Randomized Maximum Entropy Language Models

Puyang Xu, Sanjeev Khudanpur, Asela Gunawardana[#]

Department of Electrical & Computer Engineering
Center of Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218, USA
{puyangxu, khudanpur}@jhu.edu

[#] *Microsoft Research*
[#]*Redmond, WA 98052, USA*
[#] aselag@microsoft.com

***Abstract*—We address the memory problem of maximum entropy language models(MELM) with very large feature sets. Randomized techniques are employed to remove all large, exact data structures in MELM implementations. To avoid the dictionary structure that maps each feature to its corresponding weight, the feature hashing trick [1] [2] can be used. We also replace the explicit storage of features with a Bloom filter. We show with extensive experiments that false positive errors of Bloom filters and random hash collisions do not degrade model performance. Both perplexity and WER improvements are demonstrated by building MELM that would otherwise be prohibitively large to estimate or store.**

I. Introduction

Language models(LM) are crucial components in automatic speech recognition(ASR) systems. They assign probabilities to sequences of words, usually by modeling the set of conditional distributions $\{P(w|h)\}$, as shown in (1), where h_i is the word history preceding the i^{th} word w_i.

$$P(w_1, w_2, ..., w_l) = \prod_{i=1}^{l} P(w_i|h_i). \tag{1}$$

The n-gram LM is the conventional approach to parameterize $P(w|h)$, where it is assumed that the word w only depends on its previous $n-1$ words. Therefore, the number of parameters in the model depends on the number of distinct word sequences of length up to n. As the corpus size increases, it is usually hard to estimate high order n-gram LMs because of the exponential increase of the number of n-grams in the training corpus.

The maximum entropy LM(MELM) [3] is an alternative way to parameterize $P(w|h)$. It provides a principled framework to incorporate different knowledges sources in the form of feature constraints. Specifically, for word w following h, we have

$$P(w|h) = \frac{\exp \sum_i \theta_i f_i(h, w)}{\sum_{w' \in V} \exp \sum_i \theta_i f_i(h, w')}, \tag{2}$$

where f_i is the i^{th} feature function defined over the word-history pair, θ_i is the feature weight associated with f_i. As a starting point for building stronger LM, n-gram information is usually included into the model by defining n-gram features corresponding to each distinct n-grams in the training corpus, thus the memory challenge for large scale MELM is clearly no less severe than the standard n-gram LM. Furthermore, besides nice theoretical properties in terms of smoothness, the true strength of MELM lies in its flexible framework to include constraints other than n-grams. Unfortunately, despite the empirical success of adding more knowledge into MELM [4], including these features together with n-gram constraints poses considerable challenges in terms of storage. For a lot of applications where remote or disk readings are prohibitively expensive, it is very difficult to deploy MELM with large feature sets because they cannot be stored in memory on a single machine.

For the standard n-gram LM, a lot of work has focused on reducing its size. A few examples are the entropy based pruning in [5], clustering in [6], Golomb coding in [7]. People have also used various data structures such as suffix array in [8], tries in [9], to represent n-gram LMs compactly. Among these techniques, the randomized schemes [10] [11] are probably the most succinct ones. Differing from all other approaches, the randomized models do not attempt to store n-grams explicitly, *Bloom filter* [12] -type structures are employed which only store *fingerprints* of the n-grams. However, the drastic space savings come with a cost – as a randomized structure for representing sets, Bloom filters may occasionally return nonexistent items with some small quantifiable probabilities, known as false positives. On the other hand, false negatives never occur. The effect of such one sided errors has proven to be sufficiently small and does not prevent us from deriving a quality LM.

The spectacular memory performance of randomized LM inspires us to explore randomized techniques to encode MELM, which is the focus of this paper. In order to reduce the memory requirement of training and using MELM, we propose to make use of the recently introduced feature hashing techniques [1] [2]. Central to the hashing idea is allowing feature collisions. By allowing multiple features to be mapped to the same weight value, we no longer have to store in memory the expensive dictionary structure to map from f_i to θ_i. It has

been shown that given a reasonably good hash function, such random collisions have limited impact on the resulting model because the learning algorithms can usually deal with such noise in the feature representation. Besides removing exact feature mapping, as we will discuss in later sections, we also have to avoid explicit storage of the features, for which exact techniques are expensive as well. To solve this problem, we employ Bloom filters. Therefore, our approach can be briefly described as storing features with Bloom filters and mapping them randomly to the weight vector. By avoiding all of the large exact data structures in our MELM, we introduce two sources of randomness: (i) Due to false positives in Bloom filters, some features that are not defined by the model may be invoked and contribute to the probability distribution. (ii) Of all features that are invoked, either correctly or falsely, some may be randomly tied to have the same weight value.

Note that perfect hashing techniques might be an alternative to encode key-value pairs (f_i, θ_i) without using a dictionary, it has been successfully used for representing n-gram LM [11] [13]. However, the procedure to compute perfect hash function is not straightforward. As we will show in this paper, for MELM, such exact key-value correspondence is not necessary.

It is also worth pointing out that our approach is quite different from L_1 regularization or other kinds of feature selection techniques. We do not attempt to discard any features, our goal is to provide a randomized representation for arbitrary feature sets such that explicit storage and explicit mapping are not necessary. E.g., it is possible in our method that two very distinct and relevant features may accidentally be forced to share a feature weight. No attempt is made to avoid such accidents, and their consequences are measured only empirically.

II. Feature Hashing

Feature hashing [1] [2] is a method to scale up linear learning algorithms.

In linear models such as logistic regression and support vector machines, a real-valued weight has to be learned for each feature defined by the model. The features are constructed from training instances and often take the form of strings that describe certain properties of the instances. In the case of MELM, features are defined on history-word pairs (h, w). For example, in the training instance (*hello*, *world*), we can have the unigram feature *world* and the bigram feature *hello_world*.

In order to store and retrieve the weights associated with these features, a dictionary structure is usually necessary to map every string-valued feature to an integer position in a weight vector. Such dictionary structures must, however, also store the strings themselves for collision resolution—the task of dealing with multiple strings mapping to the same position. This storage requires large amounts of memory. As the feature space becomes more complex (e.g. high-order n-grams and skip n-grams), the number of possible features grows, as does the number of observed features in large training sets, thus the dictionary structure may take up considerable space.

The hashing idea does away with the storage of the input strings, and replaces the dictionary with a hash function that stochastically minimizes the likelihood of a collision between feature strings without ruling it out completely. Since the hash function is not perfect, all features that have the same hashcode will map to the same location in the weight vector, and be forced to share the same weight parameter.

The size of the weight vector is a parameter that can be adjusted to control the collision rate. As described in [1], the elimination of dictionary storage brings tremendous memory savings, permitting space allocation for a large weight vector. This in turn permits incorporating more features and/or reducing the likelihood of collisions to acceptable levels.

III. Storing Features with Bloom Filter

With the hashing technique, it is not necessary to store the features. In MELM, as we described, features are defined on the sequence of words formed by the history-word pair. Consider the sequences of length n, the number of such distinct sequences is $|V|^n$, where $|V|$ is the size of the vocabulary. For a basic n-gram MELM, if we assume every word sequence of length up to n activates a different feature, we have to contend with $|V|^n + |V|^{n-1} + ... + |V|$ possible features, even though the majority of them are never seen in the training data. Unfortunately, this set of candidate features is often too large to avoid hash collisions. For realistic weight vector sizes, such a large number of possible features significantly exacerbates the hash collision problem.

To further illustrate this point, we conducted a set of experiments on the Penn Treebank corpus. Details of the corpus are provided later when we present more results. Figure 1 shows the perplexity from using different weight vector sizes under feature hashing without storing the feature definitions, where the randomized model is denoted RMELM. The standard model contains 4.1M features derived from the corpus, requiring about 32MB of memory for storing the feature weights. The model contains n-gram features up to 5-grams and also a few other types of features. Consider only 5-gram features, if we allow a distinct feature to fire for every possible 5-word sequence, the number of features implicitly added into the model is as large as 10^{20}. If we allocate only 4.1M bins to the weight vector, on average more than 10^{13} features will be forced to share the same weight. Some of the collisions will be between the 4.1M features derived from the corpus, but most will involve features not seen in the training corpus (but which may be triggered on the test set).

As we can see in Figure 1, the loss in LM performance due to hash collisions is obvious: even with a weight vector size of 10 times the number of seen features, we achieve a perplexity of 135 for the RMELM compared to 131 for the MELM.

However, the RMELM begins to approach the perplexity of the MELM as we allocate more memory, especially after interpolation with the Kneser-Ney 5-gram LM, as shown by the (blue) RMELM+KN5 and (green) MELM+KN5 curves.

Unfortunately, it is often the case that we cannot allocate significantly more bins for the weight vector than the number

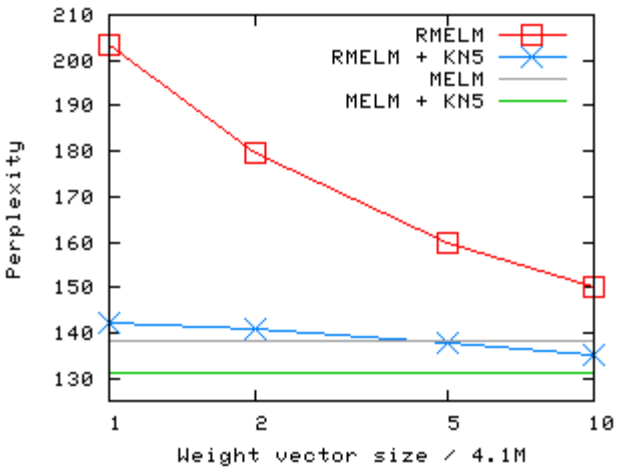

Fig. 1. Perplexity as a function of weight vector size

of seen features. For very large feature sets, such direct hashing may therefore not be advisable. For example, the size of the feature set exceeds 150M in one of our experiments on a 70M-word corpus, and storing a weight vector of size 150M in double precision would require 1.2GB of memory. If 10 times its size is needed to approach the performance of the exact MELM implementation, we would have to allocate a weight vector that takes up 12GB, which may be infeasible.

An alternative is to only include features that have been seen in the training corpus. Since the number of features observed during training is manageable, this can alleviate the collision problem. But to test the existence of a feature $f_i(\cdot)$, we need to store a table of such valid features! It would seem that storing this table would negate the cost saving from feature hashing. But Bloom filters can help alleviate this difficulty. Note that unlike [10], where the Bloom filter is used to associate key-value pairs, we only need a *set* structure that tests whether a given feature $f_i(\cdot)$ is defined in the model. Despite the risk of some false positives, the space advantage of Bloom filters over other data structures is very well known. It works by maintaining a bit vector where each added item sets k bits to 1 according to the results of k hash functions; i.e. it is not necessary to explicitly store the feature definitions. For a detailed description of Bloom filters used for LM, see [10].

IV. Binarization and Subsampling

MELM are known to be computationally expensive to train. The complexity of each iteration of training is at $O(VT)$, where V is the size of the vocabulary and T is the size of the training corpus. For large datasets, standard MELM training is often impractical. In [14], an efficient subsampling approach is proposed to accelerate MELM training. The idea is replace the original V-class problem with V binary problems, where the subsampling can be done for negative examples only. Specifically, instead of building a multi-class classifier as in (2), for each word w in the vocabulary, we build the binary classifier given by

$$P_b(w|h) = \frac{\exp \sum_i \theta_i f_i(h, w)}{1 + \exp \sum_i \theta_i f_i(h, w)}. \quad (3)$$

To obtain a valid multi-class distribution, the binary probabilities are explicitly normalized. Since the majority of the complexity for each binary classifier comes from processing negative examples, we can achieve substantial speedup by only subsampling the negatives. The authors show with extensive experiments that such subsampling strategy is more robust than subsampling for the original multi-class problem.

Given the sizes of our experiments, it is not possible to carry out the standard training efficiently, so the binarization technique is used to train different MELM throughout our experiments.

V. Experimental Results

- For the hash function used in feature hashing, we use Java's $HashCode()$ function modulo the intended size of the weight vector.
- For the Bloom filter, we use the public domain implementation from http://code.google.com/p/java-Bloomfilter/.
- Online stochastic gradient descent is used for training each of the binary classifiers, no explicit regularization is performed [15].

A. Penn Treebank Experiments

This set of experiments is intended for a detailed comparison between the randomized feature representation and the exact feature representation. The Penn Treebank corpus contains approximately 1M word tokens–section 00-20 are used for training(972K tokens), section 21-22 are the validation set(77K), section 23-24(86K) are the test set. The vocabulary size of the experiment is $10K$. To expedite training, we binarize the problem as described above, and subsample only 10% of the negative examples.

Besides n-gram features up to 5-grams, we also include skip-1 bigram, trigram and four gram features [16] . Class-based features are also added which ask about the class membership of the previous word. SRILM is used to obtain the word classes. In total, we have a set of 4.1M features.

The exact storage takes about 600MB, where more than 550MB is used to maintain a *hashmap* from features to the indices in the weight vector. In the randomized implementation, the Bloom filters only require 40-60MB of memory.

Figure 2 shows the perplexity results of randomized MELM(RMELM) compared with the exact *hashmap* implementation. For comparison, we also build a KN smoothed 5-gram LM, which gives a perplexity of 147.9. The results after interpolation with it are shown in Figure 3. The false positive rate here is the theoretical number which can be used as a parameter to initialize the Bloom filter. As we can see, besides the strong space advantage, the RMELM are also able to retain most of the gain by MELM especially after interpolation. Due to the space saving brought by removing the dictionary, we are usually able to allocate a larger weight vector to reduce the

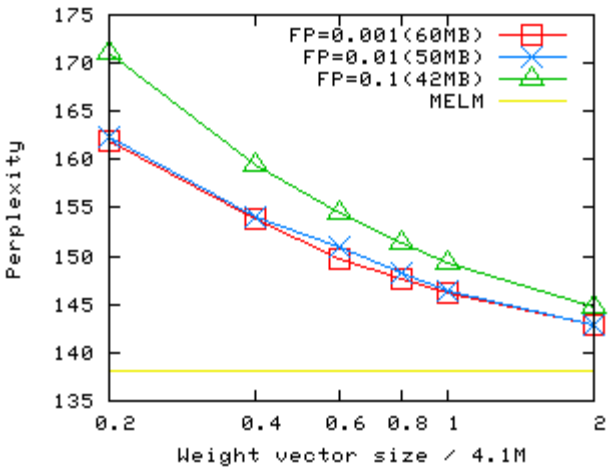

Fig. 2. RMELM with different false positive(FP) rates

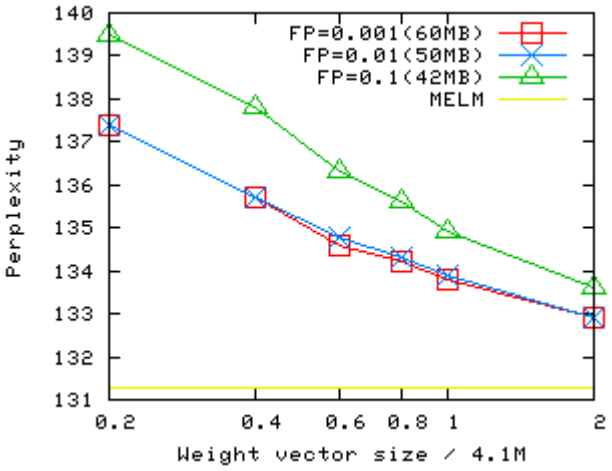

Fig. 3. RMELM with different false positive rates (Interpolated)

hash collisions and improve the results. What's also interesting is that instead of increasing the vector size, the hashing trick can in fact allow us to compress it, the perplexity differences after interpolation are not significant as we shrink the weight vector space.

This set of experiments provides us with some insight in selecting parameters. For the rest of the experiments, we will always have Bloom filters with theoretical false positive rate at 0.01, and the weight vector sizes are always set to be equal to the number of features in the model.

B. Fisher Topic Modeling

In [4], topic information is incorporated into the MELM in the form of topic dependent unigram constraints. Presumably due to the memory requirement and computational burden, topic dependent features are restricted to the unigram frequency of a selected set of words. With our proposed technique, we can easily define a lot more topic dependent features, and learn them jointly with topic independent features in a single hashed vector space.

We have an English Fisher corpus of approximately 10M words. The training set contains 7.2M words. The dev and test sets contain 1.2M words each. Each utterance is assigned to one of the 40 topics. The details of the corpus and topic assignment can be found in [17]. As vocabulary, we take the top 20K most frequent words in the training corpus, thus we need to train 20K binary classifiers, keeping only 2% of the negative examples.

When processing each (h, w) pair, besides the regular features, we also add for each feature its topic dependent version. For example, for the bigram pair (*hello*, *world*), besides the unigram feature *world* and the bigram feature *hello_world*, we also construct features *topicID-word*, *topicID-hello_world*. The number of features extracted this way is about 22M, the *hashmap* storage takes about 4.4GB, while the Bloom filter only requires 150MB. As we described, all features are hashed to a 22M dimensional weight vector, despite collisions, we are still able to benefit from the topic features added into the model.

TABLE I
PERPLEXITY OF TOPIC RMELM

Model	Dev	Eval
KN4	67.5	69.3
Topic KN4	90.4	93.3
KN4 + Topic KN4	63.8	65.6
MELM	61.4	63.4
RMELM	62.1	63.9

Table I shows the perplexity results of our RMELM with topic dependent n-gram features. As a comparison, we build for each topic a KN smoothed 4-gram LM(Topic KN4), this LM is then interpolated with the KN 4-gram LM trained on all topics(KN4). The resulting model serves as a specialized LM used only for utterances of the same topic. As we can see, our randomized scheme performs almost the same as the exact implementation, compared with the interpolated KN LM, the perplexities are slightly lower. With only n-gram features, such topic MELM is not expected to greatly outperform the interpolated n-gram LM, however, it provides a promising and efficient framework to carry out more complex topic dependent modeling.

C. ASR Experiments: Wall Street Journal

We have a set of 100-best list from the DARPA WSJ93 and WSJ92 20K open vocabulary task. The acoustic model used to generate the n-best list can be found in [18]. We use 93et and 93dt sets for evaluation, and 92et for optimization. The LM training text contains 70M words from the NYT section of English Gigaword.

Our baseline LM is a KN smoothed 5-gram LM. For the RMELM, we use the same feature set as described for the Penn Treebank experiments, namely n-grams, skip-1 n-grams within the 5-word window, and the class of the previous word. In total, we have more than 150M features, which can not fit into memory easily. *Hashmap* implementation would require approximately 30GB, with the Bloom filter, only 460MB is

needed. The weight vector takes 1.2GB. Due to the size of the corpus, aggressive subsampling is performed, only 0.5% of the negative examples are retained for each one of the 20K binary classifiers.

TABLE II
WSJ RESCORING WER

Model	Dev	Eval
KN5	12.0	17.7
RMELM + KN5	11.5	17.1

Table II shows the word error rate(WER) results of rescoring the 100-best list. Compared with the standard KN 5-gram, we are able to benefit from more fine-grained features within the 5-word window, despite the fully randomized representation. The 0.6% improvement on the test set is statistically significant.

D. ASR Experiments: English Broadcast News

We also performed ASR experiments on the English Broadcast News(BN) task. The acoustic model is trained on 430h of audio and provided by IBM as part of the 2007 IBM GALE speech transcription system [19]. For LM, we take the 50M words of EARS BN03 Closed Captions corpus as our training data. Note that it is one of the 6 sources used for LM in IBM's system, it is also the corpus that gets the largest interpolation weight (close to 0.5). We tune various parameters on *dev04f* and evaluate WER on *rt04*.

We build a KN 4-gram LM on the 50M words as our baseline. Its pruned version is used in first-pass decoding to generate lattices. For RMELM, we again use the same kinds of features as the Penn Treebank experiments except that instead of 5-grams, all features are extracted from the 4-gram window. The total number of features here is 106M. Again, heavy subsampling is performed here, only 0.5% of the negative examples are kept.

TABLE III
BN RESCORING WER

Model	dev04f	rt04
KN4	17.1	15.9
RMELM + KN4	16.1	15.0

The WER after rescoring the lattices are shown in Table III. When interpolated with the KN 4-gram, the RMELM is able to achieve 0.9% absolute WER improvement over our baseline. Although randomized in nature, the useful information seems to be retained quite well and still able to complement the standard n-gram LM nicely. Moreover, despite having more than 100M features, the memory requirement is quite moderate, only 278MB is used for storing the features, compared to more than 20GB using a *hashmap*. The weight vector takes about 800MB. We certainly have a lot of room to explore richer feature representations and expect further improvement.

VI. CONCLUSION

We describe a randomized solution to maximum entropy language modeling. Feature hashing is used to avoid the exact mapping problem which usually requires large data structures. We also replace explicit storage of the feature set with a Bloom filter. Thus, we enable the building of maximum entropy language models with much larger feature sets than was previously possible, and provide empirical evidence that doing so leads to significant improvements in predictive accuracy.

ACKNOWLEDGMENT

This research was partially supported by the National Science Foundation (Grant No 0963898) and by the JHU Human Language Technology Center of Excellence. We thank Damianos Karakos for sharing the Fisher dataset.

REFERENCES

[1] K. Ganchev and M. Dredze, "Small statistical models by random feature mixing," in *Proc. Workshop on Mobile NLP at ACL*, 2008.
[2] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. International Conference on Artificial Intelligence*, 2009.
[3] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modeling," *Computer Speech and Language*, vol. 10, pp. 187–228, 1996.
[4] S. Khudanpur and J. Wu, "Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling," *Computer Speech and Language*, vol. 14, pp. 355–372, 2000.
[5] A. Stolcke, "Entropy-based pruning of backoff language models," in *Proc. DARPA Broadcast New Transcription and Understanding Workshop*, 1998.
[6] J. Goodman and J. Gao, "Language model size reduction by pruning and clustering," in *Proc. ICSLP-00*, 2000.
[7] K. Church, T. Hart, and J. Gao, "Compressing trigram language models with golomb coding," in *Proc. EMNLP-07*, 2007.
[8] A. Emami, K. Papineni, and J. Sorensen, "Large scale distributed language modeling," in *Proc. ICASSP-07*, 2007.
[9] T. Brants, A. Popat, P. Xu, F. Och, and J. Dean, "Large language models in machine translation," in *Proc. EMNLP-07*, 2007.
[10] D. Talbot and M. Osborne, "Smoothed bloom filter language models: Tera-scale lms on the cheap," in *Proc. EMNLP-07*, 2007.
[11] D. Talbot and T. Brants, "Randomized language models via perfect hash functions," in *Proc. ACL-08*, 2008.
[12] B. Bloom, "Space/time trade-off in hash coding with allowable errors," *Commun. ACM*, vol. 13, pp. 422–426, 1970.
[13] D. Guthrie and M. Hepple, "Storing the web in memory: Space efficient language models with constant time retrieval," in *Proc. EMNLP-10*, 2010.
[14] P. Xu, A. Gunawardana, and S. Khudanpur, "Efficient subsampling for training complex language models," in *Proc. EMNLP-11*, 2011.
[15] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proc. ICML-04*, 2004.
[16] J. Goodman, "A bit of progress in language modeling," *Computer Speech and Language*, 2001.
[17] T. Hazen, F. Richardson, and A. Margolis, "Topic identification from audio recordings using word and phone recognition lattices," in *Proc. ASRU-07*, 2007.
[18] W. Wang and M. Harper, "The superarv language model: Investigating the effectiveness of tightly integrating multiple knowledge sources," in *Proc. EMNLP-02*, 2002.
[19] S. F. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at ibm under the darpa ears program," pp. 1596–1608, 2006.