On-Demand Language Model Interpolation for Mobile Speech Input

Brandon Ballinger[1], Cyril Allauzen[2], Alexander Gruenstein[1], Johan Schalkwyk[2]

[1]Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
[2]Google, 76 Ninth Avenue, New York, NY 10011, USA

`brandonb@google.com, allauzen@google.com, alexgru@google.com, johans@google.com`

Abstract

Google offers several speech features on the Android mobile operating system: search by voice, voice input to any text field, and an API for application developers. As a result, our speech recognition service must support a wide range of usage scenarios and speaking styles: relatively short search queries, addresses, business names, dictated SMS and e-mail messages, and a long tail of spoken input to any of the applications users may install. We present a method of on-demand language model interpolation in which contextual information about each utterance determines interpolation weights among a number of n-gram language models. On-demand interpolation results in an 11.2% relative reduction in WER compared to using a single language model to handle all traffic.

Index Terms: language modeling, interpolation, mobile

1. Introduction

Entering text on mobile devices is often slow and error-prone in comparison to typing on a full-sized keyboard. Google offers several features on Android aimed at making speech a viable alternative input method: search by voice, voice input into any text field, and a speech API for application developers. To search by voice, users simply tap a microphone icon on the desktop search box, or hold down the physical search button. They can speak any query, and are then shown the Google search results. To use the Voice Input feature, users tap the microphone key on the on-screen keyboard, and then speak to enter text virtually anywhere they would normally type. Users may dictate e-mail and SMS messages, fill in forms on web pages, or enter text into any application. Finally, the Android Speech API is a simple way for developers to integrate speech recognition capabilities into their own applications.

While a large portion of usage of the speech recognition service is comprised of spoken queries and dictation of SMS messages, there is a long tail of usage from thousands of other applications. Due to this diversity, choosing an appropriate language model for each utterance (recorded audio) is challenging. Two viable options are to build a single language model to handle all traffic, or to train a language model appropriate to each major use case and then choose the "best" one for each utterance, depending on the context of that utterance.

We develop and compare a third option in this paper, in which a development set of utterances from each context is used to optimize interpolation weights among a small number of component language models. Since there may be thousands of such "contexts", the language models are interpolated on-demand, either during decoding or as a post-processing rescoring phase. On-demand interpolation is performed efficiently via the use of a "compact interpolated" finite state transducer (FST), in which transition weights are dynamically computed.

	Percent of utterances
Voice input	49%
Search by Voice	44%
Speech API	7%

Table 1: *Breakdown of speech traffic on Android devices that support Voice Input, Search by Voice, and Speech API.*

2. Related Work

The technique of creating interpolated language models for different contexts has been used with success in a number of conversational interfaces [1, 2, 3] In this case, the pertinent context is the system's "dialogue state", and it's typical to group transcribed utterances by dialogue state and build one language model per state. Typically, states with little data are merged, and the state-specific language models are interpolated, or otherwise merged. Language models corresponding to multiple states may also be interpolated, to share information across similar states.

The technique we develop here differs in two key respects. First, we derive interpolation weights for thousands of recognition contexts, rather than a handful of dialogue states. This makes it impractical to create each interpolated language model offline and swap in the desired one at runtime. Our language models are large, and we only learn the recognition context for a particular utterance when the audio starts to arrive. Second, rather than relying on transcribed utterances from each recognition context to train state-specific language modes, we instead interpolate a small number of language models trained from large corpora.

3. Android Speech Usage Analysis

The challenge of supporting a variety of use cases is illustrated by examining the usage of the speech features available on Android. Table 1 breaks down the portion of utterances from the Android platform associated with the three speech features: voice input, search by voice, and the speech API. We note that this distinction isn't perfect, as some users might, for example, speak a search query into a text box in the browser using the voice input feature. In addition, a large majority of the speech API utterances come from built-in Google applications – Google Maps provides a popular voice-enabled search box, for example. Overall, we observe roughly an even split between searching and dictation.

The voice input feature encourages a wide range of usage. Since its launch in January, 2010, users have dictated text into over 8,000 distinct text fields. Table 2 shows the 10 most popular text fields. SMS is extremely popular, with usage levels an order of magnitude greater than any other application. Moreover, among the top 10 fields, 4 of them come from either the built-in SMS application, or one of the many SMS applica-

26 − 30 September 2010, Makuhari, Chiba, Japan

Text Field	Usage
SMS - Compose	63.1%
An SMS app from Market - Compose	4.9%
Browser	4.8%
Google Talk	4.5%
Gmail - Compose	3.3%
Android Market - Search	2.4%
Email - Compose	1.8%
SMS - To	1.3%
Maps - Directions Endpoint	1.0%
An SMS app from Market - Compose	1.0%

Table 2: *The 10 most popular voice input text fields and their percent usage.*
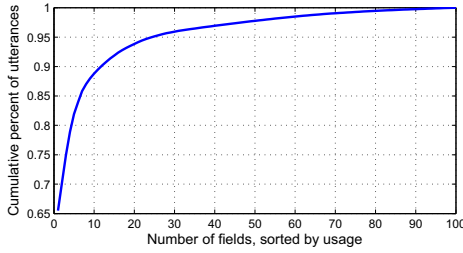

Figure 1: *Cumulative usage for the most popular 100 text fields, rank ordered by usage.*

tions available on the Android Market. Also popular are other dictation-style applications: Gmail, Email, and Google Talk. Android Market and Maps, both of which also appear in the top 10, represent different kinds of utterances – search queries. Finally, the Browser category here actually encompasses a wide range of fields – any text field on any web page.

Figure 1 shows the cumulative usage per text field of the 100 most popular text fields, rank ordered by usage. Although the usage is certainly concentrated among a handful of applications, there remains a significant tail. While increasing accuracy for the tail may not have a huge effect on the overall accuracy of the system, it's important for users to have a seamless experience using voice input: users will have a difficult time discerning that voice input may work better in some text fields than others.

4. Compact Interpolated FST

In this setting, we have a relatively small set of language models that is fixed and known in advance. At recognition time, each utterance comes with a custom set of interpolation (or mixture) weights and we need to be able to efficiently compute on-demand the corresponding interpolated model.

In a backoff language model, the conditional probability of $w \in \Sigma$ given context $h \in \Sigma^*$ is recursively defined as

$$\mathbf{P}(w \mid h) = \begin{cases} \widetilde{\mathbf{P}}(w \mid h) & \text{if } hw \in S \\ \alpha_h \mathbf{P}(w \mid h') & \text{otherwise,} \end{cases}$$

where $\widetilde{\mathbf{P}}$ is the adjusted maximum likelihood probability (derived from the training corpus), S is the skeleton of the model, α_h is the backoff weight for the context h and h' is the longest common suffix of h. The order of the model is $\max_{hw \in S} |hw|$.

Such a language model can naturally be represented by a weighted automaton over the real semiring $(\mathbb{R}, +, \times, 0, 1)$ using failure transitions [4]: the set of states is $Q =$
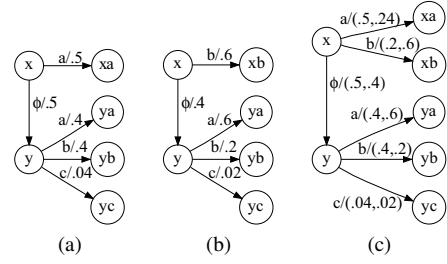

Figure 2: *Outgoing transitions from state x in (a) G_1, (b) G_2 and (c) I. For $\boldsymbol{\lambda} = (.6, .4)^T$, $\mathbf{P}_{I_{\boldsymbol{\lambda}}}(a \mid x) = .6 \times .5 + .4 \times .24$.*

$\{h \in \Sigma^* \mid \exists w \in \Sigma \text{ such that } hw \in S\}$, for each state h, there is a failure transition from h to h' labeled by ϕ and with weight α_h, and for each $hw \in S$, there is a transition from h to the longest suffix of hw that belongs to Q, labeled by w and with weight $\widetilde{\mathbf{P}}(w \mid h)$.

Given a set $\mathcal{G} = \{G_1, \ldots, G_m\}$ of m backoff language models and a vector of mixture weights $\boldsymbol{\lambda} = (\lambda_1, \ldots \lambda_m)^T$, the linear interpolation of $\mathcal{G}$ by $\boldsymbol{\lambda}$ is defined as the language model $I_{\boldsymbol{\lambda}}$ assigning the conditional probability:

$$\mathbf{P}_{I_{\boldsymbol{\lambda}}}(w \mid h) = \sum_{i=1}^{m} \lambda_i \mathbf{P}_{G_i}(w \mid h). \tag{1}$$

Using (1) directly to perform on-demand interpolation would be inefficient because for a given pair (w, h) we might need to backoff several times in several of the models and this can become rather expensive when using the automata representation. Instead, we chose to reformulate the interpolated model as a backoff model:

$$\mathbf{P}_{I_{\boldsymbol{\lambda}}}(w \mid h) = \begin{cases} \boldsymbol{\lambda}^T \mathbf{p}_{hw} & \text{if } hw \in S(\mathcal{G}), \\ f(\boldsymbol{\lambda}, \boldsymbol{\alpha}_h) \mathbf{P}_{I_{\boldsymbol{\lambda}}}(w \mid h') & \text{otherwise,} \end{cases}$$

where $\mathbf{p}_{hw} = (\mathbf{P}_{G_1}(w|h), \ldots, \mathbf{P}_{G_m}(w|h))^T$, $S(\mathcal{G}) = \cup_{i=1}^{m} S(G_i)$ and $\boldsymbol{\alpha}_h = (\alpha_h(G_1), \ldots, \alpha_h(G_m))^T$. There exists a closed-form expression of $f(\boldsymbol{\lambda}, \boldsymbol{\alpha})$ that ensure the proper normalization of the model. However, in practice we decided to approximate it by the dot product of $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}_h$: $f(\boldsymbol{\lambda}, \boldsymbol{\alpha}_h) = \boldsymbol{\lambda}^T \boldsymbol{\alpha}_h$.

The benefit of this formulation is that it perfectly fits our requirement. Since the set of models is known in advance we can precompute $S(\mathcal{G})$ and all the relevant vectors ($\mathbf{p}_{hw}$ and $\boldsymbol{\alpha}_h$) effectively building a generic interpolated model I as a model over $\mathbb{R}^m$. Given a new utterance and a corresponding vector of mixture weights $\boldsymbol{\lambda}$, we can obtain the relevant interpolated model $I_{\boldsymbol{\lambda}}$ by taking the dot product of each component vector of I with $\boldsymbol{\lambda}$.

Moreover, this approach also allows for an efficient representation of I as a weighted automaton over the semiring $(\mathbb{R}^m, +, \circ, \mathbf{0}, \mathbf{1})$ ($\circ$ denotes componentwise multiplication), the weight of each transition in the automaton being a vector in $\mathbb{R}^m$. The set of states is $Q = \{h \in \Sigma^* \mid \exists w \in \Sigma \text{ such that } hw \in S(\mathcal{G})\}$. For each state h, there is a failure transition from h to h' labeled by ϕ and with weight $\boldsymbol{\alpha}_h$, and for each $hw \in S(\mathcal{G})$, there is a transition from h to the longest suffix of hw that belongs to Q, labeled by w and with weight $\mathbf{p}_{hw}$. Figure 2 illustrates this construction.

Given a new utterance and a corresponding vector of mixture weights $\boldsymbol{\lambda}$, this automaton can be converted on-demand into a weighted automaton over the real semiring by taking the dot product of $\boldsymbol{\lambda}$ and the weight vector of each visited transition.

The OpenFst library [5] supports arbitrary semirings so we could have chosen to implement the interpolated model as a weighted automaton over $\mathbb{R}^m$. However, software engineering considerations lead us to use the CompactFst class from Open-Fst instead. This concrete class allows customizing the memory representation of the transitions. This represention is very efficient and combined with the on-demand composition algorithm of [6], it allows for the use of on-demand interpolation in the first-pass of recognition (see Section 7).

5. Interpolation Weight Optimization

To determine the mixture weights used for a particular utterance, we group utterances using the application and text field in which they are targeted, which we refer to as the *context*. For instance, utterances directed at the "subject" and "body" fields in the Gmail application form two distinct contexts, while a third context is comprised of utterances directed at the "body" field in the SMS application.

The interpolation weights for each language model component are chosen to maximize the likelihood of the development transcripts within a context. The optimization procedure is an EM-based algorithm, where at each iteration, the j'th weight is set to the fraction of the probability that the j'th language model contributed to the total mixture:

$$\lambda_j' = \frac{1}{n} \sum_{i=1}^{n} \frac{\lambda_j \mathbf{P}_j(w_i \mid h_i)}{\sum_{k=1}^{m} \lambda_k \mathbf{P}_k(w_i \mid h_i)} \qquad (2)$$

where n is the number of words in group, m is the number of language model components, and $\mathbf{P}_k(w_i \mid h_i)$ is the probability the j'th language model assigns to the i'th word in the group.

To avoid overfitting, our algorithm uses a simple back-off rule. We train interpolation weights for each field, application, and globally on the entire development set. If a particular field in the application has fewer than 10 transcripts, then the application's interpolation weights are used; if the application itself has fewer than 10 transcripts, then the global mixture is used. Experiments show that the back-off threshold has an insignificant ($< 0.1\%$ absolute) effect on accuracy. This back-off procedure provides for highly targeted weights when a significant amount of data is available for a particular field, while still providing support for the "long tail" of applications.

6. Component Language Models

We experimented with language model interpolation components constructed from a variety of spoken and written English sources, enumerated in Table 3. We extracted n-gram counts from anonymized typed e-mail and SMS messages through an automatic process in which humans did not have access to the raw data. These models were expected to match well with the expected common use case of dictated e-mail and SMS messages. A Twitter language model, which was expected to be useful for social networking "status update" usage, was created using publicly-accessible tweets that had been published on the web and downloaded by Google's web crawler. The Query model was trained on anonymized Google search queries, and is currently being used for Google's search by voice service.

In addition to text corpora, we experimented with two sources of transcribed spoken data. The Android Speech API corpus contains utterances by users of Android applications that take advantage of Google's speech API. We chose utterances from a subset of applications, for example applications

Source	Tokens	Description
Query	230B	Google search queries
SMS	2B	Anonymized SMS messages
E-mail	213M	Anonymized E-mail messages
BN	148M	Broadcast news corpora [7]
Twitter	30M	Tweets from Google's web index
Speech API	2M	Android Speech API transcripts

Table 3: *Language Model Interpolation Components.*

Task	Single pass			Rescoring	
	Global mix	LM per field	Mix per field	LM per field	Mix per field
Web search	14.6	11.8	11.7	13.1	13.2
SMS	20.9	21.0	19.0	20.9	19.2
Market Search	26.9	28.4	26.2	27.2	26.1
Browser	22.2	30.1	21.5	26.8	22.4
Maps Search	25.0	22.5	22.6	23.9	23.9
Gmail	19.2	19.8	16.0	20.0	18.5
Overall	19.7	19.1	17.5	19.5	18.7

Table 4: *WER of language model combination techniques on several representative tasks, and overall for the entire test set.*

for composing SMS messages, that we thought likely to be well-matched to mobile dictation usage. Finally, the Broadcast News transcripts [7] add coverage for a different style of speaking.

Basic text normalization was applied to each source, and n-gram models were constructed using Google's large-scale language model infrastructure [8]. The Query model is a 3-gram, and all other models are 4-grams. The vocabulary of each was limited to the one million most frequent words. Katz smoothing was employed, and entropy pruning [9] was used to significantly prune each model.

7. Experiments

In this section, we report experimental results for both individual component language models (LMs) and model combination techniques on several mobile speech tasks.

7.1. Experiment Setup

The test set consists of 42,227 transcribed utterances from several mobile voice tasks: 17,711 utterances from Google Search by Voice, 5,774 utterances from searching on Google Maps, and 18,742 utterances for Voice Input into text fields. For voice input, we report accuracy on four specific tasks: SMS (text message body), Browser (any form field on the web), Market (searching for applications in the Android Market app), and Gmail (email body). In addition, a development set of 21,000 utterances with the same distribution of tasks was used to train the mixture weights. The acoustic model is a tied-state triphone GMM-based HMM whose input features are 39 PLP-cepstral coefficients, trained using ML, MMI, and boosted-MMI objective functions as described in [10]

7.2. Language Model Combination Techniques

Table 4 gives the accuracy of five different methods for combining the component language models. The simplest method, "Global mix", decodes all utterances using a single interpolated model with interpolation weights set to minimize perplexity on the entire development set. We also tried a second baseline
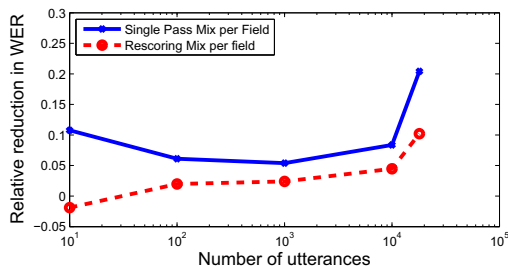
Figure 3: *Relative reduction in WER for the Single Pass and Rescoring Mix per Field conditions compared to the Global Mix baseline, grouped by field frequency in the test set.*

("LM per field"), in which we used the LM component with the lowest-perplexity for each field. Finally, we evaluated the technique described in Section 5, in which custom interpolation weights are derived for each field ("Mix per field").

Furthermore, we evaluated the latter two techniques under two conditions. First, we created a two-pass system where word lattices created by decoding with the globally-optimal mixture are rescored ("Rescoring"). Lattices had a mean arc density per word of 48.6. Second, we performed on-demand interpolation in a single decoding pass ("Single pass"). In this case, the Compact Interpolated language model FST was composed dynamically (see [6]) with a precompiled FST generated by composing C, the context-dependent phone transducer, with L, the lexicon. Unfortunately, this system currently operates at approximately 8x realtime; optimizations are in progress.

Several trends are evident. First, training per-field mixtures gives a 5.1% total relative improvement over the global mixture baseline when rescoring, and moving interpolation to the decoding pass boosts this to a 11.2% total gain. Performing interpolation in the first pass is especially important for tasks like Web search, where a single component LM optimized for the task already works quite well. For Web search, performing interpolation during the rescoring phase yields a lower accuracy than a single pass using only the Query LM. This is because the globally optimized first pass is mismatched to the task, and doesn't yield a rich enough lattice for rescoring to compensate.

Second, the benefit seems to come mostly from tasks where one of the component language models is well-matched to a task—gains for Web Search and Gmail are relatively large, while those for Market and Browser are more modest. Third, the greatest gain comes from the most popular and least popular text fields, but not the middle of the frequency distribution. Figure 3 shows the relative reduction in WER gained by using per-field mixtures. The head of the distribution is improved by both rescoring and first-pass interpolation. For the "long tail" of infrequent fields, rescoring leads to a loss of accuracy, but first-pass interpolation leads to a 11% gain. This is likely because the first-pass model under-weights language model components that are not popular in the overall test set, leading to a mismatched first and second pass in the tail.

7.3. Component Language Model Performance

Finally, we also explored using each component LM in isolation, as well as removing each LM from the mixture and recognizing with the remaining five LMs. Table 5 shows WER rates under the single-pass "Mix per field" condition. There are several points to note. First, all the mixture techniques in table 5 perform 20-29% relative better than even the best individual LM, highlighting that good performance on the mobile speech

Language Model Component	Only	Removed
Query	27.0	22.0
SMS	24.7	18.0
E-mail	25.9	17.9
Broadcast News	34.6	17.7
Twitter	28.1	17.8
Speech API	33.6	18.2

Table 5: *WER of the "mix per field" condition when including only the LM component indicated, or when removing only that LM from the on-demand interpolation.*

task requires a diverse set of data sources. Second, each LM's influence in the mixture is only loosely correlated with its individual performance. For example, the Speech API LM ranks fifth by individual performance, but removing it from the mixture leads to the second-highest accuracy drop.

8. Conclusion

We described a data structure that allows for on-demand language model interpolation, with mixture weights set at decode-time, and a training algorithm to generate mixture weights for thousands of individual text fields. These techniques together yield an 11.2% relative improvement in WER over a single statically-interpolated language model on mobile recognition tasks, with the greatest improvement coming from the most- and least- frequent fields.

9. Acknowledgements

10. References

[1] F. Wessel, A. Baader, and H. Ney, "A comparison of dialogue-state dependent language models," in *Proc. of ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, 1999, pp. 93–96.

[2] K. Visweswariah and H. Printz, "Language models conditioned on dialogue state," in *Proc. of EUROSPEECH*, 2001, pp. 251–254.

[3] W. Xu and A. Rudnicky, "Language modeling for dialog system," in *Proc. of ICSLP*, 2000, pp. 118–121.

[4] C. Allauzen, M. Mohri, and B. Roark, "Generalized algorithms for constructing statistical language models," in *Proc. of ACL*, 2003, pp. 40–47.

[5] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *CIAA*, ser. LNCS, vol. 4783, 2007, pp. 11–23, http://www.openfst.org.

[6] C. Allauzen, M. Riley, and J. Schalkwyk, "A generalized composition algorithm for weighted finite-state transducers," in *Proc. of Interspeech*, 2009, pp. 1203–1206.

[7] D. Graff, "An overview of Broadcast News corpora," *Speech Communication*, vol. 37, may 2002.

[8] T. Brants, A. C. Popat, P. Xu, F. J. Och, and J. Dean, "Large language models in machine translation," in *Proc. of EMNLP*, 2007, pp. 858–867.

[9] A. Stolcke, "Entropy-based pruning of backoff language models," in *Proc. of DARPA Braodcast News Transcription and Understanding Workshop*, 1998, pp. 270–274.

[10] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, "Google Search by Voice: A case study," in *Visions of Speech: Exploring New Voice Apps in Mobile Environments, Call Centers and Clinics*, A. Neustein, Ed. Springer, 2010 (in press).