Large-Scale Random Forest Language Models for Speech Recognition

Yi Su, Frederick Jelinek, Sanjeev Khudanpur

Center for Language and Speech Processing,
Department of Electrical and Computer Engineering,
The Johns Hopkins University, Baltimore, Maryland, USA
{suy, jelinek, khudanpur}@jhu.edu

Abstract

The random forest language model (RFLM) has shown encouraging results in several automatic speech recognition (ASR) tasks but has been hindered by practical limitations, notably the space-complexity of RFLM estimation from large amounts of data. This paper addresses large-scale training and testing of the RFLM via an efficient disk-swapping strategy that exploits the recursive structure of a binary decision tree and the local access property of the tree-growing algorithm, redeeming the full potential of the RFLM, and opening avenues of further research, including useful comparisons with n-gram models. Benefits of this strategy are demonstrated by perplexity reduction and lattice rescoring experiments using a state-of-the-art ASR system.

Index Terms: random forest language model, large-scale training, data scaling, speech recognition

1. Introduction

A language model (LM) is a crucial part of many natural language processing tasks, such as speech recognition, statistical machine translation and information retrieval, among others. With the help of several good smoothing techniques, which were systematically studied in [1], the n-gram language model gained great popularity because of its simplicity and surprisingly good performance.

Several new models have been proposed over the years with varying degrees of success. Recently the random forest language model [2], which is a natural extension of the decision tree language model [3], has been proposed and shown encouraging results in a state-of-the-art conversational telephone speech recognition system [4]. However, due to the space complexity of the model training algorithm, it has to settle for a sampling trick to accommodate a large training corpus. We propose an efficient divide-and-conquer disk swapping algorithm to alleviate this problem so that the strength of RFLM can be fully realized. Experimental results in terms of both perplexity and word error rate are presented to illustrate the benefits of this algorithm.

The rest of the paper is organized as follows: in Section 2, we briefly review the basic ideas behind the n-gram, decision tree and random forest language models, and their relatedness. Some details on the training and testing procedures of RFLM are provided in Section 3. In Section 4, we present our algorithms for training and testing RFLMs on a large scale. Experimental results appear in Section 5, and concluding remarks in Section 6.

2. Language models

A language model is a probability distribution over the set of all possible strings from a vocabulary. Let $W = w_1 \dots w_N \in V^*$ denote a string of N words, where V is the vocabulary. Using the chain rule, we have

$$P(W) = \prod_{i=1}^{N} P(w_i | w_1, \dots, w_{i-1}), \tag{1}$$

where $w_1, \dots, w_{i-1}$ is referred to as a *history*. To facilitate reliable probability estimation from data, the set of histories is divided in practice into reasonable equivalence classes. i.e.

$$P(W) \approx \prod_{i=1}^{N} P(w_i | \Phi(w_1, \dots, w_{i-1})), \tag{2}$$

where $\Phi : V^* \mapsto C$ denotes the equivalence classification.

N-gram language models: Taking the view of language modeling as equivalence classification of histories, the n-gram language model simply defines a Markovian equivalence mapping Φ_n by only looking at the last $n-1$ words of a history. i.e.,

$$P(w|h) \triangleq P(w|\Phi_n(h)) = P(w|w_{i-n+1}^{i-1}), \tag{3}$$

where h is the history and w_{i-n+1}^{i-1} represents the class of histories that share the same $(n-1)$-suffix $w_{i-n+1}, \dots, w_{i-1}$.

Decision tree language models: Starting with all histories in one equivalence class, the decision tree language model (DTLM) [3] successively refines the equivalence classification by asking questions about words in different history-positions, until some stopping criterion is reached.

$$P(w|h) \triangleq P(w|\Phi(h)) = P(w|\Phi_{DT}(h)). \tag{4}$$

Studies have shown that given modest amounts of training data, the DTLM fails to improve upon the n-gram LM [5].

Random forest language models: The random forest language model [2] is a collection of randomized DTLMs, whose trees may be viewed as i.i.d. samples from a subset of possible trees. The LM probability is defined as the average of the probabilities from all the DTLMs:

$$P(w|h) \triangleq \frac{1}{M} \sum_{j=1}^{M} P(w|\Phi_{DT_j}(h)). \tag{5}$$

Note that the n-gram LM may be seen as a naïve DTLM that refines the equivalence classification by asking only about the identities of consecutive words in the history, and the DTLM may be seen as a RFLM with only one tree.

3. Random forest language modeling

Our decision tree training procedure, following [6], consists a *growing* and a *pruning* stage. In the growing stage, we start from a single node which contains all n-gram histories in the training text and recursively split every node by an exchange algorithm [7] until further splitting does not increase the likelihood of the training text. In the pruning stage, we compute the *potential* of each node—the possible gain in held-out data likelihood from growing the node into a sub-tree. Nodes whose potential falls below a threshold are pruned, bottom-up. The questions we are currently using simply ask whether the i-th previous word w_{-i} belongs to a set of words S and $1 \leq i < n$. We will refer to n as the *order* of the RFLM.

Randomization may be used in three places in the decision tree training procedure: question set selection, exchange algorithm initialization and data sampling. Due to the greediness of the exchange algorithm, restricting the candidate questions at each node-splitting step to a *random subset* of all available questions helps find a better tree. The same argument holds for random initialization. However, random sampling of training data has not been beneficial to RFLM [8], and is used only to circumvent the space-complexity of DTLM training from large data. In this paper, we will randomize in only the fist two ways mentioned above, and address the space-complexity directly.

We use interpolated Kneser-Ney smoothing, as defined in [1], to compute the LM probabilities

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\max(C(w_i, \Phi(w_{i-n+1}^{i-1})) - D, 0)}{C(\Phi(w_{i-n+1}^{i-1}))} + \lambda(\Phi(w_{i-n+1}^{i-1}))P_{KN}(w_i|w_{i-n+2}^{i-1}), \quad (6)$$

where D is a *constant discount* and $P_{KN}(\cdot)$ is the Kneser-Ney backoff probability distribution. In Section 5.1, we will show that the version of this smoothing method, whereby three discounts D_1, D_2 and D_{3+} are used for n-grams with one, two and three or more counts respectively, further reduces perplexity as suggested in [1].

4. Large-scale training and testing

Despite several recent advances in language modeling, most state-of-the-art speech recognizers still use n-gram LMs as their first choice. The primary reason is that the amount of text available for training LMs has also been growing steadily, so that while more sophisticated LMs may be competitive with n-gram LMs on the same sized training corpus, they often require larger computational resources for training, and therefore have a hard time scaling up, giving weak learners such as n-gram LMs a chance to surpass them simply by using larger training corpora. A case in point is that the maximum entropy LM [9] gained much more deserving popularity only after efficient training algorithms like [10] were proposed. Studies like [11] also present similar analyses. Therefore, an LM will have limited utility unless it scales up nicely.

The RFLM estimation procedure of [2, 4, 8] has severe problems scaling up to training corpora beyond a few 10's of millions of words, while standard toolkits can estimate n-gram language models from 100's of millions, even billions of words. One key design decision in [8] is that all n-grams seen in training are retained in memory for estimating each (randomized) DTLM. This, by itself need not be a limitation, since machines with several GB of core memory are becoming commonplace. The problem is that as the DTLM grows, typically to millions of internal nodes, additional memory is continually needed, particularly for storing "questions" — about membership in arbitrary subsets of V — asked at each internal node. Thus the size of the eventual DTLM grows with training data and, in practice, this procedure runs out of available memory before DTLM training is terminated by its natural stopping rules. We have addressed this problem, devising a solution that permits training the RFLM on much larger corpora than previously possible.

Key insights: when the DTLM training procedure runs out of memory, the partially grown tree may be written out to disk, marking each node that has not yet been declared a *leaf*, along with the subset of the n-grams from the training data that "fall" into each *non-terminal* nodes. The DTLM training procedure may then be performed recursively for each of the unfinished nodes, again possibly writing unfinished sub-trees to disk, provided a *global* priority queue is maintained to indicate which of the non-terminal nodes should be considered next for splitting.

4.1. Algorithms

The DTLM growing and pruning stages of the original algorithm in [8] need to be modified as follows, and one more step needs to be added to achieve the goal described above:

Growing: When the number of nodes exceeds a threshold, stop growing the tree and *swap out*; Add any unfinished nodes to an agenda (priority queue); *Swap in* the top unfinished nodes according to the agenda, and continue growing.

Pruning: Move the computation of held-out data likelihood and size to the growing stage to save one pass of disk swapping; Rewrite node-potentials as functions of held-out data likelihood and size; Compute the potential for each node recursively in a bottom-up fashion and prune away nodes whose potentials are below an empirically chosen threshold.

Clean-Up: Go through the whole tree one more time to remove any internal nodes that were not pruned during the "embedded" pruning stage described above (due to unavailability of descendant node-potentials).

The first part of the algorithm ("Growing") is illustrated in Algorithm 1 and Algorithm 2. Algorithm SplitNode is unchanged from the original [8] (Page 44, Algorithm Node-Split(p)) and therefore omitted. For the second part ("Pruning"), let $l(n)$ and $s(n)$ be the log-likelihood and size of the heldout data *at* the node n, i.e., the log-likelihood and size of the heldout data if the subtree rooted at this node was pruned away. Let $L(n)$

Algorithm 1: Grow(D)

```
Input: Data D
Output: Fully grown tree T
begin
    Initialize agenda with the root node N;
    while agenda not finished do
        Take node n out of agenda;
        Swap in data d for node n from disk;
        (t, {n_i}, {d_i}) ← GrowSubtree(n, d);
        Write subtree t to disk file T;
        Swap out data {d_i} for nodes {n_i} to disk;
        Put unfinished nodes {n_i} in agenda;
    end
    Return tree T;
end
```

Algorithm 2: GrowSubtree(N, D)

Input: Node N and its data D
Output: Tree T, unfinished nodes and their data $(\{n_i\}, \{d_i\})$

```
begin
    Initialize agenda with node N;
    K ← 1;
    while agenda not finished and K ≤ threshold do
        Take node n out of agenda;
        if SplitNode(n, d) succeeds then
            Put children nodes n_L and n_R in agenda;
            K ← K + 2;
        end
    end
    Return tree T, remaining nodes {n_i} in agenda and
    their data {d_i} if there is any;
end
```

and $S(n)$ be the log-likelihood and size of the heldout data *below* the node n, i.e., the would-be log-likelihood and size of the heldout data if the entire subtree rooted at this node was retained. The *potential* of the node, $P(n)$, is defined as

$$P(n) = \frac{L(n)}{S(n)} - \frac{l(n)}{s(n)}, \tag{7}$$

where $L(n)$ and $S(n)$ can be computed recursively as

$$L(n) = \begin{cases} l(n) & \text{if node } n \text{ is a leaf;} \\ L(n_L) + L(n_R) & \text{otherwise.} \end{cases} \tag{8}$$

$$S(n) = \begin{cases} s(n) & \text{if node } n \text{ is a leaf;} \\ S(n_L) + S(n_R) & \text{otherwise.} \end{cases} \tag{9}$$

where n_L and n_R are the left and right children of the node n, respectively.

Note that although the computation of $l(n)$ and $s(n)$ is conceptually part of pruning, it is done in the growing stage of this algorithm because the move can save a whole pass of examining the tree through disk swapping. This highlights one of the many design decisions we made when dealing with the space complexity problem by trading I/O time, while striving to keep the overhead as low as possible.

5. Experiments

5.1. Modified smoothing

In the first experiment we would like to verify our hypothesis that the modified Kneser-Ney smoothing [1] improves the performance of the RFLM as it does that of the n-gram LM. We used Wall Street Journal portion of the Penn Treebank [12]. Following [2], we preprocessed the text by lowercasing words, removing punctuations and replacing numbers with the "N" symbol. Sections 00-20 (929,564 words), sections 21-22 (73,760 words) and sections 23-24 (82,430 words) were used as training, held-out and testing data, respectively. The vocabulary size was 10k, including a special token for unknown words.

We contrasted regular with modified Kneser-Ney smoothing scheme when the forest consisted of y trees, where $y = 1, 2, 3, 4, 5, 7, 10, 20, 50$ and 100. Because of the random nature of the RFLM, each experiment was repeated 10 times. The mean and standard deviation of the test data perplexities were plotted as error bars in Figure 1.

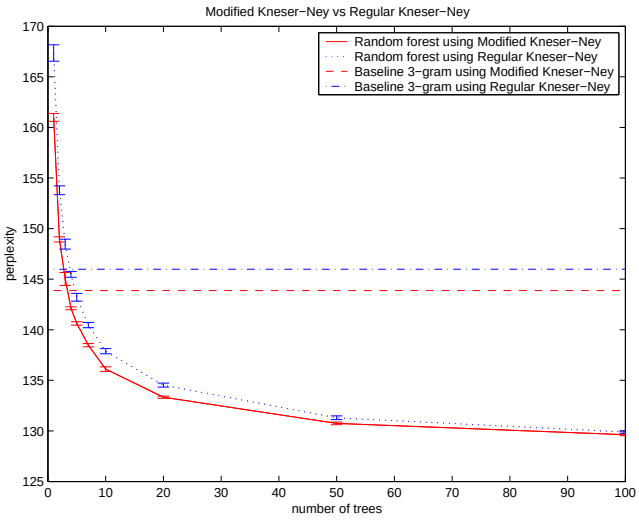

Figure 1: Smoothing RFLM: modified vs regular

We conclude from Figure 1 that the modified Kneser-Ney smoothing did improve upon the regular, as expected. More interestingly, the gap decreased as the number of trees in the forest increased, supporting the claim in [8] that the RFLM beats the n-gram LM by better smoothing. Because the additional computation incurred when switching from regular to modified smoothing was negligible, we used the latter henceforth.

5.2. Learning curves

Our second experiment was to compare the learning curves of the Kneser-Ney smoothed 4-gram language model with RFLM of the same order. For this purpose we chose to work with the 525 million words English WEB data from University of Washington, which was also used in [4]. The first x million words of the WEB data, where $x = 1, 2, 4, 8, 16, 32$ and 64, were used as training data and 2 million words from the Fisher data were used as held-out. (The largest amount of training text we could accommodate, without any counts cut-off, was about 100M words, given that the addressable space for a single process in a 32-bit machine is 4GB.) Another 2 million words from the Fisher corpus were used for testing. The vocabulary contained 30k word types. The number of trees per forest was 100.

From Fig. 2 we can see that the RFLM scaled at least as well as the n-gram LM, while keeping a more than 10% lead in perplexity.

5.3. Lattice rescoring

Building a RFLM for the IBM GALE Mandarin ASR system was exactly the challenge this work was trying to meet. The acoustic model of this recognizer was trained on 585 hours of speech with PLP features by first building a speaker-independent model, then adapting it to speaker clusters using VTLN, fMLLR and fMPE training.

Because the content of this task was mainly broadcast news (BN) and broadcast conversation (BC), the available amount of text for training a language model was huge. Specifically the text we used was organized into seven parts: xin.0 and xin.1 (Xinhua News), cna.1.0 and cna.1.1 (Central News Agency), pdcr (People's Daily and China Radio), tdt (TDT-2,3,4 and HUB-4), gwd (UWashington general web data). Each con-

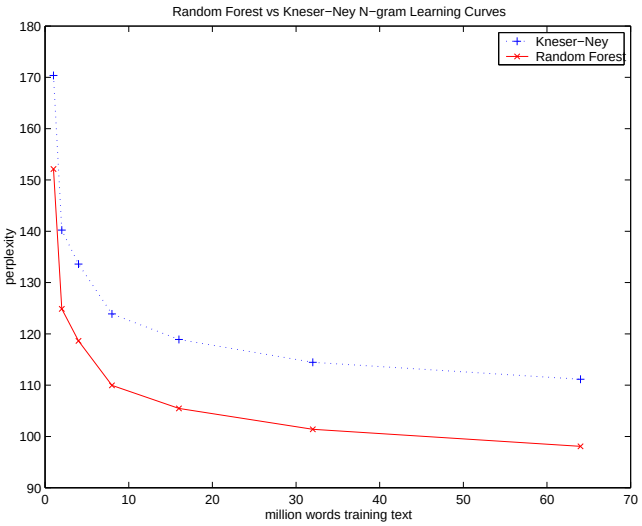

Figure 2: Learning curves in terms of perplexity

tained about 100 million words as segmented by a maximum-matching segmenter with respect to the 107k vocabulary used by the acoustic model.

We built a 4-gram RFLM with 50 trees for each of the seven parts then interpolated them together to rescore the lattices generated by the IBM system. (We used 50 trees per forest instead of 100, as we usually do, because 50 trees worked as good as 100 on the heldout data but halved the computational effort.) Interpolation weights were determined using the current one-best output on a per show basis. Note that these lattices had already been rescored once by a regular 4-gram LM ("Baseline"), which was trained on 602 million words text from 13 different sources. As shown in Table 1, we got a 0.6% absolute

Character Error Rate (%)	All	BN	BC
Baseline	18.9	14.2	24.8
RFLM	18.3	13.4	24.4

Table 1: Lattice rescoring for IBM GALE Mandarin ASR

reduction in CER, which was statistically significant in a sign test with $p < 0.001$.

6. Conclusions

We proposed an efficient disk swapping algorithm to deal with the space complexity problem in building random forest language models. By taking advantage of the recursive structure of binary decision tree and locality of the original tree growing algorithm, we successfully kept the I/O time overhead to be linear in the number of distinct training n-grams. With this approach, we empirically studied the scaling performance of the RFLM and showcased our achievement by statistically significantly improving the performance of the IBM GALE Mandarin ASR system.

7. Acknowledgments

We are grateful to Peng Xu for his RFLM source code, Lidia Mangu and Yong Qin for word lattices from their GALE ASR system, and Richard Sproat for his Chinese word segmentation script. We particularly thank Damianos Karakos for significant assistance in experimentation, and for comments on the manuscript. This research was partially supported by the DARPA GALE program (contract No. HR0011-06-2-0001).

8. References

[1] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech and Language*, vol. 13, pp. 359–394, 1999.

[2] P. Xu and F. Jelinek, "Random forests in language modeling," in *Proceedings of EMNLP 2004*, D. Lin and D. Wu, Eds. Barcelona, Spain: Association for Computational Linguistics, 2004, pp. 325–332.

[3] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer, "A tree-based statistical language model for natural language speech recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 7, pp. 1001–1008, 1989.

[4] P. Xu and L. Mangu, "Using random forest language models in the IBM RT-04 CTS system," in *Proceedings of INTERSPEECH-2005*, 2005, pp. 741–744.

[5] G. Potamianos and F. Jelinek, "A study of n-gram and decision tree letter language modeling methods," *Speech Communication*, vol. 24, no. 3, pp. 171–192, 1998.

[6] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[7] S. Martin, J. Liermann, and H. Ney, "Algorithms for bigram and trigram word clustering," *Speech Communication*, vol. 24, no. 1, pp. 19–37, 1998.

[8] P. Xu, "Random forests and the data sparseness problem in language modeling," Ph.D. dissertation, Johns Hopkins University, 2005.

[9] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modelling," *Computer Speech and Language*, vol. 10, pp. 187–228, 1996.

[10] J. Wu and S. Khudanpur, "Efficient training methods for maximum entropy language modeling," in *Proceedings of ICSLP-2000*, 2000.

[11] M. Banko and E. Brill, "Mitigating the paucity-of-data problem: Exploring the effect of training corpus size on classifier performance for natural language processing," in *Proceedings of the Conference on Human Language Technology*, 2001.

[12] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: the Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.