Kaldi- based framework to train and evaluate acoustic models with the Archimob corpus User Manual

Spitch AG

30th January 2018

Contents

1 Introduction

This document is the user manual for the basic ASR framework developed by Spitch AG for the Corpus Lab of the University of Zurich, which includes the functionality to:

- Train neural network acoustic models with the Archimob annotated corpus.
- Compile the lingware to do decoding with Kaldi acoustic models and certain vocabulary and language model.
- Evaluate the quality of the acoustic models and the lingware by decoding a set of wavefiles with annotated references.

2 Design guidelines

The whole framework was designed with the target of providing a clear separation between the Archimob specific input, and the publicly available Kaldi scripts. Even though this was a strong initial constraint for the development, it should simplify the addition of future updates of the aforementioned open source tool.

3 User guidelines

3.1 Quick overview

The framework consists of four parts:

- Data preparation: adaptation of the original Archimob files to a more appropriate format for automatic speech recognition.
- Acoustic model training: sequential training of acoustic models, from GMM (Gaussian Mixture Models) to nnet (neural network).
- Lingware compilation: reorganization of the lexicon, the language model, and part of the acoustic models in a single file more suitable for decoding.
- Models evaluation: decoding and comparison of the resulting hypotheses with some references to get an estimation of the quality of the acoustic models and lingware generated in the previous stages.

Table 1 collects a summary of the input and output information, plus the main scripts for each stage.

Step	**Input**	**Output**	**Scripts**
Data preparation	Archimob wavefiles, Exmaralda files	Archimob csv file, chunked wavefiles	extract_audio.sh, process_exmaralda_xml.py
AM training	Archimob csv file, chunked wavefiles	Acoustic models	train_AM.sh
Lingware	Acoustic models, vocabulary, language model	HCLG	compile_lingware.sh, simple_lm.sh
Evaluation	Acoustic models, references, wavefiles, HCLC	WER	decode_nnet.sh

Table 1: Framework overview

3.2 Detailed process flow

All the scripts mentioned in this section should be called from the same directory where they are located. See Section 3.3.1 for further explanation.

3.2.1 Data preparation

The input to this stage is the Archimob videos and the corresponding Exmaralda files with the annotations, as already owned by the Corpus Lab. The goal is to divide the long recordings into smaller chunks more suitable for alignment during acoustic modeling.

1. The wavefiles are extracted from the Archimob videos, with the script extract_audio.sh [1]. This script takes all the files with extension *.mp4* under an input directory, and writes the audio streams to an output directory preserving the basename and changing the extension to *.wav*.

   This is an example of how to extract the audio files:

   ```
   $archimob/extract_audio.sh input_videos full_wav
   ```

   where:

   - input_videos: folder with the Archimob videos.
   - full_wav: output folder for the audio files.

2. The script process_exmaralda_xml.py is used to compile into a single csv file all the ASR relevant information from the original Exmaralda files. Short speech segments are created from the Exmaralda turns, and the corresponding wave chunks are extracted using the timestamps information.

[1]Note that, by default, this script downsamples the waveforms to 8 KHz. If you want to use a different sampling frequency, besides changing this script, the feature extraction configuration file from Kaldi (*conf/mfcc.conf*) should be modified accordingly

The call to process_exmaralda_xml.py is like this:

```
$ archimob/process_exmaralda_xml.py −i trans/∗.exb −w full_wav −o example.csv
−O chunk_wav
```

where:

(a) trans/*.exb: list of Exmaralda files.

(b) full_wav: folder with the Archimob audio files (as generated by extract_audio.sh).

(c) example.csv: output csv file. Table 2 shows an example.

(d) chunk_wav: output folder for the chunked audio files.

Note that it is important to ensure that there is no offset among the timestamps in the Exmaralda file and the video, as this would imply desynchronization in the chunked wavefiles and transcriptions. This can be easily checked by listening to some of the chunked wavefiles and comparing the content with the chunked transcription in the csv file.

```
utt_id,transcription,speaker_id,duration,speech-in-speech,no-relevant-speech
1008-0001,"/ (?)",1008_I,6.24,0,0
1008-0002,"/ lauft /",1008_I,4.10,1,0
1008-0003,"mich würd interessiere /",1008_I,2.00,0,0
1008-0004,"wie si iri jugendzit /",1008_I,2.11,0,0
1008-0005,"als iischtiig /",1008_I,1.65,0,0
1008-0006,"ää$ erläbt hend /",1008_I,3.73,0,0
...
```

Table 2: Example of Archimob csv file

3.2.2 Acoustic model training

Acoustic modeling with the default configuration is run with a single command:

```
$ train_AM.sh −−num−jobs 80 example.csv chunk_wav out_AM
```

where:

- –num-jobs: number of jobs, for parallel processing. In the current example it is assumed that 80 virtual CPU's are available (see Section 4).
- example.csv: input csv file, as generated with process_exmaralda_xml.py.
- chunk_wav: folder with the chunked wavefiles, as generated with process_exmaralda_xml.py.
- out_AM: output folder for the script.

The final acoustic models trained with this script will be located under the folder out_AM/models/discriminative/nnet_disc.

3.2.3 Lingware compilation

The lingware for decoding is generated with this command:

```
$ compile_lingware.sh out_AM/initial_data/ling vocabulary.txt language_model.arpa
                          out_AM/models/discriminative/nnet_disc out_ling
```

where:

- out_AM/initial_data/ling: intermediate folder from the acoustic modeling stage, with the information of the phones used for training [2].
- vocabulary.txt: input file with the vocabulary to be recognized by the lingware. See Table 3 for an example. Note that it should consist only of regular words (this is, silence and noise symbols should be excluded from it).
- language_model.arpa: input language model, in arpa format (see Table 4 for an example). Note that no functionality is provided in the current framework to build this model, since its design is heavily dependent on the application domain.
- out_AM/models/discriminative/nnet_disc: intermediate folder from the acoustic modeling stage, with the discriminative nnet acoustic models. The basic nnet2 acoustic models could be used instead [3].
- out_ling: name of the output folder.

```
a
aa
aaa
aab
aabaue
aabauen
aabauschlacht
aaben
...
```

Table 3: Example of vocabulary for lingware compilation

[2]It is better to take this information from training directly than regenerating it automatically from the decoding data, since small vocabularies might not contain all the phoneset. See sections 3.3.2 and 3.3.3 for more details.

[3]In general, the closer the training data to the application domain (i.e, where the models will be used), the better the discriminative models perform. However, if the application domain is very different to the training data, the nnet2 models might be preferable.

```
\data\
ngram 1=3938
ngram 2=12397

\1-grams:
-1.735163 <\s>
-99 <s> -0.451290
-2.620098 a -0.280351
-3.445199 aa -0.082895
...
```

Table 4: Example of language model for lingware compilation

3.2.4 Evaluation

This is an example of how to evaluate the quality of some previously trained acoustic models and lingware:

```
$ decode_nnet.sh --num-jobs 80 references wav_test out_AM/models/discriminative/nnet_disc
                    out_ling out_decode
```

where:

- –num-jobs: number of jobs, for parallel processing. In the current example it is assumed that 80 virtual CPU's are available (see Section 4).
- references: file with the utterance ids of the wavefiles to be decoded, and the corresponding annotation references [4]. Table 5 shows an example.

1209-0402	s het gschäftslüüt drunder ghaa und het äu anderi drunder ghaa
1209-0403	so simpatisante
1209-0405	jjjòò
1209-0406	me het scho devoo gredt
1209-0407	und me aber me s het äifach s gfüül ghaa aso
1209-0409	nìd esoo nooch a de gränze wie gwüssi
1209-0410	gränzkantoone
...	...

Table 5: Example of references file for decoding

- wav_test: folder with the test wavefiles. Note that the filenames must match the utterance ids in the references file [5].
- out_AM/models/discriminative/nnet_disc: folder with the acoustic models. Note that it must be the same one used to compile the lingware [6].

[4]This format can be obtained by using the script *./archimob/process_exmaralda_xml.py* on the test Exmaralda files, and processing the output csv file with *./archimob/process_archimob_csv.py* and the input parameters **-p** and **-u**.

[5]In this example, the wavefile corresponding to utterance id $1209-0402$ would be $wav\_test/1209-0402.wav$.

[6]This statement is just a simplification: what is actually needed is that the GMM

- out_ling: folder with the lingware. It must have the file *HCLG.fst* under it.
- out_decode: folder for the output files.

Once the evaluation is done, the average performance and other relevant information can be checked in the output folder:

- *best_wer*: file with the best *Word Error Rate*. For example:

```
WER 39.07 [ 9270 / 23728, 1146 ins, 4820 del, 3304 sub ]
out/models/discriminative/nnet_disc/decode/wer_17_0.0
```

- *wer_details*: directory with the optimization parameters that led to that result (see Section 3.3.5), and an analysis of the errors per utterance, speaker, and word:
  - *lmwt*: optimal language model weight.
  - *wip* optimal word insertion penalty.
  - *per_utt*: it contains, for every utterance, the reference, the hypothesis, and the number of correct words, insertions, substitutions, and deletions.
  - *ops*: for every word, number of times it is correctly recognized, substituted, inserted, or deleted.

3.3 Gory details

3.3.1 Running the scripts

The scripts *train_AM.sh*, *compile_lingware.sh*, and *decode_nnet.sh* must always be called from the same directory where they are located. The reason for this strong restriction is that the underlying Kaldi scripts that are called by them also have this assumption, and making the complete flow location independent would have implied modifications in all the involved scripts, which would have made it more complicated to add features from other Kaldi recipes in the future.

3.3.2 Lexicon generation

Pronunciation dictionaries are traditionally generated as a second stage after having designed the phoneset for a target language. In the current framework, however, the process is exactly the opposite: the phoneset is just ex-

tree and transition model parts of the acoustic models used for lingware compilation and decoding are the same.

tracted from the lexicon. The justification for this decision is that Archimob is transcribed according to the Dieth rules, which already follow reasonably close the phonetic realization of each word, as uttered by a certain person in a specific moment. Therefore, pronunciations can be generated by simple concatenation of the word graphemes, with the only exception of some grapheme clusters that represent a single phoneme. Table 6 shows the first lines of the provided grapheme clusters file, *manual/clusters.txt*, plus some pronunciation examples. It contains the sequence of graphemes on the first column, and the symbol or sequence of symbols it must be mapped to [7] in the second one.

Cluster	**Symbol**	**Example**
ch	ch	aabach → */ a b a ch /*
sch	sch	aarisch → */ a r i sch /*
tsch	tcsh, z ch	bretschter → */ b r e tsch t e r /*, */ b r e z ch t e r /*
pf	pf	aapflanze → */ a pf l a n z e /*
ng	ng	afangen → */ a f a ng e n /*
ph	ph	biiphalte → */ b i ph a l t e/*
th	th	raathuus → */ r a th u s /*
ts	z	schtaats → */ sch t a z /*
tz	z	traditzioon → */ t r a d i z i o n /*
gg	gg	wäggis → */ w ä gg i s /*
...	...	...

Table 6: Excerpt from the grapheme clusters file

The script *archimob/create_simple_lexicon.py* is used to generate the lexicon in a very naive way: most of the phonetic knowledge lies on the initial Dieth pronunciations and the grapheme clusters. The script only maps clusters of graphemes to the symbols defined in the clusters file, and all the other graphemes to themselves, as shown in the examples in Table 6.

3.3.3 Coherence between the training and decoding lexica

The training and decoding lexica must be coherent, in the sense that all the phones needed for decoding must also appear in the training data. If this is not the case, the lingware compilation process described in section 3.2.3 will crash, as a result of not being able to assign any acoustic model to the new phones. If something like this happens, it is recommended to do a pre-processing of the decoding vocabulary to map unseen phones to the ones covered during training.

Phoneset coherence should not be a problem in most practical cases, as far as a reasonable amount of data is used for training. Note, however, that a

[7] If a grapheme cluster can be mapped to several symbols (for example, depending on the context), they must be separated by a comma.

strong weakness of the simple lexicon generation approach described above is that any misspelling in the vocabulary might also end up generating a new entry in the phoneset [8]. Although this would not be a critical issue for acoustic model training, it could make lingware compilation crash if the typo is in the decoding vocabulary.

3.3.4 Different acoustic models

During the training process not only a set of acoustic models is trained, but a sequence of them that are supposed to improve the performance of the previous ones:

1. **GMM mono**: Gaussian Mixture models for monophones (v.gr., / *a* /) trained with maximum likelihood.
2. **GMM tri**: Gaussian Mixture Models for basic triphones (v.gr., / *p-a+t* /), trained with maximum likelihood.
3. **GMM tri + lda**: Gaussian Mixture Models for triphones, trained with maximum likelihood, but with an initial kind of LDA transformation on the acoustic features.
4. **GMM tri + lda + MMI**: Gaussian Mixture Models for triphones, with the LDA transformation, and trained discriminatively.
5. **NNET**: p-norm neural network model, trained with maximum likelihood.
6. **NNET-DISC**: the same neural network from NNET, but trained with a discriminative criterion.

Table 7 contains the acoustic model names and the corresponding directories, assuming that the output folder in the call to *train_AM.sh* was *out_AM*.

Model name	**Directory**
GMM mono	out_AM/models/mono
GMM tri	out_AM/models/tri
GMM tri + lda	out_AM/models/tri_lda
GMM tri + lda + MMI	out_AM/models/tri_mmi
NNET	out_AM/models/nnet
NNET DISC	out_AM/models/discriminative/nnet_disc

Table 7: Acoustic models directories

[8]For example, if the word “öisi” is mistyped as “öiçi”, the grapheme **ç** would automatically be added to the phoneset with the symbol *ç*.

3.3.5 Decoding optimization

Decoding is performed in Kaldi in two steps:

1. A lattice of hypotheses paths is generated [9].
2. The best hypothesis is chosen [10], considering several combinations of two factors:
   - Language model weight: multiplier for the language model probabilities of each path in the lattice. This parameter helps to balance out the relative relevance of the acoustic and language models.
   - Word insertion penalty: fixed penalty added to each path every time a new word is recognized. Positive insertion penalties will favor paths in the lattice with few words, while negative ones will encourage paths with many words.

The decoding results will be the ones corresponding to the language model weight and word insertion penalty that minimize the global word error rate in the complete testset. Note that this corresponds rather to a classical "development" stage, and not to a pure test one. To evaluate the quality of the system, which includes the acoustic models, the lingware, and the decoder configuration, the language model weight and the word insertion penalty should be fixed to the optimal values obtained during development.

3.3.6 NFS issues

Some stages of the implementation imply heavy I/O traffic on the shared drive, with many parallel processes writing on it and reading from the same locations. As a result, it is possible that simple operations like copying a file fail with the message "Resource temporarily unavailable". If this happens, it means that either the hard drive is too slow, or that NFS has to be reconfigured to cope with these circumstances.

4 Configuration

4.1 Initial setup

- The environment variable **KALDI_ROOT** in *scripts/path.sh* should be set to the path where Kaldi is installed (v.gr., */home/ubuntu/kaldi*).

[9]See the call to *nnet-latgen-faster*, in *decode_nnet2.sh.*

[10]See *uzh/score.sh.*

- Configure the **smp** parallel environment, if the experiment is going to be run on a cluster of servers [11].
- The tool to run parallel tasks must be specified in *scripts/cmd.sh*:
  - *uzh/run.pl*: used when parallelization is done in the local machine (for example, for debugging purposes).
  - *uzh/queue.pl*: parallelization in a cluster, with Sun Grid Engine as task scheduler.
  - *uzh/slurm.pl*: parallelization in a cluster, with Slurm.

4.2 Training

All the following variables are grouped in the "Configuration" section at the header of each script.

- train_AM.sh:
  - num_jobs: number of processes for parallelization. Set it to the minimum of the number of virtual CPU's in your cluster and the number of speakers in the training data.
  - use_gpu: "true" if your cluster includes machines with GPU. Otherwise, "false".
  - num_senones: target number of models for GMM (and, hence, also the number of outputs in the neural network). Each triphone model (v.gr., **/ x-p+y/**, where **p** is the central phone, and **x** and **y** are the left and right context, respectively) is split into a subset of models (senones) according to acoustic similarity.
  - num_gaussians: total number of Gaussians to distribute along the senones.
- *run_5d.sh* [12]:
  - use_gpu: default value for the GPU decision. It is overwritten by train_AM.sh.
  - num_jobs_nnet: number of parallel processes for neural network training. It should be set to the number of GPU's in the cluster of servers, but also taking into account that in this recipe this number is also closely related to the learning rate, and changes

[11] This can be easily done in Sun Grid Engine with the call *qconf -ap smp*, and adding the total number of CPU's available in the cluster to the field **pe_slots**.

[12] For further reference, see "Improving deep neural network acoustic models using generalized maxout networks": http://www.danielpovey.com/files/2014_icassp_dnn.pdf.

in it might also affect the performance of the system. Values between five and seven are suitable.

  - mix_up: dimension to which the output layer of the neural network is increased during training.
  - initial_learning_rate: initial value for the adaptive learning rate.
  - final_learning_rate: final value for the adaptive learning rate.
  - num_hidden_layers: number of hidden layers for the neural network.
  - pnorm_input_dim: number of neurons at the input of each p–norm component.
  - pnorm_output_dim: number of neurons at the output of each p–norm component (for dimension reduction).

4.3 Decoding

- decode_nnet.sh:
  - num_jobs: number of processes for parallel decoding. Set it to the number of virtual CPU's in the cluster.

5 Requirements

5.1 Software

- Operating system: Ubuntu 14 / 16 LTS. It should also work in RedHat / Centos 7, but it was not tested.
- Kaldi [13].
- ffmpeg [14].
- MIT Language Modeling Toolkit [15]: note that this tool is only needed for the script *archimob/simple_lm.sh*, which contains an example of how to create the language model for decoding.

[13]https://github.com/kaldi-asr/kaldi, commit 8cc5c8b32a49f8d963702c6be681dcf5a55eeb2e
[14]https://www.ffmpeg.org/. Also available as a package in Ubuntu 16.
[15]https://github.com/mitlm/mitlm

5.2 Hardware

Even though Kaldi can be run on a single machine, with or without a GPU, training and decoding times are largely decreased if a cluster of servers with GPU's is used.

As a minimal configuration, a cluster of five servers is recommended, having each of them a GPU, eight virtual CPU's, and 16 GB of memory. A hard drive of several hundreds of gigabytes should be mounted on the master and shared via NFS with the other nodes.

5.3 Parallelization

If the experiment is going to be run in a cluster of servers, the Kaldi recipe needs either Sun Grid Engine (SGE) or Slurm to be installed as task scheduler. In either case, the parallel environment **smp** should also be configured in advance, same as anonymous ssh among the nodes.

6 Reference results

In order to give a reference of the performance of the Kaldi based ASR framework on the Archimob corpus, a complete experiment was run, including acoustic models training, lingware compilation, and evaluation. Part of the available annotated Archimob programs were partitioned into two independent sets:

- Training set: programs for acoustic model training and lingware generation.
- Testing set: programs for decoding.

Table 8 collects the performance statistics of the evaluation, with the partitions shown in Table 9.

Total words	Errors	Substitutions	Insertions	Deletions	WER
22675	17052	12518 (55.21%)	980 (4.32%)	3554 (15.67%)	**75.20%**

Table 8: Performance statistics

6.1 Analysis of the results

The performance obtained in the reference experiment is clearly disappointing, but it is a direct consequence of the peculiarities of the Dieth method chosen for the Archimob transcriptions. Namely:

Set	Purpose	Files
Training	Acoustic model, language model	1008, 1044, 1048, 1055, 1063, 1138, 1143, 1147, 1188, 1189, 1195, 1205, 1209, 1224, 1228, 1235, 1240, 1248, 1255, 1259, 1295, 1300
Testing	Decoding reference	1207, 1270

Table 9: Reference sets

- Dieth is not a purely phonetic method and it also depends strongly on the subjective impression of the annotators and on the orthographic canonical form for Standard German [16].
- The Dieth transcriptions are very rich in the sense of representing the actual pronunciation of each word. Not only the phonetic realization is pursued, but also the length of the phones, as mentioned before.

Even though these characteristics are desirable for other purposes, in the current context of automatic speech recognition they bring a number of drawbacks:

- The absence of a standard written form for every word creates a lot of sparsity in the training data, even for the small domain of the Archimob documentaries. This affects heavily the quality of the language model, in which the several transcriptions of the same word are actually considered as completely distinct words. As an example, the perplexity of the language model trained in Section 6 on the independent testset is **285.801**, much higher than the typical values in this kind of application [17].
- The high percentage of out of vocabulary words in the evaluation: about **8%** of the words in the test were not covered by the decoding lexicon. Commercial systems are designed to have a much smaller number of out of vocabulary words, since they have a deep impact in performance [18]. As a side experiment, decoding was repeated with a language model that included all the available data (both the training and testing partitions from Table 9), and the Word Error Rate dropped down to **33%**.
- The pronunciations dictionary: the Dieth transcriptions are too rich for automatic speech recognition. For example, the number of different vowels is probably too large, and it should be reduced.

[16]For example, the length of nasals before a plosive, like in **suntig** vs. **sunntig**, or the length of the vowels, like **schoo** vs. **scho**.

[17]For limited domain applications, such as Archimob, the language model is designed to have a perplexity around **100**, or even close to **200** if the acoustic models are very good.

[18]Note that an out of vocabulary word usually affects as well the recognition of the surrounding words.

- The phone length information included in the transcriptions cannot be modeled properly with the current technology. The Hidden Markov Models used to represent each phonetic model are well known for not predicting duration accurately, and therefore distinctions between words like **gsii** and **gsi** are based mainly on the language model probabilities, and not on the acoustics.
- Besides the subjective component of all transcriptions, in Archimob there are also inconsistencies in how the guidelines were followed. For example, wrong transcriptions like **ggsii**.

Besides the constraints coming from the transcription methods, there are two other factors that affect the quality negatively:

- The limited amount of training data: only **32 hours** of annotated waveforms is too little to train acoustic models. Unless the application domain is very restricted, at least several hundreds of hours are needed.
- The specific handling of hesitations in the current framework: during training, hesitations are mapped to the speech general model; and during decoding, they are just deleted from the references, since they usually do not have semantic value. This decision causes that, more than likely, every test utterance with a hesitation is decoded with some error.

Table 10 shows a subset of the substitution errors in the evaluation. As seen, most of them would not really be considered errors if some kind of normalized written form were available.

Reference	Hypothesis	Number
gsii	gsi	99
dän	dänn	88
gsìì	gsi	62
ghaa	gha	60
gsii	ggsii	50
daas	das	39
daa	da	37
wäiss	waiss	30
si	sii	24
gsäit	ggsait	22
choo	cho	20
gsìì	ggsii	18
gsäit	gsait	18

Table 10: Example of substitution errors during the evaluation