The Kaldi OpenKWS System: Improving Low Resource Keyword Search

Jan Trmal\textsuperscript{1,2}, Matthew Wiesner\textsuperscript{1}, Vijayaditya Peddinti\textsuperscript{1,21}, Xiaohui Zhang\textsuperscript{1}, Pegah Ghahremani\textsuperscript{1}, Yiming Wang\textsuperscript{1}, Vimal Manohar\textsuperscript{1,2}, Hainan Xu\textsuperscript{1}, Daniel Povey\textsuperscript{1,2}, Sanjeev Khudanpur\textsuperscript{1,2}

\textsuperscript{1}Center for Language and Speech Processing, Johns Hopkins University, USA
\textsuperscript{2}Human Language Technology Center of Excellence, Johns Hopkins University, USA

\{trmal,wiesner,vijay.p,xiaohui,pgahre1,ywang169,vmanohal,hxu31,dpoveyl,khudanpur\}@jhu.edu

Abstract

The IARPA BABEL program has stimulated worldwide research in keyword search technology for low resource languages, and the NIST OpenKWS evaluations are the de facto benchmark test for such capabilities. The 2016 OpenKWS evaluation featured Georgian speech, and had 10 participants from across the world. This paper describes the Kaldi system developed to assist IARPA in creating a competitive baseline against which participants were evaluated, and to provide a truly open source system to all participants to support their research. This system handily met the BABEL program goals of 0.60 ATWV and 50\% WER, achieving 0.70 ATWV and 38\% WER with a single ASR system, i.e. without ASR system combination. All except one OpenKWS participant used Kaldi components in their submissions, typically in conjunction with system combination. This paper therefore complements all other OpenKWS-based papers.

Index Terms: speech recognition, keyword search, spoken term detection, IARPA Babel, OpenKWS

1. Introduction

This paper discusses a keyword search (KWS) system developed for the IARPA Babel program. The technical goal of the last phase of the program, named Option Period 3 or OP3, was to develop an automatic speech recognition (ASR) and KWS system in a new language given only 40 hours of transcribed speech, no phonetic lexicon, and some raw document. This system would still handle out of vocabulary (OOV) keywords, we introduced the following novel features to Kaldi.

They are now available through GitHub for developing ASR and KWS systems in other low resource languages

2. Novel Features of the Kaldi ASR System

Given the OP3 data and task constraints—having only 40 hours of transcribed telephone speech, no phonetic lexicon, some raw text scraped from web—and the expectation that the KWS system would still handle out of vocabulary (OOV) keywords, we introduced the following novel features to Kaldi.

They are now available through GitHub for developing ASR and KWS systems in other low resource languages.

2.1. Grapheme-Based Pronunciation Lexicons

In the absence of a (phonemic) pronunciation lexicon, one must naturally turn to the orthography of the language to create a sub-word representations of words for acoustic modeling. Now, the documentation provided as a part of the OP3 data does contain a mapping from graphemes to phonemes (G2P), but creating a pronunciation lexicon from it is often tedious and complex, especially for inexperienced users unfamiliar the languages and for languages with complex writing systems (G2P mappings).

We therefore investigated automatically inducing sub-word representations directly from the text, without relying on language-specific knowledge. We used the Unicode database \cite{4} to convert the graphic rendering of a word into a sub-word representation for acoustic modeling: specifically, we use as “phonemes” the standard character names in Normalization Form D (NDF, Canonical Decomposition form). In this form, the characters (i.e. graphemes) are normalized to their canonical form, and multiple combining characters are arranged in a specific order. Our method thus extends the approach of \cite{3} by providing a means to handle certain non-segmental writing systems. It also differs in the mapping derived from the Unicode descriptions as noted below in Section 2.1.1.

Now, writing systems used today may be grouped into four major families: (i) alphabets, (ii) abjads, (iii) syllabaries and abugidas, and (iv) purely logographic writing systems. We handle them as follows.

Alphabets are writing systems in which the graphemes generally represent phonemes, including vowels and consonants. Examples of alphabets present in the Babel corpora are the Latin, Cyrillic, and Georgian (Mkhedruli) scripts. Character names retrieved from Unicode have names such as “GEORGIAN CAPITAL LETTER MAN”, and “LATIN CAPITAL LETTER O WITH CIRCUMFLEX AND HOOK ABOVE”. After NFD normalization, the latter character, for example, will be mapped into a sequence of three units: “LATIN CAPITAL LETTER O”, “COMBINING CIRCUMFLEX ACCENT” and “COMBINING HOOK ABOVE”.

This standard form allows us to parse composite
graphemes, and generate their estimated “phonetic” representations. It is important to realize that accents (combining characters) have language-specific functions. For example, accents can indicate stress (Catalan, Italian), length (Czech), or tone (Vietnamese) of a phone; they may indicate a modified pronunciation; and in some cases they do not affect pronunciation. **Abjads** also exhibit related graphemic and phonemic representations, but differ from alphabets in that graphemes in abjads only encode consonants and lengthened vowels. All other vowels are rarely (or not) written. As a result, extracting the correct pronunciation of a word or a sequence of words requires additional semantic and syntactic knowledge.

Currently, the only commonly used abjads are variations of the Arabic and Hebrew scripts. The only abjad present in the Babel corpus is Pashto. An example of such a grapheme is “ARABIC LETTER ALEF + ARABIC MADDAH ABOVE”. Composite characters are again decomposed into a standard (base) character, followed by combining characters.

As was the case for alphabets, the combining characters in abjads encode phonetic phenomena including emphasis, lengthening, glottal stops (Arabic), lenition (Hebrew) and pronounced case endings (Arabic). Most importantly, in almost all languages written with abjads (Arabic, Hebrew, Farsi, Pashto), the omitted vowels are occasionally written as diacritic markings, especially in pedagogical and religious contexts.

As in alphabets, we must determine the phonetic importance of diacritics, but the lack of vowels imposes additional challenges; the acoustic model we build for each normalized character will, in reality, need to represent the consonant from which it was derived as well as any surrounding vowel(s).

**Syllabaries** are writing systems in which each character represents a syllable. Examples of such are Cherokee and Katakana.

**Abugidas** are like syllabaries, except that the grapheme representing a syllable can usually be decomposed into a consonant and a vowel part. A single Tamil glyph for example, is represented as a sequence of two code-points: “TAMIL LETTER KA + TAMIL VOWEL SIGN I.”

Note that whether a certain script with these traits is represented as a syllabary or an abugida in Unicode is not based exclusively on its linguistic category—the historical organization of the character set, and overall compatibility issues are taken into account during the standardization process.

Amharic, for instance, is an abugida that is effectively treated as a syllabary, i.e. “ETHIOPIAN SYLLABLE SO” is an atomic character, and NFD will not decompose it into its linguistically obvious components “ETHIOPIAN LETTER S” and “ETHIOPIAN VOWEL SIGN O.” We, however, parse the Unicode description of the Unicode character and still generate the correct number of acoustic units.

For purely logographic writing systems such as the Han script, hieroglyphs or Mayan glyphs, Unicode character descriptions do not provide enough phonetic information. In such cases, the approach we developed does not apply, and alternative solutions must be devised.

### 2.1.1. Unicode-driven Induction of Pronunciation Lexicons

Observe from the discussion above that for each of the 4 types of writing systems, parsing the Unicode descriptions of the NFD decomposition of characters transforms the problem of finding the phoneme-like pronunciation of a word into the problem of deciding which combining characters are phonetically relevant. We ignore for now the issue of orthographic digraphs. Firstly, the Babel languages do not exhibit this property. Furthermore, prior work [5] shows that such cases can be partially accounted for by using context-sensitive (triphones) models.

For each combination of base grapheme + combining-characters seen sufficiently often in the acoustic training data, create a separate acoustic-phonetic unit. For all infrequent combinations of grapheme + combining-characters, use the base grapheme as an acoustic unit, and the combining characters as tags. When building phonetic decision-trees for triphone clustering, permit questions about each tag, so as to decide in a data-driven manner whether base characters with different tags merit separate acoustic models.

The intuition behind this approach is that redundant acoustic units are acceptable provided we have sufficient data with which to train their models. Otherwise, we let the phonetic decision-tree building algorithm determine the phonetic relevance of the combining character. Since the definition of “infrequent” is somewhat arbitrary in the HMM/GMM modeling framework, we define the threshold for rare graphemes relative to other graphemes in the language. For our experiments we use the data-driven approach on the rarest 10% of the graphemes in the acoustic training data.

Table 1 compares ASR/KWS systems for the OP3 languages: those that use graphemic lexicons built as described here versus hand-crafted phonetic lexicons. Despite the simplicity of our strategy, which does not address digraphs, diphthongs, or missing vowels in abjads, performance suffers only modestly.

<table>
<thead>
<tr>
<th>Babel OP3 Language</th>
<th>WER</th>
<th>ATWV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Phonetic</strong></td>
<td><strong>Graphemic</strong></td>
<td><strong>Phonetic</strong></td>
</tr>
<tr>
<td>Pashto</td>
<td>46.2%</td>
<td>47.2%</td>
</tr>
<tr>
<td>Guaraní</td>
<td>44.9%</td>
<td>45.8%</td>
</tr>
<tr>
<td>Igbo</td>
<td>57.1%</td>
<td>55.8%</td>
</tr>
<tr>
<td>Amháric</td>
<td>43.0%</td>
<td>43.5%</td>
</tr>
<tr>
<td>Mongolíu</td>
<td>48.2%</td>
<td>48.7%</td>
</tr>
<tr>
<td>Javaníse</td>
<td>53.8%</td>
<td>53.5%</td>
</tr>
<tr>
<td>Dholoá</td>
<td>39.4%</td>
<td>38.6%</td>
</tr>
</tbody>
</table>

2.2. Deep Neural Network-based ASR Systems

2.2.1. Cross-Entropy trained BLSTM systems

We train a hybrid hidden Markov model, deep neural network (HMM-DNN) based ASR system in the Kaldi Nnet3 framework ([7, 8]). The DNN is composed of 3 layers of bi-directional long-short-term memory (BLSTM) layers (see [9]), and is trained using the cross-entropy (CE) objective function.

The error gradients w.r.t. to the parameters of the system are computed using context-sensitive chunk back-propagation through time (chunk BPTT) similar to the approach suggested in [10]. Randomly shuffled 20-frame chunks, with context of 40 frames on both the left and right sides, were used to train the network. BPTT was performed for just 20 time steps. To stabilize the BLSTM training, norm-based gradient clipping was used. Furthermore, the error-gradients w.r.t. sigmoid and tanh activations in the BLSTMs were clipped during BPTT if they crossed a threshold.

We used the speed perturbation data augmentation approach of [11] with coefficients 0.9 and 1.1, effectively tripling the amount of the audio data to 120 hours. High-resolution 40 dimensional MFCCs, plus 3 dimensional pitch features, spliced
Table 2: Performance (WER, ATWV) comparison of different types of acoustic models. Graphemic systems for Babel OP3 languages. Notice the opposite trends for WER and ATWV.

<table>
<thead>
<tr>
<th>Language</th>
<th>LF-MMI TDNN WER</th>
<th>ATWV</th>
<th>CE BLSTM WER</th>
<th>ATWV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pashto</td>
<td>48.2% 0.377</td>
<td>47.2% 0.338</td>
<td>38.6% 0.507</td>
<td></td>
</tr>
<tr>
<td>Guarani</td>
<td>47.2% 0.497</td>
<td>45.8% 0.464</td>
<td>48.7% 0.404</td>
<td></td>
</tr>
<tr>
<td>Igbo</td>
<td>59.4% 0.288</td>
<td>55.8% 0.284</td>
<td>55.3% 0.384</td>
<td></td>
</tr>
<tr>
<td>Amharic</td>
<td>44.8% 0.506</td>
<td>43.5% 0.475</td>
<td>50.7% 0.448</td>
<td></td>
</tr>
<tr>
<td>Mongolian</td>
<td>50.7% 0.448</td>
<td>48.7% 0.404</td>
<td>53.3% 0.384</td>
<td></td>
</tr>
<tr>
<td>Javanese</td>
<td>49.9% 0.545</td>
<td>38.6% 0.507</td>
<td>40.9% 0.545</td>
<td></td>
</tr>
</tbody>
</table>

Over a context of 5 frames \((t - 2, t + 2)\) were used as the input to the neural network for every time step.

The i-vectors, when provided as an input to a neural network, have been shown to perform speaker adaptation [12]. Furthermore, they have also been shown to be useful for environment adaptation. Hence, in the proposed system, we use i-vectors as an additional input to the network at the initial layer for speaker and environment adaptation.

In addition to these, for the final Georgian system, multilingual bottleneck features (ML-BNFs) were also appended to each frame. The ML-BNF extractor is described in Section 5.

Indicative performance of this system is shown in Table 2 (cf CE-BLSTM) for the OP3 development languages.

2.2.2. Chain LF-MMI systems

Purely sequence trained TDNNs using lattice-free maximum mutual information (LF-MMI) criterion [13], which usually outperforms CE-trained systems, did not do so out of the box. We weren’t able to make the training stable enough before the OpenKWS submission deadline, so our submission system was the CE-BLSTM system described above. Subsequently, clean up of the training data was found to be necessary, as was the case in other small-data tasks, as noted in [13]. The data clean up was done using constrained decoding of the training utterances, as described in [14]. This removed around 10% (i.e. 4 hours) of the training data. Indicative performance of this system is shown in Table 2 (cf LF-MMI) for the OP3 languages.

Note from Table 2 that though the WERs of LF-MMI systems are worse than the CE-BLSTM systems, they consistently yielded better ATWVs. The likely reason is that the LF-MMI lattices have greater depth—despite the fact the pruning beams were the same for both systems. Due to stability issues we initially encountered in training the LF-MMI system, we submitted the CE-BLSTM system for OpenKWS. Subsequently, however, it has become clear that the LF-MMI TDNN system fares better; it has several additional advantages over the CE-BLSTM systems including greater training & decoding speed, comparable WER, better ATWV, etc.

3. The Kaldi Keyword Search System

The underlying KWS capability in Kaldi is word-based: word-level ASR lattices are converted into an inverted index via a factor transducer, as proposed in [15]. The KWS pipeline is conceptually based on the Babel program Base-Period (BP) and Babel Program Option Period 1 (OP1) pipeline described in [16], but was completely redesigned for OP3, with the aims of (i) enhancing OOV search, and (ii) improving score normalization.

3.1. Score Normalization: Keyword Specific Thresholding

For score normalization, we use the keyword specific threshold technique (KST) from [17]; see also [18] for a detailed discussion of other normalization techniques. The \(N_{\text{KW}}\)-term parameter needed for KST is determined experimentally by sweeping the range 1.0 to 2.0, and choosing the one that maximizes ATWV on the dev speech+keywords. The value thus chosen is used on the evaluation speech+keywords.

3.2. Fusing Multiple Methods for OOV Search

The new KWS pipeline uses multiple methods to search the ASR output for OOV keywords, as described below. Each putative keyword hit (token) is assigned a score that is a weighted combination of its scores in the hit-lists of the individual methods. The combination weights are initialized to reflect the maximum term weighted value of the individual hit-list, and then optimized on the dev speech+keywords using Powell’s method, as done in [19], to maximize ATWV.

This KWS pipeline is directly applicable for KWS system combination, if one were to build multiple ASR+KWS systems.

4. OOV Keyword Search Strategies

The new Kaldi KWS pipeline combines three approaches for handling OOV keywords.

- **Proxy search:** An OOV keyword is converted into sequences of similar sounding in-vocabulary words, and the word-index is searched. See [20] for details.
- **Phonetic search:** An OOV keyword is converted into phonetic sequences, and a phone-level index, derived from an ASR phone lattice, is searched.
- **Syllable/morpheme search:** An OOV keyword is split into a sequence of syllables or morphemes, and a syllable- or morpheme-level index (respectively) is searched. Syllabic decomposition relies on a handcrafted lexicon; for obtaining a decomposition of words into morphemes, we used Morphessor 2.0 [21]. In either case, we use the proxy search strategy described above to find putative matches of these subword sequences in the appropriate ASR subword lattices.

There are two ways to obtain subword-level lattices: one is to create the decoding graph in terms of these subword units, and run the decoding directly. The other is to convert the word-level ASR lattices into subword-level lattices using a finite state word-to-subword transducer (cf [22]). Despite the fact that direct decoding generally yields better KWS performance, we used the transduction method, which results in a straightforward pipeline that minimizes multiple ASR decoding passes.

Note that though the original (word-level) lattices contain position-dependent (i.e. word-internal versus word-boundary) tags on phonemes, these boundary markers lose their meaning after conversion to morpheme-level or phoneme-level lattices. They are treated like general tags, and do not circumscribe the keyword match.

5. Multilingual Acoustic Model Training

5.1. Data Selection for Multilingual Training

Keeping the “practicality” objective of the system in mind, we didn’t want to pursue the approach of training the multilingual
BNF extractor training time should be part of the total system statistics. The extractor from as many languages and as much data as possible, as is sometimes practiced (e.g. [23])\(^1\). Instead we used a technique similar to those used in [24, 25] that entails selecting languages close to the OpenKWS language, Georgian. Specifically, we used the confusion matrix generated from a language identification system to select the 10 languages most confusable with Georgian. The language identification system is a neural network trained to classify 2-10 second utterances as one of the 25 Babel languages. It was based using a time-delay neural network (described in [6]) followed by a statistical pooling layer. The closest languages were selected by computing the average language class posterior over all frames from a given language.

5.2. Bottleneck-features extractor

We added the FullLP data from 10 additional languages identified as being closest to Georgian—Lithuanian, Mongolian, Turkish, Kazakh, Kurmanji, Pashto, Swahili, Tok Pisin, Igbo and Dholuo—to train a multi-lingual acoustic model. The model is trained within a HMM-TDNN hybrid system where the TDNN shares all layers, other than the final affine layer, among different languages. The final softmax (output) layer is divided into language-specific parts that classify sequences from their respective language. The TDNN has 6 layers where the fifth layer has a 42 dimensional bottleneck.

In each epoch of training, the TDNN is trained using mini-batches of data sampled from all 11 languages. The sampling of the mini-batches is based on the relative frequency of data from these languages.

The high-resolution MFCC (dimension of 40), pitch features (dimension of 3, see [26]) and 100-dimensional iVectors are used as input features for this multi-lingual acoustic model. The iVector extractor used for extracting the iVectors was trained using data from all 11 languages.

The outputs of the bottleneck layer are termed as multi-lingual bottleneck features (ML-BNFs). The improvements in WER from using these features are detailed in the Table 3.

5.3. Using external text

The web text provided as part of the language resources was used to enhance the language models. The web text was cleaned and used to generate a Georgian word list to the augment the vocabulary from 30000 words to 530000 words. Using this augmented vocabulary a 4-gram LM was trained on the in-domain training data (i.e. training data transcripts). A second 4-gram LM was trained on the web text. These two LMs were then interpolated. The interpolation weight was chosen to minimize perplexity on the development set.

The web text was used to enhance the language models. The web text was cleaned and used to generate a Georgian word list to the augment the vocabulary from 30000 words to 530000 words. Using this augmented vocabulary a 4-gram LM was trained on the in-domain training data (i.e. training data transcripts). A second 4-gram LM was trained on the web text. These two LMs were then interpolated. The interpolation weight was chosen to minimize perplexity on the development set.

During the evaluation run, the data was first decoded using the trigram LM with the full 530k vocabulary and the 4-gram

---

\(^1\)One could also argue that a multilingual system trained on many languages would be useful for many other languages and tasks, which would mean that the “amortized training time” is negligible, and that argument certainly can be valid. For our setup, however, we felt that the BNF extractor training time should be part of the total system statistics.

---

<table>
<thead>
<tr>
<th>Decoding LM</th>
<th>Rescoring LM</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram (30K vocab)</td>
<td>None</td>
<td>44.1%</td>
</tr>
<tr>
<td>3-gram + 500K vocab</td>
<td>None</td>
<td>42.9%</td>
</tr>
<tr>
<td>3-gram + 500K vocab</td>
<td>Interpolated 4-gram</td>
<td>42.2%</td>
</tr>
</tbody>
</table>

Table 3: Improvements in WER from different features.

<table>
<thead>
<tr>
<th>AM</th>
<th>Features</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLSTM</td>
<td>MFCC+iVector</td>
<td>45.9%</td>
</tr>
<tr>
<td>BLSTM</td>
<td>MFCC+iVector+pitch</td>
<td>45.0%</td>
</tr>
<tr>
<td>BLSTM</td>
<td>MFCC+iVector+pitch+ML-BNF</td>
<td>44.1%</td>
</tr>
</tbody>
</table>

Table 4: Influence of adding new words from the web-scraped data and rescoring using an interpolated LM (created from training text and cleaned web-scraped text)

Table 5: Final results on the dev set and official results on eval set. ATWV(kwlist) denotes development keyword list, ATWV(kwlist3) denotes the official evaluation keyword list.

<table>
<thead>
<tr>
<th>Eval Set</th>
<th>ATWV(kwlist)</th>
<th>ATWV(kwlist3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval set</td>
<td>37.5%</td>
<td>0.705</td>
</tr>
<tr>
<td>dev set</td>
<td>42.2%</td>
<td>0.619</td>
</tr>
</tbody>
</table>

interpolated LM was used to rescore the lattices.

5.4. Official OpenKWS Evaluation Results

The official NIST results are reported in the Table 5. They easily surpass the Babel program goals: 50% WER and 0.60 ATWV. While comparing these results with those of other OpenKWS participants, please keep in mind that these were obtained using a single-pass ASR system, and the search results were obtained by combining three different hit-lists (word+proxy, phonetic, morphemic), all created by finite state transduction from a single set of word-lattices.

6. Conclusions

We presented a simple yet high performing KWS system based on single-pass decoding with a BLSTM. Several design decisions are discussed, including their quantitative impact on overall performance of the ASR+KWS pipeline. This pipeline does not need any proprietary systems and is freely available. We hope that public release of the pipeline, along with this paper, will stimulate further research, especially in KWS for low-resource languages. The consensus from the Babel program is that some languages are much harder than others and significant further research is needed to understand the causes.

7. Acknowledgement of Babel Datasets

The following IARPA Babel language packs were used to train and evaluate ASR and KWS systems: Georgian release IARPA-babel404b-v1.0a, Dholuo release IARPA-babel403b-v1.0b, Japanese release IARPA-babel402b-v1.0b, Mongolian release IARPA-babel401b-v2.0b, Amharic release IARPA-babel307b-v1.0b, Igbo release IARPA-babel306b-v2.0c, Guarani release IARPA-babel305b-v1.0 and Pashto release IARPA-babel104b-v04b.Y.

In addition, the following language packs were used for multilingual training: Lithuanian release IARPA-babel304b-v1.0b, Turkish release IARPA-babel1105b-v0.4, Kazakh release IARPA-babel302b-v1.0a, Kurmanji release IARPA-babel1205b-v1.0a, Swahili release IARPA-babel202b-v1.0d and Tok Pisin release IARPA-babel207b-v1.0e.
8. References


