LOW RESOURCE HIGH ACCURACY KEYWORD SPOTTING

by

Guoguo Chen

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

January, 2016

Abstract

Keyword spotting (KWS) is a task to automatically detect keywords of interest in continuous speech, which has been an active research topic for over 40 years. Recently there is a rising demand for KWS techniques in resource constrained conditions. For example, as for the year of 2016, USC Shoah Foundation covers audio-visual testimonies from survivors and other witnesses of the Holocaust in 63 countries and 39 languages [1], and providing search capability for those testimonies requires substantial KWS technologies in low language resource conditions, as for most languages, resources for developing KWS systems are not as rich as that for English.

Despite the fact that KWS has been in the literature for a long time, KWS techniques in resource constrained conditions have not been researched extensively. In this dissertation, we improve KWS performance in two low resource conditions: low language resource condition where language specific data is inadequate, and low computation resource condition where KWS runs on computation constrained devices.

For low language resource KWS, we focus on applications for speech data mining, where large vocabulary continuous speech recognition (LVCSR)-based KWS tech-

ABSTRACT

niques are widely used. Keyword spotting for those applications are also known as keyword search (KWS) or spoken term detection (STD). A key issue for this type of KWS technique is the out-of-vocabulary (OOV) keyword problem. LVCSR-based KWS can only search for words that are defined in the LVCSR's lexicon, which is typically very small in a low language resource condition. To alleviate the OOV keyword problem, we propose a technique named "proxy keyword search" that enables us to search for OOV keywords with regular LVCSR-based KWS systems. We also develop a technique that expands LVCSR's lexicon automatically by adding hallucinated words, which increases keyword coverage and therefore improves KWS performance. Finally we explore the possibility of building LVCSR-based KWS systems with limited lexicon, or even without an expert pronunciation lexicon.

For low computation resource KWS, we focus on wake-word applications, which usually run on computation constrained devices such as mobile phones or tablets. We first develop a deep neural network (DNN)-based keyword spotter, which is lightweight and accurate enough that we are able to run it on devices continuously. This keyword spotter typically requires a pre-defined keyword, such as "Okay Google". We then propose a long short-term memory (LSTM)-based feature extractor for query-by-example KWS, which enables the users to define their own keywords.

Primary Advisor: Sanjeev P. Khudanpur

Secondary Advisor: Daniel Povey

To Lingling Tao, my beloved wife.

Acknowledgments

I am particularly grateful to my advisor, Dr. Sanjeev Khudanpur. It is my privilege and honor to have the opportunity to study in one of the best places for speech recognition, and to have him as my advisor. During my five years in the graduate school, he has always been supportive to both my research work and my personal life, and he has given me a lot of freedom exploring various research ideas as well as career opportunities. Nothing in this dissertation would have been possible without his guidance, encouragement and support.

I am extremely thankful to Dr. Daniel Povey, who co-advised me during the past few years. It was him who brought me to the Kaldi speech recognition community, and made me realize the importance of building practical speech recognition systems. He has always been responsive whenever I have doubts on research problems, and his dedication to research has also been incredibly inspiring throughout my graduate study.

I would like to thank Dr. Damianos Karakos for his help admitting both me and my wife to Johns Hopkins University, and for his guidance in the early stage of my

graduate study. He taught me the basics of speech recognition step by step patiently, which laid down the foundations of this dissertation work.

I am also very grateful to Dr. Carolina Parada from Google speech team, for giving me the opportunity to intern there during the summer of 2013 and 2014, and for all the fruitful discussions during the two internships. It was incredible experience developing techniques and products that serve millions of people everyday, and it was during these two internships that I figured out what I would like to do after graduate school.

I thank all my dissertation committee members: Dr. Sanjeev Khudanpur, Dr. Daniel Povey, Dr. Andreas G. Andreou, Dr. Najim Dehak and Dr. Hynek Hermansky. Thank you all for being on my dissertation committee, and for all the meaningful feedbacks.

I thank all the speech folks at CLSP: Chunxi Liu, David Snyder, Hainan Xu, Ke Li, Keith Levin, Minhua Wu, Pegah Ghahrmani, Sri Harish Mallidi, Xiaohui Zhang, Vijayaditya Peddinti, Vimal Manohar, Yiming Wang and others. Research would have been boring without your company. Special thanks to Keith Levin and Vijayaditya Peddinti, our everyday "tea time" had made my graduate life a lot more fun.

I thank my senior Chinese fellows at CLSP: Daguang Xu, Feipeng Li, Ming Sun, Puyang Xu, Shuang Huang, Xuchen Yao, Yuan Cao, Ziyuan Wang. The process of pursuing Ph.D. is a long journey, and I will remember our friendship forever.

ACKNOWLEDGMENTS

Thanks to all the other CLSP students, faculties and stuff. You together have made a better CLSP, which I have always been proud of.

To my parents, my sister, and my maternal grandmother, thanks for making me who I am today. I would not have been able to finish this dissertation without your support.

To my beloved wife Lingling, thanks for the patience and sacrifice in the past five years. Together we worked many many long hours and late nights, and I am certain that I would not have achieved this far without your company. This dissertation is for you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Keyword spotting (KWS) is a task to automatically detect keywords of interest in continuous speech, which has been utilized in a broad variety of applications. A majority of such applications relate to audio indexing and speech data mining. For example, finding courses related to "speech recognition" from online lecture provider Coursera [2], where a large number of lectures are available in video format, instead of the traditional text format. Another portion of applications that are becoming increasingly important are the wake-word applications, where keyword spotting is performed to activate a device or initiate a voice interaction interface. For example, Google's voice search [3] features the keyword "Okay Google" where users can simply say "Okay Google" to initiate Google's voice search. Other popular applications include phone call routing, phone call monitoring, voice command, to name just a few. In this dissertation, we will explore various techniques to improve keyword

spotting performance for audio indexing and wake-word applications in low resource conditions.

1.1 Keyword spotting applications

Keyword spotting applications can be divided into two categories based on when the speech data and the keywords are available.

In the first category, keywords of interest are only known at the spotting stage, while the speech data is available beforehand. This type of keyword spotting technique is heavily used in applications related to speech data mining, and sometimes is also called keyword search (KWS) or spoken term detection (STD). Since keywords are unknown until the spotting stage, the focus of keyword spotting techniques for applications in this category has been on how to accurately and efficiently build inverted index from the speech data so that spotting can be performed in almost no time when keywords come.

In the second category, keywords of interest are pre-defined, while the speech data comes in real-time. Applications include voice command (e.g., telephone routing, commands on Google glass), wake-word (e.g., Okay Google, Hey Siri, Alexa), etc. Since the speech data comes in real-time, keyword spotting techniques for applications in this category usually do not rely on an inverted index of the speech data. The focus, therefore, has been on how to filter out the keywords from a snippet of the speech

data efficiently and accurately.

In this dissertation, we will discuss both categories in low resource conditions, and propose techniques to improve keyword spotting performance in low resource conditions for both categories.

1.2 Existing methods

Over the past 40 years, a lot of techniques have been proposed for keyword spotting. Below we try to summarize the literature into 3 categories.

1.2.1 Query-by-Example methods

Query-by-example (QbyE) methods are among the earliest attempts for keyword spotting [4], and the name query-by-example is self-explaining: examples of keywords, usually exist in audio format, are used to spot the keyword. QbyE methods typically consist of two steps: a *template representation* step where audio examples of the keyword are represented as templates in a certain format (posterior features, lattices, etc.), and a *template matching* step where templates are compared with the target speech utterances which have been processed in the same way. Over the past decades, research focus of QbyE has been primarily on novel template representation methods[5, 6, 7, 8, 9, 10, 11, 12, 13], while most of those methods use some variants of dynamic time warping (DTW) [14] for template matching [15, 6, 16]. QbyE methods

are usually applied to keyword spotting applications in the second category.

1.2.2 Keyword/Filler methods

The keyword/filler method sometimes is also known as acoustic keyword spotting [17], which models keyword and non-keyword (filler) explicitly in parallel using sub-word units. At spotting stage, target utterances are aligned with both the keyword model and the filler model, and decisions will be made based on the alignment cost. In [18, 19, 20, 21, 22], Hidden Markov Models (HMMs) are used to model both keywords and fillers, and spotting is done by searching through a decoding graph where keywords and fillers appear parallelly. The latter research in this category more or less follows this framework, with focus on filler word modeling [23, 24, 25, 26] and advanced scoring procedure [27, 28, 29, 30]. Discriminative training methods have also been explored in this context to directly optimize the keyword spotting performance [31, 32]. This type of keyword spotting method is often used in keyword spotting applications in the second category.

1.2.3 Large vocabulary continuous speech recognition methods

Large vocabulary continuous speech recognition (LVCSR) methods have been extensively used recently for applications such as audio indexing and speech data mining.

In this case, speech utterances are transcribed into words with LVCSR systems, which will be further indexed for efficient searching [33]. The 1-best hypothesis from LVCSR systems usually contains errors, which hurts the spotting performance. For better performance, confusion network [34, 35] or lattice [36, 37, 38] are commonly generated instead of the 1-best hypothesis for indexing [39, 40, 41, 42]. One drawback of the LVCSR-based keyword spotting methods is that the vocabulary of the LVCSR system is usually pre-defined, and if a keyword is out-of-vocabulary (OOV) with respect to the LVCSR system, there is no way the system can return anything for that keyword. Several techniques have been proposed to overcome the OOV problem, including subword modeling [43, 44], fuzzy search [45, 46], etc. Keyword spotting applications in the first category typically rely on this kind of keyword spotting technique.

1.3 Low resource keyword spotting

Despite the fact that keyword spotting has been an active research topic for over 40 years, the low resource aspect of keyword spotting has not been researched extensively. There are different types of low resource conditions depending on what kind of application the keyword spotting technique is applied to.

For the first category of keyword spotting applications where speech data is available beforehand, LVCSR-based keyword spotting methods are typically used due to their superior performance. LVCSR-based keyword spotting methods usually require

large amount of labeled training data for good spotting performance. However, there is a rising demand for developing LVCSR-based keyword spotting systems with limited language resource recently. For example, as for the year of 2016, USC Shoah Foundation covers audio-visual testimonies from survivors and other witnesses of the Holocaust in 63 countries and 39 languages [1], and providing searching capability for those testimonies requires substantial KWS technologies in low language resource conditions, as for most languages, speech recognition resources are not as rich as that for English. Recent research activities such as the Johns Hopkins University's 2012 summer workshop on zero resource speech technologies [47] and IARPA's Babel program [48] also propel the development of KWS technologies in this low language resource condition.

For the second category of keyword spotting applications where speech data is available at the spotting time, there also sees a recent focus on developing keyword spotting techniques for computation constrained devices [49, 50, 51]. For example, hands-free communication with mobile devices now becomes popular, where a KWS algorithm listens to the audio constantly and wakes up the device when pre-defined keywords are uttered (also known as wake-words). This requires the KWS algorithm to be running in a computation constrained condition, while remain highly accurate. Otherwise the KWS algorithm will quickly drain out the device's battery.

In this dissertation, we improve keyword spotting performance in low resource conditions for both categories of applications. For the first category, we focus on the

LVCSR-based keyword spotting methods, and propose various techniques to alleviate problems encountered in a low language resource condition. One of those problems is the well-known out-of-vocabulary (OOV) keyword problem: an LVCSR can only transcribe speech to the words that are defined in its lexicon, therefore if a keyword is OOV with respect to the LVCSR's original lexicon, there is no way that the LVCSR-based keyword spotting can find anything for that keyword. This out-of-vocabulary (OOV) keyword problem becomes extremely severe when the training language resource is limited. For example, in [52] the author report that OOV keyword rate hits as high as 50% when they train their LVCSR system on a 10 hours training set. For the second category, we focus on keyword spotting techniques in a low computation resource condition, and propose two new keyword spotting methods that are capable of running on computation constrained devices, and at the same time remain highly accurate.

1.4 Contributions

We make the following contributions in this dissertation through different chapters:

Chapter 2: Lexicon Value for Keyword Search

- Quantified the importance of pronunciation lexicons for LVCSR-based KWS systems in a low language resource condition.

- Discovered that lexicon augmentation had significantly greater impact on KWS performance than LVCSR.
- Discovered that utilizing the augmented lexicon in the KWS stage via approximate phonetic matching was much less effective than utilizing them in the LVCSR stage.

Chapter 3: Proxy Keyword Search

- Proposed a weighted finite-state transducer (WFST)-based framework for generating acoustically similar in-vocabulary (IV) word proxies for out-of-vocabulary (OOV) keywords. Intended for LVCSR-based KWS systems.
- Proposed a modified edit-distance transducer that allowed cost-inexpensive phone insertions and deletions at word boundaries, making proxy search more close to phonetic search.
- Evaluated the proposed method on 6 Babel languages and showed that proxy search constantly improved KWS performance by enabling OOV KWS in a regular LVCSR-based KWS system.

Chapter 4: Automatic Lexicon Expansion

- Proposed an automatic lexicon expansion technique that could expand the LVCSR's lexicon by adding a huge list of hallucinated words. Intended for LVCSR-based KWS systems.

- Evaluated the proposed method on 6 Babel languages and showed that the proposed lexicon expansion constantly improved the KWS performance because of the increased keyword coverage.

- Combined the proposed lexicon expansion with proxy keyword search and showed that it was beneficial to perform proxy keyword search for keywords that appeared in the expanded lexicon, but were not in the original lexicon.

Chapter 5: Limited Lexicon Keyword Search

- Evaluated LVCSR-based KWS systems with grapheme lexicons instead of expert pronunciation lexicons on 6 *alphabetic* Babel languages, and showed that there were small degradations for both speech recognition performance and KWS performance.

- Proposed an iterative framework that was capable of generating pronunciation lexicons for *logographic* languages given a small seed expert pronunciation lexicon.

- Evaluated on Babel language Cantonese and showed that by using a seed lexicon of 1000 words, we were able to achieve reasonably well speech recognition and KWS performance, when compared with an expert-crafted lexicon of 5000 words.

Chapter 6: Deep Neural Network Keyword Spotter

- Proposed a lightweight DNN keyword spotter that could detect pre-defined keywords, such as "Okay Google".
- Proposed algorithm was capable of running on computation constrained devices such as mobile phones, and was accurate enough for wake-word applications.
- Evaluated on a set of voice commands and showed that the proposed algorithm outperformed the keyword/filler model trained on the same data set.

Chapter 7: Long Short-Term Memory Feature Extractor

- Proposed an LSTM-based feature extractor that could embed audio segments of various length into a fixed length feature vector.
- Incorporated the proposed feature extractor to a QbyE KWS framework which enabled the keyword to be defined by just 3 keyword examples.
- Proposed a template averaging technique that could combine 3 keyword templates in the QbyE framework into just 1 template, which could reduce the detection computation by a factor of 3.
- Proposed algorithm was capable of running on computation constrained devices such as mobile phones, and was accurate enough for wake-word applications.

- Evaluated on a set of voice commands and showed that the proposed algorithm outperformed the posteriorgram + DTW model on the same data set.

1.5 Organization

Chapter 2, Chapter 3, Chapter 4 and Chapter 5 of this dissertation focus on KWS in low language resource conditions, for the keyword spotting applications in the first category. In Chapter 2, we quantify the value of pronunciation lexicon for LVCSR-based KWS systems. We show that OOV keyword problem is severe in a low language resource condition, and pronunciation lexicon plays an important role. In Chapter 3, we propose proxy keyword search that enables OOV keyword search using regular LVCSR-based KWS systems. In Chapter 4 we expand the LVCSR's lexicon by adding a huge list of hallucinated words. We demonstrate that the expanded lexicon increases keyword coverage, thus improving the KWS performance. In Chapter 5 we explore the possibility of building LVCSR-based KWS systems with limited, or even without expert pronunciation lexicon, and we propose an iterative framework to generate pronunciation lexicon for logographic languages, given a small seed expert lexicon.

Chapter 6 and Chapter 7 of this dissertation give emphasis to KWS techniques in a low computation resource condition, for keyword spotting applications in the second category. In Chapter 6 a lightweight DNN-based keyword spotter is proposed for

some pre-defined keywords, which is capable of running constantly on computation constrained devices such as mobile phones or tablets. It also has extremely high accuracy that it is qualified for wake-word applications. In Chapter 7, we propose an LSTM-based feature extractor that embeds audio segments of various length into a fixed length feature vector. A query-by-example KWS technique is further developed based on the LSTM feature extractor, which enables a keyword to be defined with just a few examples of the keyword.

1.6 Related publications

Chapter 2, Chapter 3, Chapter 5, Chapter 6 and Chapter 7 of this dissertation are based on the following publications. Chapter 4 is based on our unpublished work.

- Guoguo Chen, Sanjeev Khudanpur, Daniel Povey, Jan Trmal, David Yarowsky and Oguz Yilmaz. *Quantifying the value of pronunciation lexicons for keyword search in low resource languages.* ICASSP 2013.

- Guoguo Chen, Oguz Yilmaz, Jan Trmal, Daniel Povey and Sanjeev Khudanpur. *Using proxies for OOV keywords in the keyword search task.* ASRU 2013.

- Guoguo Chen, Carolina Parada and Georg Heigold. *Small-footprint keyword spotting using deep neural networks.* ICASSP 2014.

- Guoguo Chen, Carolina Parada and Tara N. Sainath. *Query-by-example key-*

word spotting using long short-term memory networks. ICASSP 2015.

- Guoguo Chen, Daniel Povey and Sanjeev Khudanpur. *Acoustic Data-driven pronunciation lexicon generation for logographic languages*. ICASSP 2016.

Chapter 2

Lexicon Value for Keyword Search

This chapter quantifies the value of pronunciation lexicons in large vocabulary continuous speech recognition (LVCSR) systems that support keyword search (KWS) in low resource languages. State-of-the-art LVCSR and KWS systems are developed for conversational telephone speech in Tagalog, and the baseline lexicon is augmented via three different grapheme-to-phoneme models that yield increasing coverage of a large Tagalog word list. It is demonstrated that while the increased lexical coverage — or reduced out-of-vocabulary (OOV) rate — leads to only modest (ca 1%-4%) improvements in word error rate, the concomitant improvements in actual term-weighted value are as much as 60%. It is also shown that incorporating the augmented lexicons into the LVCSR system before indexing speech is superior to using them *post facto*, e.g., for approximate phonetic matching of OOV keywords in pre-indexed lattices. These results underscore the disproportionate importance of automatic lexicon aug-

mentation for KWS in morphologically rich languages, and advocate for using them early in the LVCSR stage.

2.1 Related work

Thanks in part to the falling costs of storage and transmission, large volumes of speech such as oral history archives [1, 53] and online lectures [54, 55] are now easily accessible by large user populations via the world wide web. Unlike the text-web, however, searching speech using keywords continues to be a challenging problem. Manually transcribing the speech is often prohibitively expensive. Automatic keyword search (KWS) systems are able to address the problem in some cases, but not in others, because high performance KWS systems, in turn, rely on underlying large vocabulary continuous speech recognition (LVCSR) systems that are also expensive to develop. Good LVCSR systems utilize statistical acoustic and language models trained from large quantities of transcribed speech and "conversational" text in the search domain, and manually crafted *pronunciation lexicons* with good coverage of the collection.

We are interested in improving KWS performance in a low resource setting, i.e. where some resources are available to develop an LVCSR system — such as 10 hours of transcribed speech corresponding to about $100K$ words of transcribed text, and a pronunciation lexicon that covers the words in the training data — but accuracy

is sufficiently low that considerable improvement in KWS performance is necessary before the system is usable for searching a speech collection.

A fair amount of past research has been devoted to improving the acoustic models from un-transcribed speech [56, 57, 58, 59, 60], and to adapt language models trained from out-of-domain text to the task at hand. Such methods of improving the LVCSR performance, which subsequently improve KWS performance, are not a focus of this chapter. *We investigate the role of the pronunciation lexicon in KWS systems.*

The importance of pronunciation lexicons for LVCSR is not entirely underestimated. Several papers have addressed the problem of automatically generating pronunciations for out-of-vocabulary (OOV) words [61, 62] in order to improve LVCSR accuracy. But once a reasonably large lexicon is available, speech transcripts in most languages have a fairly small (1%-4%) OOV rate [63, 64]. Even when the OOV rate is reduced by lexicon augmentation, the former OOVs are often absent from the LVCSR transcript, due to poor triphone coverage or low LM probabilities. The impact of lexicon expansion on LVCSR accuracy, therefore, is usually very small.

Two notable exceptions to this conventional wisdom are (i) accuracy on infrequent, content-bearing words, which are more likely to be OOV, and (ii) accuracy in morphologically rich languages, e.g. Czech and Turkish. These exceptions come together in a detrimental fashion when developing KWS systems for a morphologically rich, low resource language such as Tagalog. *This is the setting in which we will quantify the impact of increasing lexical coverage on the performance of a KWS system.*

We assume a transcribed corpus of 10 hours of Tagalog conversational telephone speech, along with a $5.7K$ word pronunciation lexicon that covers all words seen in the transcripts, as our primary acoustic model (AM) training corpus. We assume that the language model (LM) training corpus is either just the transcripts ($74K$ words), or a larger corpus of $595K$ words. We first develop state-of-the-art LVCSR and KWS systems based on the given resources. We process and index a 10 hour search collection using the KWS system, and measure KWS performance using a set of 355 Tagalog keywords. We then explore three different methods for augmenting the $5.7K$ word lexicon to include additional words seen in the larger LM training corpus. The augmented lexicons are used to improve the KWS system in two different ways: reprocessing the speech with the larger lexicon, or using it during keyword search. The efficacy of the augmented lexicons is measured in terms of their impact on KWS performance, not just on LVCSR accuracy. We find that even though lexicon augmentation results in only modest reductions in word error rate (WER), the concomitant improvement in actual term-weighted value (ATWV) is often dramatically higher, particularly if the augmented lexicon is used in an early stage to generate the speech lattices used for indexing and search.

The remainder of the chapter is organized as follows. We describe our core LVCSR and KWS systems in Section 2.2, and the three lexicon augmentation methods in Section 2.3. The impact of augmented lexicons on LVCSR is reported in Section 2.4 and on KWS in Section 2.5. The main claims are reiterated in Section 2.6.

2.2 Baseline LVCSR and KWS systems

We conduct our investigations using the IARPA Babel Program Tagalog language [1]. We use a 10 hour subset of the 80 hours of conversational telephone speech in this corpus, the transcripts of this subset, and a pronunciation lexicon restricted to cover only these transcripts, to simulate low resource conditions. The Babel Tagalog collection also sets aside 10 hours of conversational telephone speech for development-testing. We use a 1.5 hour subset of this development-test set for LVCSR system tuning, e.g. acoustic and language model selection, and refer to it as the "dev" set. The entire 10 hour development-test set, which we refer to as the "eval" set, is used for KWS evaluation. Note that the LVCSR dev set is a part of the KWS eval set. We believe that any minor over-fitting that may result from this inclusion has negligible effect on KWS performance on the eval set. We use a list of 355 keywords (actually, key phrases) created by and shared among the Babel program participants.

2.2.1 Kaldi-based LVCSR system description

Our LVCSR system is built using the Kaldi tools [65]. We use standard PLP analysis to extract 13 dimensional acoustic features, and follow a typical maximum likelihood acoustic training recipe, beginning with a flat-start initialization of context-independent phonetic HMMs, and ending with speaker adaptive training (SAT) of state-clustered triphone HMMs with GMM output densities. This is followed by the

[1] Language collection release `IARPA-babel106b-v0.2f`.

training of a universal background model from speaker-transformed training data, which is then used to train a subspace Gaussian mixture model (SGMM) for the HMM emission probabilities. Finally, all the training speech is decoded using the SGMM system, and boosted maximum mutual information (BMMI) training of the SGMM parameters is performed.

Two different language models trained with the SRILM tools [66] are used in the experiments reported below: a trigram LM estimated from the transcripts of the 10 hour acoustic training data (ca $74K$ words), and a larger trigram LM estimated from the transcripts of the entire 80 hour Babel corpus (ca $595K$ words). The LMs are estimated separately for each decoding lexicon, so that their vocabulary, the probability of unseen words, etc. match decoding conditions.

The Kaldi decoder generates word lattices [38] for the eval data using the GMM+SAT, SGMM and SGMM+BMMI models. The decoding lexicon is varied systematically, from the low resource lexicon of $5.7K$ words ($8.9K$ pronunciations), through automatically augmented lexicons of three different sizes, to the full Babel reference lexicon of $23K$ words ($35K$ pronunciations). Decoding is performed with the small as well as the large LM to create contrastive sets of lattices. A matrix of word error rates is thus measured on the dev set for 3 AMs $\times$ 5 lexicons $\times$ 2 LMs.

2.2.2 OpenFst-based KWS system description

Lattices generated by the BMMI models are processed using the lattice indexing technique described in [41]. The lattices of all the utterances in the eval set are converted from individual finite-state transducers (FST) output by Kaldi to a single generalized factor transducer structure in which the start-time, end-time and lattice posterior probability of each word token in every lattice is stored as a 3-dimensional cost associated with that instance of the word. This factor transducer is, in essence, an inverted index of all word sequences seen in the collection of eval set lattices, and permits further manipulation easily using the Google OpenFst tools [67]. Interested readers are referred to [41] for details.

Given a keyword or phrase, one creates a simple finite-state machine that accepts the keyword/phrase and composes it with the factor transducer to obtain all occurrences of the keyword/phrase in the eval set lattices, along with the conversation ID, start-time and end-time and lattice posterior probability of each occurrence.

All putative instance of a keyword thus obtained are sorted according to their posterior probabilities. Furthermore, a YES/NO decision is assigned to each instance using the method proposed by [68]. Specifically, for each keyword, its expected count in the eval set is estimated by summing the posterior probabilities of all its putative hits, and a decision threshold that maximizes the expected term-weighted value is computed for each keyword. All keyword instances with posterior probabilities above this keyword-specific threshold are marked as YES.

Finally, the collection of all proposed keyword hits is evaluated against the ground truth using the NIST 2006 Spoken Term Detection evaluation protocol to compute the so called actual term-weighted value (ATWV).

2.2.3 Utilizing larger lexicons for KWS

A limitation of the word-based indexing scheme described above is that only words present in the LVCSR lexicon appear in the factor transducer. If a word in the keyword phrase is OOV relative to the lexicon, it will not be found by the FST composition step described above. And yet, the LVCSR vocabulary in low resource settings is often quite small, and the possibility that a keyword is OOV can be quite large. E.g. the Tagalog baseline vocabulary comprises only $5.7K$ words, and of the 355 phrasal keywords provided for KWS system development, 25% contain at least one OOV relative to this vocabulary.

However, if a large word list is provided, over and above the acoustic training transcripts, a number of techniques are available to generate pronunciations for them, and mitigate the possibility that a keyword is OOV.

A key goal of this chapter is to quantify the value of such lexicon augmentation to the KWS application, specifically to the improvement in ATWV from having a larger lexicon. Now, there are (at least) two ways in which one may utilize an augmented lexicon.

- If the augmented lexicon is available before the speech is processed/indexed,

one may incorporate it into the LVCSR stage. The lattices produced, and thus the factor transducer generated for search, will then contain the newly added words wherever there is sufficient evidence for them in the speech.

- An alternative to decoding all the speech with an augmented lexicon, which is sometimes inconvenient or impossible, is to use it during keyword search. Specifically, if K represents a finite-state acceptor for a keyword that is OOV relative to a baseline lexicon L_1, but in-vocabulary relative to an augmented lexicon L_2, where both L_1 and L_2 are finite-state transducers that accept phone sequences and output words, and if E is an "edit-distance" transducer that maps any phone sequence to any other phone sequence with a cost equal to their Levenshtein distance, then

$$K' = \text{Project}\left(\text{ShortestPath}\left((L_1^*)^{-1} \circ E \circ (L_2^*) \circ K\right)\right)$$

  represents the in-vocabulary keyword/phrase K' that is closest to K. One may use K' as a *proxy* for K to search lattices generated using L_1.

We investigate these two methods of utilizing an augmented lexicon L_2 for handling keywords that are OOV relative to the decoding lexicon L_1 of the low resource KWS system. We demonstrate that the first way is the preferred way to use an augmented lexicon.

2.3 Three lexicon expansion methods

We next describe the five lexicons L_1-L_5 used for generating lattices for indexation and search. L_1 ($5.7K$ words) and L_5 ($23K$ words) were manually created, while L_2-L_4 use different grapheme-to-phoneme (G2P) methods to cover progressively larger subsets of the $17K$ words in L_5 that are OOV relative to L_1.

L_1: The $5.7K$ *reference lexicon* contains $5.7K$ words ($8.9K$ pronunciations), and serves as our baseline lexicon.

L_2: The *Povey lexicon*, developed for automatically augmenting English lexicons in WSJ-like settings, which is able to cumulatively provide pronunciations for $6.6K$ of the $17K$ OOV words.

L_3: The *Yarowsky lexicon* was developed with an explicit notion of morphology. It is able to automatically generate pronunciations for $7.3K$ of the $17K$ OOV words.

L_4: The *Sequitur lexicon*, based on [69], was developed as a direct statistical grapheme-to-phoneme model. It is able to cover all $17K$ OOV words.

L_5: The $23K$ *reference lexicon* contains $23K$ words ($35K$ pronunciations), and is our most accurate lexicon.

The three G2P methods and their accuracies are summarized below. Note that the methods vary in their ability to cover the same set of $17K$ OOV words, *naturally*

yielding different-sized lexicons. But comparing different G2P methods *is not a goal* of this chapter, only the value of larger lexicons. Therefore, we do not trim the lexicons to be of equal size. Details of the methods are similarly not germane to the chapter, and are omitted.

2.3.1 The Povey lexicon

This lexicon augmentation method, originally designed for English, operates by splitting the OOV into potential prefixes and suffixes, finding the best possible match for the resulting stem-affix pair in the $5.7K$ reference lexicon, and stitching together a pronunciation from fragments of the matching lexicon entries. E.g., if the reference lexicon contains the entries `beat` ≡ `/b i t/`, `beatable` ≡ `/b i t 6 b l/` and `bear` ≡ `/b E r/`, and the word `bearable` is OOV, then it notes that `bearable` and `bear` differ in the suffix `-able`, just as `beatable` and `beat` do. Since the pronunciations of `beatable` and `beat` differ by the suffix `/6 b l/`, it generates `bearable` ≡ `/b E r 6 b l/`.

This lexicon covers $6.6K$ of the $17K$ OOVs (39%). Of the many pronunciations produced for each word, at least one exactly matches an *entire* reference pronunciation for $5.4K$ of those words (81%).

2.3.2 The Yarowsky lexicon

Our second method for lexicon augmentation is based on a novel model of synchronous `word` ≡ `/pronunciation/` transduction which utilizes all existing entries in a pronunciation lexicon to generate new candidate word/pronunciation pairs. For example, in Tagalog, the method learns that the prefix transduction `mag-` ↔`i-` of a word stem is accompanied — with probability 0.96 — by a synchronous prefix transduction `/m 6 g/-` ↔ `/? i/-` of its pronunciation. This facilitates generation of a pronunciation for an OOV word such as `magtuturo` from the pronunciation of the word `ituturo`, which is present in the $5.7K$ reference lexicon. Additional evidence for the pronunciation of `magtuturo` also obtains from the synchronous transduction of the word suffixes `-turo` ↔ `-ro` and the corresponding pronunciation suffixes `-/t u r o ?/` ↔ `-/r ?/`, and 76 other observed morphological phenomena, with a consensus probability of 0.98 for the correct pronunciation `magtuturo` ≡ `/m 6 g t u t u r o ?/`.

The algorithm requires as input only a reference lexicon, from which it infers a set of globally-optimized, performance-weighted set of *synchronous prefix and suffix transductions*. Post hoc inspection confirms that these transductions correspond to regular morphological affixations, allophonic substitutions, and variable-length prefixal and suffixal "rhymes".

This lexicon covers $7.3K$ of the $17K$ OOVs (44%). Of the many pronunciations produced for each word, at least one exactly matches an *entire* reference pronunciation for $6.4K$ of those words (88%).

2.3.3 The Sequitur lexicon

The third method of lexicon augmentation may be formalized as finding the most probable sequence of phonemes (under a source-channel model) given the sequence of graphemes. This method is implemented in the Sequitur G2P software, and is well described in [69]. We recapitulate it briefly for completeness.

The method uses so-called joint-multigram models, i.e. alignments between consecutive n graphemes ($n \geq 0$) and m phonemes ($m \geq 0$). Contrary to the usual practice, where these alignments are hand-crafted, Sequitur determines them automatically during the training phase from the input lexicon.

There are two hyper-parameters available to control the size and coverage of the augmented lexicon, namely V, the maximum number of pronunciation variants, and Q, the cumulative probability of all the generated pronunciations for a given OOV word. Multiple pronunciations are generated for a given OOV, in decreasing order of probability, until one of these targets is reached. To choose the best values of these two hyperparameters, we use the goodness criterion

$$\text{Goodness}(V, Q) = N_C - k|N_G - N_R|,$$

where N_G is the number of pronunciations variants generated at the given V and Q, N_C is the number of correct pronunciations among the N_G, and N_R is the number of reference pronunciations. The weight k controls over-generation. We set $k = 0.5$,

and find the optimal hyperparameters to be $V = 2$ and $Q = 0.8$.

This lexicon covers all $17K$ OOVs (100%). Of the many pronunciations produced for each word, at least one exactly matches an *entire* reference pronunciation for $12K$ of those words (75%).

2.4 LVCSR improvements

We perform LVCSR evaluations on the 1.5 hour dev set, and evaluate WERs for three sets of acoustic models, two language models and five different lexicons. They are reported in Table 2.1.

Lexicon	L_1	L_2	L_3	L_4	L_5
Words	$5.7K$	$12K$	$13K$	$23K$	$23K$
Pronunciations	$8.9K$	$21K$	$24K$	$39K$	$35K$
GMM+SAT acoustic models					
3gram LM $74K$	74.9	75.6	73.2	73.1	72.9
3gram LM $595K$	73.4	74.8	72.0	71.3	71.0
SGMM acoustic models					
3gram LM $74K$	71.6	70.4	70.1	69.4	68.8
3gram LM $595K$	69.3	68.7	68.2	67.4	66.4
SGMM+BMMI acoustic models					
3gram LM $74K$	71.1	70.1	69.8	68.9	68.5
3gram LM $595K$	68.9	68.2	67.4	67.0	66.2

Table 2.1: WER (%) for 5 lexicons $\times$ 3 AMs $\times$ 2 LMs.

Note that small but consistent reductions in WER result from augmenting either the lexicon or the LM alone, and reductions from lexicon and LM augmentation are additive. Note also that the gains persist even as the acoustic models improve,

demonstrating further complementarity of the three LVCSR components.

2.5 KWS Improvements

We perform KWS on the lattices produced by the BMMI acoustic models using the OpenFst-based technique summarized in Subsection 2.2.2 above.

Lexicon	L_1	L_2	L_3	L_4	L_5
Words	$5.7K$	$12K$	$13K$	$23K$	$23K$
Pronunciations	$8.9K$	$21K$	$24K$	$39K$	$35K$
SGMM+BMMI acoustic models × 3gram LM $74K$					
In-Voc keywords	0.253	0.271	0.269	0.276	0.273
OOV keywords	0.000	0.163	0.262	0.373	0.388
All keywords	0.191	0.244	0.267	0.300	0.301
SGMM+BMMI acoustic models × 3gram LM $595K$					
In-Voc keywords	0.277	0.287	0.304	0.320	0.320
OOV keywords	0.000	0.138	0.294	0.405	0.416
All keywords	0.209	0.250	0.302	0.341	0.343

Table 2.2: ATWV for in-vocabulary (268), OOV (87) and all (355) keywords, when the augmented lexicon is used in LVCSR.

To quantify the impact of using the augmented lexicons in the LVCSR stage of the KWS system, we index and search the lattices corresponding to all ten SGMM+BMMI systems in the bottom block of Table 2.1. For instance, the ATWV for the $5.7K$ reference lexicon and $74K$ word LM is obtained by using the lattices whose WER is 71.1%, ATWV for augmenting the lexicon via the Yarowsky method and using the $595K$ word LM is obtained by using the lattices whose WER is 67.4%, etc. The resulting ATWVs for 355 keywords are reported in Table 2.2, where we also provide

a breakdown of the ATWV between 87 OOV keywords relative to the $5.7K$ reference lexicon, and 268 keywords that are in-vocabulary.

Lexicon	L_1	L_2	L_3	L_4	L_5
Words	$5.7K$	$12K$	$13K$	$23K$	$23K$
Pronunciations	$8.9K$	$21K$	$24K$	$39K$	$35K$
SGMM+BMMI acoustic models × 3gram LM $74K$					
In-Voc keywords	0.253	0.253	0.253	0.253	0.253
OOV keywords	0.000	0.010	0.063	0.045	0.065
All keywords	0.191	0.194	0.206	0.202	0.207
SGMM+BMMI acoustic models × 3gram LM $595K$					
In-Voc keywords	0.277	0.277	0.277	0.277	0.277
OOV keywords	0.000	0.025	0.035	0.046	0.036
All keywords	0.209	0.215	0.217	0.220	0.218

Table 2.3: ATWV for in-vocabulary (268), OOV (87) and all (355) keywords, when the augmented lexicon is used in a pre-indexed KWS system to create proxy keywords K' for OOV keywords K.

Since OOV keywords have no chance to be found in word lattices generated using the $5.7K$ lexicon, the gains in Table 2.2 may appear to be trivial to explain. To rule out this trivial explanation, we also investigate utilizing the augmented lexicon to generate in-vocabulary *proxies* for OOV keywords, as described in Subsection 2.2.3. We construct the factor transducer from lattices generated using the $5.7K$ lexicon, but each time we encounter an OOV word in a keyword K, we use the method described in Subsection 2.2.3 to search the factor transducer using proxy in-vocabulary keywords K'. Note that we have noticed that when a proxy K' returns an unusually large number of YES'es for an OOV keyword K, they are predominantly false alarms, and hurt KWS performance, likely because K' is a frequent phrase. So we simply discard

all hits due to *high-yield* proxies, be they true or false alarms. The resulting ATWVs, again broken down between in-vocabulary and OOV keywords, are reported in Table 2.3.

2.6 Discussion and conclusion

Several interesting conclusions may be drawn from the three tables presented above.

Begin by comparing the last line in Table 2.1, where relative WER improvement is 4% (68.9% $\rightarrow$ 66.2%), with the last line in Table 2.2, where ATWV improves 64% (0.209 $\rightarrow$ 0.343). This demonstrates that lexicon augmentation has significantly greater impact on KWS performance than LVCSR.

Next, compare the first column for the BMMI acoustic models in Table 2.1 with the first column of the two "All keywords" lines in Table 2.2. The WER improves by 3% relative (71.1% $\rightarrow$ 68.9%) in Table 2.1, but the ATWV improves by only 9% (0.191 $\rightarrow$ 0.209) in Table 2.2. This demonstrates that not all WER reductions are equal: errors reduced by lexicon augmentation matter more for KWS than errors reduced by improving the LM.

Next, compare the two "All keywords" lines in Table 2.2, and note that ATWV improves due lexicon augmentation from 0.191 to 0.301 (58%) for the small LM, compared to 0.209 to 0.343 (64%) for the larger LM. This demonstrates that the KWS

improvements from lexicon augmentation are persistent even after LM improvements.

Next, compare the last lines in Tables 2.2 and 2.3, and note that ATWV improves in the former by 64%, but only 4% in the latter. This demonstrates that utilizing the augmented lexicon in the KWS stage via approximate phonetic matching (Table 2.3) is much less effective than utilizing them in the LVCSR stage (Table 2.2).

Finally, compare the ATWVs for in-vocabulary and OOV keywords in Table 2.2 to note that while much of the improvement from lexicon augmentation is on keywords that were previously OOV, there is significant (10%-15%) collateral improvement in detecting in-vocabulary keywords as well.

Chapter 3

Proxy Keyword Search

In this chapter we propose a simple but effective weighted finite-state transducer (WFST)-based framework for handling out-of-vocabulary (OOV) keywords in a speech search task. State-of-the-art large vocabulary continuous speech recognition (LVCSR) and keyword search (KWS) systems are developed for conversational telephone speech in languages such as *Tagalog*. Word-based and phone-based indices are generated for search from word lattices, the latter by using the LVCSR system's pronunciation lexicon. Pronunciations of OOV keywords are hypothesized via a standard grapheme-to-phoneme method, and OOV keywords are searched using their in-vocabulary (IV) *proxies* (word or phone sequences) generated from a WFST framework that permits incorporation of a phone confusion matrix. Empirical results when searching for the Babel/NIST evaluation keywords in the Babel 10 hour development-test speech collection show that (i) searching for word proxies in the

word index significantly outperforms searching for phonetic representations of OOV words in a phone index, and (ii) while phone confusion information yields minor improvement when searching a phone index, it can yield up to 40% improvement in terms of actual term-weighted value when searching a word index using word proxies.

3.1 Related work

Keyword search (KWS) for spoken documents has become more and more important nowadays as large speech repositories, such as oral history archives [1, 53] and online lectures [54, 2] are easily accessible. Searching for keywords in spoken documents, however, remains a challenging problem. Manual transcription of speech is usually prohibitively expensive, and given the amount of the spoken material available online, it is impractical to manually transcribe any nontrivial portion for search. Automatic KWS therefore is highly desirable. State-of-the-art KWS systems usually rely on the large vocabulary continuous speech recognition (LVCSR) systems [44, 52]. In such systems, lattices of speech segments in the search collection are generated first. An inverted index (postings list) is then created from the lattices. KWS now can be performed by searching for a given keyword via the inverted index. The KWS task should ideally be "open vocabulary". LVCSR systems however typically have a fixed vocabulary [70], making it impossible to search for out-of-vocabulary (OOV) keywords. One therefore uses as large an LVCSR vocabulary as feasible.

We are interested in KWS in low resource settings, where only limited resources are available to develop the LVCSR and KWS systems, e.g. 10 hours of transcribed speech corresponding to about 100K words of transcribed text, and a pronunciation lexicon that covers only the words in the training transcripts. Due to the low coverage of the pronunciation lexicon, OOV keyword rate in low resource settings can be extremely high, e.g., 50%, leaving a lot of room for improvement over word-based KWS methods.

One way to alleviate the OOV problem is to *preemptively* expand the LVCSR's lexicon. In other words, one adds automatically generated pronunciations of a large number of words to the LVCSR's pronunciation lexicon before lattice generation and indexing. In [52] it is shown that if one can anticipate the OOV keywords ahead of time, such a method leads to remarkable improvement in terms of KWS performance. However, advance knowledge of all possible keywords is rarely the typical operating condition for KWS systems.

Another way to handle OOV keywords is via sub-word units such as phones, syllables or word-fragments. A sub-word index can be created by either generating a sub-word lattice [40, 71, 72], or by converting a word lattice into a sub-word lattice with or without the use of an appropriate phone confusion matrix [73]. OOV keywords are then represented as sequences of sub-word units, and are searched against the sub-word index.

The idea of *query expansion* in text retrieval has also been adopted to tackle the OOV problem in speech search. A confusion matrix is used in [74, 75, 70, 76] to

generate alternative words or syllables for OOV keywords, which are used to search a word or syllable index instead of the original keywords. Unlike the idea of query expansion in the text retrieval field, wherein one augments a possibly unseen keyword with other keywords that are semantically similar (e.g. synonyms), speech search entails other keywords that are acoustically similar. We will therefore call them *proxy keywords* in this work.

Our work is most similar to that of [70], where proxy keywords are created using a phone confusion matrix. However, instead of searching for the proxy keywords in an n-best list generated by the LVCSR system, we introduce a weighted finite-state transducer (WFST)-based framework for directly matching multiple proxies against the entire index generated from word lattices. We further demonstrate that using word proxies with the word index usually outperforms searching for OOV keywords in a converted phone index via their corresponding phonetic representations.

3.2 Proxy keyword generation

Let K represent a finite-state acceptor for an OOV keyword, and L_2 a finite-state transducer for the pronunciation of the OOV keyword; e.g. pronunciations hypothesized via the joint-sequence model implemented in the *S*equitur software [69]. Let E be an edit-distance transducer that maps a phone sequence to any other phone sequence with costs estimated from a phone confusion matrix. Let L_1 denote the

pronunciation lexicon of the LVCSR system. Our WFST framework for generating a proxy keyword K' can be described as,

$$K' = \text{Project}\big(\text{ShortestPath}\big(K \circ L_2 \circ E \circ (L_1^*)^{-1}\big)\big) . \tag{3.1}$$

This framework is similar to the method proposed in [52], where Levenshtein distance was used as the cost in E. In this framework, phone confusion costs other than Levenshtein distance can easily be encoded in E, which we will explain in the next few sections. The WFST-based framework also makes it possible to carry the confusion cost to the final decision making stage. Finally, the WFST framework supports both phone and word proxies, as will be explained below.

3.2.1 OOV pronunciation generation

Pronunciations for OOV keywords are generated automatically using G2P software Sequitur [69]. The core idea of Sequitur is to align graphemes and phonemes in a set of training examples consist of "word + pronunciation" pairs to create the so-called "graphones," and then build a joint multigram model for graphone sequences. Our model is trained on the lexicon with which we build the LVCSR system. Pronunciations of an OOV keyword are "read off" the most likely graphone sequences corresponding to the grapheme sequence of the keyword.

3.2.2 Phone confusion matrix estimation

Phone confusion probabilities that can be encoded in E are obtained through a standard maximum likelihood estimation. Training data for these conditional probabilities are collected by aligning the reference phone sequences for some held-out speech to the phone sequences that correspond to its ASR hypothesis. The alignment maximizes the phone matching rate. A small subset of the development-test speech is utilized for this. Deletions and insertions are treated separately from phone substitutions, so that high rates of deletions and insertions will not adversely affect the estimation of E.

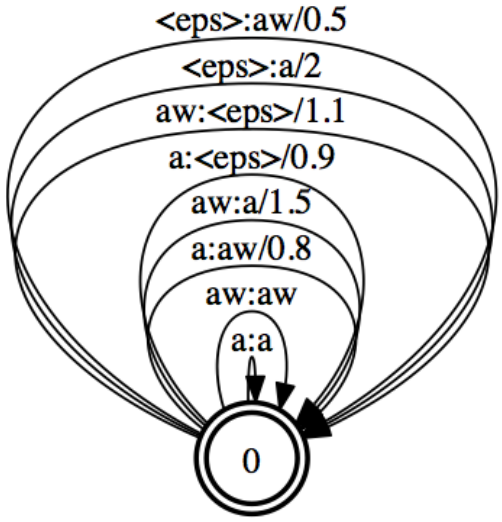

Figure 3.1: Example of a phone confusion encoding transducer E.

We encode the phone confusion statistics into the edit-distance transducer, as illustrated in Figure 3.1.

3.2.3 Phone proxy generation

Since languages usually have a closed phone set, a KWS system based on a phone index can be considered as open vocabulary. Furthermore, as claimed in [76], adding phone confusion to either the index or the phonetic representation $K \circ L_2$ of the (OOV) keyword can help improve KWS performance. Generating such phone proxies in our framework only requires a minor modification of Equation (3.1) as,

$$K'' = \text{Project}\left(\text{ShortestPath}\left(K \circ L_2 \circ E\right)\right). \tag{3.2}$$

A phone index, however, represents a much larger search space as the lexical constraint is removed. Therefore phone-based KWS system often suffers from higher false alarm rates.

3.2.4 Improved word proxy generation

Instead of searching for phone proxies K'' of an OOV keyword K in a phone index, we can generate word proxies K' and search directly against a word index. The obvious advantage is that we do not have to keep a separate index for OOV keywords — they share the index with the in-vocabulary (IV) keywords. Another potential advantage is that by imposing the lexical constraints on the permissible phone sequence via L_1^*, the search space is greatly limited, which should reduce false alarms.

However, there is a disadvantage of using Equation (3.1) to generate word proxies, as illustrated by the following made-up example in English. Suppose `balloon` ≡ `/B AH L UW N/` is an OOV word, and `some` ≡ `/S AX M/`, `Samba` ≡ `/S AA M B AH/` and `loon` ≡ `/L UW N/` are IV words. If the decoder encounters the sequence `some balloon` in the speech, it may hypothesize the word sequence `Samba loon` in that location. Now, generating K' = `Samba loon` from K = `baloon` by Equation (3.1) requires E to insert 3 phones `/S AA M/`, while generating K' = `loon` requires E to delete 2 phones `/B AH/`. So both appear to be poor proxies, even though a perfect phone sequence match for K exists in the lattice!

To address the problem described in the previous example, we modify the edit-distance transducer of Figure 3.1 to the transducer E' shown in Figure 3.2.

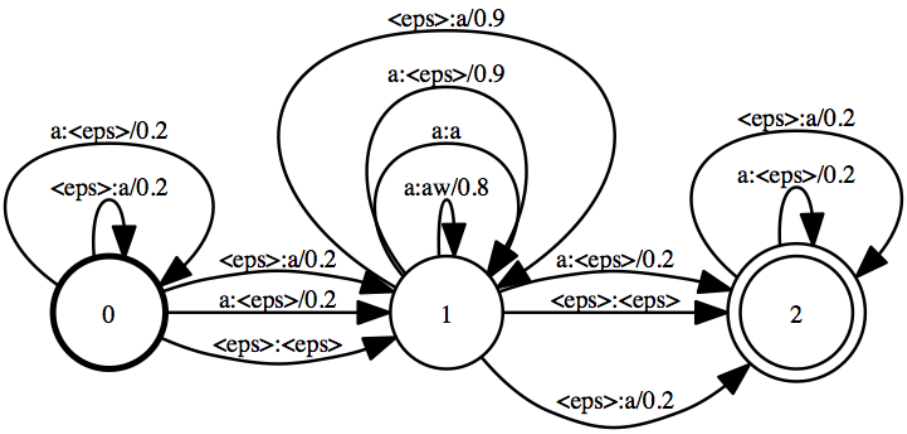

Figure 3.2: Modified phone confusion encoding transducer E', with freer edits permitted at the keyword-boundary.

In this modified transducer, insertions and deletions at the boundaries of the key-

word K are allowed at a lower cost, making search-by-word-proxy closer to phonetic search, while still retaining the lexical constraint on the phone sequence of K'. Word proxies are thus generated as,

$$K' = \text{Project}\big(\text{ShortestPath}\big(K \circ L_2 \circ E' \circ (L_1^*)^{-1}\big)\big). \tag{3.3}$$

Note that the *ShortestPath* algorithm can be computationally expensive on large WFSTs. We therefore resort to pruning in the *ShortestPath* algorithm as needed.

3.2.5 Language model impact

The proxy generation process in Equation(3.3) takes acoustic confusion into account, so that occurrences of K' in the index are good candidates for actual occurrences of K. But the word lattices from which the index is created contain both acoustic and language model scores. The language model score for K' is arguably not appropriate for comparing/sorting these occurrences. We will evaluate the impact of retaining/discarding this score in Section 3.5.

3.3 Keyword search pipeline

Our KWS pipeline is comprised of two major parts: lattice generation and lattice indexing. We build our pipeline using the open source toolkit Kaldi [65], which

includes all the necessary tools for building LVCSR systems, as well as OpenFst [67]-based keyword search systems. Code and scripts needed to reproduce the experiments in this work have been checked into the Kaldi repository and are publicly available.

3.3.1 LVCSR system

For acoustic modeling, standard PLP analysis is employed to extract 13 dimensional acoustic feature, and a maximum likelihood acoustic training recipe is followed to train speaker adaptive models. After that, two different systems are trained: a hybrid deep neural network (DNN) system and a subspace Gaussian mixture model (SGMM) system with boosted maximum mutual information (BMMI) training. For more details of those systems readers are referred to [77].

The language model is trained with the SRILM toolkit [66]. The n-gram order, smoothing method and count-cutoffs are selected to minimize the perplexity on some held-out subset. Typically, A n-gram order of 3 or 4, and smoothing techniques such as Good-Turing smoothing or Kneser-Ney smoothing are selected.

Word lattices generated by the DNN model or SGMM + MMI model are further converted to phone lattices where phonetic index is required; we do not directly perform phonetic decoding in order to get the phone lattices.

3.3.2 Indexing and search

We implemented the lattice indexing algorithm of [41] within the Kaldi toolkit. Specifically, the lattice of each utterance is converted into a finite-state acceptor with the posterior score, start-time and end-time for each word encoded as a 3-dimensional weight. An inverted index is then created from these individual acceptors, with paths to accept every possible word sequence in the original lattices. By applying standard WFST operations, one can work out the posterior score, start-time and end-time of each occurrence of a word sequence.

To carry out KWS, keywords (which include multiword key-*phrases*) are typically compiled into linear acceptors K. By composing the acceptor K with the inverted index, one obtains the posterior score, start-time and end-time of each occurrence of that keyword. The keyword acceptors do not have to be linear acceptors. They can be any acceptor, as long as each path in the acceptor represents a meaningful keyword or keyword phrase, e.g., the acceptor K' of Equation (3.3) represents multiple proxies for K. Furthermore, weights can be encoded in the keyword acceptors (such as the edit-distance supplied by E') if a single keyword has more than one representation.

Finally, a YES/NO decision is made according to the posterior scores from the search. We apply the keyword specific threshold proposed in [68], which uses the expected count of the keyword to estimate the number of the "true hits" in the formula for the actual term-weighted value (ATWV). ATWV is computed using the NIST scoring tool F4DE.

3.4 Experimental setup

3.4.1 Data

We evaluate the proposed proxy generation framework in the IARPA Babel Program (IARPA-BAA-11-02) framework, which has released conversational telephone speech corpora for several languages. In this work, we measure the system performance on Tagalog[1], Haitian[2], Lao[3], Assamese[4], Bengali[5] and Zulu[6]. We take the limited language pack (LimitedLP), which contains a 10 hour of training data, to simulate the low resource constrain. The limited language pack also comes with a 10 hour of development-test data which we use for evaluation.

3.4.2 Keyword list

We use a total of 7 keyword lists in our experiments. We start with Tagalog Eval keyword list [7], with which we show the OOV keyword statistics, and investigate proxy search performance in various settings. We then choose the best setting, and evaluate the proxy keyword search performance for the 6 languages using their corresponding

[1]Language collection release `IARPA-babel106b-v0.2g`.
[2]Language collection release `IARPA-babel201b-v0.2b`.
[3]Language collection release `IARPA-babel203b-v3.1a`.
[4]Language collection release `IARPA-babel102b-v0.5a`.
[5]Language collection release `IARPA-babel103b-v0.4b`.
[6]Language collection release `IARPA-babel206b-v0.1e`.
[7]Keyword list `IARPA-babel106b-v0.2g_conv-eval.kwlist2.xml`.

keyword lists, namely: Tagalog Dev keyword list [8], Haitian Dev keyword list [9], Lao Dev keyword list [10], Assamese Dev keyword list [11], Bengali Dev keyword list [12], Zulu Dev keyword list [13].

3.4.3 OOV keyword statistics

To better understand the importance of handling OOV keywords for keyword search in a low resource setting, we start by breaking down Tagalog Eval keyword list. Table 3.1 gives the keyword occurrence statistics of Tagalog Eval keyword list, with respect to the LVCSR's lexicon and the development-test dataset transcripts.

3805 keywords in Tagalog Eval keyword list			
1736 in-vocab keywords		2069 OOV keywords	
1067 Found in Dev	669 Not in Dev	670 Found in Dev	1399 Not in Dev

Table 3.1: Tagalog Eval keyword statistics with respect to the LVCSR's lexicon and the development-test dataset transcripts.

From Table 3.1 we can see that 2069 of 3805 keywords (54%) are OOVs with respect to the LVCSR's lexicon, which is not atypical in a low resource setting. Also, this keyword list was created for a separate (evaluation) search collection, so many keywords do not appear in the 10 hour development-test data we use as our search col-

[8] Keyword list `IARPA-babel106b-v0.2g_conv-dev.kwlist.xml`.
[9] Keyword list `IARPA-babel201b-v0.2b_conv-dev.kwlist.xml`.
[10] Keyword list `IARPA-babel203b-v3.1a_conv-dev.kwlist.xml`.
[11] Keyword list `IARPA-babel102b-v0.5a_conv-dev.kwlist.xml`.
[12] Keyword list `IARPA-babel103b-v0.4b_conv-dev.kwlist.xml`.
[13] Keyword list `IARPA-babel206b-v0.1e_conv-dev.kwlist.xml`.

lection. Indeed, only 1067 of 1736 in-vocabulary (IV) keywords, and 670 of 2069 OOV keywords occur in our search collection. Since the ATWV metric ignores keywords with zero true-positives, our KWS evaluation is *effectively* based on 1737 keywords. Yet, 670 (39%) of them are OOV. Ignoring OOV keywords therefore still significantly degrades the average ATWV.

3.5 Results

This section gives proxy keyword search performance over the 6 Babel languages. We start by experimenting with different proxy search configurations on Tagalog development-test search collection with Tagalog Eval keyword list, followed by a thorough discussion. We then choose the best configuration, and apply it to all the 6 languages.

Note that IV keywords are searched directly from the word index, while OOV keywords are searched by seeking either their word proxies using Equation (3.3) in the word index, or their phone proxies using Equation (3.2) in the corresponding phone index. As we mentioned above, we convert word lattices into phone lattices and then create the phone index. Since short keywords tend to generate much more false alarms, we only generate proxies for OOV keywords with *at least 5 phones*.

3.5.1 Comparison of 5 proxy configurations

We first explore proxy keyword search performance in 5 different configurations, as shown in Table 3.2. This part of the experiment is carried out on Tagalog with Tagalog Eval keyword list. The underlying LVCSR system is a SGMM system trained with boosted maximum mutual information. As mentioned in Section 3.4.3, there are 670 *effective* OOV keywords (i.e., the keyword actually appears in the search collection) in Tagalog Eval keyword list with respect to Tagalog development-test search collection. The richness of the generated proxies is controlled by the pruning in the *ShortestPath* step, and can either be measured by the average number of proxies per OOV keyword, or by the average number of hits retrieved (correct or false alarms) per OOV keyword. Figure 3.3 and Figure 3.4 give the proxy keyword search performance in terms of the average ATWV over the 670 *effective* OOV keywords in the 5 different configurations.

Config.	Index source	Proxy type	Uses E or E'
1	Phone lattice	Phone (K'')	No
2	Phone lattice	Phone (K'')	Yes (E)
3	Word lattice	Word (K')	No
4	Word lattice	Word (K')	Yes (E')
5	Word lattice w/ no LM scores	Word (K')	Yes (E')

Table 3.2: The numbered experimental configurations in Figures 3.3 and 3.4. The first two designate phone-based search without and with the use of phone proxies in Equation (3.2), the next two designate word-based search without and with the use of word proxies in Equation (3.3), and the last designates word-based search with word proxies after ignoring language model scores in the lattices.

Several conclusions can be drawn from Figure 3.3 and Figure 3.4. Comparing

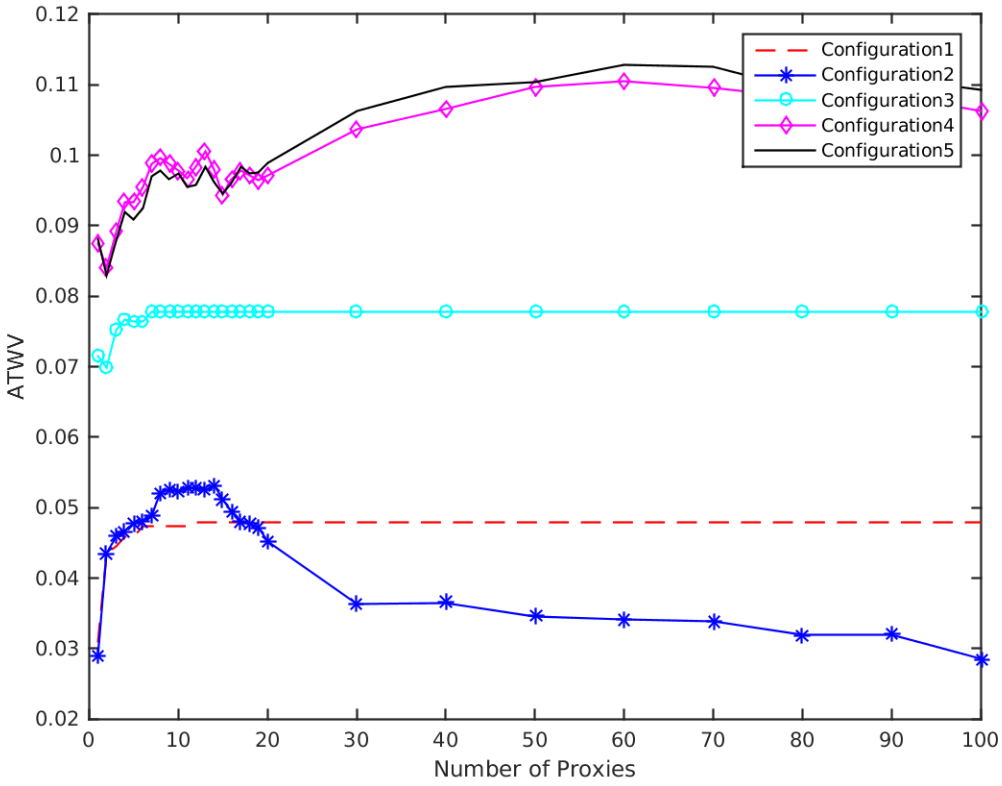

Figure 3.3: ATWV versus the number of proxies per keyword (Tagalog development-test search collection, Tagalog Eval keyword list).

Configuration 4 (magenta) with Configuration 2 (blue) in Figure 3.3 suggests that given a predetermined number of proxies per OOV keyword, searching a word index using word proxies is more effective than searching a phone index using phone proxies. Comparing Configurations 4 and 2 in Figure 3.4 suggests that for any given number of hits returned per OOV, word proxies are again the better choice. The quick drop in ATWV for Configuration 2 in both Figures 3.3 and 3.4 suggests that even some highly ranked phone proxies cause significant false alarms. This affirms the value of relying on phone sequences that satisfy lexical constraints implicitly in the word

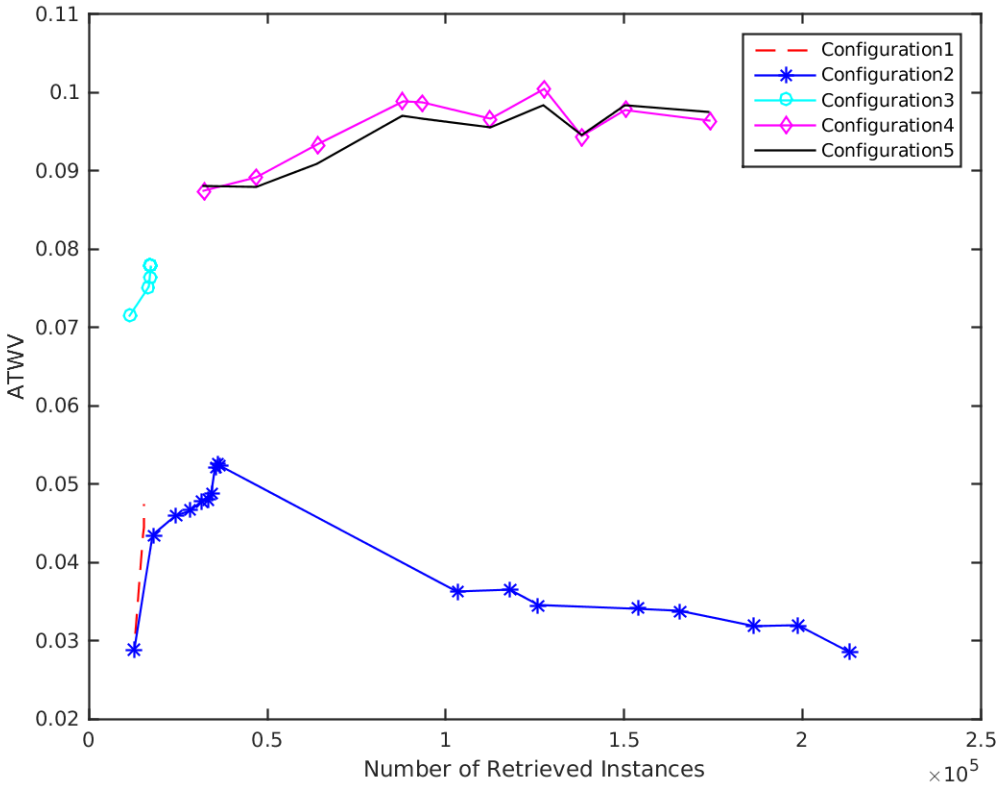

Figure 3.4: ATWV versus number of retrieved hits per keyword (Tagalog development-test search collection, Tagalog Eval keyword list).

proxies. And since we use an expected count-based threshold for YES/NO decisions, reducing the number of false alarms may indirectly increase the number of true hits marked YES.

Next, contrasting the pair of Configurations 1 & 2 against the pair of Configurations 3 & 4 in Figure 3.3 suggests that using a phone confusion transducer E or E' greatly improves KWS performance with word proxies (comparison of Configuration 3 with 4) but does not help much with phone proxies (comparison of Configuration 1 with 2). In fact, using E hurts performance in Configuration 2 when the number of

phone proxies is large. This too may be explained by the inherent false alarm problem with phone proxies. Adding the phone confusion transducer simply aggravates the situation, unless the number of proxies is severely limited, e.g. to be around 10, as suggested by Figure 3.3. On the other hand, the word proxies appear to have better precision; admitting proxies permitted by phone confusion therefore further improves the performance. We thus conclude that phone confusion information is very helpful to word proxies (40% improvement in ATWV), but should be carefully constrained when applying to phone proxies.

Next, compare Configuration 4 and Configuration 5 in Figure 3.3. The index in Configuration 5 is built without language model scores. As explained in Section 3.2.5, the mismatch between the proxies and the index may lead to some degradation in the KWS performance, because the proxies are created merely based on acoustic confusion while the index incorporates potentially incorrect language model scores. The result in Figure 3.3 shows that there is a small degradation by retaining the language model score in the index, but the degradation is usually negligible. Therefore, if only a single index can be retained/searched for both IV and OOV keywords, we suggest retaining the word index with language model scores.

Some clarification may be in order for Configuration 1 and Configuration 3, where proxies are generated without a phone confusion transducer. If only one proxy were possible for each OOV keyword without the phone edits permitted by E or E', then curves for Configuration 1 and Configuration 3 in Figure 3.3 should be horizontal

lines. However, multiple proxies are possible even without phone edits, because the lexicon L_1 contains multiple pronunciations for some words, and other legitimate ambiguities (e.g. homophones) in the phone-to-word transduction. Therefore, the curves first rise as multiple proxies are admitted by enlisting alternative shortest paths in Equation (3.2) or Equation (3.3), but quickly become horizontal lines once these limited alternatives are exhausted.

	IV Kwds	OOV Kwds	All Kwds
Tagalog	0.351	0.110	0.216 $\rightarrow$ 0.258

Table 3.3: ATWV for word search on IV keyword list and proxy search on OOV keyword list. Last column gives the overall ATWV improvement by searching OOV keywords using proxies (Tagalog development-test search collection, Tagalog Eval keyword list, proxy Configuration 4).

To close the discussion, we compare the performance of proxy keyword search on the 670 OOV keywords with that on the 1067 IV keywords, as shown in Table 3.3. We choose Configuration 4 as the ATWV performance of Configuration 4 is very close to that of Configuration 5, and Configuration 4 does not require a separate index for OOV keywords. As shown in Table 3.3, while post-facto search by proxies is still much poorer than having them in-vocabulary, the average ATWV on the full Tagalog Eval keyword list improves by 20% — from 0.216 to 0.258.

3.5.2 Proxy keyword performance

From Section 3.5.1 we can see that proxy Configuration 4 and Configuration 5 yield the best performance. While Configuration 5 requires removing the language model score from the lattices, which essentially means keeping an additional set of indices for the OOV keywords, Configuration 4 does share the same set of indices with the IV keywords, with only a little bit of performance degradation. We therefore choose proxy Configuration 4 for the rest of the experiments.

We evaluate the proxy keyword search performance for OOV keywords on all the 6 Babel languages. This time the underlying LVCSR system is a DNN system trained on top of the SAT system, and we use the Dev keyword list for all the languages. Detailed results are shown in Table 3.4.

	IV Kwds	OOV Kwds	All Kwds
Tagalog	0.2784	0.2061	0.2096 $\rightarrow$ 0.2605
Haitian	0.4664	0.1842	0.3819 $\rightarrow$ 0.4153
Lao	0.4134	0.1873	0.3629 $\rightarrow$ 0.3858
Assamese	0.3100	0.0812	0.2183 $\rightarrow$ 0.2423
Bengali	0.3345	0.0892	0.2180 $\rightarrow$ 0.2490
Zulu	0.2789	0.0693	0.1013 $\rightarrow$ 0.1454

Table 3.4: ATWV for word search on IV keyword list and proxy search on OOV keyword list. Last column gives the overall ATWV improvement by searching OOV keywords using proxies (development-test search collection, Dev keyword list, proxy Configuration 4).

From Table 3.4, it is clear that proxy search for OOV keywords is still far behind word search for IV keywords in terms of performance, but it does improve the overall search performance by extending search capability to OOV keywords without chang-

ing the word index. On the 6 languages we experimented with, proxy keyword search improves the overall ATWV by 18% relatively.

3.6 Summary

We have presented a simple, cheap and effective way to use *word proxies* to improve KWS performance for OOV keywords. Experiments were done with various Babel languages and the results suggest that such techniques are reasonably effective for handling OOV words.

Chapter 4

Automatic Lexicon Expansion

Chapter 3 attempts to alleviate the out-of-vocabulary (OOV) keyword problem for large vocabulary continuous speech recognition (LVCSR)-based low resource keyword search (KWS) methods in a *post facto* manner: it assumes the underlying LVCSR system and its corresponding KWS system are developed beforehand without any knowledge of the keywords, and at search time when an OOV keyword comes, it generates in-vocabulary (IV) *proxies* for that OOV keyword and searches for those proxies instead of the original OOV keyword. The results in Chapter 2, however, suggest that incorporating the OOV keywords (if they are available beforehand) into the LVCSR system before indexing the search collection is superior to using them *post facto*, e.g., using approximate phonetic matching of OOV keywords in pre-indexed lattices, or using our proxy keyword search technique proposed in Chapter 3. In this chapter, we attempt to alleviate the OOV keyword problem before training the

LVCSR system, by enlarging the LVCSR's pronunciation lexicon. Since it is not very practical to have prior knowledge of the keywords when building the LVCSR system, we propose to create a huge pronunciation lexicon (e.g., with one million entries) purely from the smallish original pronunciation lexicon (e.g., with a few thousand of words). A long list of hallucinated words (e.g., one million) is first generated using a grapheme-to-phoneme (G2P)-based technique. This list of hallucinated words together with the corresponding pronunciations, which can also be inferred by G2P methods, is then appended to the original pronunciation lexicon to form a huge lexicon for the LVCSR system. We show in our experiments that although most words in the hallucinated word list are not legitimate words in the language that we are working on, the list does cover an averaged 24% of true words when evaluated on a large expert-crafted lexicon (with an average of $26K$ words) from that language. Experiments also suggest that this enlarged lexicon only has a modest impact on the LVCSR's performance in terms of word error rate (WER), but it improves the keyword search performance in terms of actual term-weighted value (ATWV) by covering more keywords that are OOV w.r.t. the original pronunciation lexicon. We also show that combining the lexicon expansion technique proposed in this chapter with the proxy keyword search technique proposed in Chapter 3 further improves the keyword search performance.

4.1 Related work

Suppose we have a list of words that are not in an LVCSR's original vocabulary, adding those words to the LVCSR's vocabulary is not too difficult: first, a grapheme-to-phoneme (G2P) model can be trained on the original lexicon; then the trained G2P model can be applied to the list of words to generate candidate pronunciations for each word; some optional steps can also be taken to further refine the generated pronunciations, e.g., [78] takes an iterative approach to refine the pronunciations generated by G2P models; finally, append the list of words together with their corresponding pronunciations (generated by G2P model) to the original lexicon, and possible retrain the acoustic model and language model. Lexicons expanded this way usually yield relatively good word error rate (WER) performance when compared to an expert-crafted pronunciation lexicon with the same set of words [78]. Further more, [52] points out that expanding LVCSR's vocabulary before indexing the search collection is actually essential to keyword search performance in terms of actual term-weighted value (ATWV). The problem with keyword search is, we typically only know the keywords at search time, and adding the keywords to the LVCSR's vocabulary (if they are not already present) and then decode the search collection and rebuild the inverted index can be quite time consuming, making it impractical in real applications. We therefore propose to build LVCSR systems with a huge vocabulary, which increases the chance of covering more keywords that are OOV w.r.t. the original lexicon, thus improving the search performance. Now the problem boils down to how we can get

such a large list of words for low resource languages.

One way to get a large list of words for a certain language is to crawl the Web. Several different techniques have been proposed to retrieve relevant words from the Web. In [79], spoken term detection (a.k.a. keyword search) is used together with the Web data to recover the OOV words in the speech recognizer's output. An OOV detector is first applied to the LVCSR's output to determine the OOV regions in the lattices. Hypothesised words close to that region are then used to query the Web and words relevant to those hypothesised words will be returned by the search engine. Those returned words are then taken as keywords, and are converted to their corresponding phonetic representation to query a phonetic spoken term detection system built on the same dataset. If one of the returned words appears at the original OOV region, then it is highly likely that this word is the OOV word that is supposed to appear at that region. Other techniques more or less share the same philosophy in the sense that methods are proposed to select words retrieved from the Web. For example, in [80], local word context of the OOV words are used to retrieve the OOV words in the unlimited set of Web documents, while in [81], words in the metadata information of the audio document archive are used to argument the speech recognizer's vocabulary. This kind of technique usually works well for major languages such as English or Spanish, since a large number of resources are available on the Web for those languages. But when it comes to low resource languages, we may not be able to retrieve reasonable amount of data from the Web to achieve good performance.

Another way to expand the LVCSR's lexicon in order to cover more OOV words is to add sub-word units such as graphones [82]. This does not require retrieving a list of actual words, therefore sometimes is also called a hybrid approach in the literature. The general idea is, by adding sub-word units to the recognizer's vocabulary, OOV words will likely be represented by a combination of those sub-word units in the decoding output [82, 83, 84], which can then be recovered by various techniques, e.g., the Web-based OOV recovery method proposed in [79]. This hybrid approach has also been applied to LVCSR-based KWS systems to make the KWS system open-vocabulary (i.e., any keywords can be searched). In [85], a hybrid LVCSR system is first built with graphones added to the recognizer's vocabulary as additional sub-word units. The search collection is then decoded by this hybrid recognizer to generate lattices that contain both words and graphones. After that, all the graphones in the lattices are joined together to form words, which are possibly OOV w.r.t. the original word lexicon. Now, the lattices are converted into an inverted index and keyword search can be performed. One potential issue with this hybrid approach is that the sub-word units added to the recognizer's vocabulary are relatively short (e.g., phones, syllables), therefore the recognizer may take the sub-word units to fill in the place of IV words as well, which will likely hurt the performance of IV keywords.

In this chapter, we propose a G2P-based technique to generate a huge list of hallucinated words for lexicon expansion purpose. A trigram language model is first trained for the syllable sequences from the recognizer's lexicon (we treat syllable

sequence of one word as a sentence). This language model is then used to randomly generate a huge number of syllable sequences, which are actually pronunciations for potential words. We also train a G2P model from the original lexicon so that we can map the pronunciations to their corresponding spellings. Now that we have both spellings and pronunciations, we append them to the original lexicon to form a larger lexicon that has a better chance of covering more OOV keywords.

Our approach in this chapter is similar to the sub-word approach above in the sense that we also generate "pronunciation"-driven "words" from the original lexicon, and those "words" are not necessarily legitimate. However, instead of generating sub-word units, we generate longer units that can cover several syllables, which also have a chance to be legitimate words in that language. Since longer units usually capture more discriminative information, they are less likely to hurt the speech recognition or keyword search performance for IV words. We show in our experiments that by using our expanded lexicon, we achieve better keyword search performance in terms of ATWV, without hurting the WER performance. We also show that combining the expanded lexicon with the proxy keyword search technique proposed in Chapter 3 further improves the keyword search performance.

4.2 Automatic lexicon expansion

The general idea of our lexicon expansion method is to generate a large number of potential word pronunciations, and then map the pronunciations to their corresponding spellings using G2P models. In the next few subsections, we will first explain how G2P conversion works. We then describe how to train a pronunciation model to generate potential word pronunciations, and finally we show how we can map from pronunciations to spellings with G2P model.

4.2.1 Grapheme-to-phoneme conversion

G2P conversion is a task to map a word, represented by a sequence of graphemes, to its pronunciation, represented by a sequence of phonemes [86]. Suppose w is the grapheme sequence of a word, and $\hat{p}$ its corresponding pronunciation, G2P can be framed as follows [87]:

$$\hat{p} = \arg\max_{p} P(p|w) = \arg\max_{p} P(w, p) \tag{4.1}$$

Therefore, one can either model the conditional distribution $P(p|w)$ or the joint distribution $P(w, p)$. From Equation (4.1) it's not difficult to see that the same technique works for mapping from phonemes sequences to graphemes as well. In fact, if we swap the grapheme sequences and phoneme sequences during training, the model trained in the same way should be able to map from phoneme sequences to grapheme sequences.

We use the open source toolkit Sequitur [69] for G2P conversion in this work.

4.2.2 Pronunciation model

The core idea of our proposed lexicon expansion method is to generate a large number of potential word pronunciations. Our pronunciation model is essentially an n-gram language model for syllables. Suppose we have the following lexical entry in our lexicon (taken from Tagalog lexicon)

$$Buboy \quad \equiv \quad \text{"} \;\; b \;\; u \;\; \cdot \;\; b \;\; oj$$

where "*Buboy*" is a Tagalog word, "" b u $\cdot$ b oj" is its corresponding pronunciation with two syllables "" b u " and "b oj", "" is a stress mark, and "$\cdot$" is the syllable separator. We treat the sequence of syllables in the lexical entry like the sequences of words in a sentence, and train a language model for the syllable sequences. In the above example, "" b u $\cdot$ b oj" will be treated as a sentence with two words "" b u" and "b oj".

We use SRILM [66] to train the language model. Typically we set n-gram order to 3 and train the model with Kneser-Ney smoothing. The trained language model is capable of generating random sentences with certain probabilities. In fact, sentences generated this way are syllables sequences, which are essentially pronunciations of potential words. We typically generate 12 million sentences, and we use SRILM to

compute the probabilities of these sentences with the syllable language model.

4.2.3 Spelling generation

A lexical entry needs a spelling as well as a pronunciation. To do this we use the G2P conversion from Sequitur *in reverse* to produce the most likely spellings for each pronunciation (sentence generated in Subsection 4.2.2). Let us again take the Tagalog lexical entry as an example. In the original lexicon, we have:

$$Buboy \quad \equiv \quad " \;\; b \;\; u \quad \cdot \quad b \;\; oj$$

We reverse the grapheme sequence and the phoneme sequence in this lexical entry as follows:

$$b"u"boj \quad \equiv \quad B \;\; u \;\; b \;\; o \;\; y$$

Note that the stress mark "ˈ" has been applied to all the phones in that syllable. In our actual implementation, we map the phones to ASCII symbols first, so that phones like "*oj*" will only appear as one ASCII symbol. When we do the mapping, we treat tags (e.g., the stress mark "ˈ") separately, so each tag has its own ASCII symbol, and a phone with a tag would be rendered as two ASCII symbols.

The reversed lexicon is used to train a G2P model with Sequitur. Once we have the G2P model trained, we apply the model to all the 12 million pronunciations (sentences) generated in Subsection 4.2.2, and form lexical entries with both spellings and

pronunciations. We generate at most 3 alternative spellings for each pronunciations, and pronunciations with less than 3 phones are also removed since short words tend to create false alarms in keyword search. Of course, lexical entries that are same to those in the original lexicon are also removed.

We now define the probability of the generated lexical entry to further cut down the lexicon size. Let $P_{LM}(p)$ be the language model probability of pronunciation p from Subsection 4.2.2, let $P_{G2P}(p \rightarrow w)$ be the spelling probability assigned by G2P model, we define the probability of the new lexical entry "$w \equiv p$" as $P(w \equiv p) = P_{LM}(p)P_{G2P}(p \rightarrow w)$. We then sort all the generated lexical entries by their probabilities, and append the best one million entries to the original lexicon. Note that the probabilities of the one million lexical entries are further scaled so that probabilities of all entries sum to unity.

4.3 Language model

As suggested by the work in [52], language model also plays an important role in low resource keyword search. In [52], the authors utilize additional text data that covers the expanded lexicon for language modeling. In our work here, however, since most of the hallucinated words are not even legitimate words in that languages, additional text data that covers the hallucinated words is unfortunately not available. We therefore propose to simply add unigram probabilities to the language model for

the hallucinated words.

We foresee two potential advantages of adding unigram probabilities to the language model for those hallucinated words. First, since we only add unigram probabilities for the hallucinated words, these words are less likely to appear as the 1-best hypotheses in the decoding output, unless the acoustic model is extremely confident about their appearance. This way, the hallucinated words are less likely to hurt the original IV words performance, even if we augment the recognizer's vocabulary by one million lexical entries. Second, given the small unigram probabilities assigned to the hallucinated words in the language model, they still have a chance to appear the lattices. Various search techniques can then be adopted to improve the keyword search performance, e.g., the proxy keyword search technique as we will explain in Section 4.4.

We now explain how we assign unigram probabilities to the hallucinated words. We treat all the hallucinated words as OOV words, as they do not appear in the recognizer's original lexicon. We take a 2 hour held-out set from the search collection, and calculate the OOV rate on the recognizer's original vocabulary. This OOV rate is taken as the probability mass for all the hallucinated words. Let us denote it as P_{OOV}, we then assign probability $P(w \equiv p)P_{OOV}$ for word w in the language model, where $P(w \equiv p)$ is the *normalized* lexical entry probability from Subsection 4.2.3. We make sure that the unigram probability we assign to the hallucinated word is not more probable than the least probable word which is originally in the language model

(excluding "<s>" and "</s>" symbols).

4.4 Proxy keyword search

In Chapter 3, a technique called proxy keyword search is proposed to alleviate the OOV keyword problem in low resource keyword search. The general idea of proxy keyword search is, if an OOV word appears in the search collection, the recognizer will most likely interpret it as acoustically similar IV words that are in the recognizer's vocabulary. Therefore at search time, we can search those acoustically similar IV words as proxies of the original OOV keyword. In this work, since we only add unigram probabilities for the hallucinated words, their behavior should be similar to actual OOV words, thus applying proxy keyword search for keywords that only appear in the hallucinated word list should improve the keyword search performance.

Proxy keywords can be generated in a weighted finite-state transducer (WFST) framework. Let K represent a finite-state acceptor for a keyword (possibly OOV), and L_2 a finite-state transducer for the pronunciation of that keyword. Let E be an edit-distance transducer that maps a phone sequence to any other phone sequence with costs estimated from a phone confusion matrix. Let L_1 denote the pronunciation lexicon of the LVCSR system. The WFST procedure for generating a proxy keyword

K' can be described as,

$$K' = \text{Project}\big(\text{ShortestPath}\big(K \circ L_2 \circ E \circ (L_1^*)^{-1}\big)\big) \tag{4.2}$$

K' is then used as proxies to search the index instead of the original keyword. Typically, the edit-distance transducer is modified to allow cost-inexpensive insertions and deletions at word boundaries.

4.5 Experimental setup

4.5.1 Corpus

We evaluate the proposed lexicon expansion technique in the IARPA Babel Program (IARPA-BAA-11-02) framework, which has released conversational telephone speech corpora for several languages. In this work, we measure the system performance on Tagalog[1], Haitian[2], Lao[3], Assamese[4], Bengali[5] and Zulu[6]. We take the limited language pack (LimitedLP), which contains a 10 hour of training data, to simulate the low resource constrain. The limited language pack also comes with a 10 hour of development-test data which we use for evaluation. Note that since we do

[1]Language collection release `IARPA-babel106b-v0.2g`.
[2]Language collection release `IARPA-babel201b-v0.2b`.
[3]Language collection release `IARPA-babel203b-v3.1a`.
[4]Language collection release `IARPA-babel102b-v0.5a`.
[5]Language collection release `IARPA-babel103b-v0.4b`.
[6]Language collection release `IARPA-babel206b-v0.1e`.

not have the full list of words for each language, we take the lexicon from the full language pack (fullLP) and treat that as the "reference" word list. The lexicons from fullLP have an average number of $26K$ words. For our keyword search performance, we use the Dev keyword list for each language, namely: Tagalog Dev keyword list [7], Haitian Dev keyword list [8], Lao Dev keyword list [9], Assamese Dev keyword list [10], Bengali Dev keyword list [11], Zulu Dev keyword list [12].

4.5.2 System description

We use the open source toolkit Kaldi [65] for all our system development and experiments. Standard PLP analysis is employed to extract 13 dimensional acoustic feature, and a maximum likelihood acoustic training recipe is followed to train speaker adaptive models, which is further followed by a hybrid deep neural network (DNN) system. Lattices are then generated for the development-test set of each language using the DNN model. For the keyword search task, lattice indexing is further performed after decoding to convert lattice of each utterance into a finite-state acceptor with the posterior score, start-time and end-time for each word encoded as a 3-dimensional weight. An inverted index is then created from these individual acceptors, with paths to accept every possibly word sequence in the original lattices.

[7] Keyword list `IARPA-babel106b-v0.2g_conv-dev.kwlist.xml`.
[8] Keyword list `IARPA-babel201b-v0.2b_conv-dev.kwlist.xml`.
[9] Keyword list `IARPA-babel203b-v3.1a_conv-dev.kwlist.xml`.
[10] Keyword list `IARPA-babel102b-v0.5a_conv-dev.kwlist.xml`.
[11] Keyword list `IARPA-babel103b-v0.4b_conv-dev.kwlist.xml`.
[12] Keyword list `IARPA-babel206b-v0.1e_conv-dev.kwlist.xml`.

This way, keyword search can be done by composing the keyword acceptor with the inverted index. For details of speech recognition and keyword search systems, readers are referred to [77, 45, 52].

4.6 Results

We report word error rate (WER) for the speech recognition task, and actual term-weighted value (ATWV) for the keyword search task. WERs are reported in percentage.

4.6.1 Precision and recall

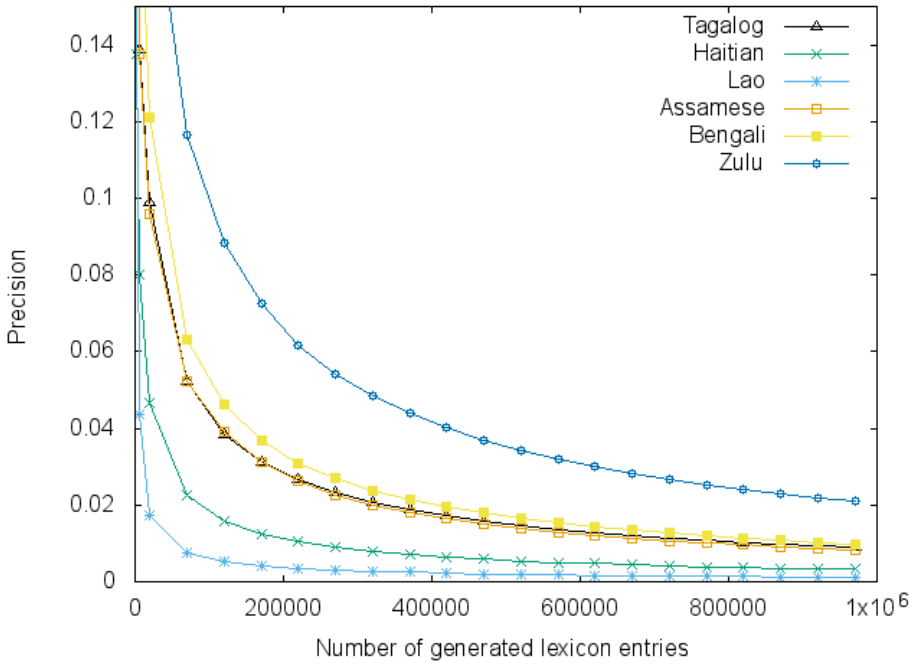

Figure 4.1: Hallucinated word precision v.s. number of generated lexical entries.

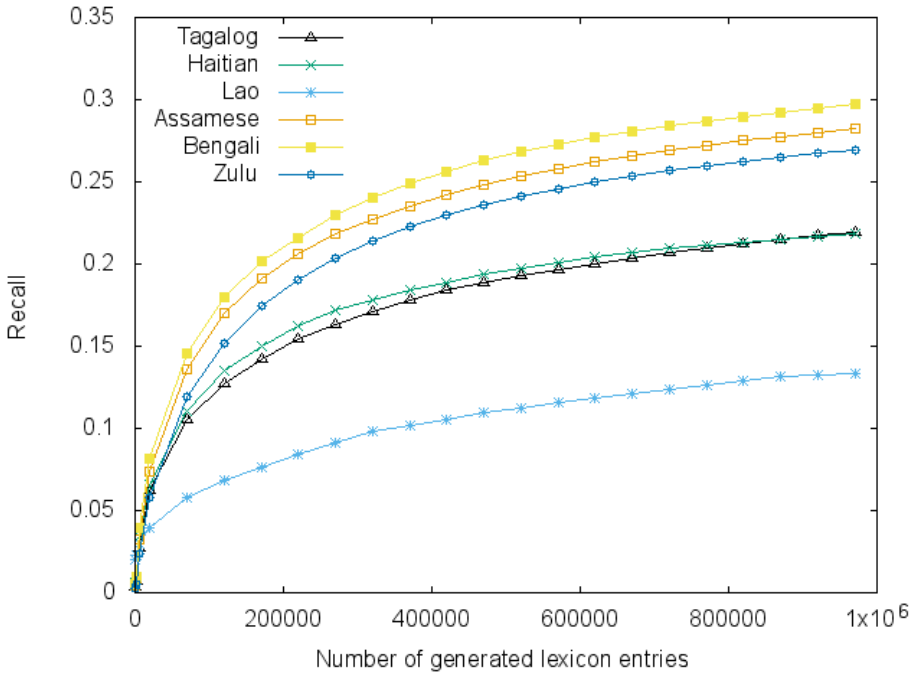

Figure 4.2: Hallucinated word recall v.s. number of generated lexical entries.

Figure 4.1 and Figure 4.2 illustrate the precision and recall of the hallucinated words as the number of expanded lexical entries goes up to one million. Since we do not have a full list of words in the 6 Babel languages that we test, we treat the lexicon form full language pack (fullLP) of each language release as the ground truth. The 6 fullLP lexicons have an average of $26K$ words, with Zulu having the largest lexicon with $61K$ words, and Lao having the smallest lexicon with only $6K$ words.

For the following experiments, we choose to expand the original with an additional one million lexical entries, as most of the languages have a recall of $20\% - 30\%$ at that operating point, except for Lao which only has a recall of 13% even after adding one million lexical entries. We do not worry too much about the low precision at

that operating point, because we only add unigram probabilities to the language model for the hallucinated words, and most of those words will not be picked up by the recognizer as the 1-best hypotheses — they will most likely appear deep in the lattices, which can be recovered by keyword search methods.

4.6.2 Speech recognition performance

	Original Lexicon	Expanded Lexicon
Tagalog	61.0	**60.5**
Haitian	**60.3**	60.5
Lao	57.9	**57.5**
Assamese	**64.0**	64.1
Bengali	**66.2**	**66.2**
Zulu	68.9	**68.3**

Table 4.1: WER (%) performance of speech recognition system with original lexicon and expanded lexicon.

Table 4.1 compares the WER performance of speech recognition with the original lexicon and the expanded lexicon. Out of the 6 Babel languages we test, the expanded lexicon system outperforms the original lexicon in 3 languages, and ties in 1 language. On average, the expanded lexicon system reduces WER by 0.2% (absolute). This confirms our previous claim that adding a huge list of hallucinated words does not hurt the speech recognition performance.

	Original Lexicon	Expanded Lexicon
Tagalog	0.2096	**0.2856**
Haitian	0.3819	**0.3944**
Lao	**0.3629**	0.3625
Assamese	0.2183	**0.2257**
Bengali	0.2180	**0.2285**
Zulu	0.1013	**0.1453**

Table 4.2: ATWV performance of keyword search system with original lexicon and expanded lexicon. Proxy keyword search is not performed for OOV keywords.

4.6.3 Keyword search performance

Figure 4.2 shows that for most of the 6 Babel languages that we test, adding one million lexical entries to the original lexicon can cover $20\% - 30\%$ of words in the corresponding fullLP lexicon. This essentially reduces the number of OOV keywords, therefore should improve the keyword search performance. Table 4.2 compares the keyword search performance of the original lexicon system and the expanded lexicon system. The expanded lexicon systems outperforms the original lexicon system in all languages by a large margin, except for Lao where the expanded lexicon system gives a little bit worse ATWV (0.0004) than the original lexicon system. This echoes with the discovery earlier that the proposed lexicon expansion method generates a relatively low precision and recall word list for Lao, when compared with other languages.

Note that in Table 4.2 we do not handle OOV keywords. In Chapter 3 proxy keyword search is proposed to alleviate OOV keyword problem for low resource keyword search. One way to combine proxy keyword search with the expanded lexicon is to also perform proxy keyword search for those keywords that appear in the expanded

	Keyword Search	Proxy Search
Tagalog	0.3853	**0.3989**
Haitian	0.2355	**0.3509**
Lao	0.0772	**0.1091**
Assamese	0.1030	**0.1686**
Bengali	0.1406	**0.2161**
Zulu	0.2744	**0.2861**

Table 4.3: ATWV performance with keyword search and proxy search in the expanded lexicon system, for keywords that are in the expanded lexicon, but are not in the original lexicon.

lexicon, but are not in the original lexicon, instead of just applying proxy keyword search to OOV keywords. Table 4.3 compares the ATWV performance of keyword search and proxy search for the set of keywords that only appear in the expanded lexicon. It is clear that proxy search outperforms the regular keyword search in all languages for that set of keywords. This is actually what we would expect because the hallucinated words do not have examples in the acoustic training data, and we only add unigram probabilities for them in the language model — they should behave like OOV words.

Table 4.4 compares the ATWV performance of the original lexicon system and the expanded lexicon system after combining the expanded lexicon system with proxy keyword search. As suggested by Table 4.3, for the expanded lexicon system we also apply proxy keyword search to those keywords that appear in the expanded lexicon, but are not in the original lexicon. The "OOV" keywords in Table 4.4 under the column "Expanded Lexicon" therefore are really only out-of-vocabulary w.r.t. the

	Original Lexicon			Expanded Lexicon		
	IV Kwds	OOV Kwds	All Kwds	IV Kwds	OOV Kwds	All Kwds
Tagalog	0.2784	0.2061	0.2605	0.3190	0.3052	**0.3156**
Haitian	0.4664	0.1842	0.4153	0.4651	0.2877	**0.4329**
Lao	0.4134	0.1873	**0.3858**	0.4104	0.1850	0.3829
Assamese	0.3100	0.0812	0.2423	0.3036	0.1137	**0.2474**
Bengali	0.3345	0.0892	0.2490	0.3240	0.1588	**0.2665**
Zulu	0.2789	0.0693	0.1454	0.2958	0.1841	**0.2246**

Table 4.4: ATWV performance of the proposed lexicon expansion method, OOV keywords are out-of-vocabulary w.r.t. to the original lexicon, and are searched with proxies.

original lexicon. From Table 4.4 we can see that with regular keyword search for IV keywords, and proxy keyword search for OOV keywords, the expanded lexicon systems outperforms the original lexicon systems in all languages except for Lao, where the expanded lexicon system's ATWV is a little bit lower than that of the original lexicon system. For Tagalog and Zulu, the expanded lexicon even improves the search performance of IV keywords.

4.7 Summary

We have proposed a G2P-based technique that automatically expands the lexicon for low resource languages by adding a huge list of hallucinated words learned from the original lexicon. Experiments on 6 Babel languages suggest that although the expanded lexicon only leads to modest gain for speech recognition tasks, its contribution to keyword search is remarkable, as it covers more keywords that are out-of-

vocabulary w.r.t. the original lexicon. Experiments also show that combining the expanded lexicon with proxy keyword search for OOV keywords further improves the keyword search performance.

Chapter 5

Limited Lexicon Keyword Search

Handcrafted pronunciation lexicons are widely used in modern speech recognition systems as well as keyword search systems that rely on automatic speech recognition systems. Designing a pronunciation lexicon, however, requires tremendous amount of expert knowledge and effort, which is not practical when applying keyword search and speech recognition techniques to low resource languages. In this chapter, we first show that while it is relatively easy for alphabetic languages to completely avoid expert pronunciation lexicons by developing grapheme-based speech recognition and keyword search systems, avoiding expert pronunciation lexicon for logographic languages is still pretty difficult, as the graphemes in such languages do not necessarily imply the phonetic representation of the words, and the number of graphemes to model is often quite large, e.g., a few thousand. We then propose to develop speech recognition and keyword search systems for logographic languages with only a small expert

pronunciation lexicon. An iterative framework is proposed to generate and refine the phonetic transcripts of the training data, which will then be aligned to their word-level transcripts for grapheme-to-phoneme (G2P) model training. The G2P model trained this way covers graphemes that appear in the training transcripts (most of which are usually unseen in a small expert lexicon for logographic languages), therefore is able to generate pronunciations for all the words in the transcripts. The proposed lexicon generation procedure is further evaluated on Cantonese speech recognition and keyword search tasks. Experiments show that starting from an expert lexicon of only $1K$ words, we are able to generate a lexicon that works reasonably well when compared with an expert-crafted lexicon of $5K$ words.

5.1 Related work

In the past few years there has been an increased interest in developing speech recognition and keyword search systems for low resource languages. Building a speech recognition system for a new language usually requires three major resources: first, transcribed speech data for acoustic modeling; second, optional additional text data for language modeling; and finally a lexicon that maps words to sub-word modeling units, typically, phonemes. While it is relatively easy to collect transcribed speech data and text data, the creation of the pronunciation lexicon is often expensive as it requires large amount of expert knowledge and effort. The pronunciation lexicon,

therefore, is the Achilles heel when building speech recognition and keyword search systems for low resource languages.

A lot of techniques have been proposed in the literature to reduce the expert effort needed in lexicon design for automatic speech recognition. One solution is to model graphemes instead of phonemes as the sub-word units, which completely removes the necessity of a phonetic pronunciation lexicon in speech recognition. Such techniques have found success in languages with alphabetic (a.k.a. segmental) writing systems [88, 89], but cannot naturally be extended to other writing systems, e.g., logographic, as the graphemes in those languages do not necessarily imply the phonetic representation of the words, and the number of graphemes is often quite large, e.g., a few thousand. Other researchers have been looking into techniques that generate pronunciation lexicons in a data-driven and stochastic manner. In [90, 91], a hierarchical Bayesian model is proposed to jointly discover the phonetic inventory as well as the grapheme-to-phoneme (G2P) mapping rules using only transcribed speech data. The authors show encouraging results in their papers, but the pronunciation lexicon discovery process itself is quite time consuming with the proposed model, making it the bottleneck when rapid development of speech recognition systems is desired. A more practical technique for logographic languages is to start from a small expert pronunciation lexicon, enlarge it by learning the pronunciations of additional words and incorporate them into the existing speech recognition system. In [92, 93, 94, 95, 78, 96], the expert lexicon is used to train a G2P model, with which

pronunciations of additional words are generated, and added to the existing lexicon. The enlarged lexicon can then be refined in a data-driven manner.

Most of the above mentioned techniques are developed for languages with alphabetic writing systems, for example, English and Spanish. But as we will show in Section 5.2, developing speech recognition and keyword search systems for low resource alphabetic languages without an expert pronunciation lexicon is quite doable — grapheme-based systems for those languages only give a small amount of performance degradation when compared to the corresponding expert pronunciation lexicon-based systems. We therefore will focus on logographic languages in this chapter. We are interested in developing speech recognition and keyword search systems for logographic languages with only a small expert pronunciation lexicon. We follow the general techniques in [92, 93, 94, 95, 78, 96], where G2P conversion is used to generate pronunciations for out-of-vocabulary (OOV) words. For logographic languages, due to the large number of unique graphemes, G2P models trained on a small seed lexicon, as proposed in [78], typically are not able to generate pronunciations for all the words in the training transcripts. Previous work on pronunciation modeling for logographic languages such as Mandarin Chinese mostly only focus on pronunciation variants [97, 98], and does not address the problem of unseen graphemes. In this chapter, we propose to incorporate the phonetic transcripts of the training data into G2P modeling through an iterative framework, so that all the graphemes that appear in the training transcripts will be modeled. We start from the initial expert lexicon

and build a bootstrap speech recognition system, with which we generate phonetic transcripts for the training data. These phonetic transcripts are aligned to their word-level transcripts using a many-to-many alignment algorithm [99], which can then be used for G2P modeling and lexicon update. This procedure is carried out iteratively, and is able to generate pronunciations for words in the training transcripts.

We make three contributions through this chapter. First, we re-visit grapheme-based speech recognition and keyword search systems, and show that while such systems can yield competible results for alphabetic languages without an expert pronunciation lexicon, directly applying them to logographic languages does not work properly. Second, we propose an iterative framework that is capable of generating pronunciations for any words in logographic languages, as long as the word appears in the transcribed training data. This allows us to build speech recognition systems with a small expert pronunciation lexicon. Third, the proposed iterative framework is evaluated on Cantonese speech recognition and keyword search tasks, and we show that the lexicon generated from the proposed framework performs reasonably well when compared with an expert-crafted pronunciation lexicon.

The remainder of this chapter is organized as follows. We re-visit grapheme-based speech recognition and keyword search systems in Section 5.2. We describe our pronunciation generation method in Section 5.3, and explain how we handle multiple pronunciations in our speech recognition system in Section 5.4. We then illustrate our proposed iterative lexicon generation framework in Section 5.5. The experimental

setup is detailed in Section 5.6, and results are provided in Section 5.7. Finally we reiterate our main claims in Section 5.8.

5.2 Grapheme systems for alphabetic languages

A common way to avoid expert pronunciation lexicon when developing speech recognition systems is to use graphemes as the modeling units instead of phonemes [100, 101, 102]. In [89], the authors use the unicode of each grapheme as the modeling unit, and show on several low resource languages (i.e., the training data available is limited, e.g., 10 hours) that their unicode-based grapheme systems deliver comparable, and complementary performance to phonetic lexicon-based systems.

However, almost all the grapheme-based approaches proposed in the literature are intended for alphabetic languages, and their extension to other writing systems is not very trivial. Take logographic languages as an example, graphemes in such languages usually do not have an obvious relation with their corresponding pronunciations, making it difficult to bridge the written words and their underlying acoustic realization. Also, logographic languages typically have a large set of unique graphemes (e.g., a few thousand), which makes it impractical to build context-dependent systems on top of the modeling units.

We start this chapter by exploring the grapheme-based approaches for low resource

speech recognition and keyword search systems. We follow [89] and take the unicode of the grapheme as the modeling unit. The rest of our speech recognition and keyowrd search pipeline is the same as that described in Section 5.6.2. Experiments are carried out on 7 Babel languages, namely Tagalog[1], Haitian[2], Lao[3], Assamese[4], Bengali[5] Zulu[6] and Cantonese[7]. We use the 10 hour subset of the full language pack to simulate the low resource condition. The first 6 languages are alphabetic languages, while the last one Cantonese is logographic. The speech recognition part is evaluated in terms of word error rate (WER), and the keyword search part is evaluated using actual term-weighted value (ATWV). We use the "dev" keyword list for all the languages in their corresponding language collection release, and we take the 10 hour development-test set for each language as the evaluation set. Detailed results are given in Table 5.1.

From Table 5.1, we can see that for the 6 alphabetic languages, by switching the expert lexicon to grapheme lexicon, there is a small amount of performance degradation in terms of both WER and ATWV, but the performance is not very far away when compared with that of the expert lexicon. For the logographic language Cantonese, switching modeling units from phonemes to graphemes caused an explosion in size when compiling the decoding graph, so we did not perform WER and ATWV

[1]Language collection release `IARPA-babel106b-v0.2g`.
[2]Language collection release `IARPA-babel201b-v0.2b`.
[3]Language collection release `IARPA-babel203b-v3.1a`.
[4]Language collection release `IARPA-babel102b-v0.5a`.
[5]Language collection release `IARPA-babel103b-v0.4b`.
[6]Language collection release `IARPA-babel206b-v0.1e`.
[7]Language collection release `IARPA-babel101b-v0.4c`.

	Expert Lexicon		Grapheme Lexicon	
	WER	ATWV	WER	ATWV
Tagalog	61.0	0.2096	61.9	0.1969
Haitian	60.3	0.3819	61.0	0.3617
Lao	57.9	0.3629	59.0	0.3323
Assame	64.0	0.2183	63.7	0.2065
Bengali	66.2	0.2180	67.1	0.1867
Zulu	68.9	0.1013	69.3	0.0918
Cantonese	53.8	0.2623	-	-

Table 5.1: WER (%) and ATWV performance when utilizing expert lexicon and grapheme lexicon, DNN system.

evaluation for the Cantonese grapheme system. Generally speaking, grapheme-based approaches are good candidates for alphabetic languages when expert pronunciation lexicon is not available during speech recognition and keyword search system development, but are not very helpful when it comes to logographic languages.

5.3 Pronunciation generation for logographic languages

A more practical way to reduce the expert effort when building speech recognition and keyword search systems for low resource logographic languages is to start from a small expert lexicon, and then learn the pronunciations automatically for the rest of the words. The most commonly used technique for such pronunciation learning task is grapheme-to-phoneme (G2P) conversion.

G2P conversion is a task to map a word, represented by a sequence of graphemes, to its pronunciation, represented by a sequence of phonemes [86]. Suppose w is the grapheme sequence of a word, and $\hat{p}$ its corresponding pronunciation, G2P can be framed as follows [87]:

$$\hat{p} = \arg\max_{p} P(p|w) = \arg\max_{p} P(w, p) \tag{5.1}$$

Therefore, one can either model the conditional distribution $P(p|w)$ or the joint distribution $P(w, p)$. In this work, we choose to model the later, and break it down into three steps [103]: (i) an alignment step that aligns phonemes to graphemes, (ii) a modeling step that takes the alignments and creates a pronunciation model, and (iii) a decoding step that generates pronunciations for new words using the pronunciation model.

The G2P formulation generally works for both alphabetic and logographic languages. For logographic languages, since there are usually a large number of unique graphemes, special care should be taken during model training. Below we explain how we generate pronunciations for logographic languages when we build a speech recognition system with a small expert lexicon.

5.3.1 G2P training data

G2P training data usually consists of a set of grapheme sequences and their corresponding phoneme sequences. For G2P conversion in speech recognition tasks, oftentimes the pronunciation lexicon is used for G2P training.

In our task, since we are building speech recognition systems for logographic languages and we only have a small expert lexicon, training the G2P model using the lexicon will result in a large number of unseen graphemes in the training data, and the G2P model trained this way will not be able to generate pronunciations for all the words in the training data.

In order to cover as many graphemes as possible, we propose to train the G2P model on the phonetic training transcripts, in addition to the pronunciation lexicon. A speech recognition system can first be trained with the small expert lexicon that we start with. Training data will then be decoded into phoneme sequences using this initial speech recognizer. Now for each utterance in the training data, we have a sequence of graphemes, and their corresponding phonemes. We use those grapheme and phoneme sequences together with the pronunciation lexicon to train the G2P model, which will be able to generate pronunciations for all the words in the training transcripts. The phoneme sequences generated by the initial recognizer will generally be noisy, but it can be refined through an iterative procedure, which we will explain in details in Section 5.5.

5.3.2 G2P alignment

Grapheme and phoneme sequences provided in the G2P training data have to be aligned into modeling units called "graphones" before building the pronunciation model. For logographic languages such as Cantonese, one grapheme typically corresponds to multiple phonemes, we adopt a many-to-many alignment algorithm proposed in [99]. We use the open source toolkit Phonetisaurus [103] to generate our alignment, which implements a weighted finite-state transducer (WFST) version of the many-to-many aligner. In our Cantonese experiments, we allow at most 1 grapheme, and at most 4 phonemes in a single graphone. A typical Cantonese graphone generated by Phonetisaurus looks like "中 / dz1 u:1 N1", where "中" is a single grapheme (Chinese character), and "dz1 u:1 N1" is the phoneme sequence aligned to it.

5.3.3 Joint n-gram model

After aligning graphemes and phonemes into graphones, a 4-gram language model is trained for the graphone sequences using SRILM [66]. The language model is further converted into a WFST G, whose input labels are graphemes, output labels are phonemes, and weights are the n-gram scores. The WFST G serves as the pronunciation model.

5.3.4 Pronunciation generation

Given a grapheme sequence w of a certain word, generating pronunciation p for the word is essentially finding the best path through G that has input label sequence w, as described in Equation (5.2).

$$p = \text{ShortestPath}(\text{Determinize}(\text{Project}(w \circ G))) \tag{5.2}$$

For our Cantonese experiments, we generate at most 5 different pronunciations for a single word.

5.4 Handling multiple pronunciations

Since multiple pronunciations are generated for words that are not already in the small expert lexicon, we explicitly model pronunciation and inter-word silence probability as that has been found helpful when pronunciation variants exist in the lexicon [104]. Unlikely pronunciations are further pruned away based on the estimated pronunciation probabilities. We incorporate the estimated pronunciation and inter-word silence probabilities into the lexicon transducer, which will be used in both training and decoding.

5.4.1 Pronunciation probability estimation

We estimate the pronunciation probabilities for a word with multiple pronunciations via simple relative frequency [105, 106, 107]. Let $w.p_i$ be the i^{th} pronunciation of word w, $1 \leq i \leq N_w$, and N_w is the number of different *baseform* pronunciations of word w in the lexicon. Let $C(w, w.p_i)$ be the count of "w $w.p_i$" pairs in the aligned training data. The probability of a pronunciation $w.p_i$ given the word w is simply

$$\pi(w.p_i|w) = \frac{C(w, w.p_i) + \lambda_1}{\sum_{i=1}^{N_w}(C(w, w.p_i) + \lambda_1)}, \tag{5.3}$$

where λ_1 is a smoothing constant that we typically set to 1. An undesirable consequence of (5.3) is that a word with several equiprobable pronunciations is unfairly handicapped w.r.t words that have a single pronunciation. Max-normalization, whereby the pronunciation probabilities are scaled so that the most likely pronunciation of each word has "probability" 1, has been found helpful in speech recognition [108]. This suggests using

$$\pi'(w.p_i|w) = \frac{\pi(w.p_i|w)}{\max_{1\leq i\leq N_w} \pi(w.p_i|w)}. \tag{5.4}$$

We do max-normalization for pronunciation probabilities in all our experiments. The quantity $\pi'(w.p_i|w)$ is of course not a well defined probability any more.

5.4.2 Silence probability estimation

This subsection explains how we model the probability of silence preceding or following certain pronunciation. For a given sequence of words, we assume there is either a silence or non-silence event between two consecutive words. Since such an event usually depends on the neighbouring words, we further assume that it only depends on the two surrounding words, i.e., we model the event using $P(s \mid w.p_i, w'.p_j)$ and $P(n \mid w.p_i, w'.p_j)$, where $w.p_i$ and $w'.p_j$ are the surrounding pronunciations, s and n represent silence and non-silence event. For computation simplicity, we decompose this into two parts: (i) probability of inter-word silence (or non-silence) following the pronunciation, and (ii) probability of inter-word silence (or non-silence) preceding the pronunciation.

We use $P(s_r|w.p)$ to denote the probability of inter-word silence following the pronunciation $w.p$, and $P(n_r|w.p)$ for the complementary probability of non-silence following $w.p$. We compute $P(s_r|w.p)$ from training data counts as

$$P(s_r|w.p) = \frac{C(w.p\,s) + \lambda_2 P(s)}{C(w.p) + \lambda_2}, \tag{5.5}$$

where $C(w.p\,s)$ is the count of the sequence "$w.p\,s$" in the training data alignment, $C(w.p)$ is the count of pronunciation $w.p$, $P(s) = C(s)/(C(s) + C(n))$ is the overall probability of inter-word silence, and λ_2 is a smoothing constant that we set to 2 for experiments.

Directly modeling the probability of inter-word silence (or non-silence) preceding the pronunciation will cause double counting problem. We therefore compute it as correction term instead

$$F(s_l|w'.p) = \frac{C(s\ w'.p) + \lambda_3}{\widetilde{C}(s\ w'.p) + \lambda_3}, \text{ and} \tag{5.6}$$

$$F(n_l|w'.p) = \frac{C(n\ w'.p) + \lambda_3}{\widetilde{C}(n\ w'.p) + \lambda_3}, \tag{5.7}$$

where $\widetilde{C}(s\ w'.p)$ and $\widetilde{C}(n\ w'.p)$ are the "mean" counts of silence or non-silence preceding $w'.p$, estimated according to $\widetilde{C}(s\ w'.p) = \sum_v C(v\ *\ w'.p)P(s_r|v)$, where the sum is over all pronunciations v in the lexicon, the symbol "$*$" in $C(v\ *\ w'.p)$ denotes either s or n, and $P(s_r|v)$ is computed using Equation ((5.5)). λ_3 is a smoothing constant that we set to 2 for experiments reported here.

Putting it all together, we estimate the inter-word silence (or lack thereof) given the neighbouring words as follows

$$P(s \,|\, w.p_i, w'.p_j) \approx P(s_r \,|\, w.p_i) \times F(s_l \,|\, w'.p_j), \text{ and}$$

$$P(n \,|\, w.p_i, w'.p_j) \approx P(n_r \,|\, w.p_i) \times F(n_l \,|\, w'.p_j).$$

5.4.3 Pronunciation selection

Pruning of pronunciations is performed after estimating the pronunciation and silence probabilities. For each word, we only keep the pronunciations with probability

higher than 0.6. Note that this is the "max-normalized" probability, therefore each word may have multiple pronunciations after pruning.

5.5 Iterative framework

Our pronunciation generation procedure follows a general iterative learning schedule. Unlike [78] and [94], where the iterative procedure is more focused on selecting the best pronunciation given the pronunciations generated by the G2P model, our framework attempts to refine the phonetic transcripts generated by the speech recognizer, which will then be used to train the G2P model together with the small expert lexicon, as described in Section 5.3.1.

Figure 5.1 illustrates our proposed iterative framework for lexicon generation and acoustic modeling. We start from a small expert pronunciation lexicon of 500 or 1000 words. Pronunciations of the words that are already in the small expert lexicon are kept untouched throughout the whole process. The small expert lexicon is used to train an initial G2P model, which is then applied to the words in the training transcripts to create an extended lexicon. We generate at most 5 pronunciations for each word in the training transcripts, if it is not already in the expert lexicon. Note that the initial extended lexicon typically cannot cover all the words in the training transcripts, due to the large number of unseen graphemes that are not covered by the small expert lexicon. Those words are treated as OOV words in the first iteration.

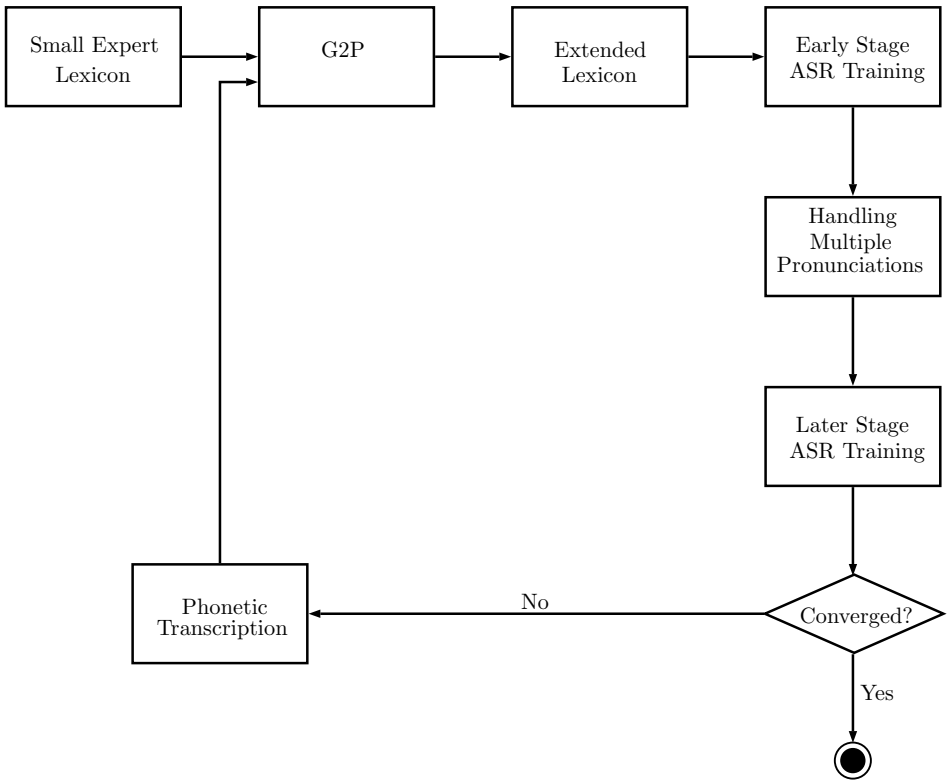

Figure 5.1: An iterative framework for lexicon generation and acoustic modeling.

For later iterations, since we add phonetic transcripts for G2P training, the extended lexicon will be able to cover all the words in the training transcripts.

Early stages of acoustic model training are carried out with this extended lexicon, typically till the speaker adaptive training stage. Since the extended lexicon contains multiple pronunciations for each word, we estimate the pronunciation and inter-word silence probabilities as described in Section 5.4, which has found its success when pronunciation variants exist in the lexicon [104]. We further prune away pronunciations with low (less than 0.6) "max-normalized" probabilities. The updated lexicon, as well as the pronunciation and inter-word silence probabilities are incorporated into a new

lexicon transducer for later stages of acoustic modeling as well as decoding.

After training the speech recognition system, if the word error rate (WER) performance on some held-out dataset converges to a stable point, we stop the process. Otherwise, we use the trained system to decode the training data into phoneme sequences, which will be sent back to re-train the G2P system, and start the process again, till convergence. Note that in our experiments we set the language model weight to 0 during the phonetic decoding process, but it may also make sense to use a n-gram phone language model.

5.6 Experimental setup

5.6.1 Corpus

We evaluate the proposed framework using IARPA Babel language Cantonese [8]. The 10 hour subset of the full language pack is used to conduct our experiments. The language pack also comes with an expert lexicon that contains $5.9K$ lexical entries and $5K$ unique words, which covers all the words in the training transcripts. The number of unique characters covered by this lexicon is $2K$. To simulate the situation where only a small expert lexicon is available for speech recognition system development, we create two seed lexicons by randomly selecting 500 and 1000 words from the original expert lexicon. These two lexicons have $0.63K$ and $0.95K$ unique

[8]Language collection release `IARPA-babel101b-v0.4c`.

characters respectively. For the keyword search task, the "dev" keyword list [9] is used for evaluation.

5.6.2 System description

We use the open source toolkit Kaldi [65] for all our system development and experiments. Standard PLP analysis is employed to extract 13 dimensional acoustic feature, and a maximum likelihood acoustic training recipe is followed to train speaker adaptive models. This is followed by the modeling of pronunciation and inter-word silence probabilities, which updates the lexicon and prunes away unlikely pronunciations. From this point, two different systems are trained: a hybrid deep neural network (DNN) system and a subspace Gaussian mixture model (SGMM) system with boosted maximum mutual information (BMMI) training. For the keyword search task, lattice indexing is further performed after decoding to convert lattice of each utterance into a finite-state acceptor with the posterior score, start-time and end-time for each word encoded as a 3-dimensional weight. An inverted index is then created from these individual acceptors, with paths to accept every possible word sequence in the original lattices. This way, keyword search can be done by composing the keyword acceptor with the inverted index. For details of speech recognition and keyword search systems, readers are referred to [77, 45, 52].

[9] Keyword list `IARPA-babel101b-v0.4c_conv-dev.kwlist.xml`.

5.7 Results

We report word error rate (WER) for the speech recognition task, and actual term-weighted value (ATWV) for the keyword search task. WERs are reported in percentage.

5.7.1 Performance

	$L1$ (500 words)		$L2$ (1000 words)	
	WER	ATWV	WER	ATWV
baseline	75.6	0.0279	68.1	0.0908
Iteration0	72.9	0.0676	65.7	0.1381
Iteration1	61.6	0.1395	58.9	0.1787
Iteration2	61.8	0.1519	58.4	0.1859
Iteration3	**60.9**	**0.1538**	**57.8**	**0.1891**
oracle	*54.2*	*0.2377*	*54.2*	*0.2377*

Table 5.2: WER and ATWV performance of lexicons from different iterations, SGMM BMMI system.

	$L1$ (500 words)		$L2$ (1000 words)	
	WER	ATWV	WER	ATWV
baseline	74.3	0.0332	67.7	0.1086
Iteration0	71.6	0.0862	65.1	0.1495
Iteration1	61.8	0.1491	58.4	0.1959
Iteration2	62.1	0.1659	59.1	0.2108
Iteration3	**61.4**	**0.1764**	**57.2**	**0.2216**
oracle	*53.8*	*0.2623*	*53.8*	*0.2623*

Table 5.3: WER and ATWV performance of lexicons from different iterations, DNN system.

Table 5.2 and Table 5.3 present the WER and ATWV performance of the auto-

matically generated lexicons from various iterations. We evaluate two seed lexicons, one with size 500 ($L1$), and the other with size 1000 ($L2$), as described in Section 5.6.1. The "baseline" in the two tables corresponds to the speech recognition system trained only with the seed lexicon, and the "oracle" represents the system trained with the full $5K$ words expert lexicon. Iteration0 corresponds to the system trained with the initial extended lexicon, which does not contain all the words from the training transcripts. Starting from Iteration1, phonetic transcripts can be generated by the system, so the G2P model trained on that can generate pronunciations for all the words. The iterative procedure is carried out for three times, excluding Iteration0. Further iterations do not help in our experiments.

Let us start by looking at the ATWV numbers. It is clear from the table that ATWV is increasing through the iterations. This implies that the pronunciation lexicon is indeed improving through the iterative framework. The WER improvements, however, are not as steady as the ATWV improvements, although the best numbers are all achieved at Iteration3. We suspect that this is partly because of decoding noise, and partly because we discard the pronunciations when we update the lexicon with the newly trained G2P model. It might make sense to combine the old and new lexicons, and let the speech recognizer pick the best pronunciation.

Note that there is a performance jump from Iteration0 to Iteration1. This is because the initial extended lexicon trained from the expert lexicon only covers a small portion of words in the training transcripts, while the lexicon in Iteration1

covers all the words.

It is encouraging to see that, starting from a lexicon of 1000 words, which is just one-fifth of the original lexicon, we are able to achieve a WER of 57.2 and ATWV of 22.16 with our proposed iterative framework. This closes 76% of the WER gap and 74% of the ATWV gap between the baseline and the oracle lexicon.

5.7.2 Impact of phonetic transcripts quality

	SAT transcripts		DNN transcripts	
	WER	ATWV	WER	ATWV
baseline	67.7	0.1086	67.7	0.1086
Iteration0	65.1	0.1495	65.1	0.1495
Iteration1	60.2	0.1883	58.4	0.1959
Iteration2	59.4	0.2046	59.1	0.2108
Iteration3	**59.2**	**0.2066**	**57.2**	**0.2216**
oracle	*53.8*	*0.2623*	*53.8*	*0.2623*

Table 5.4: WER and ATWV performance on the DNN system, with lexicons trained on the SAT transcripts or DNN transcripts ($L2$ seed lexicon).

Phonetic transcripts of training data can either be generated by a model from speaker adaptive training (SAT), or by DNN model. The later typically yields better quality. Table 5.4 gives the performance comparison of lexicons generated by the two models. The table suggests that generating high quality phonetic transcripts is essential in our framework.

5.8 Summary

We explored building grapheme-based speech recognition and keyword search systems for low resource alphabetic languages. Experiments on 6 Babel languages show that one can avoid the expert pronunciation lexicon by building grapheme-based systems, with only a small performance degradation. We then presented an iterative framework that is capable of generating pronunciation lexicons for logographic languages. This allows us to rapidly build speech recognition systems with limited expert lexicon for those languages. Experiments on Cantonese suggest that by using a seed lexicon of 1000 words, we are able to achieve reasonably well speech recognition and keyword search performance, when compared with an expert-crafted lexicon of 5000 words.

Chapter 6

Deep Neural Network Keyword Spotter

The recent rise of speech recognition technologies on mobile devices calls for keyword spotting algorithms with a small memory footprint, low computational cost, and high precision, which can be used to active the device without a touch. In this chapter, we propose a simple approach based on deep neural networks for this purpose. A deep neural network is trained to directly predict the keyword(s) or subword units of the keyword(s) followed by a posterior handling method producing a final confidence score. Keyword recognition results achieve 45% relative improvement with respect to a competitive Hidden Markov Model-based system, while performance in the presence of babble noise shows 39% relative improvement.

6.1 Related work

Thanks to the rapid development of smartphones and tablets, interacting with technology using voice is becoming commonplace. For example, Google offers the ability to search by voice [3] on Android devices and Apple's iOS devices are equipped with a conversational assistant named Siri. These products allow a user to tap a device and then speak a query or a command.

We are interested in enabling users to have a fully hands-free experience by developing a system that listens continuously for specific keywords to initiate voice input. This could be especially useful in situations like driving. The proposed system must be highly accurate, low-latency, small-footprint, and runs in computationally constrained environments such as modern mobile devices. Running the system on the device avoids latency and power implications with connecting to the server for recognition.

Keyword Spotting (KWS) aims at detecting predefined keywords in an audio stream, and it is a potential technique to provide the desired hands-free interface. There is an extensive literature in KWS, although most of the proposed methods are not suitable for low-latency applications in computationally constrained environments. For example, several KWS systems [68, 109, 72] assume offline processing of the audio using large vocabulary continuous speech recognition systems (LVCSR) to generate rich lattices. In this case, their task focuses on efficient indexing and searching for keywords in the lattices. These systems are often used to search large

databases of audio content. We focus instead on detecting keywords in the audio stream without any latency.

A commonly used technique for keyword spotting is the Keyword/Filler Hidden Markov Model (HMM) [21, 18, 20, 25, 110]. Despite being initially proposed over two decades ago, it remains highly competitive. In this generative approach, an HMM model is trained for each keyword, and a filler model HMM is trained from the non-keyword segments of the speech signal (fillers). At runtime, these systems require Viterbi decoding, which can be computationally expensive depending on the HMM topology. Other recent work explores discriminative models for keyword spotting based on large-margin formulation [111, 112] or recurrent neural networks [113, 114]. These systems show improvement over the HMM approach. The large-margin formulation-based methods, however, require processing of the entire utterance to find the optimal keyword region, which increases detection latency. We have also been working on recurrent neural networks for keyword spotting, but with a different application, which will be described in Chapter 7.

We propose a simple discriminative KWS approach based on deep neural networks that is appropriate for mobile devices. We refer to it as Deep KWS . A deep neural network is trained to directly predict the keyword(s) or subword units of the keyword(s) followed by a posterior handling method producing a final confidence score. In contrast with the HMM approach, this system does not require a sequence search algorithm (decoding), leading to a significantly simpler implementation, reduced run-

time computation, and smaller memory footprint. It also makes a decision every 10 ms, minimizing latency. We show that the Deep KWS system outperforms a standard HMM-based system on both clean and noisy test sets, even when a smaller amount of data is used for training.

6.2 Deep KWS system

Our proposed Deep KWS framework is illustrated in Figure 6.1.

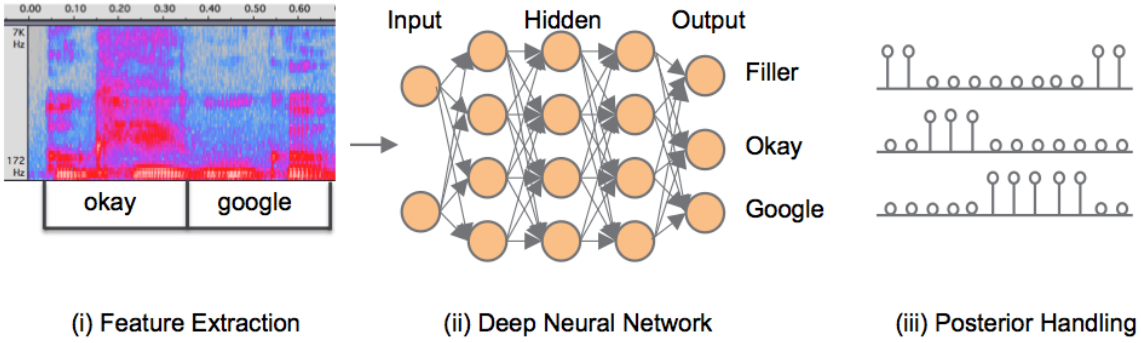

Figure 6.1: Framework of Deep KWS system, components from left to right: (i) Feature Extraction (ii) Deep Neural Network (iii) Posterior Handling.

The framework consists of three major components: (i) a feature extraction module, (ii) a deep neural network, and (iii) a posterior handling module. The feature extraction module (i) performs voice-activity detection and generates a vector of features every frame (10 ms). These features are stacked using the left and right context to create a larger vector, which is fed as input to the DNN (Section 6.2.1). We train a DNN (ii) to predict posterior probabilities for each output label from the stacked features. These labels can correspond to entire words or sub-words for the keywords

(Section 6.2.2). Finally, a simple posterior handling module (iii) combines the label posteriors produced every frame into a confidence score which is then used for detection (Section 6.2.3).

In the example of Figure 6.1, the audio contains the key-phrase "okay google". The DNN in this case only has 3 output labels: "okay", "google", and "filler", and it generates frame-level posterior scores shown in (iii). The posterior handling module combines these scores to provide a final confidence score for the given window.

6.2.1 Feature extraction

The feature extraction module is common to our proposed Deep KWS system and the baseline HMM system.

To reduce computation, we use a voice-activity detection system and only run the KWS algorithm in voice regions. The voice-activity detector, described in [115], uses 13-dimensional PLP features and their deltas and double-deltas as input to a trained 30-component diagonal covariance GMM, which generates speech and non-speech posteriors at every frame. This is followed by a hand-tuned state machine (SM), which performs temporal smoothing by identifying regions where speech posteriors exceed a threshold.

For the speech regions, we generate acoustic features based on 40-dimensional log-filterbank energies computed every 10 ms over a window of 25 ms. Contiguous frames are stacked to add sufficient left and right context. The input window is asymmetric

since each additional frame of future context adds 10 ms of latency to the system. For our Deep KWS system, we use 10 future frames and 30 frames in the past. For the HMM baseline system we use 5 future frames and 10 frames in the past, as this provided the best trade-off between accuracy, latency, and computation [116].

6.2.2 Deep neural network

The deep neural network model is a standard feed-forward fully connected neural network with k hidden layers and n hidden nodes per layer, each computing a non-linear function of the weighted sum of the output of the previous layer. The last layer has a softmax which outputs an estimate of the posterior of each output label. For the hidden layers, we have experimented with conventional logistic and rectified linear unit (ReLU) functions [117], and consistently found that ReLU outperforms logistic on our development set, while reducing computation. We present results with ReLU activations only.

The size of the network is also dictated by the number of output labels. In the following sub-sections we describe in detail the label generation and training for our neural network. We also describe a learning technique that further improves the KWS performance.

6.2.2.1 Labeling

For our baseline HMM system, as in previous work [25, 110, 118] the labels in the output layer of the neural network are context-dependent HMM states. More specifically the baseline system uses 2002 context dependent states selected as described in [116].

For the proposed Deep KWS , the labels can represent entire words or sub-word units in the keyword/key-phrase. We report results with full word labels, as these outperform sub-word units. These labels are generated at training time via forced alignment using our $50M$ parameter LVCSR system [119]. Using entire word labels as output for the network, instead of the HMM states, has several advantages: (i) smaller inventory of output labels reduces the number of neural network parameters in the last layer, which leads to less computation, (ii) a simple posterior handling method can be used to make a decision (as explained in Section 6.2.3), and (iii) whole word models can achieve better performance, assuming the training data is adequate for each word label.

6.2.2.2 Training

Suppose p_{ij} is the neural network posterior for the i^{th} label and the j^{th} frame x_j (see Section 6.2.1), where i takes values between $0, 1, ..., n-1$, with n the number of total labels and 0 the label for non-keyword. The weights and biases of the deep neural network, θ, are estimated by minimizing the cross-entropy training criterion

over the labeled training data $\{x_j, i_j\}_j$.

$$F(\theta) = \sum_j \log p_{i_j j}. \tag{6.1}$$

The optimization is performed with the software framework DistBelief [120, 121] that supports distributed computation on multiple CPUs for deep neural networks. We use asynchronous stochastic gradient descent with an exponential decay for the learning rate.

6.2.2.3 Transfer learning

Transfer learning refers to the situation where (some of) the network parameters are initialized with the corresponding parameters of an existing network, and are not trained from scratch [122, 123]. Here, we use a deep neural network for speech recognition with suitable topology to initialize the hidden layers of the network. All layers are updated in training. Transfer learning has the potential advantage that the hidden layers can learn a better and more robust feature representation by exploiting larger amount of data and avoiding bad local optima [122]. In our experiments we find this to be the case.

6.2.3 Posterior handling

The DNN explained in Section 6.2.2 produces frame level label posteriors. In this section we discuss our proposed simple, yet effective, approach to combine DNN posteriors into keyword/key-phrase confidence scores. A decision then will be made if the confidence exceeds some predefined threshold. We describe the confidence computation assuming a single keyword. However, it can be easily modified to detect multiple keywords simultaneously.

6.2.3.1 Posterior smoothing

Raw posteriors from the neural network are noisy, so we smooth the posteriors over a fixed time window of size w_{smooth}. Suppose p'_{ij} is the smoothed posterior of p_{ij} (Equation (6.1)). The smoothing is done with the following formula

$$p'_{ij} = \frac{1}{j - h_{smooth} + 1} \sum_{k=h_{smooth}}^{j} p_{ik} \tag{6.2}$$

where $h_{smooth} = \max\{1, j - w_{smooth} + 1\}$ is the index of the first frame within the smoothing window.

6.2.3.2 Confidence

The confidence score at j^{th} frame is computed within a sliding window of size w_{max}, as follows

$$confidence = \sqrt[n-1]{\prod_{i=1}^{n-1} \max_{h_{max} \leq k \leq j} p'_{ik}} \tag{6.3}$$

where p'_{ij} is the smoothed state posterior in Equation (6.2), $h_{max} = \max\{1, j - w_{max} + 1\}$ is the index of the first frame within the sliding window. We use $w_{smooth} = 30$ frames, and $w_{max} = 100$, as this gives the best performance on our development set; the performance however is not very sensitive to the window size. Also Equation (6.3) does not enforce the order of the label sequence, we do not bother enforcing it because the stacked frames fed as input to the neural network help encode contextual information.

6.3 Baseline HMM KWS system

We implement a standard Keyword-Filler Hidden Markov Model as our baseline. The basic idea is to create a HMM for the keyword and a HMM to represent all non-keyword segments of the speech signal (filler model). There are several choices for the filler model, from fully connected phonetic units [18] to a full LVCSR system where the lexicon excludes the keyword [124]. Obviously, the latter approach yields a better filler model, however it requires higher computational cost at runtime, and significantly larger memory footprint. Given the constraints of our application, we

implemented a triphone-based HMM model as filler. In contrast to previous work [18, 124], our implementation uses a Deep Neural Network to compute the HMM state densities.

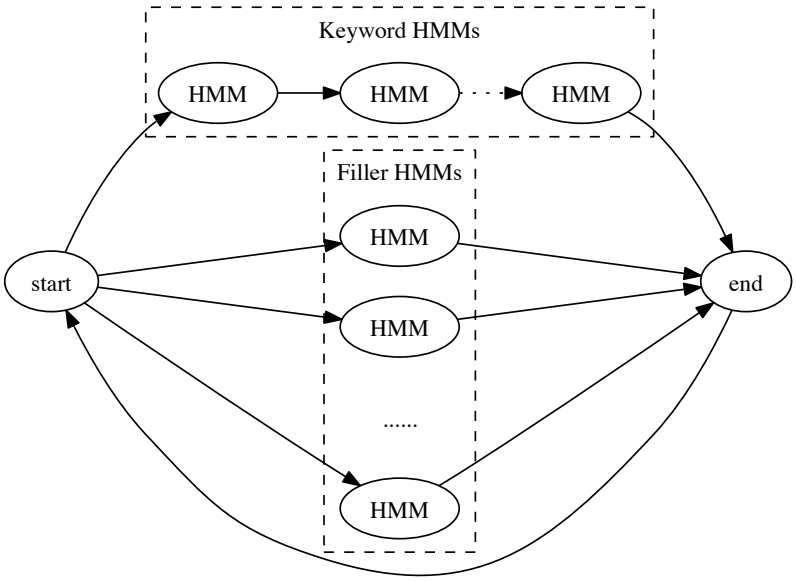

Figure 6.2: HMM topology for KWS system, which consists of a keyword model and a triphone filler model.

The Keyword-Filler HMM topology is shown in Figure 6.2. Keyword detection is achieved by running Viterbi decoding with this topology and checking if the best path passes through the Keyword HMM or not. The trade-off between false alarms (a keyword is not present but the KWS system gives a positive decision) and false rejects (a keyword is present but the KWS system gives a negative decision) is controlled by the transition probability between keyword and filler models. High transition probability leads to high false alarm rate and vice versa.

An important advantage of the Keyword-Filler model is that it does not require keyword-specific data at training time. It simply learns a generative model for all triphone HMM states through likelihood maximization on general speech data. Knowledge of the keyword can be introduced only at runtime, by specifying the keyword in the decoder graph. However, if keyword-specific data is available for training, one can improve system performance using transfer learning (Section 6.2.2), i.e., by initializing the acoustic model network with a network trained on the general speech data and then continue training it using the keyword-specific data.

6.4 Experimental setup

Experiments are performed on a data set which combines real voice search queries as negative examples and phrases including the keywords, sometimes followed by queries, as positive examples. A full list of the keywords evaluated is shown in Table 6.1. We train a separate Deep KWS and build a separate Keyword-Filler HMM KWS system for each key-phrase. Results are presented in the form of a modified receiver operating characteristic (ROC) curves, where we replace true positive rate with the false reject rate on Y-axis. Lower curves are better. The ROC for the baseline system is obtained by sweeping the transition probability for the Keyword HMM path in Figure 6.2. For the Deep KWS system, the ROC is obtained by sweeping the confidence threshold. We generate a curve for each keyword and average the curves

answer call	dismiss alarm
go back	ok google
read aloud	record a video
reject call	show more commands
snooze alarm	take a picture

Table 6.1: Keywords used in evaluation.

vertically (at fixed FA rates) over all keywords tested. Detailed comparison is given at 0.5% FA rate, which is a typical operating point for practical applications.

We compare the Deep KWS system and the HMM system with different size of neural networks (Section 6.5.2), evaluate the effect of transfer learning for both systems (Section 6.5.1), and show performance changes in the presence of babble noise (Section 6.5.3).

6.4.1 Data

We use two sets of training data. The first set is a general speech corpus, which consists of $3,000$ hours of manually transcribed utterances (referred to as VS data). The second set is a keyword specific data (referred to as KW data), which includes around $2.3K$ training examples for each keyword, and $133K$ negative examples comprised of anonymized voice search queries or other short phrases. For the keyword "*okay google*", $40K$ positive examples are available for training.

The evaluation set contains roughly $1K$ positive examples for each keyword and $70K$ negative examples, representing 1.4% of positive to negative ratio, to match

expected application usage. Again, for keyword "*okay google*" we use instead $2.2K$ positive examples. The noisy test set is generated by adding babble noise to this test set with a $10db$ Signal to Noise Ratio (SNR). Finally, we use a similar size non-overlapping set of positive and negative examples as development set to tune decoder parameters and DNN input window size parameters.

6.5 Results

6.5.1 Initial results

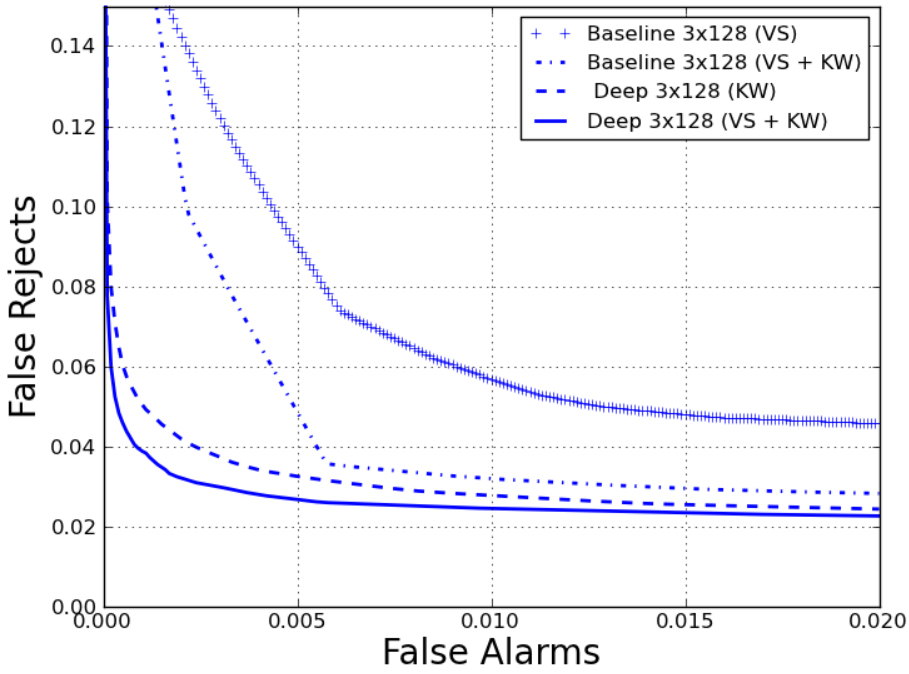

Figure 6.3: HMM vs. Deep KWS system with 3 hidden layers, 128 hidden nodes neural network.

We first evaluate the performance of the smaller neural networks trained for the

baseline HMM and the Deep KWS systems. Both systems use the frontend described in 6.2.1. They both use a network with 3 hidden layers and 128 hidden nodes per layer with ReLU non-linearity. However, the number of parameters for the two networks differs. The DNN acoustic model used for the baseline HMM system uses an input window size of 10 left frames and 5 right frames, and outputs $2,002$ HMM states, resulting in around $373K$ parameters. The Deep KWS uses instead 30 left frames and 10 right frames, but only produces word labels reducing the output label inventory to 3 or 4 depending on the key-phrase evaluated. The total number of parameters for Deep KWS is no larger than $244K$ parameters.

Figure 6.3 shows the performance for both systems. Baseline 3x128 (VS) refers to the HMM system with a DNN acoustic model trained on the voice search corpus. Baseline 3x128 (VS + KW) is this same system after adapting the DNN acoustic model using keyword specific data. Deep 3x128 (KW) refers to the proposed Deep KWS system trained on keyword specific data. Finally, Deep 3x128 (VS + KW) shows the performance when we initialize the Deep 3x128 KW network with a network trained on VS data as explained in Section 6.2.2.

It is clear from the results that the proposed Deep KWS outperforms the baseline HMM KWS system even when it is trained with less data and has fewer number of parameters. For example, see Deep 3x128 (KW) vs Baseline 3x128 (VS + KW) in Figure 6.3. The gains are larger at very low false alarm rate, which is a desirable operating point for our application. At 0.5% FA rate, Deep 3x128 (VS + KW)

system achieves 45% relative improvement with respect to Baseline 3x128 (VS + KW). Training a network on the KW data takes only a couple of hours, while training it on VS + KW takes about a week using our DistBelief framework described in Section 6.2.2.

6.5.2 Model size

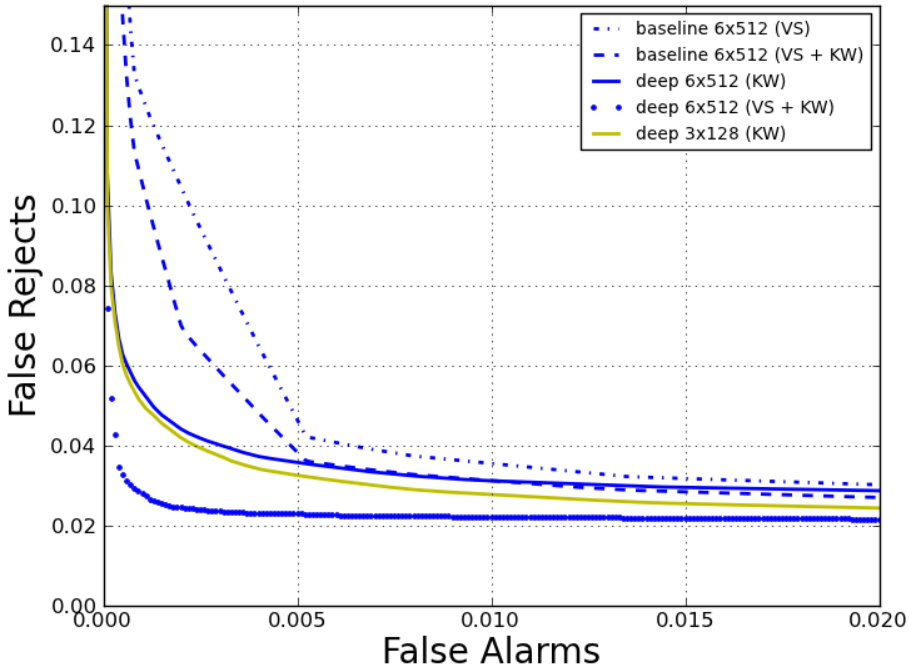

Figure 6.4: HMM vs. Deep KWS system with 6 hidden layers, 512 hidden nodes neural network.

Figure 6.4 presents the performance when evaluating both systems with a 6x512 network. In this case the number of parameters for the baseline increases to $2.6M$ while that of the Deep models reaches $2.1M$. Deep 6x512 (KW) system, actually performs worse than the smaller 3x128 models, we conjecture this is due to not

having enough KW data to train the larger number of parameters. However when both systems are trained on VS + KW data, we observe a consistent improvement with respect to their corresponding 3x128 systems. Here again, the Deep KWS system has superior performance to the baseline.

6.5.3 Noise robustness

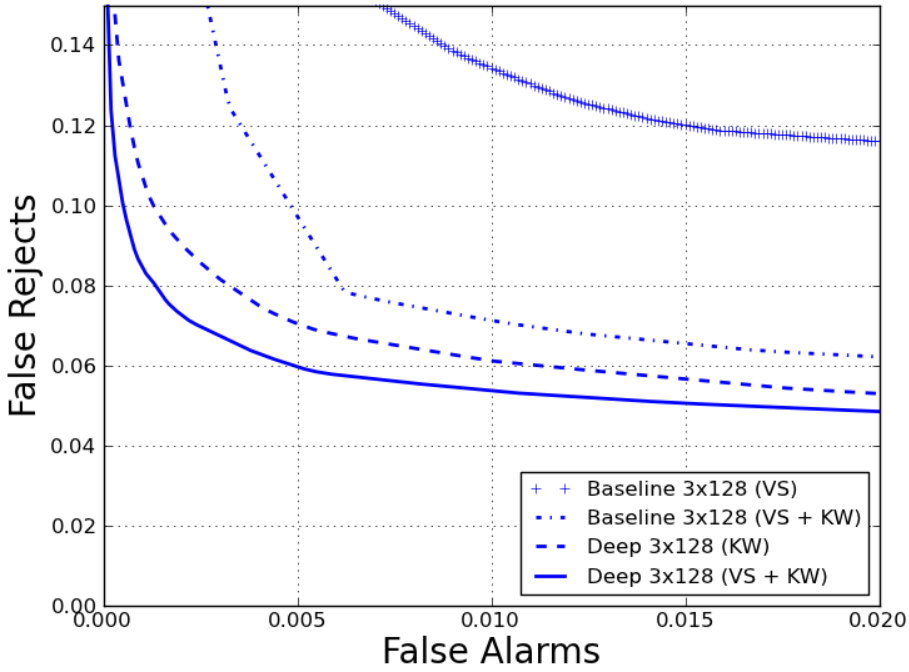

Figure 6.5: HMM vs. Deep KWS system with 3 hidden layers, 128 hidden nodes neural network on NOISY data.

We also test the same models on a noisy test set, generated by adding babble noise to the original test set with a 10*db* SNR. Results are shown in Figure 6.5 and Figure 6.6. Comparing Baseline 3x128 (VS + KW) in Figure 6.3 and Figure 6.5, at 0.5% FA rate, the FR rate of the HMM doubles from 5% FR to 10% FR. The Deep KWS

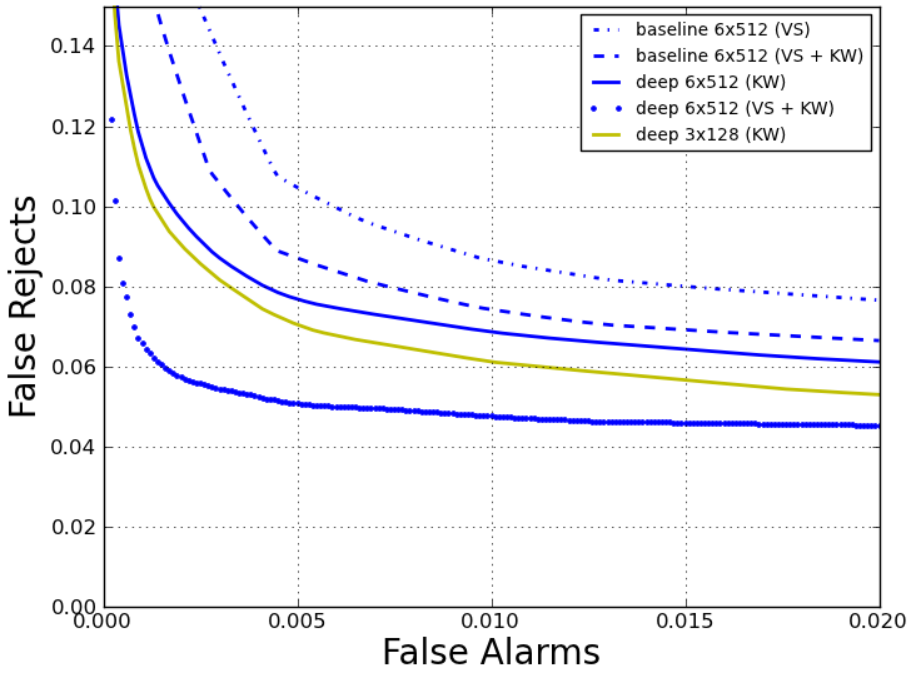

Figure 6.6: HMM vs. Deep KWS system with 6 hidden layers, 512 hidden nodes neural network on NOISY data.

system suffers similar degradation. However it achieves 39% relative improvement with respect to the baseline.

6.6 Summary

We have presented a new deep neural network-based framework for keyword spotting. Experimental results show that the proposed framework outperforms the standard HMM-based system on both clean and noisy conditions. We further demonstrate that a Deep KWS model trained with only the KW data yields better search performance then the baseline HMM KWS system trained with both KW and VS data. The

Deep KWS system also leads to a simpler implementation by removing the necessity of a decoder, as well as reduced runtime computation, and a smaller model, and thus is favored for our embedded application.

Since the detection application we are working on only requires a real-time YES/NO decision, the proposed framework in this work does not model the keyword ending time explicitly. We will extend the proposed method to model keyword boundary in the future work.

Chapter 7

Long Short-Term Memory Feature Extractor

In this chapter we present a novel approach to query-by-example keyword spotting (KWS) using a long short-term memory (LSTM) recurrent neural network-based feature extractor. In our approach, we represent each keyword using a fixed-length feature vector obtained by running the keyword audio through a word-based LSTM acoustic model. We use the activations prior to the softmax layer of the LSTM as our keyword-vector. At runtime, we detect the keyword by extracting the same feature vector from a sliding window and computing a simple similarity score between this test vector and the keyword vector. With clean speech, we achieve 86% relative false rejection rate reduction at 0.5% false alarm rate when compared to a competitive phoneme posteriorgram with dynamic time warping KWS system, while the reduction

in the presence of babble noise is 67%. Our system has a small memory footprint, low computational cost, and high precision, making it suitable for on-device applications.

7.1 Related work

With the growing popularity of voice control in mobile devices, the need for high performance, small footprint, and low computational cost keyword spotting (KWS) methods is becoming increasingly important [49]. In such applications, KWS usually serves as a frontier of voice search: it listens to the audio continuously and initiates the voice search if a specific keyword is detected, thus providing a fully hands-free experience when interacting with devices.

A common use is to have a pre-defined keyword to activate devices. For example, Google's voice search [3] uses the phrase "Okay Google" to initiate the search interface and Apple's conversational assistant Siri features the keyword sequence "Hey Siri". However, this general phrase makes the experience less personal, and usually requires additional speaker identification if the user does not want others to easily activate their device. In this work we seek a keyword spotting method that allows users to define their own keyword; for example, a user may select a keyword by saying the word or phrase a few times during enrollment. After enrollment, the user-specified keyword can be used to activate the device.

We are interested in small memory footprint and low computational cost solu-

tions, suitable for on-device applications. While KWS is an active research area, most techniques are not suitable with our constraints. A common KWS approach is the Keyword/Filler Hidden Markov Model (HMM) [21, 18, 20, 25, 110]. It first builds a special decoding graph that contains both keywords and filler words, and then uses Viterbi decoding to determine the best path through the graph. This requires prior knowledge of the keywords, which is not appropriate for our problem. Another area of research relies on using large vocabulary continuous speech recognition (LVCSR) systems to decode the audio into lattices or confusion networks, and search the keywords from there [68, 109, 52]. The keywords, however, are only limited to the words that are already defined in the LVCSR vocabulary. While effort has been made to make such systems keyword-independent [72, 125, 85, 45], there is usually a performance degradation when the keywords are out of vocabulary. In addition, these approaches are relatively expensive because of the LVCSR system.

Given our goal of detecting user-specified keywords, query-by-example (QbyE) is the most appropriate KWS technique. QbyE methods usually take several examples of the keywords as templates, and compare the test audio segment against the templates to make detection decisions. In [10] example keywords are decoded with an LVCSR system to get their lattice representation as templates. This is computationally expensive since the LVCSR system involves speaker adaptation, discriminative features and model transformations [126]. It is also inappropriate for our memory constraints. In [13] graph-based method is proposed to embed audio segments of

arbitrary length into a fixed-dimensional space, but dynamic time warping (DTW) is performed between the test audio segment and all the training segments in order to compute the embedding, which can be slow given large number of training segments. In [6, 8], Gaussian or phoneme posteriorgrams are generated as templates from example keywords, and DTW is used to compare the templates. Though this type of DTW-based method has well-known inadequacies [13], it is the most appropriate KWS baseline for our application.

Given the constraints of our problem and the limitations of previous QbyE approaches, we propose a novel LSTM-based feature extractor. In our approach, first an LSTM is trained with whole word output targets. Next, for each audio segment, a fixed-length representation of the audio is created by taking the activations from the last hidden layer of the LSTM and stacking them over a fixed number of frames. This embeds audio segments of different length into a fixed-dimensional space, therefore vector distance can be used for similarity measurement. Our method only requires a forward pass computation of the neural network, followed by a vector distance computation, and therefore is more efficient than [10] where an LVCSR is involved and [13] where multiple DTW computations are necessary. It also requires less computation than [6, 8] since vector distance is used instead of DTW.

To understand the behavior of our proposed LSTM KWS system, we first conduct experiments on a clean enrollment and evaluation set. We find that the LSTM KWS system reduces the false rejection rate by 86% relatively at 0.5% false alarm rate,

when compared to the DTW KWS system. Next, we explore its behavior at the presence of noise. We add 10 dB babble noise to the evaluation set, and achieve 67% relative false rejection rate reduction at the same false alarm rate. We further add 10 dB cafe noise to the enrollment set, and the reduction is 37%. Our proposed system is consistently better than the standard DTW KWS system in various environments.

7.2 LSTM feature extractor system

The general idea of our proposed system is to embed audio segments of varying lengths into a fixed-dimensional representation. Given the success of deep learning [127], and the power of LSTMs for sequence modeling [128], we choose an LSTM to learn this embedding. The LSTM is attractive because the state of the LSTM can encode information about past history, and intuitively after processing a complete audio segment, this LSTM state encodes information about the complete sequence. This idea was motivated by face verification work in [129], with the key difference in our work being that we use the LSTM state to embed a fixed-length representation of a variable length input sequence, as opposed to having a fixed-length input representation and thus using a convolutional neural network.

7.2.1 Feature extraction

To reduce computation, we use a voice-activity detection system and only run the KWS algorithm in voice regions. The voice-activity detector, described in [115], uses 13-dimensional PLP features along with their deltas and double-deltas as input to a 30-component diagonal covariance GMM-trained system, which generates speech and non-speech posteriors at every frame. This is followed by a hand-tuned state machine, which performs temporal smoothing by identifying regions where speech posteriors exceed a threshold.

For the speech regions, we generate 40-dimensional log-filterbank energies computed every 10 ms over a window of 25 ms. For the deep neural network (DNN)-based system, contiguous frames are stacked to add sufficient left and right context. The input window is asymmetric since each future frame adds 10 ms of latency to the system. We use 5 future frames and 10 past frames, to provide the best trade-off between accuracy, latency, and computation [116]. For the LSTM-based system, no frames are stacked.

7.2.2 Long short-term memory

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) used to model long-range dependencies [128]. RNNs are used for a variety of sequence-labeling tasks. Specifically, given an input sequence $\mathbf{x} = \{x_1, \ldots, x_T\}$, an RNN

computes a sequence of hidden vectors $\mathbf{h} = \{h_1, \ldots, h_T\}$ and outputs $\mathbf{y} = \{y_1, \ldots, y_T\}$ for all time steps $\{1, \ldots, T\}$. An LSTM uses special memory cells to model the temporal sequence, which allows it to more effectively exploit long-range context than an RNN.

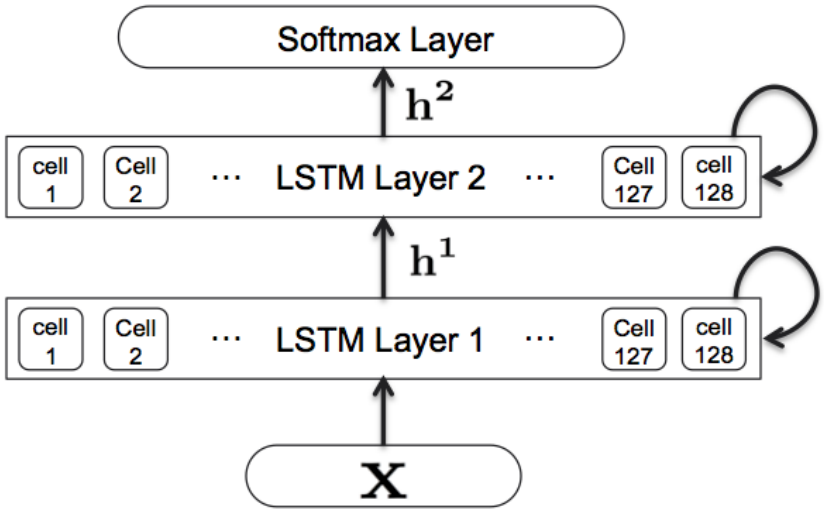

Figure 7.1: LSTM architecture.

In our work, we train a 2-layer LSTM, as shown in Figure 7.1. The network has $15k$ output targets, representing whole word units. We exclude the evaluation keywords from the $15k$ output targets to make the feature extractor independent of the test keywords. The network is trained with the cross-entropy criterion, using the asynchronous stochastic gradient descent (ASGD) optimization strategy described in [121]. The weights in each network are randomly initialized. The learning rate is chosen specific to each network, and is chosen to be the largest value such that training remains stable. Learning rates are exponentially decayed during training.

7.2.3 LSTM feature extractor

After training the LSTM, given an audio segment, we use the LSTM to determine a fixed-dimensional feature vector to represent the audio signal. Our fixed length representation is created by removing the softmax layer and using the hidden units from the 2^{nd} LSTM layer. More specifically, given an acoustic feature $\mathbf{x}$ with T frames, the hidden units from the second layer of the LSTM are given as

$$\mathbf{h^2} = \{h_1^2, \ldots, h_T^2\}. \tag{7.1}$$

Here $h_i^2 \in \Re^n$, where n represents the number of LSTM cells. As each $h_i^2 \in \mathbf{h^2}$ encodes information up to time i, there is no need to use all state vectors $\mathbf{h^2}$. We create a fixed-length representation $\mathbf{f}$ by choosing the last k state vectors, as denoted by

$$\mathbf{f} = \{h_{T-k+1}^2, \ldots, h_T^2\} \tag{7.2}$$

Here $\mathbf{f}$ has a fixed dimensionality of $n \times k$. The parameter k can be estimated from the enrollment templates. In our experiments, we choose k to be the averaged number of frames of all the templates as we want to encode as much information as possible. Zeros are padded in front of $\mathbf{f}$ if the segment length T is smaller than the desired template length k.

7.2.4 LSTM KWS

Now that we have described how to create a fixed-length representation from an audio segment, in this section we describe the enrollment and verification phases of our LSTM KWS system, as illustrated in Figure 7.2. In the enrollment phase, an utterance is spoken three times. For each utterance, the activations from the last hidden layer of the LSTM are calculated per frame, and the last k activations are used to create a fixed feature vector $\mathbf{f}$. Note that since we have three enrollment templates, we can keep all the three as separate templates, or average them into one single vector. We will show in our experiments of how template averaging impacts performance.

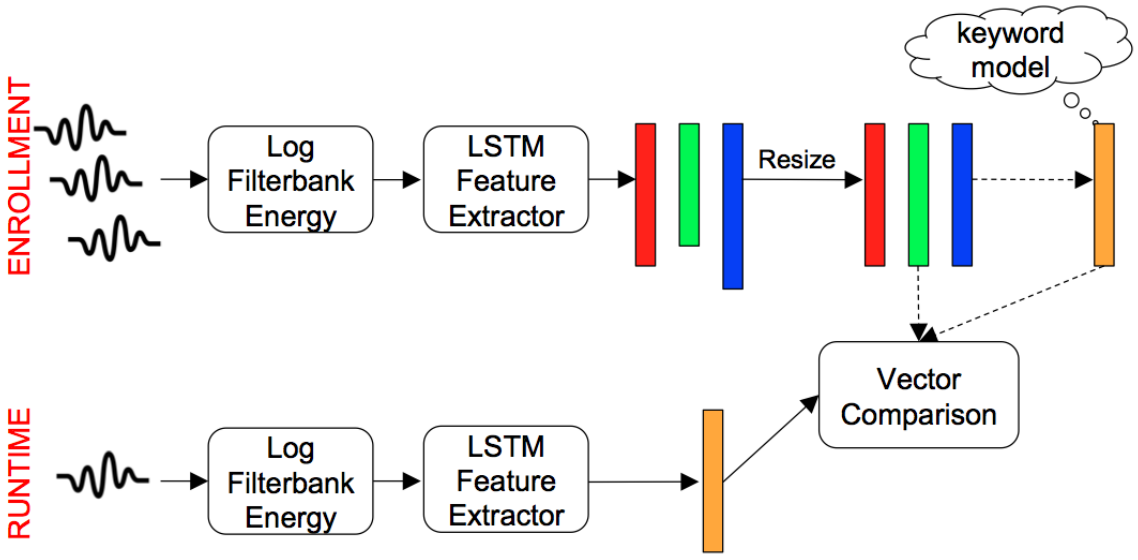

Figure 7.2: Framework of the LSTM KWS system.

At runtime, another LSTM feature vector is generated in the same way from a sliding window, and Cosine distance is used to measure the similarity between

the keyword template(s) and the sliding window. Decisions are made based on the similarity score.

7.3 DTW baseline system

DTW QbyE systems usually consist of two steps. First, features are extracted at the frame level [13, 6, 8]. Second, DTW is performed to compare the feature matrix of the templates and the test segment. A confidence score is then computed from the DTW alignment cost.

We use phoneme posteriorgram features in our implementation. Specifically, we label the training data by taking the forced-aligned 14,336 context-dependent (CD) state labels, and remapping them to a set of 43 phonemes, including the silence phone. We then train a neural network (i.e., DNN, LSTM) with 43 phoneme outputs. After training, each input frame of the template and test segments is converted to a posteriorgram by passing this frame through the network. Another way of generating phoneme posteriorgrams is to train a network to predict all 14,336 CD states, and then map them to 43 phonemes when computing the posteriorgrams. This, however, greatly increases the number of model parameters due to the large output layer.

A full picture of the baseline DTW KWS system is shown in Figure 7.3. In the enrollment phase, log filterbank energy features are extracted at the frame level for the three keyword examples, which are then fed to the neural network to generate

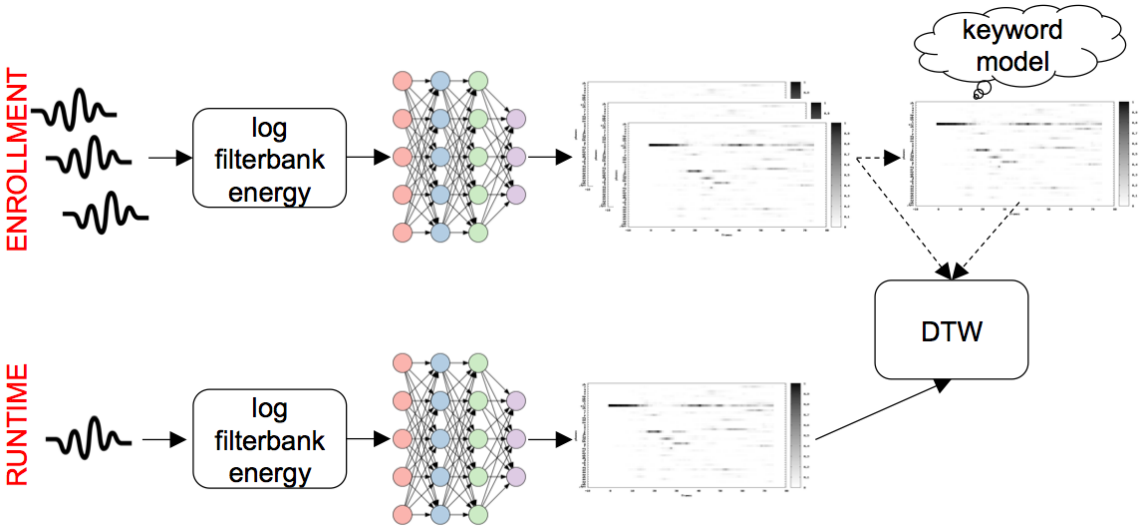

Figure 7.3: Framework of the baseline DTW KWS system.

keyword phoneme posteriorgrams. In the runtime phase, sliding window phoneme posteriorgrams are computed in the same way, and DTW is performed to compare the sliding window posteriorgrams and the keyword posteriorgrams. Detection decisions are then made based on the DTW alignment scores. Note that for the baseline we also have three keyword templates. We can either keep all the three templates, or average the three templates into one using the technique proposed in Section 7.4.

7.4 Template averaging

It has been shown in [6] that increasing the number of enrollment templates usually leads to performance improvements. In our work, we use three enrollment templates.

There are various techniques to perform evaluation with a test utterance given multiple enrollment templates. For example, in [6], the test audio is compared against

each of the enrollment templates, generating one score for each template. These scores are then merged into one final score for decision making. Evaluating against three templates also increases the computational cost, which is not favored by on-device applications. Therefore, we propose to combine multiple templates into a single template for scoring.

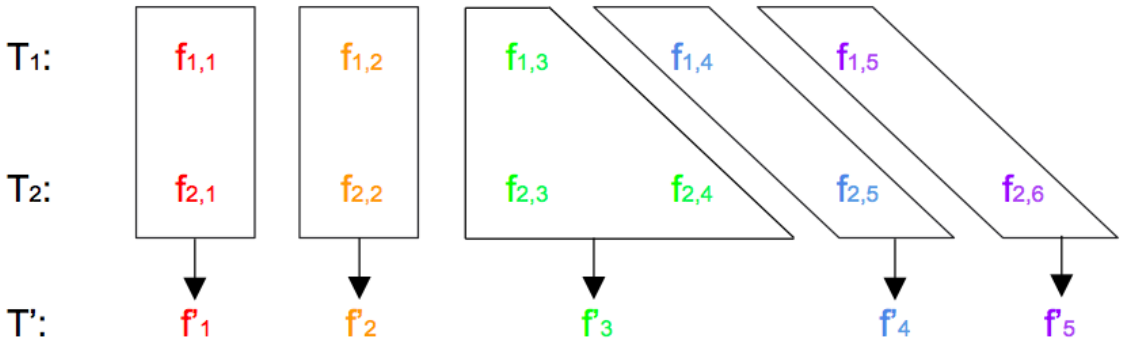

Figure 7.4: Example of template averaging.

Figure 7.4 shows how we combine templates. We use two templates in the figure for simplicity, where T_1 is the first template, and T_2 is the second template. We first align the second template T_2 to the first template T_1. For the DTW KWS system, since the templates T_1 and T_2 have different length, we use DTW to align them. Figure 7.4 shows the aligned frames in the same box. For the LSTM KWS system, however, templates are fixed length, so we align them by their dimensionality index. Once frames are aligned, we compute the combined frame by averaging them. For example, $f'_3 = (f_{1,3} + f_{2,3} + f_{2,4})/3$. The combined template T' will be used as the only template at runtime. If more templates are available, we always combine the new template with the previous combined one.

The proposed template averaging method relies heavily on the quality of the first template. In our experiments we ignore this effect and simply choose the first template randomly. Another option is to choose the longest template as the first one, as proposed in [130].

7.5 Experimental setup

7.5.1 Keywords for evaluation

Table 7.1 lists keywords used in our experiments. Our test-set combines anonymized voice search queries as negative examples, and utterances including these keywords as positive examples.

hello genie	hi galaxy
okay glass	change watch face
open settings	show agenda
show alarms	show step count

Table 7.1: Keywords used in evaluation.

We build one keyword model for each (speaker, keyword) combination. It is out of scope for this work to evaluate impostor performance (same keyword, different speaker), so positive examples only include utterances from that selected speaker and keyword.

Results are reported in the form of a modified version of receiver operating char-

acteristic (ROC) curves [49], lower curves are better. False alarm and false rejection counts are collected from all the models to compute the false rejection rate at a certain false alarm rate. The curve is obtained by sweeping the decision threshold.

7.5.2 Training

The neural network models are trained with $2,500$ hours of speech data, anonymized and manually transcribed. This dataset does not have any assumption on the keywords that will be evaluated. All models are trained using the cross-entropy criterion, with asynchronous stochastic gradient descent (ASGD) [121]. The weights in each network are randomly initialized and learning rates are exponentially decayed.

The enrollment set contains 3 keyword examples for each (speaker, keyword) combination. The evaluation set has $9k$ positive examples for the 8 selected keywords, and $36k$ negative examples. We use many more negative than positive examples to simulate the expected application usage.

7.6 Results

7.6.1 Initial results

Figure 7.5 shows the performance of the proposed LSTM Feature Extractor and 2 DTW KWS systems. The LSTM Feature Extractor system takes a 40-dimensional

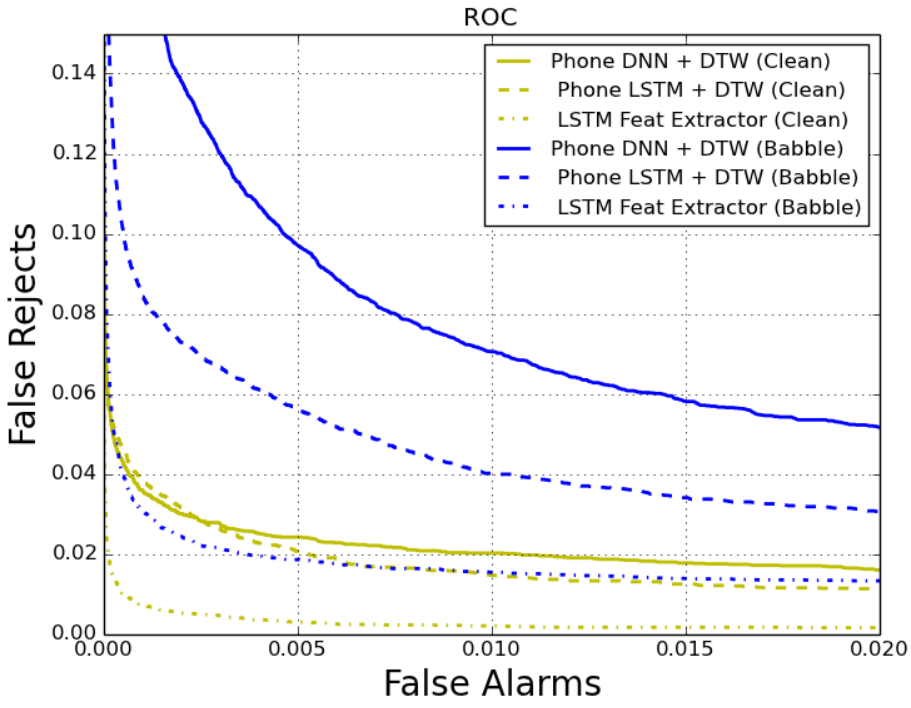

Figure 7.5: LSTM feature extractor vs. baseline.

input feature vector and has 2 LSTM layers, each with 128 cells, resulting in $152k$ parameters. For the DTW KWS systems, we compare obtaining posteriors from a DNN and an LSTM. The LSTM-posterior model also takes a 40-dimensional input feature vector, followed by 2 LSTM layers, each with 128 cells, and 43 phoneme output targets, resulting in $157k$ total parameters. The DNN-posterior model has 5 layers, each with 128 hidden units, and 43 phoneme targets. It uses 40-dimensional feature with 10 history frames and 5 future frames, resulting in a DNN with $152k$ parameters. Those network topologies are mainly chosen to match each other's parameter size, and are not explicitly tuned for performance.

We first use clean enrollment and evaluation data. In Figure 7.5, at roughly the

same parameter size, the *Phone LSTM + DTW* system outperforms the *Phone DNN + DTW* system. The proposed *LSTM Feat Extractor* system improves significantly over both DTW systems. At 0.5% false alarm rate, which is the desired operating point in our application, the *LSTM Feat Extractor* system yields 86% and 88% relative improvements respectively over the DTW systems.

We also explore the robustness of the proposed KWS approach by adding 10 dB of babble noise to the evaluation set. Results in Figure 7.5 show a performance degradation for all the systems due to noise. However, the general story does not change: the *Phone LSTM + DTW (Babble)* system is better than the *Phone DNN + DTW (Babble)* system (a lot better in this noisy case), while the *LSTM Feat Extractor (Babble)* system outperforms both of them.

7.6.2 Template averaging

Figure 7.6 compares the performance with and without template averaging. In this figure, the dashed curves are generated with template averaging, i.e., all the three templates are averaged into one template, and is used as the only template during runtime. The solid curves are generated without template averaging. Figure 7.6 suggests that template averaging does not degrade performance significantly. This is good news for on-device applications since we can reduce the runtime computation greatly (3x) without hurting the performance.

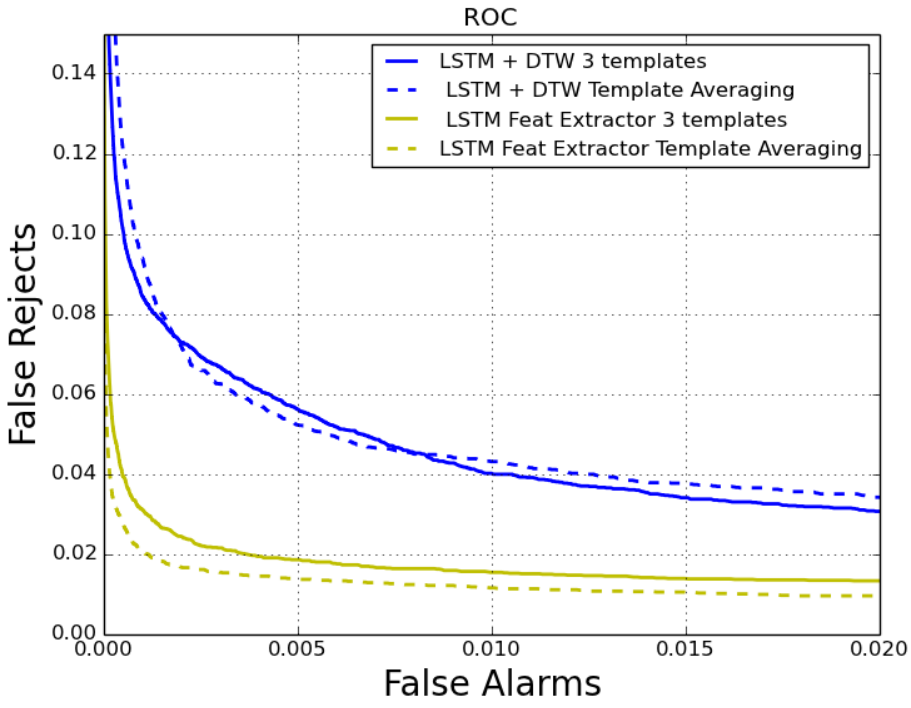

Figure 7.6: Impact of template averaging.

7.6.3 Whole word modeling

In Figure 7.7 we evaluate the importance of training the LSTM feature extractor with whole word output targets. The *Phone LSTM + DTW (Babble)* and *Word LSTM Feat Extractor (Babble)* curves are the same curves from Figure 7.5. For the *Phone LSTM Feat Extractor (Babble)* curve, we take the Phone LSTM model, remove the output layer and treat it as a feature extractor for keyword spotting. The performance of this system is much worse than the original LSTM feature extractor trained with whole word output targets, and it is also worse than the *Phone LSTM + DTW* system where we take the LSTM from. This implies that whole word modeling is critical in our LSTM KWS system.

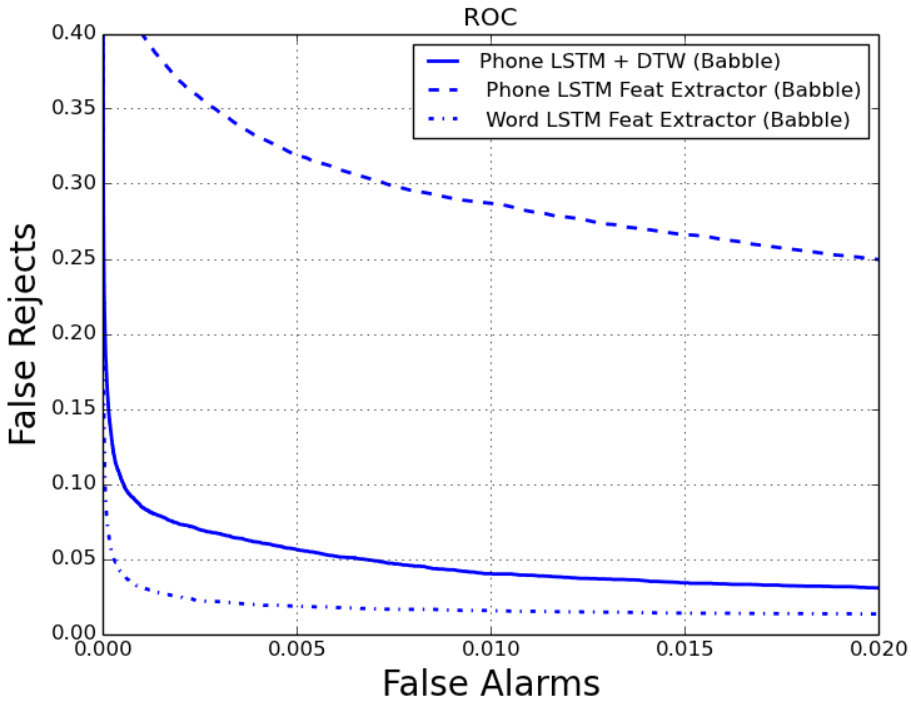

Figure 7.7: Impact of whole word modeling.

7.6.4 Noisy enrollment

Finally, we show the robustness of our proposed KWS approach when adding different noise sources at enrollment and evaluation time. To simulate this, we add 10 dB of babble noise to the clean evaluation data, and also add a different 10 dB cafe noise to the clean enrollment data, so that the keyword models built from the noisy enrollment data are corrupted. As shown in Figure 7.8, the performance degrades dramatically for all the systems. For example, at 0.5% (0.005) false alarm rate, the two DTW KWS systems both give 99.8% false rejection, basically rejecting everything, while the LSTM KWS system gives a false rejection rate of 63%.

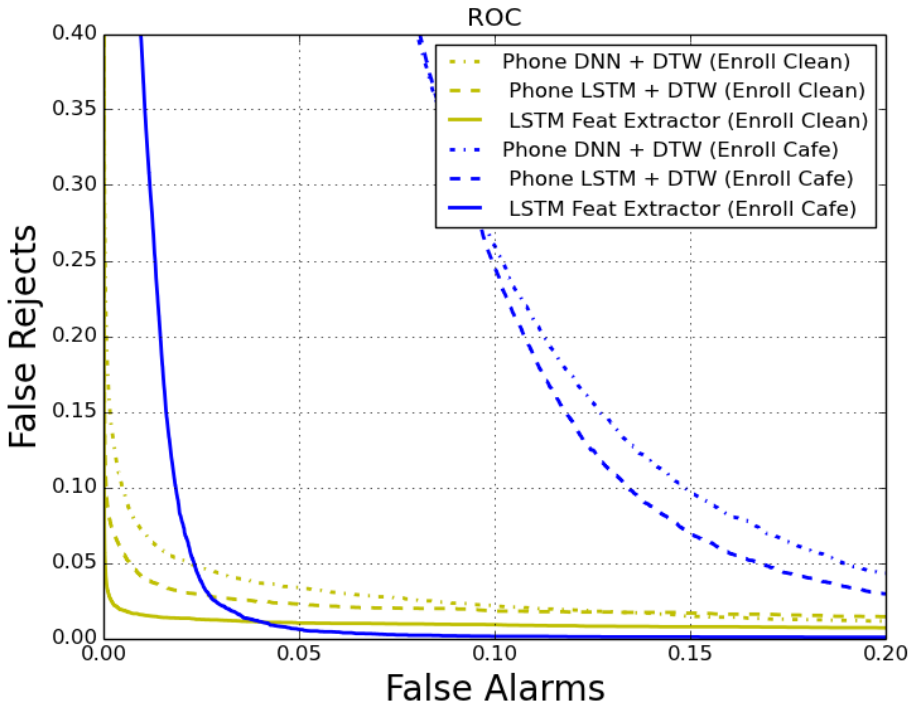

Figure 7.8: Impact of noisy enrollment.

7.7 Summary

We presented an LSTM feature extractor for the QbyE task in KWS. Experimental results showed that our method outperformed the standard phoneme posteriorgram + DTW system. We also proposed a template averaging technique, which allows us to combine multiple templates into one without performance degradation on our dataset. This technique is especially important for on-device applications since it can reduce the runtime computation cost. We tested the proposed QbyE system along with the baseline systems on various enrollment/evaluation conditions, and showed that it is important to have a clean enrollment environment for QbyE tasks.

As a future direction, we will look into ways to further improve our LSTM feature

extractor. For example, dimensionality reduction can be applied to reduce the stacked feature vector. We will also compare our method with the latest DTW QbyE systems, as described in [130, 131].

Chapter 8

Conclusion and Future Direction

8.1 Conclusion

In this dissertation we explored KWS in two low resource conditions: low language resource condition where language specific data for KWS model training is inadequate, and low computation resource condition where KWS runs on computation constrained devices.

8.1.1 Low language resource

For low language resource KWS, we focus on applications for speech data mining, where large vocabulary continuous speech recognition (LVCSR)-based KWS techniques are widely used. Keyword spotting for those applications are also known as keyword search (KWS) or spoken term detection (STD).

A key issue for this type of KWS technique is the out-of-vocabulary (OOV) keyword problem. Since LVCSR can only transcribe speech data to words that are defined in its lexicon, the search capability of LVCSR-based KWS systems is also limited to this lexicon, which is typically very small in a low language resource condition. This essentially means a large portion of the keywords will likely be out-of-vocabulary, and a regular LVCSR-based KWS system will not be able to search for those keywords.

We started by investigating the importance of pronunciation lexicon for LVCSR-based KWS systems in a low language resource condition. We assumed a large list of words was available, and compared 3 different grapheme-to-phoneme techniques. We discovered that lexicon augmentation had significantly greater impact on KWS performance than LVCSR. We also discovered that utilizing the augmented lexicon in the KWS stage via approximate phonetic matching was much less effective than utilizing them in the LVCSR stage.

One issue with utilizing the augmented lexicon in the LVCSR stage is, typically we do not have prior knowledge of what keywords we are going to search for. In fact, keyword search is supposed to be open-vocabulary. We therefore proposed a weighted finite-state transducer (WFST)-based framework for generating acoustically similar in-vocabulary (IV) word proxies for out-of-vocabulary (OOV) keywords. This was motivated by the fact that speech recognizer usually interprets OOV words as acoustically similar IV words. We further proposed a modified edit-distance transducer that allowed cost-inexpensive phone insertions and deletions at word boundaries, making

proxy search more close to phonetic search, while retaining the lexical constrain. Experiments on 6 Babel languages showed that the proposed proxy search technique constantly improved KWS performance by enabling OOV search in a regular LVCSR-based KWS system.

Our investigation of the lexicon value in LVCSR-based KWS also suggested the importance of utilizing a large lexicon, if available, in the LVCSR stage. In a practical low language resource condition, a large list of words may not be available, not to mention the large lexicon. We therefore proposed an automatic lexicon expansion technique that could expand the LVCSR's lexicon by adding a huge list of hallucinated words (e.g. 1 million), which greatly increased the keywords coverage. Experiments on 6 Babel languages showed that the proposed lexicon expansion method constantly improved the KWS performance because of the increased keyword coverage. We further combined the proposed lexicon expansion method with proxy keyword search technique and discovered that it was beneficial to perform proxy keyword search for keywords that only appeared in the expanded lexicon.

Finally, we explored the possibility of building LVCSR-based KWS systems without expert pronunciation lexicon. We started by evaluating LVCSR-based KWS systems with grapheme lexicons instead of expert pronunciation lexicons. Experiments on 6 *alphabetic* Babel languages showed that there were small degradations for both speech recognition performance and KWS performance. But the same experiment for a *logographic* language Cantonese was not successful due to the large number of unique

graphemes in *logographic* languages. We therefore proposed an iterative framework that was capable of generating pronunciation lexicons for *logographic* languages given a small seed expert pronunciation lexicon. Experiments on Babel language Cantonese showed that by adopting the proposed iterative framework with a seed lexicon of 1000 words, we were able to close 76% of the speech recognition performance gap (measured by word error rate) and 74% of the keyword search performance gap (measured by actual term-weighted value) between a KWS system with only the seed lexicon of 1000 words and a KWS system with an expert-crafted lexicon of 5000 words.

8.1.2 Low computation resource

For low computation resource KWS, we focus on wake-word applications, which usually run on computation constrained devices such as mobile phones or tablets. This type of application typically requires the underlying keyword spotting algorithms to be extremely accurate since they will be running constantly in a wake-word application. It is worth noting that LVCSR-based KWS techniques are inappropriate for those applications, since they are usually computationally expensive.

We first proposed a lightweight DNN keyword spotter that could detect pre-defined keywords, such as "Okay Google". The proposed algorithm was capable of running on computation constrained devices such as mobile phones, and was accurate enough for wake-word applications. Experiments on a set of voice commands showed that the proposed algorithm outperformed the keyword/filler model trained on the same

data set.

One issue with the DNN keyword spotter is that the model is trained for a specific keyword, and the training requires a large amount of keyword specific data. This is somewhat inconvenient if we would like to have a large number of models, e.g., user specific wake-words.

We then proposed a query-by-example (QbyE) framework, which only required 3 keyword examples to create the keyword model. We proposed an LSTM-based feature extractor for the QbyE framework, which could embed audio segments of various length into a fixed length feature vector. We also proposed a template averaging technique that could combine 3 keyword templates in the QbyE framework into just 1 template, which essentially reduced the detection computation complexity by a factor of 3. Experiments on a set of voice commands showed that the proposed algorithm outperformed the posteriorgram + DTW model trained on the same data set. Also it was capable of running on computation constrained devices such as mobile phones, and was accurate enough for wake-word applications.

8.2 Future direction

We believe the following directions can further improve the KWS performance if we incorporate them with the techniques proposed in this dissertation, thus listing them as future directions.

- **Language model.** This is for LVCSR-based KWS. For all of our LVCSR-based KWS experiments, we used a simple Kneser-Ney n-gram language model. However, more advanced language model techniques have been proposed in the literature. In [132], recurrent neural network-based language model (RNNLM) is proposed, which significantly outperforms traditional n-gram language models for speech recognition tasks. Applying RNNLM to LVCSR-based KWS should also improve the search performance. In [133], class-based n-gram language model is proposed. This kind of model will be very helpful in Chapter 4 when we estimate the language model probabilities for the hallucinated words. Other language models such as the cache-based language model [134] should generally be helpful to low language resource LVCSR-based KWS as well.

- **Score normalization.** This is for LVCSR-based KWS. One key step of LVCSR-based KWS is to make a final decision based on scores of all the retrieved instances. In this dissertation, we have proposed various techniques to improve KWS in a low language resource condition, but scores from those techniques may not be compatible with each other. For example, for the proxy search technique in Chapter 3, we incorporate phone confusion cost to the final score; for the lexicon expansion technique in Chapter 4, we include pronunciation scores (which are actually derived from the syllable language model and the pronunciation G2P model). These scores are not directly compatible with each other, and will cause problems if we do system combination. In [135], various

score normalization methods are compared. We should also apply some of those methods to our techniques.

- **Computation complexity evaluation metric.** This is for general KWS methods with computation constrain. In our experiments in this dissertation, we used the model size to estimate the computation complexity — smaller model typically leads to less computation. This method is intuitive but not rigorous. Adding other rigorous metrics will be helpful, e.g., the power consumption.

- **Multilingual feature extractor.** This is for LSTM-based feature extractor. In Chapter 7 the LSTM feature extractor was trained on English corpus, and technically the framework would only work well for English keywords. This is inconvenient if support of multiple languages is desired. One plausible extension is to train a multilingual feature extractor. For example, in [136] universal phonological features are used for multilingual speech recognition. Similar philosophies can be adopted in our feature extractor training.

Bibliography

[1] USC Shoah Foundation. Visited 1/23/2016. [Online]. Available: https://sfi.usc.edu/about/institute

[2] Coursera. Visited 1/23/2016. [Online]. Available: https://www.coursera.org/

[3] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, "Your word is my command: Google search by voice: a case study," in *Advances in Speech Recognition*. Springer, 2010, pp. 61–90.

[4] J. S. Bridle, "An efficient elastic template method for detecting given words in running speech," in *Proceedings of the British Acoustical Society Meeting*, 1973, pp. 1–4.

[5] P. Fousek and H. Hermansky, "Towards ASR based on hierarchical posterior-based keyword recognition," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 1. IEEE, 2006.

[6] Y. Zhang and J. R. Glass, "Unsupervised spoken keyword spotting via segmental DTW on Gaussian posteriorgrams," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2009, pp. 398–403.

[7] M. Huijbregts, M. McLaren, and D. Van Leeuwen, "Unsupervised acoustic sub-word unit detection for query-by-example spoken term detection," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 4436–4439.

[8] T. J. Hazen, W. Shen, and C. White, "Query-by-example spoken term detection using phonetic posteriorgram templates," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2009, pp. 421–426.

[9] W. Shen, C. M. White, and T. J. Hazen, "A comparison of query-by-example methods for spoken term detection," DTIC Document, Tech. Rep., 2009.

[10] C. Parada, A. Sethy, and B. Ramabhadran, "Query-by-example spoken term detection for OOV terms," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2009, pp. 404–409.

[11] H. Wang, T. Lee, and C.-C. Leung, "Unsupervised spoken term detection with acoustic segment model," in *Proceedings of the International Conference on Speech Database and Assessments (Oriental COCOSDA)*. IEEE, 2011, pp. 106–111.

[12] J. Tejedor, I. Szöke, and M. Fapso, "Novel methods for query selection and query combination in query-by-example spoken term detection," in *Proceedings of the International Workshop on Searching Spontaneous Conversational Speech*. ACM, 2010, pp. 15–20.

[13] K. Levin, K. Henry, A. Jansen, and K. Livescu, "Fixed-dimensional acoustic embeddings of variable-length segments in low-resource settings," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2013, pp. 410–415.

[14] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978.

[15] C.-a. Chan and L.-s. Lee, "Unsupervised spoken-term detection with spoken queries using segment-based dynamic time warping," in *Proceedings of INTERSPEECH*. ISCA, 2010, pp. 693–696.

[16] Y. Zhang and J. R. Glass, "An inner-product lower-bound estimate for dynamic time warping," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 5660–5663.

[17] A. Mandal, K. P. Kumar, and P. Mitra, "Recent developments in spoken term detection: a survey," *International Journal of Speech Technology*, vol. 17, no. 2, pp. 183–198, 2014.

[18] R. C. Rose and D. B. Paul, "A hidden Markov model based keyword recognition system," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 1990, pp. 129–132.

[19] J. G. Wilpon, L. Rabiner, C.-H. Lee, and E. Goldman, "Automatic recognition of keywords in unconstrained speech using hidden Markov models," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 38, no. 11, pp. 1870–1878, 1990.

[20] J. Wilpon, L. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 1991, pp. 309–312.

[21] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden Markov modeling for speaker-independent word spotting," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 1989, pp. 627–630.

[22] L. D. Wilcox and M. Bush, "Training and search algorithms for an interactive wordspotting system," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2. IEEE, 1992, pp. 97–100.

[23] J. Junkawitsch, L. Neubauer, H. Hoge, and G. Ruske, "A new keyword spotting

algorithm with pre-calculated optimal thresholds," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, vol. 4. IEEE, 1996, pp. 2067–2070.

[24] A. S. Manos and V. W. Zue, "A segment-based wordspotter using phonetic filler models," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2. IEEE, 1997, pp. 899–902.

[25] M.-C. Silaghi and H. Bourlard, "Iterative posterior-based keyword spotting without filler models," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. Citeseer, 1999, pp. 213–216.

[26] A. Tavanaei, H. Sameti, and S. H. Mohammadi, "False alarm reduction by improved filler model and post-processing in speech keyword spotting," in *Proceedings of Machine Learning for Signal Processing (MLSP) Workshop*. IEEE, 2011, pp. 1–5.

[27] R. A. Sukkar and J. G. Wilpon, "A two pass classifier for utterance rejection in keyword spotting," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2. IEEE, 1993, pp. 451–454.

[28] M. Weintraub, "LVCSR log-likelihood ratio scoring for keyword spotting," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1. IEEE, 1995, pp. 297–300.

[29] Y. Benayed, D. Fohr, J.-P. Haton, and G. Chollet, "Confidence measures for keyword spotting using support vector machines," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1. IEEE, 2003, pp. 588–591.

[30] H. Ketabdar, J. Vepa, S. Bengio, and H. Bourlard, "Posterior based keyword spotting with a priori thresholds," in *Proceedings of INTERSPEECH*. ISCA, 2006.

[31] R. A. Sukkar, A. R. Setlur, M. G. Rahim, and C.-H. Lee, "Utterance verification of keyword strings using word-based minimum verification error (WB-MVE) training," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1. IEEE, 1996, pp. 518–521.

[32] M. G. Rahim, C.-H. Lee, and B.-H. Juang, "Discriminative utterance verification for connected digits recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 3, pp. 266–277, 1997.

[33] J. S. Garofolo, C. G. Auzanne, and E. M. Voorhees, "The TREC spoken document retrieval track: a success story," *NIST SPECIAL PUBLICATION SP*, no. 246, pp. 107–130, 2000.

[34] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus in speech recognition: word error minimization and other applications of confusion networks," *Computer Speech & Language*, vol. 14, no. 4, pp. 373–400, 2000.

[35] D. Hakkani-Tur and G. Riccardi, "A general algorithm for word graph matrix decomposition," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 1. IEEE, 2003, pp. 596–599.

[36] M. Tomita, "An efficient word lattice parsing algorithm for continuous speech recognition," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 11. IEEE, 1986, pp. 1569–1572.

[37] F. Weng, A. Stolcke, and A. Sankar, "Efficient lattice representation and generation," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*. Citeseer, 1998.

[38] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlicek, Y. Qian *et al.*, "Generating exact lattices in the WFST framework," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 2012, pp. 4213–4216.

[39] C. Allauzen, M. Mohri, and M. Saraclar, "General indexation of weighted automata: application to spoken utterance retrieval," in *Proceedings of the Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval at HLT-NAACL*. Association for Computational Linguistics, 2004, pp. 33–40.

[40] M. Saraclar and R. Sproat, "Lattice-based search for spoken utterance retrieval," *Urbana*, vol. 51, p. 61801, 2004.

[41] D. Can and M. Saraclar, "Lattice indexing for spoken term detection," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 8, pp. 2338–2347, 2011.

[42] J. Chiu, Y. Wang, J. Trmal, D. Povey, G. Chen, and A. Rudnicky, "Combination of FST and CN search in spoken term detection," in *Proceedings of INTERSPEECH*. ISCA, 2014.

[43] K. Ng and V. W. Zue, "Subword-based approaches for spoken document retrieval," *Speech Communication*, vol. 32, no. 3, pp. 157–186, 2000.

[44] I. Szoke, L. Burget, J. Cernocky, and M. Fapso, "Sub-word modeling of out of vocabulary words in spoken term detection," in *Proceedings of Spoken Language Technology (SLT) Workshop*. IEEE, 2008, pp. 273–276.

[45] G. Chen, O. Yilmaz, J. Trmal, D. Povey, and S. Khudanpur, "Using proxies for OOV keywords in the keyword search task," in *Proceedings of the Automatic Speech Recognition and Understanding (ASRU) Workshop*. IEEE, 2013, pp. 416–421.

[46] M. Saraclar, A. Sethy, B. Ramabhadran, L. Mangu, J. Cui, X. Cui, B. Kingsbury, and J. Mamou, "An empirical study of confusion modeling in keyword search for low resource languages," in *Proceedings of the Automatic Speech Recognition and Understanding (ASRU) Workshop*. IEEE, 2013, pp. 464–469.

[47] A. Jansen, E. Dupoux, S. Goldwater, M. Johnson, S. Khudanpur, K. Church, N. Feldman, H. Hermansky, F. Metze, R. C. Rose *et al.*, "A summary of the 2012 JHU CLSP workshop on zero resource speech technologies and models of early language acquisition," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 8111–8115.

[48] IARPA Babel Program. Visited 1/23/2016. [Online]. Available: http://www.iarpa.gov/index.php/research-programs/babel

[49] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4087–4091.

[50] G. Chen, C. Parada, and T. N. Sainath, "Query-by-example keyword spotting using long short-term memory networks," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015.

[51] M. Sun, V. Nagaraja, B. Hoffmeister, and S. Vitaladevuni, "Model shrinking for embedded keyword spotting," in *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 369–374.

[52] G. Chen, S. Khudanpur, D. Povey, J. Trmal, D. Yarowsky, and O. Yilmaz, "Quantifying the value of pronunciation lexicons for keyword search in low re-

source languages," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013.

[53] StoryCorps. Visited 1/23/2016. [Online]. Available: http://storycorps.org/

[54] TED: The Ideas Worth Spreading. Visited 1/23/2016. [Online]. Available: http://www.ted.com/

[55] MITVideo. Visited 1/23/2016. [Online]. Available: http://video.mit.edu/

[56] J. Ma and R. Schwartz, "Unsupervised versus supervised training of acoustic models," in *Proceedings of INTERSPEECH*, 2008, pp. 2374–2377.

[57] L. Wang, M. J. Gales, and P. C. Woodland, "Unsupervised training for mandarin broadcast news and conversation transcription," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, vol. 4. IEEE, 2007, pp. IV–353.

[58] F. Wessel and H. Ney, "Unsupervised training of acoustic models for large vocabulary continuous speech recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 13, no. 1, pp. 23–31, 2005.

[59] L. Lamel, J.-L. Gauvain, and G. Adda, "Investigating lightly supervised acoustic model training," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, vol. 1. IEEE, 2001, pp. 477–480.

[60] T. Kemp and A. Waibel, "Unsupervised training of a speech recognizer: recent experiments," in *Proceedings of Eurospeech*, 1999.

[61] P. Vozila, J. Adams, Y. Lobacheva, and R. Thomas, "Grapheme to phoneme conversion and dictionary verification using graphonemes," in *Proceedings of Eurospeech*, 2003.

[62] L. Galescu and J. F. Allen, "Bi-directional conversion between graphemes and phonemes using a joint n-gram model," in *Proceedings of ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 2001.

[63] R. Rosenfield, "Optimizing lexical and ngram coverage via judicious use of linguistic data," in *Proceedings of Eurospeech*, 1995.

[64] J. Psutka, P. Ircing, J. V. Psutka, J. Hajic, W. J. Byrne, and J. Mírovskỳ, "Automatic transcription of Czech, Russian, and Slovak spontaneous speech in the MALACH project," in *Proceedings of INTERSPEECH*, 2005, pp. 1349–1352.

[65] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz, J. Silovský, G. Stemmer, and K. Veselý, "The Kaldi speech recognition toolkit," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2011.

[66] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, 2002.

[67] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: a general and efficient weighted finite-state transducer library," in *Proceedings of the International Conference on Implementation and Application of Automata (CIAA)*. Springer-Verlag, 2007, pp. 11–23.

[68] D. R. Miller, M. Kleber, C.-L. Kao, O. Kimball, T. Colthurst, S. A. Lowe, R. M. Schwartz, and H. Gish, "Rapid and accurate spoken term detection," in *Proceedings of INTERSPEECH*. ISCA, 2007, pp. 314–317.

[69] M. Bisani and H. Ney, "Joint-sequence models for grapheme-to-phoneme conversion," *Speech Communication*, vol. 50, no. 5, pp. 434 – 451, 2008.

[70] B. Logan, J.-M. V. Thong, and P. J. Moreno, "Approaches to reduce the effects of OOV queries on indexed spoken audio," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 899–906, 2005.

[71] O. Siohan and M. Bacchiani, "Fast vocabulary-independent audio search using path-based graph indexing," in *Proceedings of INTERSPEECH*. ISCA, 2005.

[72] J. Mamou, B. Ramabhadran, and O. Siohan, "Vocabulary independent spoken term detection," in *Proceedings of SIGIR*. ACM, 2007, pp. 615–622.

[73] U. V. Chaudhari and M. Picheny, "Improvements in phone based audio search

via constrained match with high order confusion estimates," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2007, pp. 665–670.

[74] B. Logan and J.-M. Van Thong, "Confusion-based query expansion for OOV words in spoken document retrieval," in *Proceedings of INTERSPEECH*. ISCA, 2002.

[75] Y.-C. Li, W.-K. Lo, H. M. Meng, and P. Ching, "Query expansion using phonetic confusions for Chinese spoken document retrieval," in *Proceedings of the International Workshop on Information Retrieval with Asian Languages (IRAL)*. ACM, 2000, pp. 89–93.

[76] P. Karanasou, L. Burget, D. Vergyri, M. Akbacak, and A. Mandal, "Discriminatively trained phoneme confusion model for keyword spotting," in *Proceedings of INTERSPEECH*. ISCA, 2012.

[77] J. Trmal, G. Chen, D. Povey, S. Khudanpur, P. Ghahremani, X. Zhang, V. Manohar, C. Liu, A. Jansen, D. Klakow *et al.*, "A keyword search system using open source software," in *Proceedings of Spoken Language Technology (SLT) Workshop*. IEEE, 2014.

[78] L. Lu, A. Ghoshal, and S. Renals, "Acoustic data-driven pronunciation lexicon for large vocabulary speech recognition," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2013, pp. 374–379.

[79] C. Parada, A. Sethy, M. Dredze, and F. Jelinek, "A spoken term detection framework for recovering out-of-vocabulary words using the web," in *Proceedings of INTERSPEECH*, 2010.

[80] S. Oger, V. Popescu, and G. Linares, "Using the world wide web for learning new words in continuous speech recognition tasks: Two case studies," in *Proceedings of the International Conference on Speech and Computer (SPECOM)*, 2009, pp. 76–81.

[81] A. Allauzen and J.-L. Gauvain, "Open vocabulary ASR for audiovisual document indexation," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005, pp. 1013–1016.

[82] M. Bisani and H. Ney, "Open vocabulary speech recognition with flat hybrid models," in *Proceedings of INTERSPEECH*, 2005, pp. 725–728.

[83] A. Rastrow, A. Sethy, B. Ramabhadran, and F. Jelinek, "Towards using hybrid word and fragment units for vocabulary independent LVCSR systems," in *Proceedings of INTERSPEECH*, 2009, pp. 1931–1934.

[84] C. Parada, M. Dredze, A. Sethy, and A. Rastrow, "Learning sub-word units for open vocabulary speech recognition," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2011, pp. 712–721.

[85] M. Akbacak, D. Vergyri, and A. Stolcke, "Open-vocabulary spoken term detection using graphone-based hybrid recognition systems," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2008, pp. 5240–5243.

[86] S. Jiampojamarn, *Grapheme-to-phoneme conversion and its application to transliteration.* University of Alberta, 2011.

[87] S. F. Chen *et al.*, "Conditional and joint models for grapheme-to-phoneme conversion." in *Proceedings of INTERSPEECH*, 2003.

[88] D. Harwath and J. Glass, "Speech recognition without a lexicon-bridging the gap between graphemic and phonetic systems," in *Proceedings of INTERSPEECH*, 2014.

[89] M. J. Gales, K. M. Knill, and A. Ragni, "Unicode-based graphemic systems for limited resource languages," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

[90] C. ying Lee, Y. Zhang, and J. Glass, "Joint Learning of Phonetic Units and Word Pronunciations for ASR," in *Proceedings of the 2013 Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2013, pp. 182–192.

[91] C.-y. Lee, T. J. O'Donnell, and J. Glass, "Unsupervised lexicon discovery from

acoustic input," *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 389–403, 2015.

[92] F. Beaufays, A. Sankar, S. Williams, and M. Weintraub, "Learning name pronunciations in automatic speech recognition systems," in *Proceedings of ICTAI*. IEEE, 2003, pp. 233–240.

[93] X. Li, A. Gunawardana, and A. Acero, "Adapting grapheme-to-phoneme conversion for name recognition," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2007, pp. 130–135.

[94] N. Goel, S. Thomas, M. Agarwal, P. Akyazi, L. Burget, K. Feng, A. Ghoshal, O. Glembek, M. Karafiát, D. Povey *et al.*, "Approaches to automatic lexicon learning with limited training examples," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2010, pp. 5094–5097.

[95] R. Rasipuram and M. Magimai Doss, "Combining acoustic data driven G2P and letter-to-sound rules for under resource lexicon generation," in *Proceedings of INTERSPEECH*, no. EPFL-CONF-192596, 2012.

[96] I. McGraw, I. Badr, and J. R. Glass, "Learning lexicons from speech using a pronunciation mixture model," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 21, no. 2, pp. 357–366, 2013.

[97] X. Lei, W. Wang, and A. Stolcke, "Data-driven lexicon expansion for mandarin broadcast news and conversation speech recognition," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2009, pp. 4329–4332.

[98] W. Byrne, V. Venkataramani, T. Kamm, T. F. Zheng, Z. Song, P. Fung, Y. Lui, and U. Ruhi, "Automatic generation of pronunciation lexicons for mandarin spontaneous speech," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2001, pp. 569–572.

[99] S. Jiampojamarn, G. Kondrak, and T. Sherif, "Applying many-to-many alignments and hidden markov models to letter-to-phoneme conversion." in *Proceedings of the Human Language Technology Conference of the NAACL (HLT-NAACL)*, vol. 7, 2007, pp. 372–379.

[100] S. Kanthak and H. Ney, "Context-dependent acoustic modeling using graphemes for large vocabulary speech recognition," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2. IEEE, 2002, pp. 845–848.

[101] M. Killer, S. Stüker, and T. Schultz, "Grapheme based speech recognition." in *Proceedings of INTERSPEECH*, 2003.

[102] P. Charoenpornsawat, S. Hewavitharana, and T. Schultz, "Thai grapheme-based speech recognition," in *Proceedings of the Human Language Technology Confer-*

ence of the NAACL (HLT-NAACL). Association for Computational Linguistics, 2006, pp. 17–20.

[103] J. R. Novak, N. Minematsu, and K. Hirose, "WFST-based grapheme-to-phoneme conversion: Open source tools for alignment, model-building and decoding," in *Proceedings of 10th International Workshop on Finite State Methods and Natural Language Processing*, 2012, p. 45.

[104] G. Chen, H. Xu, M. Wu, D. Povey, and S. Khudanpur, "Pronunciation and silence probability modeling for ASR," in *Proceedings of Interspeech*, 2015.

[105] T. Hain, "Implicit modelling of pronunciation variation in automatic speech recognition," *Speech Communication*, vol. 46, no. 2, pp. 171–188, 2005.

[106] B. Peskin, M. Newman, D. McAllaster, V. Nagesha, H. Richards, S. Wegmann, M. Hunt, and L. Gillick, "Improvements in recognition of conversational telephone speech," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 1. IEEE, 1999, pp. 53–56.

[107] T. Hain, P. Woodland, G. Evermann, and D. Povey, "The CU-HTK march 2000 Hub5e transcription system," in *Proceedings of Speech Transcription Workshop*, vol. 1, 2000.

[108] E. Fosler, M. Weintraub, S. Wegmann, Y.-H. Kao, S. Khudanpur, C. Galles, and M. Saraclar, "Automatic learning of word pronunciation from data," in

Proceedings of the International Conference on Spoken Language Processing, 1996.

[109] S. Parlak and M. Saraclar, "Spoken term detection for Turkish broadcast news," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2008, pp. 5244–5247.

[110] M.-C. Silaghi, "Spotting subsequences matching an HMM using the average observation probability criteria with application to keyword spotting," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20. AAAI Press/MIT Press, 2005, p. 1118.

[111] D. Grangier, J. Keshet, and S. Bengio, "Discriminative keyword spotting," *Automatic speech and speaker recognition: large margin and kernel methods*, pp. 175–194, 2009.

[112] S. Tabibian, A. Akbari, and B. Nasersharif, "An evolutionary based discriminative system for keyword spotting," in *Proceedings of the International Symposium on Artificial Intelligence and Signal Processing (AISP)*. IEEE, 2011, pp. 83–88.

[113] K. Li, J. Naylor, and M. Rossen, "A whole word recurrent neural network for keyword spotting," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2. IEEE, 1992, pp. 81–84.

[114] S. Fernández, A. Graves, and J. Schmidhuber, "An application of recurrent neural networks to discriminative keyword spotting," in *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Springer, 2007, pp. 220–229.

[115] T. Hughes and K. Mierle, "Recurrent neural networks for voice activity detection," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 7378–7382.

[116] X. Lei, A. Senior, A. Gruenstein, and J. Sorensen, "Accurate and compact large vocabulary speech recognition on mobile devices," in *Proceedings of INTERSPEECH*. ISCA, 2013.

[117] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, "On rectified linear units for speech processing," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 3517–3521.

[118] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 4688–4691.

[119] N. Jaitly, P. Nguyen, A. Senior, and V. Vanhoucke, "Application of pretrained

deep neural networks to large vocabulary speech recognition," in *Proceedings of INTERSPEECH*. ISCA, 2012.

[120] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 8595–8598.

[121] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Proceedings of Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

[122] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[123] G. Heigold, V. Vanhoucke, A. Senior, P. Nguyen, M. Ranzato, M. Devin, and J. Dean, "Multilingual acoustic models using distributed deep neural networks," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 8619–8623.

[124] M. Weintraub, "Keyword spotting using SRI's DECIPHER large vocabulary speech recognition system," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2. IEEE, 1993, pp. 463–466.

[125] R. G. Wallace, R. J. Vogt, and S. Sridharan, "A phonetic search approach to the 2006 NIST spoken term detection evaluation," in *Proceedings of INTERSPEECH*. ISCA, 2007.

[126] H. Soltau, G. Saon, and B. Kingsbury, "The IBM Attila speech recognition toolkit," in *Proceedings of Spoken Language Technology Workshop (SLT)*. IEEE, 2010, pp. 97–102.

[127] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[128] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 6645–6649.

[129] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014, pp. 1701–1708.

[130] L. J. Rodriguez-Fuentes, A. Varona, M. Penagarikano, G. Bordel, and M. Diez, "High-performance query-by-example spoken term detection on the SWS 2013

evaluation," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 7819–7823.

[131] I. Szöke, M. Skácel, and L. Burget, "BUT QUESST 2014 system description," in *Proceedings of CEUR Workshop*, vol. 2014, no. 1263. CEUR-WS.org, 2014, pp. 1–2.

[132] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of INTERSPEECH*, 2010, pp. 1045–1048.

[133] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[134] R. Kuhn and R. De Mori, "A cache-based natural language model for speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 6, pp. 570–583, 1990.

[135] D. Karakos, R. Schwartz, S. Tsakalidis, L. Zhang, S. Ranjan, T. Tim Ng, R.-C. Hsiao, G. Saikumar, I. Bulyko, L. Nguyen *et al.*, "Score normalization and system combination for improved keyword spotting," in *Proceedings of the Automatic Speech Recognition & Understanding (ASRU) Workshop*. IEEE, 2013, pp. 210–215.

[136] L. Deng, "Integrated-multilingual speech recognition using universal phonological features in a functional speech production model," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2. IEEE, 1997, pp. 1007–1010.

Vita

Guoguo Chen received the B.Eng. degree in Electronic Engineering from Tsinghua University in 2010. He enrolled in the Electrical and Computer Engineering Ph.D. program at Johns Hopkins University in August 2010, and has been a member of the Center for Language and Speech Processing since then. His research focuses on automatic speech recognition and keyword spotting. During his graduate study at Johns Hopkins University, he had the opportunity to work as a summer intern at Google's mobile speech group, and developed the prototype of Google's hotword detection system. He is also an active developer for the open source speech recognition toolkit Kaldi, and the Computational Network Toolkit CNTK.

Starting from January 2016, Guoguo will be running KITT.AI with Dr. Xuchen Yao and Dr. Kenji Sagae, focusing on spoken and natural language technologies.