Free on-line speech recogniser based on Kaldi ASR toolkit producing word posterior lattices

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

Abstract

This paper presents an extension of the Kaldi automatic speech recognition toolkit to support on-line recognition. The resulting recogniser supports acoustic models trained using state-of-the-art acoustic modelling techniques. As the recogniser produces word posterior lattices, it is particularly useful in statistical dialogue systems, which try to exploit uncertainty in the recogniser's output. Our experiments show that the on-line recogniser performs significantly better in terms of latency when compared to a cloud-based recogniser.

1 Introduction

There are many choices of speech recognisers, but we find no alternative with both a permissive license and on-line recognition suitable for a spoken dialogue system. The Google speech recognition service[1] provides state-of-the-art quality for many tasks (Morbini et al., 2013) and may be used for free; however, the licensing conditions are not clear, adaptation of acoustic and language models to a task at hand is not possible and the service is not officially supported.

Another option is Nuance cloud based recognition[2]; however, again adjustments to the system are not possible. Moreover, it is a paid service.

When considering local ASR systems, we found no viable alternatives either. The HTK toolkit does not provide on-line large vocabulary decoders suitable for real-time decoding. OpenJulius can be used with custom-built acoustic and language models and for on-line decoding (Akinobu, 2014). 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 a free software and a license must be purchased for commercial applications (Rybach et al., 2011).

As a result, we implemented a lightweight modification of the *LatticeFasterDecoder* from the Kaldi toolkit and created an on-line recogniser with an interface that is suitable for statistical dialogue systems. The Kaldi toolkit as well as the on-line recogniser is distributed under the Apache 2.0 license[3]. Our on-line recogniser may use acoustic models trained using the state-of-the-art techniques, such as Linear Discriminant Analysis (LDA), Maximum Likelihood Linear Transform (MLLT), Boosted Maximum Mutual Information (BMMI), Minimum Phone Error (MPE). It produces word posterior lattices which can be easily converted into high quality n-best lists. The recogniser's speed and latency can be effectively controlled off-line by optimising a language model and during decoding by beam thresholds.

In the next section, the Kaldi recognition toolkit is briefly described. Section 3 describes the implementation of the *OnlineLatgenRecogniser*. Section 4 evaluates the accuracy and speed of the recogniser. Finally, Section 5 concludes this work.

2 The Kaldi toolkit

The Kaldi toolkit[4] is a speech recognition toolkit distributed under a free license (Povey et al., 2011). The toolkit is based on Finite State Transducers, implements state-of-the-art acoustic modelling techniques, is computationally efficient, and is already widely adapted among research groups.

[1]The API is available at https://www.google.com/speech-api/v1/recognize, and its use described in a blog post at http://mikepultz.com/2013/07/google-speech-api-full-duplex-php-version/.

[2]http://www.nuancemobiledeveloper.com/

[3]http://www.apache.org/licenses/LICENSE-2.0

[4]http://sourceforge.net/projects/kaldi

Its only major drawback was the lack of on-line recognition support. Therefore, it could not be used directly in applications such as spoken dialogue systems. Kaldi includes an on-line recognition application; however, hard-wired timeout exceptions, audio source fixed to a sound card, and a specialised 1-best decoder limit its use to demonstration of Kaldi recognition capabilities only.

3 OnlineLatgenRecogniser

The standard Kaldi interface between the components of the toolkit is based on a batch processing paradigm, where the components assume that the whole audio signal is available when recognition starts. However, when performing on-line recognition, one would like to take advantage of the fact that the signal appears in small chunks and can be processed incrementally. When properly implemented, this significantly reduces recogniser output latency.

3.1 C++ implementation

To achieve this, we implemented Kaldi's *DecodableInterface* supporting incremental speech pre-processing, which includes speech parameterisation, feature transformations, and likelihood estimation. In addition, we subclassed *LatticeFasterDecoder* and split the original batch processing interface.

The newly implemented *OnlineLatgenRecogniser* makes use of our incremental speech pre-processing and modified *LatticeFasterDecoder*. It 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 C++ example in Listing 1 shows a typical use of the *OnlineLatgenRecogniser* interface. When audio data becomes available, it is queued into the recogniser's buffer (line 11) and immediately decoded (lines 12-14). If the audio data is supplied in small enough chunks, the decoding of queued data is finished before new data arrives. When the recognition is finished, the recogniser prepares for lattice extraction (line 16). Line 20 shows how to obtain word posterior lattice as an OpenFST object. The *getAudio()* function represents a separate process supplying speech data. Please note that the recogniser's latency is mainly determined by the time spent in the *GetLattice* function.

Please note that we do not present here the functions controlling the input stream of audio chunks passed to the decoder and processing the output because these differ according to use case. An example of a nontrivial use case is in a dialogue system through a thin Python wrapper (see Section 3.2).

```
1  OnlineLatgenRecogniser rec;
2  rec.Setup(...);
3
4  size_t decoded_now = 0;
5  size_t max_decoded = 10;
6  char *audio_array = NULL;
7
8  while (recognitionOn())
9  {
10   size_t audio_len = getAudio(audio_array);
11   rec.AudioIn(audio_array, audio_len);
12   do {
13     decoded_now = rec.Decode(max_decoded);
14   } while(decoded_now > 0);
15 }
16 rec.PruneFinal();
17
18 double tot_lik;
19 fst::VectorFst<fst::LogArc> word_post_lat;
20 rec.GetLattice(&word_post_lat, &tot_lik);
21
22 rec.Reset();
```

Listing 1: Example of the decoder API

The source code of the *OnlineLatgenRecogniser* is available in Kaldi repository[5].

3.2 Python extension

In addition, we developed a Python extension exporting the *OnlineLatgenRecogniser* C++ interface. This can be used as an example of bringing Kaldi's on-line speech recognition functionality to higher-level programming languages. This Python extension is used in the Alex Dialogue Systems Framework (ADSF, 2014), an open-source language and domain independent framework for developing spoken dialogue systems. The *OnlineLatgenRecogniser* is deployed in an application which provides information about public

[5]https://sourceforge.net/p/kaldi/code/HEAD/tree/sandbox/oplatek2/src/dec-wrap/

transport and weather in the Czech republic and is available on a public toll-free telephone number.

4 Evaluation

4.1 Acoustic and language model training

The *OnlineLatgenRecogniser* is evaluated on a corpus of audio data from the Public Transport Information (PTI) domain. In PTI, users can interact in Czech with a telephone-based dialogue system to find public transport connections (UFAL-DSG, 2014). The PTI corpus consist of approximately 12k user utterances with a length varying between 0.4 s and 18 s with median around 3 s. The data were divided into training, development, and test data where the corresponding data sizes were 9496, 1188, 1188 respectively. For evaluation, a domain specific the class-based language model with a vocabulary size of approximately 52k and 559k 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.

Since the PTI acoustic data amounts to less then 5 hours, the acoustic training data was extended by an additional 15 hours of telephone out-of-domain data from the VYSTADIAL 2013 - Czech corpus (Korvas et al., 2014). The acoustic models were obtained by BMMI discriminative training with LDA and MLLT feature transformations. The scripts used to train the acoustic models are publicly available in ASDF (2014) as well as in Kaldi[6] and a detailed description of the training procedure is given in Korvas et al. (2014).

4.2 Experiments

We focus on evaluating the speed of the *OnlineLatgenRecogniser* and its relationship with the accuracy of the decoder, namely:

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

Accuracy and speed of the *OnlineLatgenRecogniser* are controlled by the *max-active-states*, *beam*, and *lattice-beam* parameters (Povey et al., 2011). *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 setting RTF smaller than 1.0. However, in practice, it is useful if the RTF is even smaller because other processes running on the machine can influence the amount of available computational resources. Therefore, we target the RTF of 0.6 in our setup.

We used grid search on the development set to identify optimal parameters. Figure 1 (a) shows the impact of the *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. Observing Figure 1 (a), we set *beam* to 13 as this setting balances the WER, 95th RTF percentile, and the average RTF. Figure 1 (b) shows the impact of the *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 of 200 ms, which is considered natural in a dialogue (Skantze and Schlangen, 2009). *Lattice-beam* does not affect WER, but larger *lattice-beam* improves the oracle WER of generated lattices (Povey et al., 2012).

Figure 2 shows the percentile graph of the RTF and latency measures over the development set. For example, the 95th percentile is the value of a measure such that 95% of the data has the measure below that value. One can see from Figure 2 that 95% of development 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, *OnlineLatgenRecogniser* decodes the utterances with a WER of about 21%.

Please note that *OnlineLatgenRecogniser* only extends the batch Kaldi decoder for incremental speech processing interface. It uses the same code as the batch Kaldi decoder to compute speech parametrisation, frame likelihoods, and state-level lattices. Therefore, the accuracy of *OnlineLatgenRecogniser* is equal to that of the batch Kaldi decoder given the same parameters.

[6]`http://sourceforge.net/p/kaldi/code/HEAD/tree/trunk/egs/vystadial_en/`

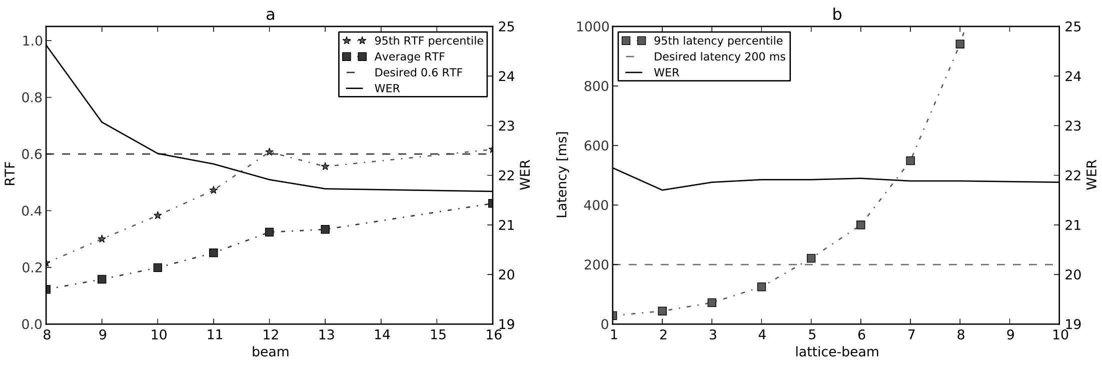

Figure 1: The left graph (a) shows that WER decreases with increasing *beam* and the average RTF linearly grows with the beam. Setting the maximum number of active states to 2000 stops the growth of the 95th RTF percentile at 0.6, indicating that even in the worst case, we can guarantee an RTF around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

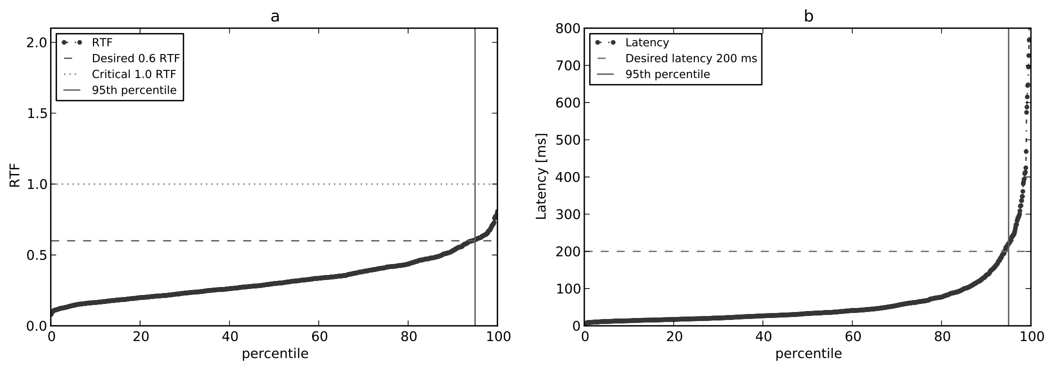

Figure 2: The percentile graphs show RTF and Latency scores for development data for *max-active-sates*=2000, *beam*=13, *lattice-beam*=5. Note that 95 % of utterances were decoded with the latency lower that 200ms.

In addition, we have also experimented with Google ASR service on the same domain. The Google ASR service decodes 95% of test utterances with latency under 1900 ms and WER is 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.

5 Conclusion

This work presented the *OnlineLatgenRecogniser*, an extension of the Kaldi automatic speech recognition toolkit. The *OnlineLatgenRecogniser* is distributed under the Apache 2.0 license, and therefore it is freely available for both research and commercial applications. The recogniser and its Python extension is stable and intensively used in a publicly available spoken dialogue system (UFAL-DSG, 2014). Thanks to the use of a standard Kaldi lattice decoder, the recogniser produces high quality word posterior lattices. The training scripts for the acoustic model and the *OnlineLatgenRecogniser* code are currently being integrated in the Kaldi toolkit. Future planned improvements include implementing more sophisticated speech parameterisation interface and feature transformations.

Acknowledgments

We would also like to thank Daniel Povey and Ondřej Dusěk 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

ADSF. 2014. The Alex Dialogue Systems Framework. `https://github.com/UFAL-DSG/alex`.

Lee Akinobu. 2014. Open-Source Large Vocabulary CSR Engine Julius. `http://julius.sourceforge.jp/en_index.php`.

Matěj Korvas, Ondřej Plátek, Ondřej Dušek, Lukáš Žilka, and Filip Jurčíček. 2014. Free English and Czech telephone speech corpus shared under the CC-BY-SA 3.0 license. In *Proceedings of the Eigth International Conference on Language Resources and Evaluation (LREC 2014)*.

Fabrizio Morbini, Kartik Audhkhasi, Kenji Sagae, Ron Artstein, Doan Can, Panayiotis G. Georgiou, Shrikanth S. Narayanan, Anton Leuski, and David Traum. 2013. Which ASR should I choose for my dialogue system? In *Proc. SIGDIAL*, August.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. 2011. The kaldi speech recognition toolkit. In *Proc. ASRU*, pages 1–4.

Daniel Povey, Mirko Hannemann, Gilles Boulianne, Lukas Burget, Arnab Ghoshal, Milos Janda, Martin Karafiát, Stefan Kombrink, Petr Motlicek, Yanmin Qian, et al. 2012. Generating exact lattices in the WFST framework. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4213–4216. IEEE.

David Rybach, Stefan Hahn, Patrick Lehnen, David Nolden, Martin Sundermeyer, Zoltan Tüske, Siemon Wiesler, Ralf Schlüter, and Hermann Ney. 2011. RASR-The RWTH Aachen University Open Source Speech Recognition Toolkit. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*.

Gabriel Skantze and David Schlangen. 2009. Incremental dialogue processing in a micro-domain. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 745–753. Association for Computational Linguistics.

UFAL-DSG. 2014. The Alex Dialogue Systems Framework - Public Transport Information. `https://github.com/UFAL-DSG/alex`.