

Chapter 8

Empirical Exploration of Language Modeling for the google.com Query Stream as Applied to Mobile Voice Search

Ciprian Chelba and Johan Schalkwyk

Abstract Mobile is poised to become the predominant platform over which people access the World Wide Web. Recent developments in speech recognition and understanding, backed by high bandwidth coverage and high quality speech signal acquisition on smartphones and tablets are presenting the users with the choice of speaking their web search queries instead of typing them. A critical component of a speech recognition system targeting web search is the language model. The chapter presents an empirical exploration of the google.com query stream with the end goal of high quality statistical language modeling for mobile voice search. Our experiments show that after text normalization the query stream is not as “wild” as it seems at first sight. One can achieve out-of-vocabulary rates below 1% using a 1 million word vocabulary, and excellent n -gram hit ratios of 77/88% even at high orders such as $n = 5/4$, respectively. A more careful analysis shows that a significantly larger vocabulary (approx. 10 million words) may be required to guarantee at most 1% out-of-vocabulary rate for a large percentage (95%) of users. Using large scale, distributed language models can improve performance significantly—up to 10% relative reductions in word-error-rate over conventional models used in speech recognition. We also find that the query stream is non-stationary, which means that adding more past training data beyond a certain point provides diminishing returns, and may even degrade performance slightly. Perhaps less surprisingly, we have shown that locale matters significantly for English query

C. Chelba, Ph.D. (✉)
Staff Research Scientist, Google, Inc.,
1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
e-mail: ciprianchelba@google.com

J. Schalkwyk, M.Sc.
Principal Staff Engineer, Google, Inc., 76 Ninth Avenue, 4th Floor, New York,
NY 10011, USA

data across USA, Great Britain and Australia. In an attempt to leverage the speech data in voice search logs, we successfully build large-scale discriminative N-gram language models and derive small but significant gains in recognition performance.

Introduction

Mobile web search is a rapidly growing area of interest. Internet-enabled smartphones account for an increasing share of mobile devices sold throughout the world, and most models offer a web browsing experience that rivals desktop computers in display quality. Users are increasingly turning to their mobile devices when searching the web, driving efforts to enhance the usability of web search on these devices.

Although mobile device usability has improved, typing search queries can still be cumbersome, error-prone, and even dangerous in some usage scenarios. To address these problems, Google introduced voice search in November 2008. The goal of Google voice search is to recognize any spoken search query, and be capable of handling anything that Google search can handle.

We present an empirical exploration of `google.com` query stream language modeling for voice search. We describe the normalization of the typed query stream resulting in out-of-vocabulary (OOV) rates below 1% for a 1 million word vocabulary. We present a comprehensive set of experiments that guided the design decisions for a voice search service. In the process we re-discovered a less known interaction between Kneser-Ney smoothing and entropy pruning, and found empirical evidence that hints at non-stationarity of the query stream, as well as strong dependence on various English locales—USA, Britain and Australia.

In an attempt to leverage the large amount of speech data made available by the voice search service, we present a distributed framework for large-scale discriminative language models that can be integrated within a large vocabulary continuous speech recognition (LVCSR) system using lattice rescoring. We intentionally use a weakened acoustic model in a baseline LVCSR system to generate candidate hypotheses for voice search data; this allows us to utilize large amounts of unsupervised data to train our models. We propose an efficient and scalable MapReduce framework that uses a perceptron-style distributed training strategy to handle these large amounts of data. We report small but significant improvements in recognition accuracies on a standard voice search data set using our discriminative reranking model. We also provide an analysis of the various parameters of our models including model size, types of features, size of partitions in the MapReduce framework with the help of supporting experiments.

We will begin by defining the language modeling problem and typical metrics for comparing language models. We will then describe a series of experiments which explore the dimensions along which Voice Search language models may be refined.

Language Modeling Basics

A statistical language model estimates the prior probability values $P(W)$ for strings of words W in a vocabulary \mathcal{V} whose size is usually in the tens or hundreds of thousands. Typically the string W is broken into sentences, or other segments such as utterances in automatic speech recognition, which are assumed to be conditionally independent. For the rest of this chapter, we will assume that W is such a segment, or sentence. With $W = w_1, w_2, \dots, w_n$ we get:

$$P(W) = \prod_{i=1}^n P(w_i | w_1, w_2, \dots, w_{i-1}) \quad (8.1)$$

Since the parameter space of $P(w_k | w_1, w_2, \dots, w_{k-1})$ is too large, the language model is forced to put the *context* $W_{k-1} = w_1, w_2, \dots, w_{k-1}$ into an *equivalence class* determined by a function $\Phi(W_{k-1})$. As a result,

$$P(W) \cong \prod_{k=1}^n P(w_k | \Phi(W_{k-1})) \quad (8.2)$$

Research in language modeling consists of finding appropriate equivalence classifiers Φ and methods to estimate $P(w_k | \Phi(W_{k-1}))$.

The most successful paradigm in language modeling uses the $(n - 1)$ -gram equivalence classification, that is, defines

$$\Phi(W_{k-1}) \doteq w_{k-n+1}, w_{k-n+2}, \dots, w_{k-1}$$

Once the form $\Phi(W_{k-1})$ is specified, only the problem of estimating $P(w_k | \Phi(W_{k-1}))$ from training data remains. In most practical cases, $n=3$ which leads to a *trigram* language model.

Perplexity as a Measure of Language Model Quality

A *statistical language model* can be evaluated by how well it predicts a string of symbols W_t —commonly referred to as *test data*—generated by the source to be modeled.

Assume we compare two models M_1 and M_2 using the same vocabulary¹ \mathcal{V} . They assign probability $P_{M_1}(W_t)$ and $P_{M_2}(W_t)$, respectively, to the sample test string W_t . The test string has neither been used nor seen at the estimation step

¹Language models estimated on different vocabularies cannot be directly compared using perplexity, since they model completely different probability distributions.

of either model and it was generated by the same source that we are trying to model. “Naturally”, we consider M_1 to be a better model than M_2 if $P_{M_1}(W_t) > P_{M_2}(W_t)$.

A commonly used quality measure for a given model M is related to the entropy of the underlying source and was introduced under the name of *perplexity* (PPL) (Jelinek 1997):

$$PPL(M) = \exp\left(-\frac{1}{N} \sum_{k=1}^N \ln[P_M(w_k | W_{k-1})]\right) \quad (8.3)$$

To give intuitive meaning to perplexity, it represents the number of guesses the model needs to make in order to ascertain the identity of the next word, when running over the test word string from left to right. It can be easily shown that the perplexity of a language model that uses the uniform probability distribution over words in the vocabulary \mathcal{V} equals the size of the vocabulary; a good language model should of course have lower perplexity, and thus the vocabulary size is an upper bound on the perplexity of a given language model.

Very likely, not all words in the test string W_t are part of the language model vocabulary. It is common practice to map all words that are out-of-vocabulary to a distinguished *unknown word* symbol, and report the out-of-vocabulary (OOV) rate on test data—the rate at which one encounters OOV words in the test string W_t —as yet another language model performance metric besides perplexity. Usually the unknown word is assumed to be part of the language model vocabulary—*open vocabulary* language models—and its occurrences are counted in the language model perplexity calculation, Eq. (8.3). A situation far less common in practice is that of *closed vocabulary* language models where all words in the test data will always be part of the vocabulary \mathcal{V} .

Smoothing

Since the language model is meant to assign non-zero probability to unseen strings of words (or equivalently, ensure that the cross-entropy of the model over an arbitrary test string is not infinite), a desirable property is that:

$$P(w_k | \Phi(W_{k-1})) > \epsilon > 0, \forall w_k, W_{k-1}, \quad (8.4)$$

also known as the *smoothing* requirement.

A large body of work has accumulated over the years on various smoothing methods for n -gram language models that ensure this to be true. The two most widespread smoothing techniques are probably Kneser-Ney (1995) and Katz (1987); Goodman (2001) provides an excellent overview that is highly recommended to any practitioner of language modeling.

Query Language Modeling for Voice Search

A typical voice search language model used in our system for the US English query stream is trained as follows:

- Vocabulary size: 1M words, OOV rate 0.57%
- Training data: 230B words, a random sample of anonymized queries from google.com that did not trigger spelling correction

The resulting size, as well as its performance on unseen query data (10k queries) when using Katz smoothing is shown in Table 8.1. We note a few key aspects:

- The first pass LM (15 million n -grams) requires very aggressive pruning—to about 0.1% of its unpruned size—in order to make it usable in static FST-based (Finite State Transducer-based) ASR decoders (Automatic Speech Recognition decoders)
- The perplexity hit taken by pruning the LM is significant, 50% relative; similarly, the 3-g hit ratio is halved
- The impact on WER due to pruning is significant, yet lower in relative terms—10% relative, as we show in section “Effect of Language Model Size on Speech Recognition Accuracy”
- The unpruned model has excellent n -gram hit ratios on unseen test data: 77% for $n=5$, and 97% for $n=3$
- The choice of $n=5$ is because using higher n -gram orders yields diminishing returns: a 7-g LM is 4 times larger than the 5-g LM trained from the same data and using the same vocabulary, at no gain in perplexity.

For estimating language models at this scale we have used the distributed language modeling tools built for statistical machine translation (Brants and Xu 2009; Brants et al. 2007) based on the MapReduce infrastructure described in section “Language Modeling Basics”. Pruned language models used in the first pass of the ASR decoder are converted to ARPA (Paul and Baker 1992) and/or FST (Allauzen et al. 2007) format using an additional MapReduce pass with a single reducer, which can optionally apply the language model compression techniques described in Harb et al. (2009).

The next section describes the text normalization that allows us to use a 1 million word vocabulary and obtain out-of-vocabulary (OOV) rates lower than 1%, as well as the excellent n -gram hit ratios presented in Table 8.1.

We then present experiments that show the temporal and spatial dependence of the English language models. Somewhat unexpectedly, using more training data does not result in an improved language model despite the fact that it is extremely well matched to the unseen test data. Additionally, the English language models built from training data originating in three locales (USA, Britain, and Australia) exhibit strong locale-specific behavior, both in terms of perplexity and OOV rate.

We will then present speech recognition experiments on a voice search test set.

Table 8.1 Typical voice search LM, Katz smoothing: the LM is trained on 230 billion words using a vocabulary of 1 million words, achieving out-of-vocabulary rate of 0.57% on test data

Order	No. n-grams	Pruning	PPL	n-gram hit-ratios
3	15M	Entropy (Stolcke)	190	47/93/100
3	7.7B	None	132	97/99/100
5	12.7B	Cut-off (1-1-2-2-2)	108	77/88/97/99/100

Privacy Considerations

Before delving into the technical aspects of our work, we wish to clarify the privacy aspects of our work with respect to handling user data.

All of the query data used for training, and testing models is strictly anonymous; the queries bear no user-identifying information. The only data saved after training are vocabularies, or n-gram counts. When working with session data, such as the experiments reported in section “Optimal Size, Freshness and Time-Frame for Voice Search Vocabulary”, we are even stricter: the evaluation on test data is done by counting on streamed filtered query logs, without saving any data.

Text Normalization

In order to build a language model for spoken query recognition we boot-strap from written queries to `google.com`. Written queries provide a data-rich environment for modeling of queries. This requires robustly transforming written text into spoken form.

Table 8.2 lists a couple of example queries and their corresponding spoken equivalents. Written queries contain a fair number of cases which require special attention to convert to spoken form. Analyzing the top million vocabulary items before text normalization we see approximately 20% URLs and 20+% numeric items in the query stream. Without careful attention to text normalization the vocabulary of the system will grow substantially.

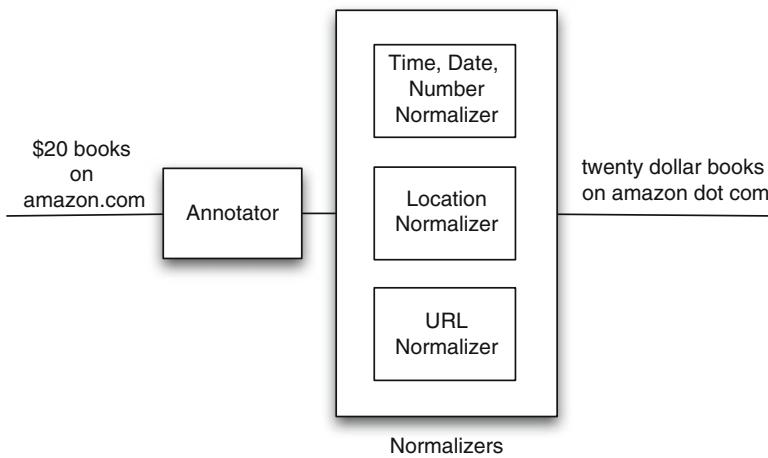
We adopt a finite state approach to text normalization. Let $T(\textit{written})$ be an acceptor that represents the written query. Conceptually the spoken form is computed as follows

$$T(\textit{spoken}) = \textit{bestpath}(T(\textit{written}) \circ N(\textit{spoken}))$$

where $N(\textit{spoken})$ represents the transduction from written to spoken form. Note that composition with $N(\textit{spoken})$ might introduce multiple alternate spoken representations of the input text. For the purpose of computing n -grams for spoken language modeling of queries we use the *bestpath* operation to select a single most likely interpretation.

Table 8.2 Example written queries and their corresponding spoken form

Written query	Spoken query
weather scarsdale, ny	weather scarsdale new york
	weather in scarsdale new york
bankofamerica.com	bank of america dot com
81 walker rd	eighty one walker road
10:30am	ten thirty A M
at&t	A T and T
espn	E S P N

**Fig. 8.1** Block diagram for context aware text normalization

The text normalization is run in multiple phases. Figure 8.1 depicts the text normalization process. In the first step we annotate the data. In this phase we categorize parts (sub strings) of queries into a set of known categories (e.g. time, date, url, location).

Since the query is annotated, it is possible to perform context-aware normalization on the substrings. Each category has a corresponding text normalization transducer $N_{cat}(spoken)$ that is used to normalize the substring. Depending on the category we either use rule based approaches or a statistical approach to construct the text normalization transducer. For numeric categories like date, time and numbers it is easy enough to describe $N(spoken)$ using context dependent rewrite rules. For the URL normalizer $N_{url}(spoken)$ we train a statistical word decomposer that segments the string into its word constituents. For example, one reads the URL cancercentersofamerica.com as “cancer centers of america dot com”. The URL decomposing transducer (decomposer) is built from the annotated data. Let Q

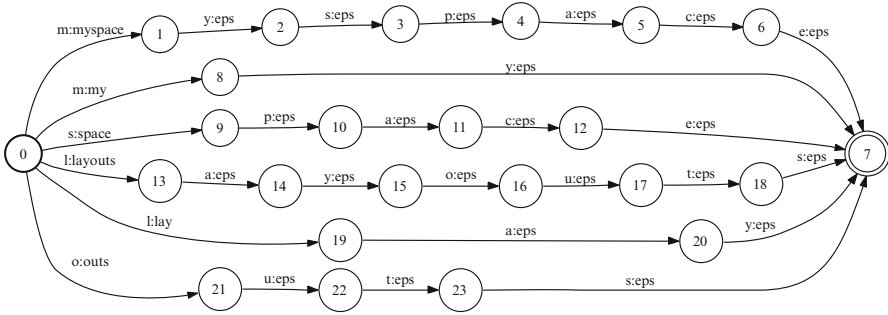


Fig. 8.2 $T(S)$ for the set of words $S = \{my, space, myspace, lay, outs, layouts\}$ where ‘{ eps}’ denotes ϵ

be the set of queries in this Table, and let U be the set of substrings of these queries that are labeled URLs.

For a string s of length k let $I(s)$ be the transducer that maps each character in s to itself; i.e., the i -th transition in $I(s)$ has input and output label $s(i)$. $I(s)$ represents the word segmented into characters. Further, let $T(s)$ be the transducer that maps the sequence of characters in s to s ; i.e., the first transition in $T(s)$ has input $s(1)$ and output s , and the i -th transition, where $i \neq 1$, has input $s(i)$ and output ϵ . $T(s)$ represents the transduction of the spelled form of the word to the word itself. For a set of strings S , we define

$$T(S) = \bigoplus_{s \in S} T(s)$$

where \bigoplus is the union operation on transducers. $T(S)$ therefore represents the transduction of the spelling of the word to the word itself for the whole vocabulary. Figure 8.2 illustrates the operation of $T(\cdot)$.

The queries in Q and their frequencies are used to train an LM L_{BASE} . Let V_{BASE} be its vocabulary. We build the decomposer as follows:

1. For each $u \in U$, define $N(u)$ as,

$$N(u) = \text{bestpath}(I(u) \circ T^*(V_{BASE}) \circ L_{BASE}) \tag{8.5}$$

where ‘ $*$ ’ is the Kleene Closure, and ‘ \circ ’ is the composition operator.

2. $N(U) = \bigoplus_{u \in U} N(u)$ is the URL decomposer.

The transducer $I(u) \circ T^*(V_{BASE})$ in (8.5) represents the lattice of all possible segmentations of u using the words in V_{BASE} , where each path from the start state to a final state in the transducer is a valid segmentation. The composition with the LM L_{BASE} scores every path. Finally, $N(u)$ is the path with the highest probability; i.e. the most likely segmentation.

As an example, Fig. 8.3 depicts $I(u) \circ T^*(V_{BASE})$ for $u = myspace\ layouts$. Each path in this lattice is a valid decomposition, and in Table 8.3 we list a sample of

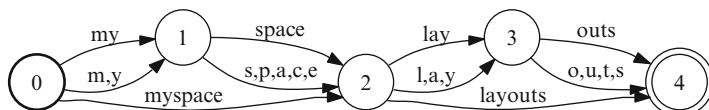


Fig. 8.3 The lattice $I(u) \circ T^*(V_{\text{BASE}})$ of all possible segmentations for $u = \text{myspacelayouts}$ using words in V_{BASE}

Table 8.3 Sample segmentations from Fig. 8.3. The one in bold represents the highest probability path as determined by the composition with L_{BASE}

Possible segmentations
myspace layouts
my space layouts
my space lay outs
my space l a y outs

these paths. After scoring all the paths via the composition with L_{BASE} , we choose the best path to represent the spoken form of the URL.

Language Model Refinement

Query Stream Non-stationarity

Our first attempt at improving the language model was to use more training data: we used a significantly larger amount of training data (BIG) vs. the most recent 230 billion (230B) prior to September 2008. The 230B corpus is the most recent subset of BIG. As test data we used a random sample consisting of 10k queries from Sept to Dec 2008.

The first somewhat surprising finding was that this had very little impact in OOV rate for 1M word vocabulary: 0.77% (230B vocabulary) vs. 0.73% (BIG vocabulary). Perhaps even more surprising however is the fact that the significantly larger training set did not yield a better language model, despite the training data being clearly well matched, as illustrated in Table 8.4. In fact, we observed a significant reduction in PPL (10%) when using the more recent 230B data. Pruning masks this effect, and the differences in PPL and WER become insignificant after reducing the language model size to approximately 10 million 3-g.

Since the vocabulary, and training data set change between the two rows, the PPL differences need to be analyzed in a more careful experimental setup.

Table 8.4 Pruned and unpruned 3-g language model perplexity when trained on the most recent 230 billion words, and a much larger amount of training data prior to test data, respectively

Training set	Test	Set PPL
	Unpruned	Pruned
230B	121	205
BIG	132	209

A superficial interpretation of the results seems to contradict the “there’s no data like more data” dictum, recently reiterated in a somewhat stronger form in Banko and Brill (2001), Och (2005) and Halevy et al. (2009).

Our experience has been that supply of “more data” needs to be matched with increased demand on the modeling side, usually by increasing the model capacity—typically achieved by estimating more parameters. Experiments reported in section “Effect of Language Model Size on Speech Recognition Accuracy” improve performance by *keeping the amount of training data constant* (albeit very large), and *increasing the n -gram model size* by adding more n -grams at fixed n , as well as increasing the model order n . As such, it may well be the case that the increase in PPL for the BIG model is in fact due to limited capacity in the 3-g model.

More investigation is needed to disentangle the effects of query stream non-stationarity from possible mismatched model capacity issues. A complete set of experiments needs to:

- Let the n -gram order grow as large as the data allows;
- Build a sequence of models trained on exactly the same amount of data obtained by sliding a time-window of varying length over the query stream, and control for the ensuing vocabulary mismatches.

Effect of Language Model Size on Speech Recognition Accuracy

The work described in Harb et al. (2009) and Allauzen et al. (2009) enables us to evaluate relatively large query language models in the 1st pass of our ASR decoder by representing the language model in the OpenFst (Allauzen et al. 2007) framework. Figures 8.4 and 8.5 show the PPL and word error rate (WER) for two language models (3- and 5-g, respectively) built on the 230B training data, after entropy pruning to various sizes in the range 15 million–1.5 billion n -grams. Perplexity is evaluated on the test set described in section “Query Stream Non-stationarity”; word error rate is measured on another test set representative for the voice search task.

As can be seen, perplexity is very well correlated with WER, and the size of the language model has a significant impact on speech recognition accuracy: increasing the model size by two orders of magnitude reduces the WER by 10% relative.

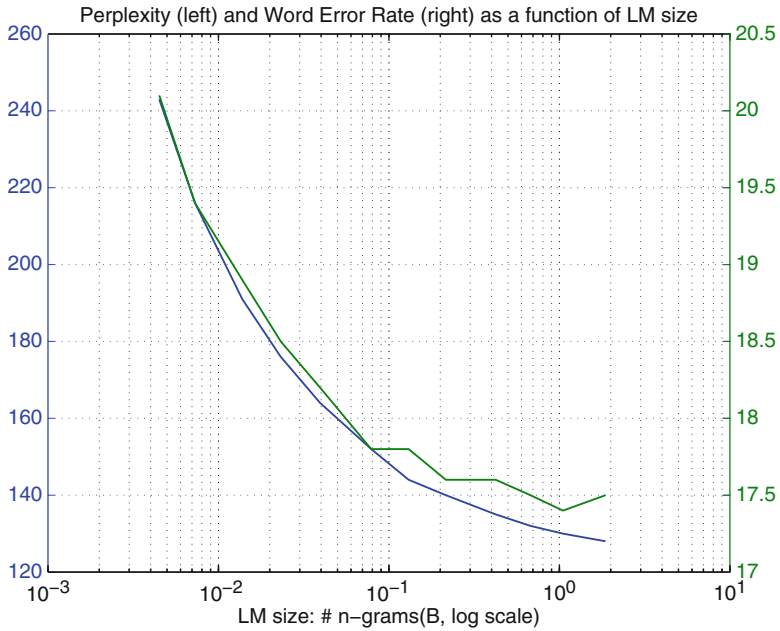


Fig. 8.4 Three-gram language model perplexity and word error rate as a function of language model size; *lower curve is PPL*

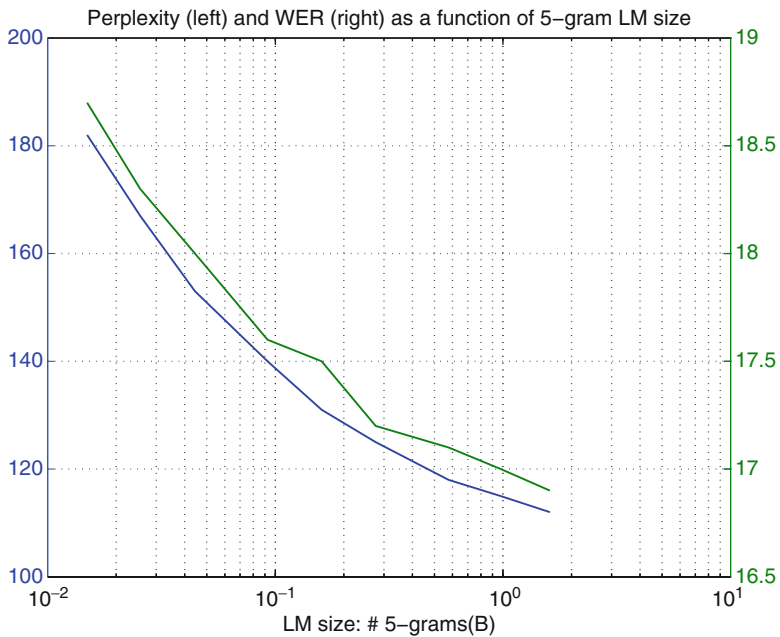


Fig. 8.5 Five-gram language model perplexity and word error rate as a function of language model size; *lower curve is PPL*

Table 8.5 Speech recognition language model performance when used in the 1st pass or in the 2nd pass—lattice rescoring

Pass	Language model	PPL	WER
1st	15M 3-g	191	18.7
1st	1.6B 5-g	112	16.9
2nd	15M 3-g	191	18.8
2nd	1.6B 5-g	112	16.9
2nd	12.7B 5-g	108	16.8

Table 8.6 Out of vocabulary rate: locale specific vocabulary halves the OOV rate

Training		Test	Locale
locale	USA	GBR	AUS
USA	0.7	1.3	1.6
GBR	1.3	0.7	1.3
AUS	1.3	1.1	0.7

We have also implemented lattice rescoring using the distributed language model architecture described in Brants et al. (2007), see the results presented in Table 8.5. This enables us to validate empirically the hypothesis that rescoring lattices generated with a relatively small first pass language model (in this case 15 million 3-g, denoted 15M 3-g in Table 8.5) yields the same results as 1st pass decoding with a large language model. A secondary benefit of the lattice rescoring setup is that one can evaluate the ASR performance of much larger language models.

Locale Matters

We also built locale specific English language models using training data prior to September 2008 across 3 English locales: USA (USA), Britain (GBR, about a quarter of the USA amount) and Australia (AUS, about a quarter of the GBR amount). The test data consisted of 10k queries for each locale, sampled randomly from Sept to Dec 2008.

Tables 8.6–8.8 show the results. The dependence on locale is surprisingly strong: using an LM on out-of-locale test data doubles the OOV rate and perplexity, either pruned or unpruned.

We have also built a combined model by pooling data across locales, with the results shown on the last row of Table 8.8. Combining the data negatively impacts all locales, in particular the ones with less data. The farther the locale from USA (as seen on the first line, GBR is closer to USA than AUS), the more negative the impact of lumping all the data together, relative to using only the data from that given locale.

Table 8.7 Perplexity of unpruned LM: locale specific LM halves the PPL of the unpruned LM

Training		Test	Locale
Locale	USA	GBR	AUS
USA	132	234	251
GBR	260	110	224
AUS	276	210	124

Table 8.8 Perplexity of pruned LM: locale specific LM halves the PPL of the unpruned LM. Pooling all data is suboptimal

Training		Test	Locale
Locale	USA	GBR	AUS
USA	210	369	412
GBR	442	150	342
AUS	422	293	171
combined	227	210	271

The Case for Discriminative Language Modeling

The language model is a critical component of an automatic speech recognition (ASR) system that assigns probabilities or scores to word sequences. It is typically derived from a large corpus of text via maximum likelihood estimation in conjunction with some smoothing constraints. N-gram models have become the most dominant form of LMs in most ASR systems. Although these models are robust, scalable and easy to build, we illustrate a limitation with the following example from voice search. We expect a low probability for an ungrammatical or implausible word sequence. However, for a trigram like “a navigate to”, a backoff trigram LM gives a fairly large LM log probability of -0.266 because both “a” and “navigate to” are popular words in voice search! Discriminative language models (DLMs) attempt to directly optimize error rate by rewarding features that appear in low error hypotheses and penalizing features in misrecognized hypotheses. In such a model, the trigram “a navigate to” receives a negative weight of -6.5 thus decreasing its chances of appearing as an ASR output. There have been numerous approaches towards estimating DLMs for large vocabulary continuous speech recognition (LVCSR) (Gao et al. 2005; Roark et al. 2007; Zhou et al. 2006).

There are two central issues that we discuss regarding DLMs. Firstly, DLM training requires large amounts of parallel data (in the form of correct transcripts and candidate hypotheses output by an ASR system) to be able to effectively compete

with n-gram LMs trained on large amounts of text. This data could be simulated using voice search logs from a baseline ASR system that are filtered by confidence score to obtain reference transcripts. However, this data is perfectly discriminated by first pass features such as the acoustic and language model scores, and leaves little room for learning. We propose a novel training strategy using lattices generated with a weaker acoustic model (henceforth referred to as *weakAM*) than the one used to generate reference transcripts for the unsupervised parallel data (referred to as the *strongAM*). This provides us with enough errors to derive large numbers of potentially useful word features; it is akin to using a weak LM in discriminative acoustic modeling to give more room for diversity in the word lattices resulting in better generalization (Schlüter et al. 1999). We conduct experiments to verify whether *weakAM*-trained models provide performance gains on rescored lattices from a standard test set generated using *strongAM* (discussed in section “Evaluating ASR Performance on *v-search-test* Using DLM Rescoring”).

The second issue is that discriminative estimation of LMs is computationally more intensive than regular N-gram LM estimation. The advent of distributed learning algorithms (Hall et al. 2010; Mann et al. 2009; McDonald et al. 2010) and supporting parallel computing infrastructure like MapReduce (Ghemawat and Dean 2004) has made it feasible to use large amounts of parallel data for training DLMs. We implement a distributed training strategy for the perceptron algorithm introduced by McDonald et al. (2010) using the MapReduce framework. Our design choices for the MapReduce implementation are specified in section “MapReduce Implementation Details” along with its modular nature thus enabling us to experiment with different variants of the distributed structured perceptron algorithm. Some of the descriptions in this paper have been adapted from previous work Jyothi et al. (2012).

The Distributed DLM Framework: Training and Implementation Details

Learning Algorithm

We aim to allow the estimation of large scale distributed models, similar in size to the ones in Brants et al. (2007). To this end, we make use of a distributed training strategy for the structured perceptron to train our DLMs (McDonald et al. 2010). Our model consists of a high-dimensional feature vector function Φ that maps an (utterance, hypothesis) pair (x, y) to a vector in R^d , and a vector of model parameters, $\mathbf{w} \in R^d$. Our goal is to find model parameters such that given x , and a set of candidate hypotheses \mathcal{Y} (typically, as a word lattice or an N-best list that is obtained from a first pass recognizer), $\operatorname{argmax}_{y \in \mathcal{Y}} \mathbf{w} \cdot \Phi(x, y)$ would be the $y \in \mathcal{Y}$ that minimizes the error rate between y and the correct hypothesis for x . For our experiments, the feature vector $\Phi(x, y)$ consists of AM and LM costs for y from the lattice \mathcal{Y} for x , as

well as “word level n-gram features” which count the number of times different N-grams (of order up to 5 in our experiments) occur in y .

In principle, such a model can be trained using the conventional structured perceptron algorithm (Collins 2002). This is an online learning algorithm which continually updates \mathbf{w} as it processes the training instances one at a time, over multiple training epochs. Given a training utterance $\{x_i, y_i\}$ ($y_i \in \mathcal{Y}_i$ has the lowest error rate with respect to the reference transcription for x_i , among all hypotheses in the lattice \mathcal{Y}_i for x_i), if $\tilde{y}_i^* := \operatorname{argmax}_{y \in \mathcal{Y}_i} \mathbf{w} \cdot \Phi(x_i, y)$ is not y_i , then \mathbf{w} is updated to increase the weights corresponding to features in y_i and decrease the weights of features in \tilde{y}_i^* . During evaluation, we use parameters averaged over all utterances and over all training epochs. This was shown to give substantial improvements in previous work Collins (2002) and Roark et al. (2007).

Unfortunately, the conventional perceptron algorithm takes impractically long for the amount of training examples we have. We make use of a distributed training strategy for the structured perceptron that was first introduced in McDonald et al. (2010). The iterative parameter mixing strategy used in this paradigm can be explained as follows: the training data $\mathcal{T} = \{x_i, y_i\}_{i=1}^N$ is suitably partitioned into \mathcal{C} disjoint sets $\mathcal{T}_1, \dots, \mathcal{T}_c$. Then, a structured perceptron model is trained on each data set in parallel. After one training epoch, the parameters in the \mathcal{C} sets are mixed together (using a “mixture coefficient” μ_i for each set \mathcal{T}_i) and returned to each perceptron model for the next training epoch where the parameter vector is initialized with these new mixed weights. This is formally described in Algorithm 1; we call it “Distributed Perceptron”. We also experiment with two other variants of distributed perceptron training, “Naive Distributed Perceptron” and “Averaged Distributed Perceptron”. These models easily lend themselves to implementations using the distributed infrastructure provided by the MapReduce framework. The following section describes this infrastructure in greater detail.

MapReduce Implementation Details

We propose a distributed infrastructure using MapReduce (Ghemawat and Dean 2004) to train our large-scale DLMs on terabytes of data. The MapReduce (Ghemawat and Dean 2004) paradigm, adapted from a specialized functional programming construct, is specialized for use over clusters with a large number of nodes. Chu et al. (2007) have demonstrated that many standard machine learning algorithms can be phrased as MapReduce tasks, thus illuminating the versatility of this framework. In relation to language models, Brants et al. (2007) recently proposed a distributed MapReduce infrastructure to build Ngram language models having up to 300 billion n -grams. We take inspiration from this and use the MapReduce infrastructure for our DLMs. Also, the MapReduce paradigm allows us to easily fit different variants of our learning algorithm in a modular fashion by only making small changes to the MapReduce functions.

Algorithm 1 Distributed Perceptron [19]

Require: Training samples $\mathcal{T} = \{x_i, y_i\}_{i=1}^N$

- 1: $\mathbf{w}^0 := [0, \dots, 0]$
- 2: Partition \mathcal{T} into \mathcal{C} parts, $\mathcal{T}_1, \dots, \mathcal{T}_\mathcal{C}$
- 3: $[\mu_1, \dots, \mu_\mathcal{C}] := [\frac{1}{\mathcal{C}}, \dots, \frac{1}{\mathcal{C}}]$
- 4: **for** $t := 1$ to T **do**
- 5: **for** $c := 1$ to \mathcal{C} **do**
- 6: $\mathbf{w} := \mathbf{w}^{t-1}$
- 7: **for** $j := 1$ to $|\mathcal{T}_c|$ **do**
- 8: $\tilde{y}_{c,j}^t := \operatorname{argmax}_y \mathbf{w} \cdot \Phi(x_{c,j}, y)$
- 9: $\delta := \Phi(x_{c,j}, y_{c,j}) - \Phi(x_{c,j}, \tilde{y}_{c,j}^t)$
- 10: $\mathbf{w} := \mathbf{w} + \delta$
- 11: **end for**
- 12: $\mathbf{w}_c^t := \mathbf{w}$
- 13: **end for**
- 14: $\mathbf{w}^t := \sum_{c=1}^{\mathcal{C}} \mu_c \mathbf{w}_c^t$
- 15: **end for**
- 16: **return** \mathbf{w}^T

In the MapReduce framework, any computation is expressed as two user-defined functions: *Map* and *Reduce*. The *Map* function takes as input a key/value pair and processes it using user-defined functions to generate a set of intermediate key/value pairs. The *Reduce* function receives all intermediate pairs that are associated with the same key value.

The distributed nature of this framework comes from the ability to invoke the *Map* function on different parts of the input data simultaneously. Since the framework assures that all the values corresponding to a given key will be accumulated at the end of all the *Map* invocations on the input data, different machines can simultaneously execute the *Reduce* to operate on different parts of the intermediate data.

Any MapReduce application typically implements *Mapper/Reducer* interfaces to provide the desired *Map/Reduce* functionalities. For our models, we use two different Mappers (as illustrated in Fig. 8.6) to compute feature weights for one training epoch. The *Rerank-Mapper* receives as input a set of training utterances and has the capacity to request feature weights computed in the previous training epoch. *Rerank-Mapper* then computes feature updates for the given training data (the subset of the training data received by a single *Rerank-Mapper* instance will be henceforth referred to as a “Map chunk”). We also have a second *Identity-Mapper* that receives feature weights from the previous training epoch and directly maps the inputs to outputs which are provided to the *Reducer*. The *Reducer* combines the outputs from both *Rerank-Mapper* and *Identity-Mapper* and outputs the feature weights for the current training epoch. These output feature weights are persisted on disk in the form of SSTables that are an efficient abstraction to store large numbers of key-value pairs.

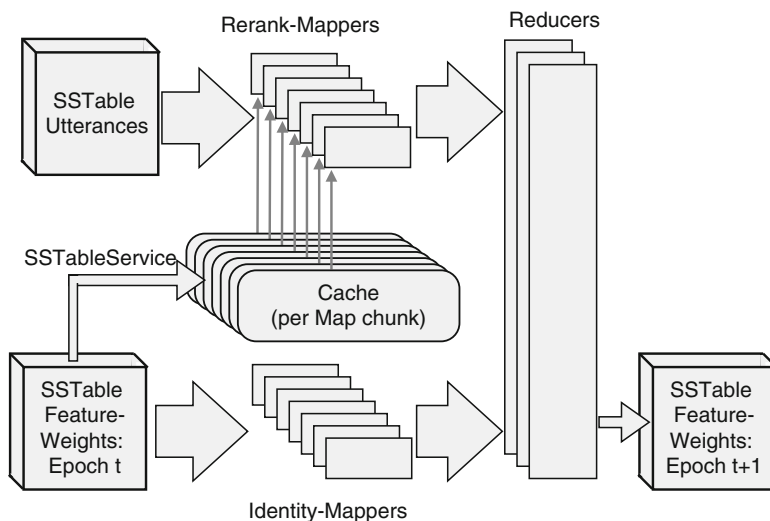


Fig. 8.6 MapReduce implementation of reranking using discriminative language models

The features corresponding to a Map chunk at the end of training epoch need to be made available to *Rerank-Mapper* in the subsequent training epoch. Instead of accessing the features on demand from the SSTables that store these feature weights, every *Rerank-Mapper* stores the features needed for the current Map chunk in a cache. Though the number of features stored in the SSTables are determined by the total number of training utterances, the number of features that are accessed by a *Rerank-Mapper* instance are only proportional to the chunk size and can be cached locally. This is an important implementation choice because it allows us to estimate very large distributed models: the bottleneck is no longer the total model size but rather the cache size that is in turn controlled by the Map chunk size. Section “Evaluating Our DLM Rescoring Framework on *weakAM-dev/test*” discusses in more detail about different model sizes and the effects of varying Map chunk size on recognition performance.

Figure 8.6 is a schematic diagram of our entire framework; Fig. 8.7 shows a more detailed representation of a single *Rerank-Mapper*, an *Identity-Mapper* and a *Reducer*, with the pseudocode of these interfaces shown inside their respective boxes. *Identity-Mapper* gets feature weights from the previous training epoch as input (\mathbf{w}') and passes them to the output unchanged. *Rerank-Mapper* calls the function `Rerank` that takes an N-best list of a training utterance (`utt.Nbest`) and the current feature weights (\mathbf{w}_{curr}) as input and reranks the N-best list to obtain the best scoring hypothesis. If this differs from the correct transcript for `utt`, `FeatureDiff` computes the difference in feature vectors corresponding to the two hypotheses (we call it δ) and \mathbf{w}_{curr} is incremented with δ . `Emit` is the output function of a

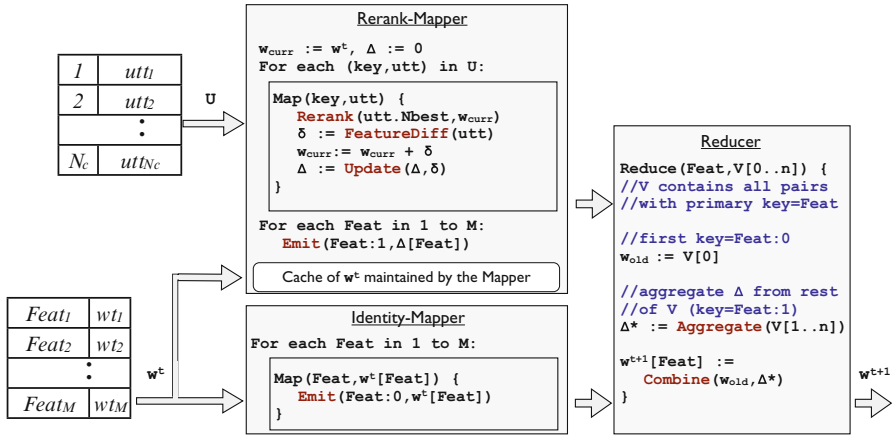


Fig. 8.7 Details of the Mapper and Reducer

Naive Distributed Perceptron:

- Update(Δ, δ) returns $\Delta + \delta$.
- Aggregate($[\Delta_1^t, \dots, \Delta_c^t]$) returns $\Delta^* = \sum_{c=1}^C \Delta_c^t$.
- Combine(w_{NP}^{t-1}, Δ^*) returns $w_{NP}^{t-1} + \Delta^*$.

Distributed Perceptron:

- Update and Combine are as for the *Naive Distributed Perceptron*.
- Aggregate($[\Delta_1^t, \dots, \Delta_c^t]$) returns $\Delta^* = \sum_{c=1}^C \mu_c \Delta_c^t$.

Averaged Distributed Perceptron: Here, $w^t = (w_{AV}^t, w_{DP}^t)$, and $\Delta = (\beta, \alpha)$ contain pairs of values; α is used to maintain w_{DP}^t and β , both of which in turn are used to maintain w_{AV}^t (α_c^t plays the role of Δ_c^t in *Distributed Perceptron*). Only w_{AV}^t is used in the final evaluation and only w_{DP}^t is used during training.

- Update($(\beta, \alpha), \delta$) returns $(\beta + \alpha + \delta, \alpha + \delta)$.
- Aggregate($[\Delta_1^t, \dots, \Delta_c^t]$) where $\Delta_c^t = (\beta_c^t, \alpha_c^t)$, returns $\Delta^* = (\beta^*, \alpha^*)$ where $\beta^* = \sum_{c=1}^C \beta_c^t$, and $\alpha^* = \sum_{c=1}^C \mu_c \alpha_c^t$.
- Combine($(w_{AV}^{t-1}, w_{DP}^{t-1}), (\beta^*, \alpha^*)$) returns $({}_{t-1}^{t-1} w_{AV}^{t-1} + \frac{1}{t} w_{DP}^{t-1} + \frac{1}{Nt} \beta^*, w_{DP}^{t-1} + \alpha^*)$.

Fig. 8.8 Update, Aggregate and Combine procedures for the three variants of the distributed perceptron algorithm

Mapper that outputs a processed key/value pair. For every feature Feat, both *Identity-Mapper* and *Rerank-Mapper* also output a secondary key (0 or 1, respectively); this is denoted as Feat:0 and Feat:1. At the *Reducer*, its inputs arrive sorted according to the secondary key; thus, the feature weight corresponding to Feat from the previous training epoch produced by *Identity-Mapper* will necessarily arrive before Feat’s current updates from the *Rerank-Mapper*. This ensures that w^{t+1} is updated correctly starting with w^t . The functions Update, Aggregate and Combine are explained in the context of three variants of the distributed perceptron algorithm in Fig. 8.8.

MapReduce Variants of the Distributed Perceptron Algorithm

Our MapReduce setup described in the previous section allows for different variants of the distributed perceptron training algorithm to be implemented easily. We experimented with three slightly differing variants of a distributed training strategy for the structured perceptron, *Naive Distributed Perceptron*, *Distributed Perceptron* and *Averaged Distributed Perceptron*; these are defined in terms of Update, Aggregate and Combine in Fig. 8.8 where each variant can be implemented by plugging in these definitions from Fig. 8.8 into the pseudocode shown in Fig. 8.7. We briefly describe the functionalities of these three variants. The weights at the end of a training epoch t for a single feature f are $(w_{NP}^t, w_{DP}^t, w_{AV}^t)$ corresponding to *Naive Distributed Perceptron*, *Distributed Perceptron* and *Averaged Distributed Perceptron*, respectively; $\phi(\cdot, \cdot)$ correspond to feature f 's value in Φ from Algorithm 1. Below, $\delta_{c,j}^t = \phi(x_{c,j}, y_{c,j}) - \phi(x_{c,j}, \tilde{y}_{c,j}^t)$ and $\mathcal{N}_c =$ number of utterances in Map chunk \mathcal{T}_c .

At the end of epoch t , the weight increments in that epoch from all map chunks are added together and added to w_{NP}^{t-1} to obtain w_{NP}^t .

Here, instead of adding increments from the map chunks, at the end of epoch t , they are averaged together using weights μ_c , $c = 1$ to \mathcal{C} , and used to increment w_{DP}^{t-1} to w_{DP}^t .

In this variant, firstly, all epochs are carried out as in the Distributed Perceptron algorithm above. But at the end of t epochs, all the weights encountered during the whole process, over all utterances and all chunks, are averaged together to obtain the final weight w_{AV}^t . Formally,

$$w_{AV}^t = \frac{1}{\mathcal{N} \cdot t} \sum_{t'=1}^t \sum_{c=1}^{\mathcal{C}} \sum_{j=1}^{\mathcal{N}_c} w_{c,j}^{t'},$$

where $w_{c,j}^t$ refers to the current weight for map chunk c , in the t th epoch after processing j utterances and \mathcal{N} is the total number of utterances. In our implementation, we maintain only the weight w_{DP}^{t-1} from the previous epoch, the cumulative increment $\gamma_{c,j}^t = \sum_{k=1}^j \delta_{c,k}^t$ so far in the current epoch, and a running average w_{AV}^{t-1} . Note that, for all c, j , $w_{c,j}^t = w_{DP}^{t-1} + \gamma_{c,j}^t$, and hence

$$\begin{aligned} \mathcal{N}t \cdot w_{AV}^t &= \mathcal{N}(t-1)w_{AV}^{t-1} + \sum_{c=1}^{\mathcal{C}} \sum_{j=1}^{\mathcal{N}_c} w_{c,j}^t \\ &= \mathcal{N}(t-1)w_{AV}^{t-1} + \mathcal{N}w_{DP}^{t-1} + \sum_{c=1}^{\mathcal{C}} \beta_c^t \end{aligned}$$

where $\beta_c^t = \sum_{j=1}^{\mathcal{N}_c} \gamma_{c,j}^t$. Writing $\beta^* = \sum_{c=1}^{\mathcal{C}} \beta_c^t$, we have $w_{AV}^t = \frac{t-1}{t} w_{AV}^{t-1} + \frac{1}{t} w_{DP}^{t-1} + \frac{1}{\mathcal{N}t} \beta^*$.

Experiments and Results

Our DLMS are evaluated in two ways: (1) we extract a development set (*weakAM-dev*) and a test set (*weakAM-test*) from the speech data that is re-decoded with a *weakAM*, and (2) we use a standard voice search test set (*v-search-test*) (Strope et al. 2011) to evaluate actual ASR performance on voice search. More details regarding our experimental setup along with a discussion of our experiments and results are described in the rest of the section.

Experimental Setup

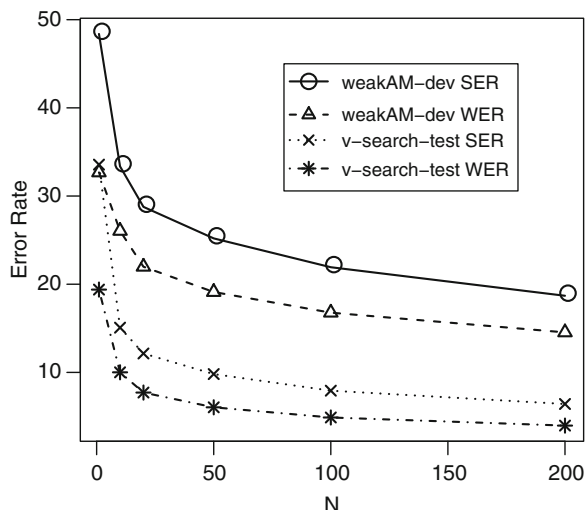
We generate training lattices using speech data that is re-decoded with a *weakAM* acoustic model and the baseline language model. We use maximum likelihood trained single mixture Gaussians for our *weakAM*. And, we use a sufficiently small baseline LM (~21 million n-grams) to allow for sub-real time lattice generation on the training data with a small memory footprint, without compromising on its strength. Chelba et al. (2010) demonstrate that it takes much larger LMs to get a significant relative gain in WER. Our largest discriminative language models are trained on 87,000h of speech, or ~350 million words (*weakAM-train*) obtained by filtering voice search logs at 0.8 confidence, and re-decoding the speech data with a *weakAM* to generate N-best lists. We set aside a part of this *weakAM-train* data to create *weakAM-dev* and *weakAM-test*: these data sets consist of 328,460/316,992 utterances, or 1,182,756/1,129,065 words, respectively.

We use a manually-transcribed, standard voice search test set (*v-search-test*) consisting of 27,273 utterances, or 87,360 words to evaluate actual ASR performance using our *weakAM*-trained models. All voice search data used in the experiments is anonymized.

Figure 8.9 shows oracle error rates, both at the sentence and word level, using N-best lists of utterances in *weakAM-dev* and *v-search-test*. These error rates are obtained by choosing the best of the top N hypotheses that is either an exact match (for sentence error rate) or closest in edit distance (for word error rate) to the correct transcript. The N-best lists for *weakAM-dev* are generated using a weak AM and N-best lists for *v-search-test* are generated using the baseline (strong) AM. Figure 8.9 shows these error rates plotted against a varying threshold N for the N-best lists. Note there are sufficient word errors in the *weakAM* data to train DLMS; also, we observe that the plot flattens out after N=100, thus informing us that N=100 is a reasonable threshold to use when training our DLMS.

Experiments in section “Evaluating Our DLM Rescoring Framework on *weakAM-dev/test*” involve evaluating our learning setup using *weakAM-dev/test*. We then investigate whether improvements on *weakAM-dev/test* translate to *v-search-test* where N-best are generated using the *strongAM*, and scored against *manual* transcripts using fully fledged text normalization instead of the string edit

Fig. 8.9 Oracle error rates at word/sentence level for *weakAM-dev* with the weak AM and *v-search-test* with the baseline AM



distance used in training the DLM. More details about the implications of this text normalization on WER can be found in section “Evaluating ASR Performance on *v-search-test* Using DLM Rescoring”.

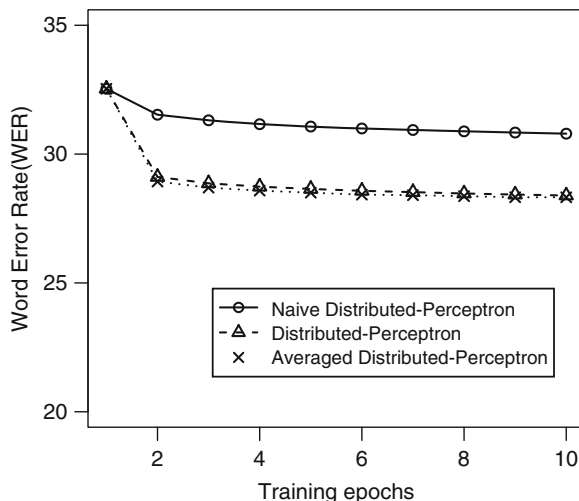
Evaluating Our DLM Rescoring Framework on weakAM-dev/test

Improvements on *weakAM-dev* Using Different Variants of Training for the DLMs

We evaluate the performance of all the variants of the distributed perceptron algorithm described in section “MapReduce Implementation Details” over 10 training epochs using a DLM trained on $\sim 20,000$ h of speech with trigram word features. Figure 8.10 shows the drop in WER for all the three variants. We observe that the *Naive Distributed Perceptron* gives modest improvements in WER compared to the baseline WER of 32.5%. However, averaging over the number of Map chunks as in the *Distributed Perceptron* or over the total number of utterances and total number of training epochs as in the *Averaged Distributed Perceptron* significantly improves recognition performance; this is in line with the findings reported in Collins (2002) and McDonald et al. (2010) of averaging being an effective way of adding regularization to the perceptron algorithm.

Our best-performing *Distributed Perceptron* model gives a 4.7% absolute ($\sim 15\%$ relative) improvement over the baseline WER of 1-best hypotheses in *weakAM-dev*. This, however, could be attributed to a combination of factors: the use of large amounts of additional training data for the DLMs or the discriminative

Fig. 8.10 Word error rates on *weakAM-dev* using *Perceptron*, *Distributed Perceptron* and *AveragedPerceptron* models



nature of the model. In order to isolate the improvements brought upon mainly by the second factor, we build an ML trained backoff trigram LM (ML-3 g) using the reference transcripts of all the utterances used to train the DLMs. The N-best lists in *weakAM-dev* are reranked using ML-3 g probabilities linearly interpolated with the LM probabilities from the lattices. We also experiment with a log-linear interpolation of the models; this performs slightly worse than rescoring with linear interpolation.

Impact of Varying Orders of N-gram Features

Table 8.9 shows that our best performing model (DLM-3 g) gives a significant $\sim 2\%$ absolute ($\sim 6\%$ relative) improvement over ML-3 g. We also observe that most of the improvements come from the unigram and bigram features. We do not expect higher order N-gram features to significantly help recognition performance; we further confirm this by building DLM-4 g and DLM-5 g that use up to 4- and 5-g word features, respectively. Table 8.10 gives the progression of WERs for 6 epochs using DLM-3 g, DLM-4 g and DLM-5 g showing minute improvements as we increase the order of Ngram features from 3 to 5.

Impact of Model Size on WER

We experiment with varying amounts of training data to build our DLMs and assess the impact of model size on WER. These are evaluated on the test set derived from the *weakAM* data (*weakAM-test*). Table 8.11 shows each model along with its size

Table 8.9 WERs on *weakAM-dev* using the baseline 1-best system, ML-3 g and DLM-1/2/3 g

Data set	Baseline (%)	ML-3 g (%)	DLM-1 g (%)	DLM-2 g (%)	DLM-3 g (%)
<i>weakAM-dev</i>	32.5	29.8	29.5	28.3	27.8

Table 8.10 WERs on *weakAM-dev* using DLM-3 g, DLM-4 g and DLM-5 g of 6 training epochs

Iteration	DLM-3 g (%)	DLM-4 g (%)	DLM-5 g (%)
1	32.53	32.53	32.53
2	29.52	29.47	29.46
3	29.26	29.23	29.22
4	29.11	29.08	29.06
5	29.01	28.98	28.96
6	28.95	28.90	28.87

Table 8.11 WERs on *weakAM-test* using DLMs of varying sizes

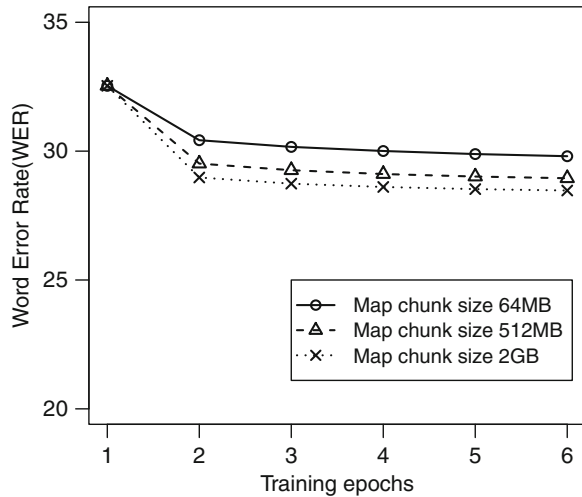
Model	Size (in millions)	Coverage (%)	WER (%)
Baseline	21	–	39.08
Model 1	65	74.8	34.18
Model 2	135	76.9	33.83
Model 3	194	77.8	33.74
Model 4	253	78.4	33.68

(measured in total number of word features), coverage on *weakAM-test* in percent of tokens (number of word features in *weakAM-test* that are in the model) and WER on *weakAM-test*. As expected, coverage increases with increasing model size with a corresponding tiny drop in WER as the model size increases. “Larger models”, built by increasing the number of training utterances used to train the DLMs, do not yield significant gains in accuracy. We need to find a good way of adjusting the model capacity with increasing amounts of data.

Impact of Varying Map Chunk Sizes

We also experiment with varying Map chunk size to determine its effect on WER. Figure 8.11 shows WERs on *weakAM-dev* using our best *Distributed Perceptron* model with different Map chunk sizes (64MB, 512MB, 2GB). For clarity, we examine two limit cases: (a) using a single Map chunk for the entire training data is equivalent to the conventional structured perceptron and (b) using a single training instance per Map chunk is equivalent to batch training. We observe that moving from 64MB to 512MB significantly improves WER and the rate of improvement in WER decreases when we increase the Map chunk size further to 2GB. We attribute

Fig. 8.11 Word error rates on *weakAM-dev* using varying Map chunk sizes of 64MB, 512MB and 2GB



these reductions in WER with increasing Map chunk size to on-line parameter updates being done on increasing amounts of training samples in each Map chunk. The larger number of training samples per Map chunk accounts for greater stability in the parameters learned by each Map chunk in our DLM.

Evaluating ASR Performance on *v-search-test* Using DLM Rescoring

We evaluate our best *Distributed Perceptron* DLM model on *v-search-test* lattices that are generated using a strong AM. We hope that the large relative gains on *weakAM-dev/test* translate to similar gains on this standard voice search data set as well. Table 8.12 shows the WERs on both *weakAM-test* and *v-search-test* using Model 1 (from Table 8.11).² We observe a small but statistically significant ($p < 0.05$) reduction ($\sim 2\%$ relative) in WER on *v-search-test* over reranking with a linearly interpolated ML-3 g. This is encouraging because we attain this improvement using training lattices that were generated using a considerably weaker AM.

It is instructive to analyze why the relative gains in performance on *weakAM-dev/test* do not translate to *v-search-test*. Our DLMs are built using N-best outputs from the recognizer that live in the “spoken domain” (SD) and the manually transcribed *v-search-data* transcripts live in the “written domain” (WD). The normalization of training data from WD to SD is as described in Chelba et al. (2010);

²We also experimented with the larger Model 4 and saw similar improvements on *v-search-test* as with Model 1.

Table 8.12 WERs on *weakAM-test* and *v-search-test*

Data set	Baseline (%)	ML-3 g (%)	DLM-3 g (%)
<i>weakAM-test</i>	39.1	36.7	34.2
<i>v-search-test</i>	14.9	14.6	14.3

inverse text normalization (ITN) undoes most of that when moving text from SD to WD, and it is done in a heuristic way. There is $\sim 2\%$ absolute reduction in WER when we move the N-best from SD to WD via ITN; this is how WER on *v-search-test* is computed by the voice search evaluation code. Contrary to this, in DLM training we compute WERs using string edit distance between test data transcripts and the N-best hypotheses and thus we ignore the mismatch between domains WD and SD. It is quite likely that part of what the DLM learns is to pick N-best hypotheses that come closer to WD, but may not truly result in WER gains after ITN. This would explain part of the mismatch between the large relative gains on *weakAM-dev/test* compared to the smaller gains on *v-search-test*. We could correct for this by applying ITN to the N-best lists from SD to move to WD before computing the oracle best in the list. An even more desirable solution is to build the LM directly on WD text; text normalization would be employed for pronunciation generation, but ITN is not needed anymore (the LM picks the most likely WD word string for homophone queries at recognition).

Optimal Size, Freshness and Time-Frame for Voice Search Vocabulary

In this section, we investigate how to optimize the vocabulary for a voice search language model. The metric we optimize over is the out-of-vocabulary (OOV) rate since it is a strong indicator of user experience; the higher the OOV rate, the more likely the user is to have a poor experience. Clearly, each OOV word will result in at least one error at the word level,³ and in exactly one error at the whole query/sentence level. In ASR practice, OOV rates below 0.01 (1%) are deemed acceptable since typical WER values are well above 10%.

As shown in Chelba et al. (2010), a typical vocabulary for a US English voice search language model (LM) is trained on the US English query stream, contains about 1 million words, and achieves out-of-vocabulary (OOV) rate of 0.57% on unseen text query data, after query normalization.

In a departure from typical vocabulary estimation methodology, (Jelinek 1990; Venkataraman and Wang 2003), the web search query stream not only provides us

³The approximate rule of thumb is 1.5 errors for every OOV word, so an OOV rate of 1% would lead to about 1.5% absolute loss in word error rate (WER).

with training data for the LM, but also with session level information based on 24-h cookies. Assuming that each cookie corresponds to the experience of a web search user over exactly 1 day, we can compute per-one-day-user OOV rates, and directly correlate them with the voice search LM vocabulary size (Kamvar and Chelba 2012).

Since the vocabulary estimation algorithms are extremely simple, the work presented here is purely experimental. Our methodology is as follows:

- Select as training data \mathcal{T} a set of queries arriving at the `google.com` front-end during time period T ;
- Text normalize the training data, see section “A Note on Query Normalization”;
- Estimate a vocabulary \mathcal{V} by thresholding the 1-g count of words in \mathcal{T} such that it exceeds C , $\mathcal{V}(T, C)$;
- Select as test data \mathcal{T} a set of queries arriving at the `google.com` front-end during time period E ; E is a single day that occurs after T , and the data is subjected to the exact same text normalization used in training;
- We evaluate both *aggregate* and *per-cookie* OOV rates, and report the aggregate OOV rate across all words in \mathcal{T} , as well as the percentage of cookies in \mathcal{T} that experience an OOV rate that is less or equal than 0.01 (1%).

We aim to answer the following questions:

- How does the vocabulary size (controlled by the threshold C) impact both *aggregate* and *per-cookie* OOV rates?
- How does the vocabulary freshness (gap between T and E) impact the OOV rate?
- How does the time-frame (duration of T) of the training data \mathcal{T} used to estimate the vocabulary $\mathcal{V}(T, C)$ impact the OOV rate?

A Note on Query Normalization

We build the vocabulary by considering all US English queries logged during T . We break each query up into words, and discard words that have non-alphabetic characters. We perform the same normalization for the test set. So for example if the queries in \mathcal{T} were: `gawker.com`, `pizza san francisco`, `baby food,4chan status` the resulting vocabulary would be `pizza`, `san`, `francisco`, `baby`, `food`, `status`. The query `gawker.com` and the word `4chan` would not be included in the vocabulary because they contain non-alphabetic characters.

We note that the above query normalization is extremely conservative in the sense that it discards a lot of problematic cases, and keeps the vocabulary sizes and OOV rates smaller than what would be required for building a vocabulary and language model that would actually be used for voice search query transcription. As a

Table 8.13 Vocabulary size as a function of count threshold

Threshold	Vocabulary size
15	3,643,583
30	2,277,696
60	1,429,888
120	901,213
240	569,330
480	361,776
960	232,808

result, the vocabulary sizes that we report to achieve certain OOV values are very likely just lower bounds on the actual vocabulary sizes needed, were correct text normalization (see Chelba et al. 2010 for an example text normalization pipeline) to be performed.

Experiments

The various vocabularies used in our experiment are created from queries issued during a 1-week–1-month period starting on 10/04/2011. The vocabulary is comprised of the words that were repeated C or more times in \mathcal{T} . We chose seven values for C : 960, 480, 240, 120, 60, 30 and 15. As C decreases, the vocabulary size increases; to preserve user privacy we do not use C values lower than 15. For each training set \mathcal{T} discussed in this paper, we will create seven different vocabularies based on these thresholds.

Each test set \mathcal{T} is comprised of queries associated with a set of over 10 million cookies during a 1-day period. We associate test queries by cookie-id in order to compute user-based (per-cookie) OOV rate.

All of our data is strictly anonymous; the queries bear no user-identifying information. The only query data saved after training are the vocabularies. The evaluation on test data is done by counting on streamed filtered query logs, without saving any data.

Vocabulary Size

To understand the impact of vocabulary size on OOV rate, we created several vocabularies from the queries issued in the week $T = 10/4/2011 - 10/10/2011$. The size of the various vocabularies as a function of the count threshold is presented in Table 8.13; Fig. 8.12 shows the relationship between the logarithm of the size of the vocabulary and the aggregate OOV rate—a log-log plot of the same data points

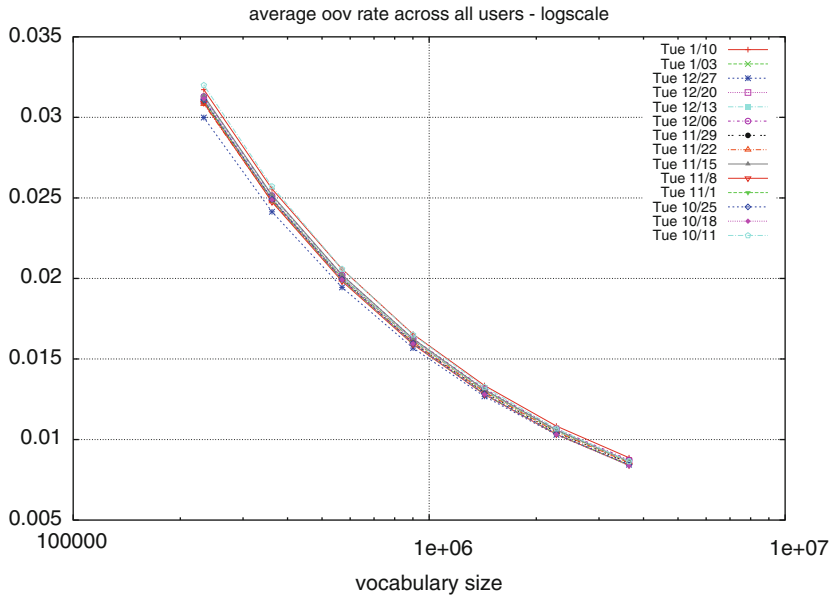


Fig. 8.12 Aggregate OOV rate as a function of vocabulary size (log-scale), evaluated on a range of test sets collected every Tuesday between 2011/10/11 and 2012/01/03

would reveal a “quasi-linear” dependency. We have also measured the percentage of cookies/users for a given OOV rate (0.01, or 1%), and the results are shown in Fig. 8.13. At a vocabulary size of 2.25 million words ($C = 30$, aggregate OOV = 0.01), over 90% of users will experience an OOV rate of 0.01.

Vocabulary Freshness

To understand the impact of the vocabulary freshness on the OOV rate, we take the seven vocabularies described above ($T = 10/4/2011 - 10/10/2011$ and $C = 960, 480, 240, 120, 60, 30, 15$) and investigate the OOV rate change as the lag between the training data T and the test data E increases: we used the 14 consecutive Tuesdays between 2010/10/11 and 2011/01/20 as test data. We chose to keep the day of week consistent (a Tuesday) across this set of E dates in order to mitigate any confounding factors with regard to day-of-week.

We found that within a 14-week time span, as the *freshness* of the vocabulary decreases, there is no consistent increase in the aggregate OOV rate (Fig. 8.12) nor any significant decrease in the percentage of users who experience less than 0.01 (1%) OOV rate (Fig. 8.13).

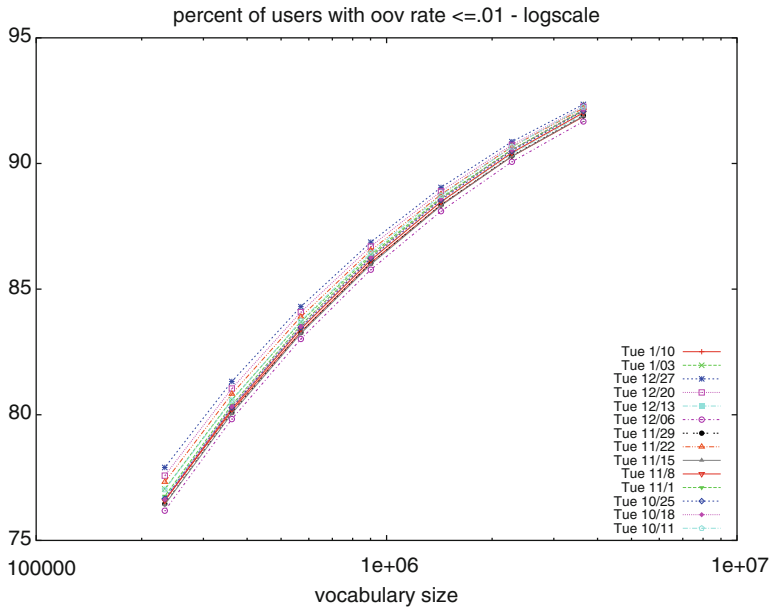


Fig. 8.13 Percentage of cookies/users with OOV rate less than 0.01 (1%) as a function of vocabulary size (log-scale), evaluated on test sets collected every Tuesday between 2011/10/11 and 2012/01/03

Vocabulary Time Frame

To understand how the duration of T (the time window over which the vocabulary is estimated) impacts OOV rate, we created vocabularies over the following time windows:

- 1 week period between 10/25/2011 and 10/31/2011
- 2 week period between 10/18/2011 and 10/31/2011
- 3 week period between 10/11/2011 and 10/31/2011
- 4 week period between 10/04/2011 and 10/31/2011

We again created seven threshold based vocabularies for each T . We evaluate the aggregate OOV rate on the date $E = 11/1/2011$, see Fig. 8.14, as well as the percentage of users with a per-cookie OOV rate below 0.01 (1%), see Fig. 8.15. We see that the shape of the graph is fairly consistent across T time windows, and a week of training data is as good as a month.

More interestingly, Fig. 8.15 shows that aiming at an operating point where 95% the percentage of users experience OOV rates below 0.01 (1%) requires significantly larger vocabularies, approx. 10 million words.

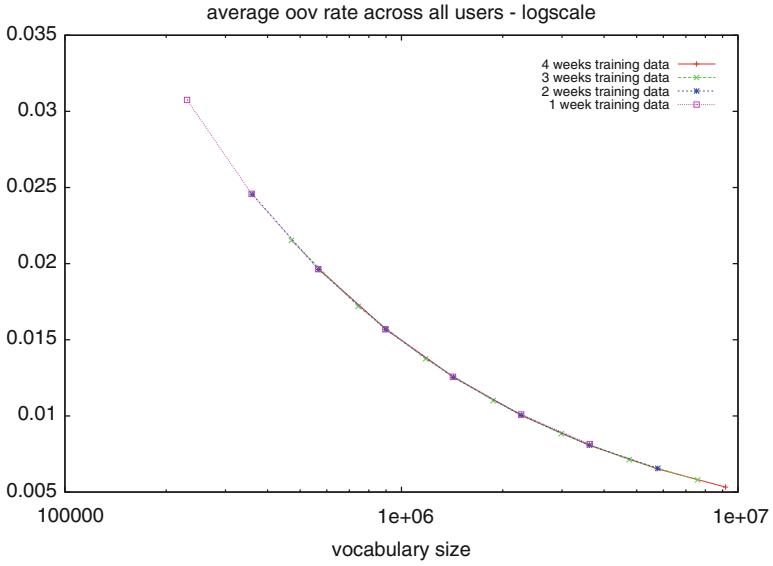


Fig. 8.14 Aggregate OOV rate on 11/1/2011 over vocabularies built from increasingly large training sets

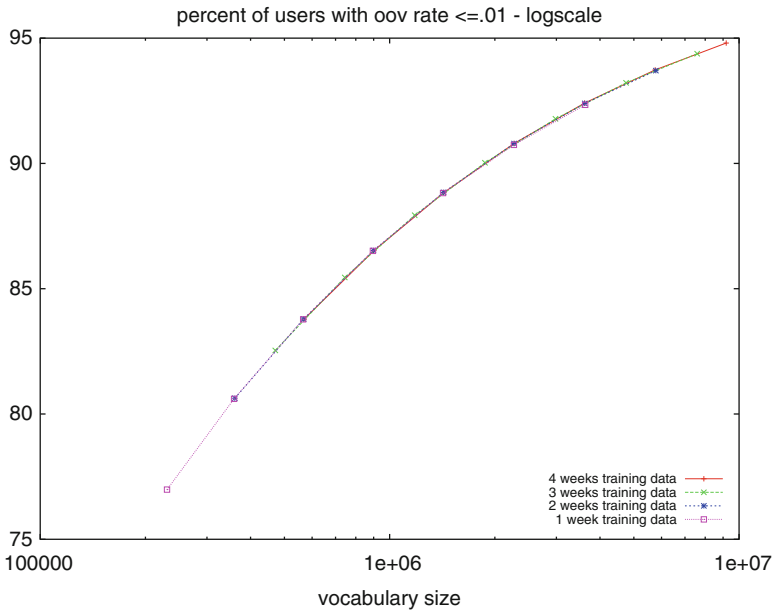


Fig. 8.15 Percentage of cookies/users with OOV rate less than 0.01 (1%) on 11/1/2011 over vocabularies built from increasingly large training sets

Conclusions: Language Modeling for Voice Search

Our experiments show that with careful text normalization the query stream is not as “wild” as it seems at first sight. One can achieve excellent OOV rates for a 1 million word vocabulary, and n -gram hit ratios of 77/88% even at $n = 5 / 4$, respectively.

Experimental evidence suggests that the query stream is non-stationary, and that more data does not automatically imply better models even when the data is clearly matched to the test data. More careful experiments are needed to adjust model capacity and identify an optimal way of blending older and recent data—attempting to separate the stationary/non-stationary components in the query stream. Less surprisingly, we have shown that locale matters significantly for English query data across USA, Great Britain and Australia.

We generally see excellent correlation of WER with PPL under various pruning regimes, as long as the training set and vocabulary stays constant.

As for leveraging the speech logs data for better language modeling, we successfully build large-scale discriminative N -gram language models with lattices regenerated using a weak AM and derive small but significant gains in recognition performance on a voice search task where the lattices are generated using a stronger AM. We use a very simple weak AM and this suggests that there is room for improvement if we use a slightly better “weak AM”. Also, we have a scalable and efficient MapReduce implementation that is amenable to adapting minor changes to the training algorithm easily and allows for training large LMs. The latter functionality will be particularly useful if we generate the contrastive set by sampling from text instead of re-decoding logs (Jyothi and Fosler-Lussier 2010).

A more careful analysis of vocabulary estimation for voice search shows that a significantly larger vocabulary (approx. 10 million words) seems to be required to guarantee a 0.01 (1%) OOV rate for 95% of the users.

Studies on the www pages side (Brants) show that after just a few million words, vocabulary growth is close to a straight line in the logarithmic scale; the vocabulary grows by about 69% each time the size of the text is doubled even when using one trillion words of training data. Since queries are used for finding such pages, the growth in query stream vocabulary size is easier to understand.

We also find that 1 week is as good as 1 month of data for estimating the vocabulary, and that there is very little drift in OOV rate as the test data (1 day) shifts during the 3 months following the training data used for estimating the vocabulary.

References

- Allauzen C, Riley M, Schalkwyk J, Skut W, Mohri M. (2007) OpenFst: a general and efficient weighted finite-state transducer library. In: Proceedings of the ninth international conference on implementation and application of automata, (CIAA 2007). Lecture notes in computer science, vol 4783. Springer, pp 11–23. <http://www.openfst.org>
- Allauzen C, Schalkwyk J, Riley M (2009) A generalized composition algorithm for weighted finite-state transducers. In: Proceedings of Interspeech, Brighton, pp 1203–1206

- Banko M, Brill E (2001) Mitigating the paucity-of-data problem: exploring the effect of training corpus size on classifier performance for natural language processing. In: Proceedings of the first international conference on human language technology research, HLT '01, San Diego. Association for Computational Linguistics, Stroudsburg, pp 1–5
- Brants T Vocabulary growth. In: Kordoni V (ed) Festschrift for Hans Uszkoreit. CSLI Publications, to appear, Stanford, CA 94305
- Brants T, Xu P (2009) Distributed language models. In: HLT-NAACL tutorial abstracts, Boulder pp 3–4
- Brants T, Popat AC, Xu P, Och FJ, Dean J (2007) Large language models in machine translation. In: Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL), Prague, pp 858–867
- Chelba C, Schalkwyk J, Brants T, Ha V, Harb B, Neveitt W, Parada C, Xu P (2010) Query language modeling for voice search In: Proceedings of SLT, Berkeley
- Chu CT, Kim SK, Lin YA, Yu YY, Bradski G, Ng AY, Olukotun K (2007) Map-reduce for machine learning on multicore. Proc NIPS 19:281
- Collins M (2002) Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In: Proceedings of EMNLP, Philadelphia
- Gao J, Yu H, Yuan W, Xu P (2005) Minimum sample risk methods for language modeling. In: Proceedings of EMNLP, Vancouver
- Ghemawat S, Dean J (2004) Mapreduce: Simplified data processing on large clusters. In: Proceedings of OSDI, San Francisco
- Goodman J (2001) A bit of progress in language modeling, extended version. Technical Report, Microsoft Research
- Halevy A, Norvig P, Pereira F (2009) The unreasonable effectiveness of data. IEEE Intell Syst 24(2):8–12
- Hall KB, Gilpin S, Mann G (2010) MapReduce/Bigtable for distributed optimization. In: NIPS LCCC Workshop, Whistler, BC
- Harb B, Chelba C, Dean J, Ghemawat S (2009) Back-off language model compression. In: Proceedings of Interspeech, Brighton. ISCA, pp 325–355
- Jelinek F (1990) Self-organized language modeling for speech recognition. In: Waibel A, Lee K-F (eds) Readings in speech recognition. Morgan Kaufmann Publishers, San Mateo, pp 450–506
- Jelinek F (1997) Information extraction from speech and text. MIT Press, Cambridge, MA, pp 141–142. Chap. 8
- Jyothi P, Fosler-Lussier E (2010) Discriminative language modeling using simulated ASR errors. In: Proceedings of Interspeech, Makuhari
- Jyothi P, Johnson L, Chelba C, Strobe B (2012) Distributed discriminative language models for Google voice-search. In: Proceedings of ICASSP, Kyoto
- Kamvar M, Chelba C (2012) Optimal size, freshness and time-frame for voice search vocabulary. Google Tech Report
- Katz S (1987) Estimation of probabilities from sparse data for the language model component of a speech recognizer. IEEE Trans Acoust Speech Signal Process 35:400–401
- Kneser R, Ney H (1995) Improved backing-off for m-gram language modeling. Proc IEEE Int Conf Acoust Speech Signal Process 1:181–184
- Mann G, McDonald R, Mohri M, Silberman N, Walker D (2009) Efficient large-scale distributed training of conditional maximum entropy models. In: Proceedings of NIPS, Vancouver
- McDonald R, Hall K, Mann G (2010) Distributed training strategies for the structured perceptron. In: Proceedings of NAACL, Los Angeles
- Och FJ (2005) Statistical machine translation: foundations and recent advances. In: Presentation at MT-Summit. Phobert, Thailand
- Paul DB, Baker JM (1992) The design for the Wall Street Journal-based CSR corpus. In: Proceedings of the workshop on speech and natural language, HLT '91, Harriman, New York. Association for Computational Linguistics, Stroudsburg, pp 357–362
- Roark B, Saraçlar M, Collins M, Johnson M (2007) Discriminative n-gram language modeling. Computer Speech and Language, 21(2): 373–392

- Schlüter R, Müller B, Wessel F, Ney H (1999) Interdependence of language models and discriminative training. In: Proceedings of ASRU, Keystone
- Strope B, Beeferman D, Gruenstein A, Lei X (2011) Unsupervised testing strategies for ASR. In: Proceedings of Interspeech, Florence
- Venkataraman A, Wang W (2003) Techniques for effective vocabulary selection. Arxiv preprint cs/0306022
- Zhou Z, Gao J, Soong FK, Meng H (2006) A comparative study of discriminative methods for reranking LVCSR N-best hypotheses in domain adaptation and generalization. In Proceedings of ICASSP, Toulouse