set does not include "sp," you should delete "sp" from this dictionary as well.

hmmList: a list of all the HMMs generated in the "transcription preparation" step in Tool\_trainFA.dcf. This list needs to match the model set (whether there is a "sp" or not)

SrcDir\_hmm: this is the folder to load the initial HMMs. In our example, all the initial HMMs are stored in "hmms\fhmm" folder. These HMMs will be used to do the first round of forced alignment.

Conf\_FA: this is the configuration file for the decoder. Copy "hvite.conf" from "files needed" folder to "toolconfs" folder, and specify the file path here.

Conf\_embd: this is the configuration file for embedded training.

There are three output files/folders generated.

aligned\_folder: this is the folder to store the aligned transcriptions in each iteration. If Clean\_up is turned on, the old aligned transcriptions will be cleaned up first. A series of transcription files named as aligned\_1.mlf, aligned\_2.mlf,... will be generated. In our example, we specify this folder to be "labs."

TgtDir\_hmm: this is the folder to store the retrained models in each iterations. In our example, a "refined\_hmm" folder will be generated inside hmms folder, as specified. The refined models are used to do forced alignment in the next iteration.

Stat\_embd: this is a statistic file generated by embedded training.

At this point, a set of aligned MLF files have been genereated. The final "aligned\_N.mlf" (in our example, N=6) is the "perfect" phone transcription. In addition, a set of well trained HMM models are stored in the folder "hmms\refined\_hmm."

Since you have got a set of well trained HMMs, they can be used to align new data. Here are the main steps in aligning new transcriptions using a well trained model set.

a. Use the feature extraction tool (Tool\_Compute\_Feat.dcf) to extract features. Note that the feature type needs to match those used to train the models; for example, they are both 39 MFCC features.

b. In the tool for training (Tool\_trainFA.dcf), turn on "Trans\_prep," and "Gen\_Word\_MLF," turn off "Gen\_Phn\_MLF." So, the word level transcription of the new data will be generated. Turn off "Train\_on."

c. Set FA\_on to "y" in Tool\_FA.dcf.

d. Set FA\_iteration to 1. Since the models are well trained, you normally do not need multiple iterations. Of course, you can do that if you want.

e. Set Embd\_iteration to 0. Again, since the models are well trained, you do not need to retrain them.

f. Set SrcDir\_hmm to "hmms\refined\_hmm," where the well trained HMMs are stored. Also, specify the list of HMMs in hmmList, in accordance to the refined HMMs.

g. Set "Init\_phoneMLF" to "labs\aligned\_1.mlf." Since there is actually no initial phone transcription for the new data (because the models exist already, so no need to generate initial phone transcription to train any models), yet a place holder is still needed, we can put the aligned MLF here. Of course, in the progress report, it will tell you no change was made between the "initial transcription" and aligned.mlf, since they are the same files. So, you can ignore the progress report in this case.

Now, you are all done with forced alignment. The final aligned transcription will be used in subsequent tools.

A working example is provided. The main output of the forced alignment tool package is put in the folder “files generated.” The database used is 863 Mandarin Chinese database. The phonetic transcriptions of 74 women speakers, namely, 34616 sentences, were forced aligned. The wave files, as well as the feature files are not included, since they are too big. The tool setup files used are exactly the same as the ones provided in the Tool\_FA folder, as described in this manual. If you go through the steps in this manual using the same data, and same files as provided, you will get identical outputs. For feature extraction, 42 features were generated using the tfrontm MFCC method. These features consist of 12 DCTCs, 1 log energy, 1 pitch (14 static features), and delta, and delta-delta terms. For training, 174 tonal phones were trained, including 27 Initials, 145 tonal Finals, and two silence models ( sil and sp ).

First, open “Log” folder. Inside this folder, you can see the progress reports for feature extraction ( progress\_feat.log ), training initial models ( progress\_trainFA.log ), and forced alignment ( progress\_FA.log ). In progress\_feat.log, it records that the frontend selected was “User,” and the feature dimension was 42. In progress\_trainFA.log, you can see what happened in order. First, the tool did data preparation. This converted word transcriptions to word MLF, and then converted this word MLF to phone MLF. Then, the training started. The mixture splitting sequence as well as the iterations for each mixture splitting was recorded. The flat start initialization was performed first, and the embedded training followed. At the beginning of the embedded training, there was no “sp” model. After the first low order model (1 mixture) had been trained, the “sp” model was introduced, and the silence model was fixed. Then, a list of current number of mixtures being trained was given. After the desired order was achieved (16 mixtures), the models were copied to “hmms\fhmm” folder from “hmms\hmm2” folder. Finally, in progress\_FA.log, you can see the progress of the forced alignment and model refinement. As you see, the forced alignment iteration was specified to be 6, and each time the phonetic transcriptions were updated, the models were retrained 3 times. In the last iteration, only forced alignment was performed, and the models were not refined (as a 0 was specified for the last iteration). After the forced alignment began, the tool checked how much change was made between the updated phonetic transcription and the old transcription obtained in the last round. For example, after the first iteration, it shows that the accuracy was 98.74%. This means that 1.26% of all the phones in the initial transcriptions (before forced alignment) were corrected by the first round forced alignment. Next, the models were retrained using this updated transcription. Then, the forced alignment was performed again, using the refined models. This time, the accuracy became 99.64%. This means that 0.36% of all the phones in the old transcriptions (obtained in the first round) were corrected by the second round forced alignment. This processing kept going until the desired number of iterations was achieved. Note that as the models kept being refined by the updated transcriptions, the accuracy between two consecutive transcriptions got higher. This shows that fewer and fewer phonetic pronunciations were updated by forced alignment, as most of them were already corrected by previous iterations.

Next, open “labs” folder. You can see 6 phone level MLF files generated by the forced alignment iterations. They were named as aligned\_1.mlf, aligned\_2.mlf, etc. In our example, aligned\_6.mlf is assumed to be the “perfect” version, since it was generated by the last iteration. This version will be used in other parts of the ASR system. The files phone\_nosp.mlf, and phone\_sp.mlf are the initial phone transcriptions without and with “sp.” The file word.mlf is the word level MLF file of all the sentences.

Finally, open “hmms” folder. “fhmm” folder stores the HMM models before model refinement by forced alignment step. In other words, these models were trained by the initial phone transcriptions. In order to get these models, the initialization and embedded training were conducted. The models after initialization step were stored in “hmm1,” and the models after embedded training were stored in “hmm2.” The “refined\_hmm” folder stores the refined models. This set of refined models are well trained. It can be used to forced align new data as described by steps a-g above. The “proto” folder stores the model prototype. This prototype defines the model topology. In either fhmm, hmm1, hmm2, or refined\_hmm folders, you can find two files, “hmmdefs” and “macros.” “hmmdefs” has all the parameters of all the models. “macros” defines some “global” parameters which are identical for all models, such as the feature vector dimension, and a variance floor. If any variance of any state in a model falls below this floor, it will be clamped to this floor. “macros” was generated in the initialization step and remained unchanged in all subsequent steps. “hmmdefs” got updated after each embedded training iteration.