Personalizing Model M for Voice-search

Geoffrey Zweig[1], *Shuangyu Chang*[2]

[1]Microsoft Research, Redmond, WA
[2]Speech at Microsoft, Mountain View, CA
{gzweig, shchang}@microsoft.com

Abstract

Model M is a recently proposed class based exponential n-gram language model. In this paper, we extend it with personalization features, address the scalability issues present with large data sets, and test its effectiveness on the Bing Mobile voice-search task. We find that Model M by itself reduces both perplexity and word error rate compared with a conventional model, and that the personalization features produce a further significant improvement. The personalization features provide a very large improvement when the history contains a relevant query; thus the overall effect is gated by the number of times a user re-queries a past request.

Index Terms: voice search, language modeling, speech recognition, personalization

1. Introduction

Model M is a recently proposed class based exponential n-gram language model [1, 2] which has shown improvements across a variety of LVCSR tasks [2, 3, 4]. The key ideas present are the modeling of word n-gram probabilities with a log-linear model, and the use of word-class information in the definition of the features. Assuming an n-gram model on words w, the form of the basic exponential language model is

$$P(w_i|w_{i-n+1}\dots w_{i-1}) = \frac{\exp(\lambda_{w_{i-n+1}\dots w_{i-1}w_i} + \dots + \lambda_{w_{i-1}w_i} + \lambda_{w_i})}{\sum_{w'} \exp(\lambda_{w_{i-n+1}\dots w_{i-1}w'} + \dots + \lambda_{w_{i-1}w'} + \lambda_{w'})} \quad (1)$$

In other words, it is a log-linear model in which the features are 0/1 and each suffix in the n-gram activates a feature. The model is trained to maximize the data probability, subject to $L1$ and $L2$ regularization. While this model supports arbitrary length n-grams, for simplicity of notation, for the remainder of this paper we will assume trigram models, with the extension to higher order n-grams being immediate.

In [1], it was observed that the weighted sum of the training-set perplexity and the $L1$ norm of the λs is a good predictor of test set perplexity. This was used to motivate a class-based exponential language model in which each word w is assigned deterministically to a single class c, and n-gram probabilities are estimated as the product of a class part and a word part

$$P(w_i|w_{i-2}w_{i-1}) = P(c_i|c_{i-2}c_{i-1}, w_{i-2}w_{i-1})P(w_i|w_{i-2}w_{i-1}, c_i).$$

Both components are themselves exponential n-gram models:

$$P(w_i|w_{i-2}w_{i-1}, c_i) = \frac{\exp(\lambda_{w_{i-2}w_{i-1}w_i} + \lambda_{w_{i-1}w_i} + \lambda_{w_i})}{\sum_{w' \in c_i} \exp(\lambda_{w_{i-2}w_{i-1}w'} + \lambda_{w_{i-1}w'} + \lambda_{w'})} \quad (2)$$

$$P(c_i|c_{i-2}c_{i-1}, w_{i-2}w_{i-1}) = \frac{\exp(\begin{array}{r}\lambda_{c_{i-2}c_{i-1}c_i} + \lambda_{c_{i-1}c_i} + \lambda_{c_i} \\ + \lambda_{w_{i-2}w_{i-1}c_i} + \lambda_{w_{i-1}c_i}\end{array})}{\sum_{c'} \exp(\begin{array}{r}\lambda_{c_{i-2}c_{i-1}c'} + \lambda_{c_{i-1}c'} + \lambda_{c'} \\ + \lambda_{w_{i-2}w_{i-1}c'} + \lambda_{w_{i-1}c'}\end{array})} \quad (3)$$

Here we have used c_i to represent the class of word w_i and $w' \in c_i$ to range over the members of class c_i. A λ parameter may be defined for each n-gram pattern in the training data, or restricted to commonly occurring patterns. The word classing may be done with a variety of methods; in this work, we use the original IBM classing mechanism [5]. In [2], empirical evidence is presented which indicates that the $L1$ norm of a model of this form is generally smaller than that of a purely word-based exponential model providing equal perplexity; better test set performance is observed as well.

Note that Model M is factored into separate class and word models. In an alternative formulation, a similar model could be formed simply by adding class features to an unfactored model of the form of Eq'n. 1. In fact, while this approach is more general, the Model M approach makes an efficient implementation much more straightforward. In a basic n-gram model, the normalization must be computed over all the words in the vocabulary, often 10^5 or more. In Model M, however, there are two much smaller normalizations: first the normalization over classes (typically on the order of 10^2) and then the normalization over the words in a *specific class*. Assuming the words are equally divided across classes, this results in around two orders of magnitude speedup relative to a naive implementation of an unfactored model.

To date, Model M has shown improvements on a number of standard speech recognition tasks: Wall Street Journal [2, 6], GALE Arabic data [4, 3], Voicemail and Broadcast News [3]. These tasks all fit well the standard paradigm of building a single language model, and then using it unchanged for the entire data-set. In this paper, we turn to a task in which we would like a highly dynamic model that is adjusted on an utterance-by-utterance basis. Specifically, we examine the problem of speech recognition for Bing Mobile voice-search [7]. In this task, we have an extremely large number of individual users, each with their own unique history of interaction with the system, which we would like to leverage to improve performance.

In principle, personalization might be viewed as an extreme form of domain adaptation subject to the MDI and MAP adaptation described in [8, 9, 10, 2, 3]. In these approaches, a general

language model is taken as a prior on a domain specific model, and the domain specific model is trained in that context. In the context of voice-search, however, this is undesirable because it is infeasible to train a separate, complex model for millions of users. Instead, we propose a very fast method that can be applied "on-the-fly," by adding a handful of feature weights, which are determined once only, by minimizing error rate on a development set.

In the context of voice-search, a second challenge relates simply to reducing the computational complexity to the point where it is feasible to train on billion-plus word data-sets. Model M itself takes the first step by allowing for a decomposition of the computations into two parts - one where the normalization is proportional to the number of classes, and one where it is proportional to the average number of members in a class. We further adapt ideas from [11] to reduce the computation involved in the second part to the average number of *bigram successors* of a word, where those successors are further restricted to belong to a specific class.

The remainder of this paper is organized as follows. In section 2, we describe our implementation of Model M computations. In section 3, we describe the personalization features we use. Section 4 describes the Bing Mobile data-set and our experimental results. Discussion and concluding remarks are offered in Section 5.

2. Computational Efficiency

2.1. Training Data Organization

Depending on the data-source and amount of data, a factor of 20 to 70 speedup can obtained by careful data organization. First, we apply the standard pre-processing step in which every ngram occurrence is explicitly written to a file, and then the occurrences of a specific ngram are made contiguous by sorting. Each ngram is then represented once with a count. All gradient computations involving the ngram are simply weighted by this count, which results in a factor of 6 to 15 reduction in computation. Explicitly replacing unknown words by <unk> at this step results in a further reduction. Another speedup results from realizing that the normalization factors need only be computed for the class part of the model when the $n-1$ gram word prefix changes, and for the word part of the model when either the $n-1$ gram word prefix changes, or the class of the predicted word changes. To exploit this fact, the data is sorted by $n-1$ gram word history, with ties broken on the basis of the class of the last word. Then normalizers are only computed as necessary. This typically results in about a factor of 3 speedup in training [1].

2.2. Bigram Restrictions

In our experiments, the number of classes is typically 200, and the vocabulary 150k words. Thus, the word normalization on average involves summing over about 750 words, which is more than the typical number of bigram successors that a word has. We may take advantage of the fact that most word pairs are never seen in order to reduce this to a computation proportional to the number of bigram successors of the second to last word. Recall that $\lambda_{w_{i-2}w_{i-1}w'}$ and $\lambda_{w_{i-1}w'}$ only exist for n-grams that occur in the training data. Thus, if we define $succs(c_i, w_{i-1})$ to be the words in class c_i that have been seen following w_{i-1}, we may write:

$$\sum_{w' \in c_i} \exp(\lambda_{w_{i-2}w_{i-1}w'} + \lambda_{w_{i-1}w'} + \lambda_{w'}) = \sum_{w' \in c_i} \exp(\lambda_{w'}) \quad (4)$$

$$+ \sum_{w' \in succs(c_i, w_{i-1})} \exp(\lambda_{w_{i-2}w_{i-1}w'} + \lambda_{w_{i-1}w'} + \lambda_{w'}) \quad (5)$$

$$- \sum_{w' \in succs(c_i, w_{i-1})} \exp(\lambda_{w'}) \quad (6)$$

In this, line (4) is computed once per class, after each round of parameter re-estimation. Lines (5) and (6) are computed on-demand, and only require looking at bigram successors of the second-to-last word. This results in a 30% to 40% speedup.

2.3. R-prop Initialization

We implement $L1$ and $L2$ regularization along the lines of [1], and have found the R-prop [12] gradient descent method more efficient in this context than generalized iterative scaling (GIS) [13]. R-prop maintains a step size for each dimension, and dynamically adjusts this – when the gradient maintains sign on successive iterations, the step size is increased; when it changes sign, the step size is reduced. While effective, we have found that it is sensitive to the initial step sizes, and as a efficiency measure, we initialize the step size by running GIS for two iterations, and using the absolute value of the change in the second step. This results in slightly better solutions in a fixed number of iterations.

2.4. Caching

During training, parameter values change, so new normalizers must be computed at each iteration. A significant speedup results at test time, however, by caching the normalizers used during training so that they do not need to be recomputed. Table 1 gives a rough idea of our estimates of the speedups from the various methods, for two data sets: a 244M word set used in the rest of our experiments, and a 2G word mobile text query set. Some absolute training times are presented in Sec. 4; at test time, with caching, we achieve 50 to 100k ngram computations per second for typical models.

3. Personalization Features

In previous work, we have explored the use of personalization by rescoring n-best lists at the sentence level [14], and modeling repetition with a cache-based language model [15, 16]. Here, we explore the possibility of dynamically creating a personalized version of Model M which operates n-gram by n-gram and could potentially be used in first pass decoding.

In principle, this personalization can be done by adapting a general domain model using minimum discrimination information (MDI) [8, 2] or MAP techniques [9, 10]. However, these methods require re-training for each user and are thus prohibitive. Instead, we take the approach of adding a new binary feature for every feature in the original model, and tying together the feature weights so that a small number of parameter values can be estimated once, with reference to held out data. The feature meaning and tying is as follows:

- For every original feature, there is a personalization feature whose value is 1 when the feature occurs for some

[1] After submission, Sethy et al. presented a paper "Distributed Training of Large Scale Exponential Language Models" (ICASSP 2011) describing further data organization methods.

Speedup	244M words	2G words
Factored Normalization	~160x	~160x
Count & <unk> Compression	7.4	20
Sorting by Class	2.8	3.4
Bigram Restrictions	1.4	1.3
Caching Normalizer	31	42

Table 1: Approximate effect of different speedups for two training set sizes. Times faster than without method.

average words per utterance	3.3
fraction having a history	96%
utterances with an exact match in history	11%
fraction of utterances with at least one matched word	55%
average word matches per utterance	1.4
fraction of words seen in history	41%

Table 2: Bing Mobile data characteristics.

n-gram in the user's history.

- The weight of a feature is determined by the feature's length - there is one weight for unigram features, one for bigrams, and so on.
- The same weights are used with the class-prediction and word-prediction parts of the model. (Note that the class-prediction part has features that involve both words and classes.)

This use of feature augmentation for adaptation is also found in [17]; however that work still requires training a complete new model for each condition. Through the parameter tying described, we effectively introduce just a handful of actual features (despite the notionally large number), thus enabling once-only optimization on the development data. We compare this type of model with an interpolation model in which the likelihoods of Model M and a unigram cache-based model are linearly interpolated.

4. Experimental Results

4.1. Bing Mobile Data

To test Model M and our personalization approaches, we use data from the Bing Mobile voice-search application [7], collected in October 2010. The Bing Mobile application provides users with a voice interface to the Bing search engine, and supports all types of queries. While it is sometimes difficult to categorize a query, approximately 40% are what have been traditionally called local business queries, e.g. "Merchandise Mart Chicago," somewhat over 50% are web queries, e.g. "Show me pictures of galaxies," perhaps 5% are street addresses, and a small number are non-intentional, e.g. "This is John Doe can you hear me?". From the October data, we randomly selected 3000 development set utterances and 6000 test utterances. The language model training data consists of 50M queries comprising 244M words, taken from the logs of users, exclusive of all users in the dev and test sets. These logs include both text queries and high-confidence recognition results from voice queries. The baseline sentence error rates are produced by using the deployed acoustic model in association with a language model built with the 244M word training data set. An unpruned language model was built, followed by Stolcke pruning [18] to a manageable size. N-best lists were generated for rescoring, using a confidence measure based cutoff which makes N variable; the average depth is 3.8. Table 2 illustrates some of the characteristics of the development set. Figure 1 shows how much the utterances overlap with the user histories, both at the whole-utterance and word levels.

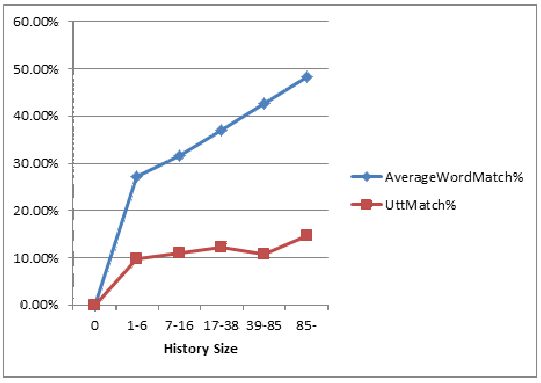

Figure 1: Word and utterance matches by history size.

Class 1	Class 2	Class 3
walmart	on	art
craigslist	with	williams
coffee	from	jones
starbucks	by	brothers
target	about	hunter
mcdonald's	without	energy
costco	between	twins
gamestop	into	castle
walgreens	around	bush
lowes	during	parker

Table 3: Most frequent members of three classes.

4.2. Results

4.2.1. Model M: Classes and Perplexity

Table 3 illustrates examples of three different classes. Some, such as the first two, appear relatively meaningful; the first is dominated by frequently requested businesses, and the second a nice list of English prepositions. Other classes, such as the third, have members without an obvious relationship.

Table 4 shows the relationship between the number of classes, perplexity and training time. With few classes, the computation is dominated by the normalization of the word part of the model; with many classes, by the normalization of the class part of the model. In these and other experiments, we have found 200 classes to be a good compromise.

In Table 5, we present perplexity results for both standard, and Model M based LMs. The standard language models were built with the CMU toolkit, using Good Turing smoothing (GT). For Model M, we created features for all word unigrams and bigrams seen in the data; trigram features were only created when seen two or more times, and fourgram features when seen three or more times. The same cutoffs were used with the conventional language model. We see that Model M typically provides a 10% improvement in perplexity, and that it is better able to use a higher-order model.

Number of Classes	Perplexity	# Features	Runtime
50	162	28.8M	20.6 hrs
100	160	30.3	17.4
200	160	32.5	17.6
400	158	35	23.1

Table 4: Perplexity as a function of number of classes. 150k vocabulary. Runtime is training time for 20 iterations on a 3.33GHz Xeon, no parallelization. 244M words training data.

Training Sents	3gm GT	4gm GT	3gm M	4gm M
5M	224	222	199	190
10M	204	202	180	170
20M	191	188	169	159
50M	179	176	157	146

Table 5: Development set perplexity - effect of amount of training data and n-gram level.

4.2.2. Personalization and Error Rates

We now turn to the use of personalization features. Recall from section 3 that with a trigram model there are six weights to estimate. These weights were optimized by hand on the development data, and Table 6 shows the effect of Model M and personalization on the test set sentence error rate. The baseline decoding was done with a Stolcke-pruned model that has fewer n-grams than Model M. To provide a better comparison, we built a conventional unpruned model with the same cutoffs (Unpruned ARPA). Model M itself produces about a 0.3% absolute gain over the larger LM, and interpolation with the original scores increases this further. With personalization features, Model M improves the error rate by 1% and 1.2% absolute for the 3-gram and 4-gram models respectively. Finally, when we augment the histories with the correct word sequence and re-optimize the personalization feature weights, there is a dramatic improvement, essentially to the error-rate floor that is imposed by the use of n-best lists.

The approach of personalization by adding features is general and can be extended in other ways, e.g. with localization features. However, it has the disadvantage that the fast normalization of Sec. 2.2 cannot be used. Therefore, we experimented with an alternative form of personalization, where we used the user histories to create a unigram cache language model for each user. The history based probability was then interpolated with the standard Model M probability, resulting in similar error rates as with feature addition.

5. Conclusion

The Voice-search task is significantly different from other language modeling tasks: the utterances tend to be very short, yet unconstrained and open-domain, and we have a user's specific history to exploit. We have seen that Model M, which has previously been applied in more standard LVCSR tasks, is also quite effective in this domain, and can be extended with user-personalization in a simple, efficient way to improve performance.

Model	3gm	4gm
Baseline	44.2%	44.1%
Unpruned ARPA	44.0	43.8
Model M	43.7	43.5
+ Interpolation	43.5	42.9
+ Histories	43.0	42.6
+ Correct Histories	35.8	35.8
N-best Oracle	34.8	34.6

Table 6: Sentence Error Rates under various conditions.

Acknowledgments

The authors thank Stanley Chen for his helpful comments.

6. References

[1] S. Chen, "Performance prediction for exponential language models," in *NAACL-HLT*, 2009.

[2] ——, "Shrinking exponential language models," in *NAACL-HLT*, 2009.

[3] S. Chen, L. Mangu, B. Ramabhadran, R. Sarikaya, and A. Sethy, "Scaling shrinkage-based language models," in *ASRU*, 2009.

[4] A. Emami, S. Chen, A. Ittycheriah, H. Soltau, and B. Zhao, "Decoding with shrinkage-based language models," in *Interspeech*, 2010.

[5] P. Brown, V. D. Pietra, P. deSouza, J. Lai, and R. Mercer, "Class-based n-gram models of natural language," *Computational Linguistics*, vol. 18, no. 4, 1992.

[6] S. Chen and S. Chu, "Enhanced word classing for model m," in *Interspeech*, 2010.

[7] A. Acero, N. Bernstein, R. Chambers, Y. Ju, X. Li, J. Odell, P. Nguyen, O. Scholz, and G. Zweig, "Live Search for Mobile: Web Services by Voice on the Cellphone," in *Proc. of ICASSP*, 2007.

[8] S. D. Pietra, V. D. Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Tansactions Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, 1997.

[9] C. Chelba and A. Acero, "Adaptation of maximum entropy capitalizer: Little data can help a lot," in *EMNLP*, 2004.

[10] Y.-H. Sung, C. Boulis, and D. Jurafsky, "Maximum conditional likelihood linear regression and maximum a posteriori for hidden conditional random fields speaker adaptation," in *ICASSP*, 2008.

[11] J. Wu and S. Khudanpur, "Efficient training methods for maximum entropy language modeling," in *Interspeech*, 2000.

[12] M. Reidmiller, "Rprop - Description and Implementation Details," University of Karlsruhe, Tech. Rep., January 1994.

[13] J. Darroch and D. Ratcliff, "Generalized iterative scaling for log-linear models," *Ann. Math. Statist.*, vol. 43, no. 5, 1972.

[14] D. Bolanos and G. Zweig, "Multi-scale personalization for voice-search applications," in *HLT-NAACL*, 2009.

[15] G. Zweig, D. Bohus, X. Li, and P. Nguyen, "Structured Models for Joint Decoding of Repeated Utterances," in *Proc. Interspeech*, 2008.

[16] G. Zweig, "New Methods for the Analysis of Repeated Utterances," in *Proc. Interspeech*, 2009.

[17] H. Daume, "Frustratingly easy domain adaptation," in *ACL*, 2007.

[18] A. Stolcke, "Entropy-based pruning of backoff language models," in *DARPA Broadcast News Transcription and Understanding Workshop*, 1998.