Integration of an on-line Kaldi Speech Recogniser to the Alex Dialogue Systems Framework

Ondřej Plátek and Filip Jurčíček

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
{oplatek,jurcicek}@ufal.mff.cuni.cz
https://ufal.mff.cuni.cz

Abstract. This paper describes the integration of an on-line Kaldi speech recogniser into the Alex Dialogue Systems Framework (ADSF). As the Kaldi *OnlineLatgenRecogniser* is written in C++, we first developed a Python wrapper for the recogniser so that the ADSF, written in Python, could interface with it. Training scripts for acoustic and language modelling were developed and integrated into ADSF, and acoustic and language models were build. Finally, optimal recogniser parameters were determined and evaluated. The dialogue system Alex with the new speech recogniser is evaluated on Public Transport Information (PTI) domain.

Keywords: automatic speech recognition, Kaldi, Alex, dialogue systems

1 Introduction

The Alex Dialogue Systems Framework (ADSF) is used for development of our experimental Czech spoken dialogue system in the Public Transport Information (PTI) domain [8]. In PTI, the dialogue system provides information about all kinds of public transport in the Czech Republic, publicly available at a toll-free 800 899 998 number.

This paper describes and evaluates the integration of the Kaldi *OnlineLatgenRecogniser* in the ADSF. Important goal of ADSF is to process the speech incrementally. Incremental speech processing recognise the incoming speech while a user speaks and the Kaldi *OnlineLatgenRecogniser* implements on-line interface which allows incremental audio processing. As a result, the ASR output can be obtained with minimal latency.

In the next section, the Python interface of *OnlineLatgenRecogniser* is described and alternative ASR implementations are discussed. Integration of *PyOnlineLatgenRecogniser* into Alex is described in Section 3. Section 4 details the PTI domain, acoustic and language model training. The evaluation itself is presented in Section 5. Finally, Section 6 concludes this work.

2 Automatic Speech Recognition Implementations

There are various options for automatic speech recognition. Among the most popular are: OpenJulius and RWTH decoder.

P. Sojka et al. (Eds.): TSD 2014, LNAI 8655, pp. 603–610, 2014.

The OpenJulius decoder can be used with custom-built acoustic and language models and for on-line speech recognition [2]. However, OpenJulius suffers from software instability when producing lattices and confusion networks. Therefore, it is not suitable for practical use. The RWTH decoder is not free software and a license must be purchased for commercial applications [6].

In the following text, we present ASR implementations which are integrated in ADSF. As of now, Google cloud based speech recognition service and *OnlineLatgen-Recogniser* are available in ADSF.

2.1 Google ASR

The main advantage of Google's cloud-based ASR is that it can be used off-the-shelf without any additional modifications and is relatively fast. However, the Word Error Rate (WER) is rather high on the PTI domain. The Google ASR API is available at `https://www.google.com/speech-api/v1/recognize`[1].

2.2 PyOnlineLatgenRecogniser

The *PyOnlineLatgenRecogniser* is a thin Python wrapper of the Kaldi *OnlineLatgen-Recogniser*, which is implemented in C++ [12]. The *OnlineLatgenRecogniser* implements on-line interface to speech parametrisation, feature transformations, and Kaldi *LatticeFasterDecoder*. It supports state of the art acoustic models for non-speaker adaptive recognition. Namely, we are able to use MFCC speech parametrisation, $\Delta - \Delta\Delta$ or LDA+MLLT feature transformation, generative training and MMI or MPE discriminative training [7].

The *OnlineLatgenRecogniser* implements the following interface:

- *AudioIn* – queueing new audio for pre-processing,
- *Decode* – decoding a fixed number of audio frames,
- *PruneFinal* – preparing internal data structures for lattice extraction,
- *GetLattice* – extracting a word posterior lattice and returning log likelihood of processed audio,
- *Reset* – preparing the recogniser for a new utterance,

The minimalistic Python example in Listing 1.1 shows usage of the *PyOnlineLatgenRecogniser* and the decoding of a single utterance. The audio is passed to the recogniser in small chunks (line 4), so the decoding (line 5 and 8) can be performed as user speaks. When no more audio data is available a likelihood and a word posterior lattice is extracted from the recogniser (line 10).

The word posterior lattices are returned as instances of the OpenFST [3] class. The OpenFST implementation for Python is provided by *pyfst* library [13]. In the ADSF we implemented conversion of the word posterior lattices to an n-best list. The implementation is efficient since the OpenFST shortest path algorithm is used on small lattices.

[1] Its use is described in a blog at `http://mikepultz.com/2013/07/google-speech-api-full-duplex-php-version/`

Listing 1.1. The snippet implements incremental recognition with *PyOnlineLatgenRecogniser's* interface abstracting from anaudio source and parameters details.

```
d = PyGmmLatgenWrapper()
d.setup(argv)
while audio_to_process():
    d.audio_in(get_audio_chunks())
    dec_t = d.decode(max_frames=10)
    while dec_t > 0:
        decoded_frames += dec_t
        dec_t = d.decode(max_frames=10)
d.prune_final()
lik, lat = d.get_lattice()
```

3 Integration of PyOnlineLatgenRecogniser into ADSF

The integration of the Kaldi real-time recognizer into the Alex framework consisted of implementing the following features:

1. The *KaldiASR* as subclass of *ASRInterface*, so that the Alex's ASR component can use *PyOnlineLatgenRecogniser*.
2. The training scripts for acoustic and language models.
3. The scripts for evaluation and decoding graph creation so that various settings, language and acoustic models can be tested.

This section discuss the implementation of *KaldiASR* class which uses *PyOnlineLatgenRecogniser's* functionality. The acoustic model training and the evaluation are described in next sections.

The ASR component in the ADSF runs as a separate process, and speech recognition is triggered by Voice Activity Detection (VAD) decisions. The ADSF uses voice activity detection to split a stream of incoming audio signal into speech and silence segments. Once speech segments are identified, they are sent to an ASR component to be recognised.

The ADSF defines abstract interface for the ASR unit as illustrated in Listing 1.2. The two most important methods are *rec_in* and *hyp_out*.

Listing 1.2. ASRInterface

```
class ASRInterface(object):

    def rec_in(self, frame):

    def flush(self):

    def hyp_out(self):

    def rec_wav(self, pcm):
        self.rec_in(pcm)
        return self.hyp_out()
```

The *rec_wav* method in Listing 1.2 nicely illustrates how the two methods *rec_in* and *hyp_out* are used for decoding. The *rec_in* queues in audio input and decodes the audio using beam search [5]. The *hyp_out* method extract ASR hypothesis. Since the *rec_wav* method is used only for testing purposes, it sends all input audio to the speech recogniser

at once. However, the audio is passed in small chunks to *PyOnlineLatgenRecogniser* in real-time applications, so the *rec_in* method decodes the audio as a user speaks.

When VAD recognises the end of speech, the *hyp_out* method is called in order to extract ASR hypothesis. *PyOnlineLatgenRecogniser* extracts word posterior lattices where its probabilities are computed using the forward-backward algorithm. The resulting lattice is converted to an n-best list and returned to hyp_out method which is processed by spoken language understanding component.

The latency of the ASR unit depends on the time spent in *hyp_out* method. The *hyp_out* method itself spend most of the time on preparing a word lattice. The word lattice is extracted by Kaldi determinisation algorithm [5]. The *lattice-beam* affects a level of approximations. The higher value of the *lattice-beam* parameter the better determinised lattice is extracted with longer latency.

The hypothesis extraction takes a nearly constant amount of time for variable utterance length and fixed *lattice-beam*. We fixed the *lattice-beam* value so it takes around 60 ms on average.

The Alex dialogue system frequently handles several spoken requests immediately one after another. Nevertheless, at the end of each utterance the *hyp_out* method is called and the ASR hypothesis is extracted. Since the user already speaks when the lattice is extracted, *rec_in* cannot be called and the audio is buffered. Consequently, the *rec_in* should decode the buffered audio faster than the user speaks.

In extreme cases, the *flush* method may be used in order to throw away the buffered audio input and reset the decoding. Skipping some user requests is arguably a better strategy than baffling the user with responses to a request which was asked a long time ago.

4 Training Acoustic and Language Models for the PTI Domain

The *OnlineLatgenRecogniser* is evaluated on a corpus of audio data from the Public Transport Information (PTI) domain. In PTI, users can interact in Czech language with a telephone-based dialogue system to find public transport connections [8]. The PTI corpus consists of approximately 12,000 user utterances with a length varying between 0.4 s and 18 s with a median around 3 s. The data were divided into training, development, and test data sections where the corresponding data sizes were 9,496, 1,188, 1,188 respectively. For evaluation, a domain specific class-based language model with a vocabulary size of approximately 52,000 and 559,000 n-grams was estimated from the training data. Named entities e.g., cities or bus stops, in class-based language model are expanded before building a decoding graph which is used for beam search in *OnlineLatgenRecogniser*. The perplexity of the resulting language model evaluated on the development data is about 48.

Since the PTI acoustic data amounts to less then 5 hours, the acoustic training data was extended by additional 15 hours of telephone out-of-domain data from VYSTADIAL 2013 – Czech corpus [9]. The best acoustic models were obtained by BMMI discriminative training with LDA and MLLT feature transformations. A detailed description of the training procedure is given in [9].

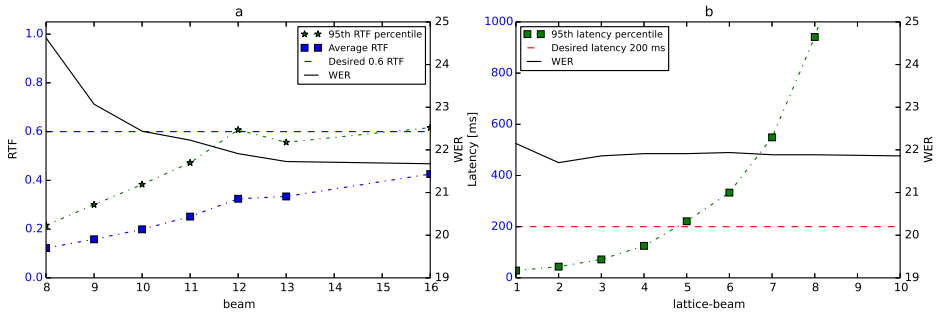

Fig. 1. The left graph (a) shows that WER decreases with increasing *beam* and the average RTF linearly grows with the beam. Setting the maximum number of active states to 2000 stops the growth of the 95th RTF percentile at 0.6, indicating that even in the worst case, we can guarantee an RTF around 0.6. The right graph (b) shows how latency grows in response to increasing *lattice-beam*.

5 Evaluation of *PyOnlineLatgenRecogniser* in ADSF

We focus on evaluating the speed of the *OnlineLatgenRecogniser* and its relationship to the accuracy of the decoder. We evaluate the following measures:

- Real Time Factor (RTF) of decoding – the ratio of the recognition time to the duration of the audio input,
- Latency – the delay between the end of utterance and the availability of the recognition results,
- Word Error Rate (WER).

The accuracy and speed of the *OnlineLatgenRecogniser* are controlled by the *max-active-states*, *beam*, and *lattice-beam* parameters [7]. *Max-active-states* limits the maximum number of active tokens during decoding. *Beam* is used during graph search to prune ASR hypotheses at the state level. *Lattice-beam* is used when producing word level lattices after the decoding is finished. It is crucial to tune these parameters optimally to obtain good results.

In general, one aims for a RTF smaller than 1.0. Moreover, it is useful in practice if the RTF is even smaller because other processes running on the machine can influence the amount of available computational resources. Therefore,we target in our setup the RTF of 0.6.

We used grid search on the test set to identify the optimal parameters. Figure 1 (a) shows the impact of *beam* on the WER and RTF measures. In this case, we set *max-active-states* to 2000 in order to limit the worst case RTF to 0.6. Based on the results shown in Figure 1 (a), we set *beam* to 13 for further experiments as this setting balances the WER. Figure 1 (b) shows the impact of *lattice-beam* on WER and latency when *beam* is fixed to 13. We set *lattice-beam* to 5 based on Figure 1 (b) to obtain the 95th latency percentile[2] of 200 ms, which is considered natural in a dialogue [1].

[2] For example, the 95th percentile is the value of a measure such that 95% of the data has the measure below that value.

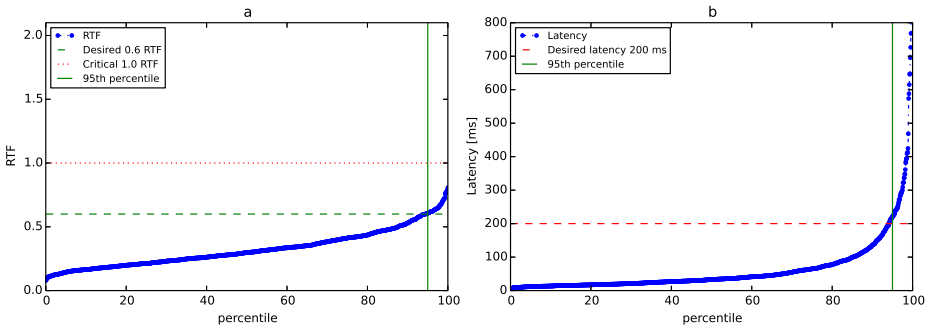

Fig. 2. The percentile graphs show RTF and Latency scores for test data for *max-active-sates*=2000, *beam*=13, *lattice-beam*=5. Note that 95 % of utterances were decoded with a latency lower than 200ms.

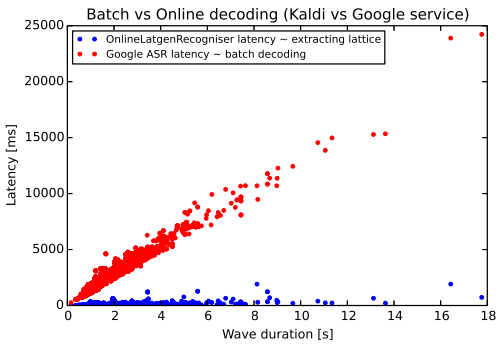

Fig. 3. Relation between latency and utterance length. Comparing on-line decoder (OnlineLatgenRecogniser) and batch decoding (Google cloud ASR service).

Lattice-beam does not affect WER, but larger *lattice-beam* improves the oracle WER of generated lattices [5]. Richer lattices may improve SLU performance.

Figure 2 shows the percentile graph of the RTF and latency measures over the test set. One can see from Figure 2 that 95% of test utterances is decoded with RTF under 0.6 and latency under 200 ms. The extreme values are typically caused by decoding long noisy utterances where uncertainty in decoding slows down the recogniser. Using this setting, the *OnlineLatgenRecogniser* decodes the test utterances with a WER of about 21%.

In addition, previously used ASR engine, Google ASR service, is evaluated. The Google ASR service decoded the test utterances from the PTI domain with a 95% latency percentile of 1900ms and reached WER about 48%. The high latency is presumably caused by the batch processing of audio data and network latency, and the high WER is likely caused by a mismatch between Google's acoustic and language models and the test data.

6 Conclusion

This work described the integration of the Kaldi *OnlineLatgenRecogniser* into the Alex Dialogue Systems Framework (ADSF). The *OnlineLatgenRecogniser* offers state-of-the-art accuracy as well as outstanding real-time and latency performance. The source code of the *OnlineLatgenRecogniser*, acoustic modelling scripts, and ADSF is available for download under the permissive Apache 2.0 license. The evaluation showed that the *OnlineLatgenRecogniser* significantly outperforms Google ASR service in terms of accuracy and latency in the PTI domain.

Overall, *OnlineLatgenRecogniser*, and the Kaldi toolkit offer an excellent alternative to Google ASR or OpenJulius when considered in the context of spoken dialogue systems.

Acknowledgments We would also like to thank Daniel Povey and Ondřej Dušek for their useful comments and discussions. We also thank the anonymous reviewers for their helpful comments and suggestions.

This research was funded by the Ministry of Education, Youth and Sports of the Czech Republic under the grant agreement LK11221, by the core research funding of Charles University in Prague. The language resources presented in this work are stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2010013).

References

1. G. Skantze, D. Schlangen: Incremental dialogue processing in a micro-domain, In Proc. ECACL, pp. 745–753, (2009)
2. L. Akinobu: Open-Source Large Vocabulary CSR Engine Julius, `http://julius.sourceforge.jp/en_index.php` (2014)
3. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, M. Mohri: OpenFst: A general and efficient weighted finite-state transducer library. In Proc. CIAA, pp. 11–23, (2007)
4. D. Huggins-Daines, M. Kumar, A. Chan, A. Black and M. Ravishankar and A. Rudnicky: Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices. In Proc. ICASSP, pp. I–I, (2006, December)
5. D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlicek, Y. Qian at al.: Generating exact lattices in the WFST framework. In Proc. ICASSP, pp. 4213–4216 (2012)
6. Rybach, David and Hahn, Stefan and Lehnen, Patrick and Nolden, David and Sundermeyer, Martin and Tüske, Zoltan and Wiesler, Siemon and Schlüter, Ralf and Ney, Hermann: The RASR-The RWTH Aachen University open source speech recognition toolkit. In Proc. IEEE Automatic Speech Recognition and Understanding Workshop (2011)
7. Povey, D., et.al: The Kaldi speech recognition toolkit. In Proc. ASRU, pp. 1–4, Hawaii, US (2011, December)
8. Public Transport Information System for Czech Republic, `https://ufal.mff.cuni.cz/alex-dialogue-systems-framework/ptics`
9. Korvas, M, Plátek, O, Dušek, O, Žilka, L, Jurčíček, F: Free English and Czech telephone speech corpus shared under the CC-BY-SA 3.0 license, In Proceedings of International Conference on Language Resources and Evaluation - to be published (2014)

10. The Kaldi ASR toolkit, http://sourceforge.net/projects/kaldi (2014)
11. The Alex Dialogue Systems Framework, https://github.com/UFAL-DSG/alex (2014)
12. The OnlineLatgenRecogniser, https://github.com/UFAL-DSG/pykaldi (2014)
13. The pyfst library: OpenFst in Python, http://pyfst.github.com/ (2014)