LARGE-SCALE DISTRIBUTED LANGUAGE MODELING

Ahmad Emami

IBM T J Watson Research Center
1101 Kitchawan Rd.
Yorktown Heights, NY 10598
emami@us.ibm.com

*Kishore Papineni**

Yahoo! Research
45 W. 18th St.
New York, NY 10011
kpapi@yahoo-inc.com

Jeffrey Sorensen

IBM T J Watson Research Center
1101 Kitchawan Rd.
Yorktown Heights, NY 10598
sorenj@us.ibm.com

ABSTRACT

A novel distributed language model that has no constraints on the n-gram order and no practical constraints on vocabulary size is presented. This model is scalable and allows for an arbitrarily large corpus to be queried for statistical estimates. Our distributed model is capable of producing n-gram counts on demand. By using a novel heuristic estimate for the interpolation weights of a linearly interpolated model, it is possible to dynamically compute the language model probabilities. The distributed architecture follows the client-server paradigm and allows for each client to request an arbitrary weighted mixture of the corpus. This allows easy adaptation of the language model to particular test conditions. Experiments using the distributed LM for re-ranking N-best lists of a speech recognition system resulted in considerable improvements in word error rate (WER), while integration with a machine translation decoder resulted in significant improvements in translation quality as measured by the BLEU score.

Index Terms— Statistical language modeling, Speech recognition, Statistical machine translation, Client-server systems, Distributed memory systems

1. INTRODUCTION

Statistical language models have a major role in applications such as Automatic Speech Recognition (ASR) [1], Statistical Machine Translation (SMT) [2], and Information Retrieval. These applications depend critically upon models of natural language to evaluate decoding hypotheses. A statistical language model assigns a probability $P(W)$ to any given word string $W = w_1 w_2 \dots w_m$, typically factored as

$$P(W) = P(w_1 w_2 \dots w_m) = P(w_1) \prod_{k=2}^{m} P(w_k | W_1^{k-1}) \quad (1)$$

where the sequence of words $w_1 w_2 \dots w_j$ is denoted by W_1^j. In n-gram models, which are the state-of-the-art, the probability $P(w_i|W_1^{i-1})$ is approximated by conditioning only on the last $n-1$ words; that is $P(w_i|W_1^{i-1}) \approx P(w_i|W_{i-n}^{i-1})$.

Conventional n-gram models are plagued by high dimensionality, and have a large number of parameters which must be estimated. In order to estimate these parameters, a large amount of training data is required. For these reasons, language models are usually very expensive in terms of physical resources consumed. Some techniques used to mitigate these burdens have included class-based language models, as well as methods such as count cut-offs or entropy pruning that reduce the language model size [3].

However, as corpus size increases, it becomes increasingly difficult, if not impossible, to build a language model off-line. Given the large amount of training text required to estimate n-gram models, and the rate at which new data becomes available, it is becoming increasingly necessary to develop infrastructures that can handle large amounts of training data, specially for n-gram models of high orders ($n > 3$).

In this paper we present a distributed language model in the client-server paradigm that can readily handle very large amounts of training data. The infrastructure is scalable, and more data can be incrementally added without the need to change or re-compute any of the pre-existing modules, and the model can be easily grown as more data becomes available. Furthermore, the infrastructure allows for multiple clients, each with different needs to be served simultaneously. Even decoders with dynamically changing needs can be handled.

In conventional n-gram models, the decisions about the specifications of the model are usually made before it is estimated from the training data. These specifications include vocabulary size, n-gram order, and smoothing method. Since, in our distributed model, the actual corpus in its entirety is stored, there are no restriction on the n-gram order or the mixture of corpora used, and any client can choose any specification upon connection to the server. In this regard, the distributed architecture has usage beyond just language modeling and has applications in any problem where n-gram counts are required.

Our storage architecture makes use of the characteristics of present day server systems, and by distributing the storage of the corpus across multiple machines, we allow for many new potential dynamic combinations of data from multiple independent corpora.

We use our distributed model in re-ranking the N-best lists from a ASR system, getting up to 5.3% relative improvement in Word Error Rate (WER). Furthermore, using our distributed language model with an SMT decoder resulted in up to 3.3 BLEU points (7.1% relative) improvement in translation performance.

2. DISTRIBUTED ARCHITECTURE

Figure 1 illustrates the architecture of the distributed language model. A similar distributed architecture was independently proposed in [4], and used for re-ranking N-best lists output by a machine translation decoder. The data is divided among several machines, which we refer to as the *LM workers*. There is no training data stored on the server machine; its sole job is to facilitate communication between the clients and the workers and to compute the language model probabilities.

*while at IBM TJ Watson Research Center

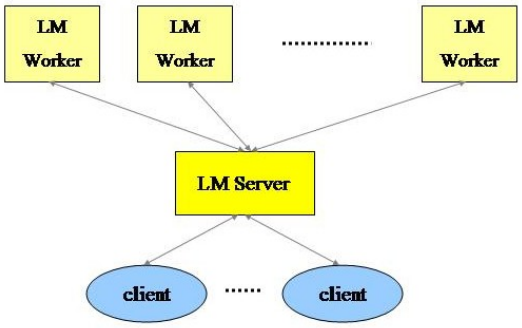

Fig. 1. Clients can have different weight vectors

The communication between the server and the workers is implemented using the Message Passing Interface (MPI) library [5]. Communication between the clients and the server is implemented via TCP/IP socket networking. This allows a more flexible architecture where clients can connect to and disconnect from the server at will and independently. Clients usually send queries in large batches so as to minimize the TCP/IP communication overhead.

Each worker is capable of storing either a whole corpus or a stand-alone, smoothed n-gram language model. Section 2.1 details the use of suffix arrays for storing a corpus and for efficient lookups of n-gram counts. A client can request any mixture of the workers, specifying arbitrary weights for each of the component corpora or language models.

2.1. Suffix Arrays

Calculating the number of occurrences of a sequence of words in a large corpus can be done if the corpus is suitably indexed. A *suffix tree* is a data structure that can be built in time and storage proportional to the length of the corpus. However, as in many applications, we found the similar *suffix array* data structure to be preferable due to the ease of construction [6].

To build the suffix array index, an index of offsets is initialized to point to every offset in the corpus. This index is then *reverse* lexicographically sorted effectively creating a sorted view of the entire corpus.

To find the number of n-grams matching a particular history the algorithm explained in Algorithm 1 is used. The `OffsetEqual(List,Value,Offset)` function returns the sublist of the indicated portion of the corpus that has the word, at the specified offset, equal to the current word of the history. Because the corpus is lexicographically sorted, this can be implemented using a binary search in $O(N \log C)$ time, when N is the size of the n-gram and C is the total length of the corpus. It should also be noted that computing the count $C(w1w2w3)$ also computes $C(w2w3)$ and $C(w3)$ as intermediate results.

The corpus is divided so that each portion, along with its index, will fit within the available physical RAM of each host in the distributed language mode. And, because we do not count n-grams that cross sentence or document boundaries, the total count for the entire corpus can be computed by aggregating the counts from each of the clients.

Algorithm 1 n-gram(h)

Require: h a reverse iterator pointing to the current word {Returns the number of occurances of the n-gram}
  $B \Leftarrow$ begining of reverse sorted corpus
  $E \Leftarrow$ end of reverse sorted corpus + 1
  $i \Leftarrow 0$
  while $E > B$ and h.hasNext() **do**
    (B, E) = OffsetEqual((B, E), h.next(), i)
    $i \Leftarrow i + 1$
  end while
  return $E - B$

2.2. Dynamic Smoothing

n-gram language models require many parameters to be estimated before the model can be used. These parameters include estimates of word string probabilities as well as the parameters used for smoothing. For example, in back-off language models, the weights for backing-off from higher order to lower order n-grams need to be estimated. Similarly, in linearly interpolated models,

$$P_{intp}(w_i|W_{i-n+1}^{i-1}) = \lambda_{W_{i-n+1}^{i-1}} P_{ml}(w_i|W_{i-n+1}^{i-1}) + (1 - \lambda_{W_{i-n+1}^{i-1}}) P_{intp}(w_i|W_{i-n+2}^{i-1}) \quad (2)$$

where the maximum likelihood estimate P_{ml} is interpolated with the lower order smoothed model P_{intp}, which is defined in a similar fashion. A model is typically built iteratively starting with the lowest order probabilities [7]. Estimating a distinct $\lambda_{W_{i-n+1}^{i-1}}$ for each W_{i-n+1}^{i-1} is not practical; in practice the $\lambda_{W_{i-n+1}^{i-1}}$ values are grouped into bins based on the word counts $C(W_{i-n+1}^{i-1})$, where the final weight in each bin is assumed to be equal [1].

Normally, in linearly interpolated models, the interpolation weights $\lambda_{W_{i-n+1}^{i-1}}$ are estimated on some *held-out* data using the EM algorithm [8]. In a similar fashion, back-off models use *ad-hoc* discounting and back-off parameters that are estimated, or assigned manually, before the language model is used.

For simplicity, we have decided to use linearly interpolation as the smoothing method for our distributed LM architecture. It is also possible to use a back-off smoothing framework, at the cost of a bit more involved lookups and computations. However, we carried out some simple experiments with different smoothing techniques and decided that the choice of smoothing method is less relevant as we move to very large corpora.

Noting that the $\lambda_{W_{i-n+1}^{i-1}}$ are a function of only the counts $C(W_{i-n+1}^{i-1})$, we have instead opted for a closed-form heuristic formula for computing $\lambda_{W_{i-n+1}^{i-1}}$. After plotting the weights $\lambda_{W_{i-n+1}^{i-1}}$ estimated using EM on a held-out set against counts $C(W_{i-n+1}^{i-1})$ we noticed that the plot can be approximately fitted to a *log-linear* curve, from which we estimated the interpolation weights for the distributed language model:

$$\lambda_{W_{i-n+1}^{i-1}} = 0.1 \ log_{10}(C(W_{i-n+1}^{i-1})) + 0.3 \quad (3)$$

from which all the $n+1$ interpolation weights for the model in Equation 2 are computed in an iterative manner starting from the highest order n-gram.

The clear advantage of this heuristic formula is that $\lambda_{W_{i-n+1}^{i-1}}$ is directly computable from word counts in the training data. It should

Training	λ estimation	Perplexity
I	EM	193
I	heuristic	216
II	EM	128
II	heuristic	137

Table 1. Perplexities for different interpolation weight estimation methods

also be noted that there is no lookup overhead here, since the count $C(W_{i-n+1}^{i-1})$ needs to be extracted anyway to estimate the probability $P_{ml}(w_i|W_{i-n+1}^{i-1})$.

Table 1 shows perplexities of linear interpolated 5-gram models using both the bucketed EM estimate and heuristic interpolation weights. There are 2 different training data sets used, denoted by I and II in the table and consisting of approximately 200M and 2.9G words respectively. The held-out data used for EM estimation of the weights is a corpus of 35M words. The vocabulary size is 128K and the perplexities are reported on an unseen test set of 58M words.

As can be seen in the table the linearly interpolated model with heuristic interpolation weights has comparable perplexity to those estimated using the EM algorithm. It should also be noted that EM maximizes the performance of the model for likelihood, and hence, the perplexity. We expect the difference between the heuristic and EM estimated weights to be smaller when used for a practical task such as ASR or SMT where the performance is measured by metrics which are not fully correlated with perplexity.

2.3. Model Details

Assume there are m_1 workers with suffix array corpora and m_2 stand-alone smoothed n-gram models for a total of $m_1 + m_2$ workers. Assuming that the j^{th} client is using the worker mixture vector $\left(s_1^j, s_2^j, \cdots, s_k^j\right)$, then the aggregate count is simply $C^j(W_1^n) = \sum_{i=1}^{m_1} s_i^j C_i(W_1^n)$ where $C_i(W_1^n)$ is the count of n-gram W_1^n for the corpus stored at worker i.

In case the client requested probabilities, the aggregate probability $P_a^j(w_n|W_1^{n-1})$ is computed according to equations 2 and 3 above, using the aggregate statistics C^j. Taking into account the stand-alone smoothed LM workers, the final probability is defined as follows:

$$P^j(w_n|W_1^{n-1}) = \alpha^j P_a^j(w_n|W_1^{n-1}) + (1-\alpha^j)\sum_{i=1}^{m_2} \frac{s_i^j}{\sum_{l=1}^{m_2} s_l^j} P_w^i(w_n|W_1^{n-1}) \quad (4)$$

where $P_w^i(w_n|W_1^{n-1})$ is the probability from the i^{th} stand-alone worker, and $\alpha^j = \frac{\sum_{i=1}^{m_1} s_i^j}{\sum_{i=1}^{m_1+m_2} s_i^j}$.

3. EXPERIMENTS AND RESULTS

We carried out experiments using our distributed language model for both speech recognition and machine translation tasks. In the machine translation experiments, the distributed language model was directly integrated into the decoder. In the speech recognition experiments, however, the distributed model was used only in re-ranking N-best lists created by a speech recognition decoder using standard language models.

	RT-03	Dev-04	RT-04
baseline	9.5	17.7	15.0
5-gram DLM	9.0	17.4	14.5

Table 2. ASR N-best re-ranking

Model	MT-03	MT-05
Phrase-based Decoder 3gm	48.84	46.92
Phrase-based Decoder 5gm DLM	51.10	50.25
Max-Ent Decoder 3gm	48.95	48.02
Max-Ent Decoder 5gm DLM	51.35	51.30

Table 3. Arabic BLEU Results

3.1. Speech Recognition

We carried experiments re-ranking the N-best lists output by a speech recognizer. The system description is similar to that given in [9]. The lattices are generated using a two-pass speech recognition system that performs a speaker-independent recognition pass; then a series of speaker adaptation steps, including vocal tract length normalization, feature-space MLLR, and MLLR; and concludes with a speaker-adapted recognition pass. The speaker-independent and speaker-adapted acoustic models are trained on a 450-hour corpus comprising the 1996 and 1997 English Broadcast News Speech collections and the English broadcast audio from TDT-4. The language model used by the decoder was a 4-gram modified Kneser-Ney model trained on 192M words and interpolated with five other 4-gram models trained on topic specific subsets of the original 192M words data. The final merged model was pruned to a total of 3.2M n-grams. The final speaker-adapted recognition pass performed in-memory lattice re-scoring using a larger, 30M n-gram language model that differed from the one used to produce the decoding graph only in the degree of pruning.

Table 2 shows the results of using the distributed language model, in this case a 5-gram trained on 4 billion words, for re-ranking the 500-best lists from two EARS English Broadcast News evaluation sets (RT-03 and RT-04) and one development set (Dev-04f). As can be seen using the larger distributed LM results in considerable improvements in Word Error Rate across all the three sets. It should be noted that out of the 500 hypotheses per utterance in average only 9.5 were unique word strings and the distributed model achieved the shown reduction by effectively only re-raking these remaining few hypotheses.

3.2. Machine Translation

The phrase decoder we use is inspired by [10]. It is a multi-stack, multi-beam search decoder with as many stacks as the length of the source sentence and a beam associated with each stack. It is described in more details in [11]. The training data for the Arabic translation system consisted of 4.6M sentence pairs, totaling 124M Arabic, and 143M English words. The MT-03 and MT-05 test sets consist of 663 and 1056 segments respectively. We also used a different decoder based on the Maximum Entropy (Max-Ent) approach. The Max-Ent decoder is a novel method of using simple translation blocks, which allows the translation problem to be cast into a classification problem where instead of encoding the context in long phrasal blocks, it is cast as feature function on simple blocks [12].

Model	MT-03	MT-04	MT-05
3-gram	26.93	28.72	26.10
5-gram DLM	29.10	30.09	29.18

Table 4. Chinese BLEU Results

For the Chinese translation system CE Training data there were about 8.4M sentence pairs, totaling 212.5M Chinese and 241.2M English words. We only used the phrase-based decoder for the Chinese to English translation experiments. It should be noted that the training data was sampled for each test set; the actual training data being used is in average 400K sentences in size. The test sets MT-03, MT-04, and MT-05 are 919, 1788, and 1082 sentences long respectively.

The results for the Arabic-English and Chinese-English experiments are given in Tables 3 and 4 respectively. The 3-gram LM is a linearly interpolated model trained with a vocabulary size of 128K on about 2.8 billion words with bigram and trigram with count less than 3 pruned. The 5-gram distributed model was trained on 2.3 billion words with a vocabulary size of just over a million. As can be seen, using the distributed model that allows for larger corpus and higher order n-gram with no pruning (not to mention larger vocabulary size) results in significant improvements across all systems and test-set. For example, in the case of the Arabic-English system, the phrase-based decoder achieves a 3.3 BLEU point improvement in using a 5-gram language model over a 3-gram for both MT-03 and MT-05 test sets.

When compared with the ASR experiments, it is clear that we achieve better improvements with the distributed LM for the SMT task. One explanation is that for the SMT experiments, the distributed language model was directly integrated into the decoder, whereas in the ASR experiments the distributed model was used only to re-rank very shallow ($N \approx 9.5$) N-best lists. Another explanation is that the tokenization and vocabulary selection for the distributed language model were optimized with only the SMT tasks in mind.

4. DISCUSSION AND FUTURE WORK

In this paper we presented a novel distributed architecture for storing large-scale and scalable corpora and language models. The distributed architecture is capable of handling arbitrarily large corpora, while placing no restrictions on either the n-gram order or vocabulary size. Computing n-gram counts of arbitrary order was made possible by storing the original corpus in the suffix array format. By developing a novel closed-form heuristic formula for interpolation weights, we were able to compute the conditional probability for any given n-gram dynamically.

We applied the distributed language model in re-ranking the N-best list of a state-of-the-art ASR system achieving up to 5.3% relative improvement in WER. We also used the distributed LM directly integrated in an SMT decoder, gaining across different systems and test-sets, considerable and consistent improvements in translation quality as measured by the BLEU score.

Our distributed architecture allows for each client to choose the subset of the corpus, with arbitrary weights for each component corpus. This allows for a flexible language modeling architecture where the mixture of corpora is selected and changed dynamically as the test-set changes.

It should be noted that since the distributed model produces raw n-gram counts, it can be utilized for more than just producing conditional n-gram probabilities $P(w_1|W_1^{n-1})$ (e.g. computing co-occurrence and mutual information), and can find applications beyond language modeling.

The next step in extending this work is to integrate the distributed model in the ASR decoder, as was done for the SMT system. In the future, we plan to develop other smoothing methods for the distributed model (e.g. back-off). We are also planning to develop algorithms where the corpora mixture vector is adapted automatically to the test-set. We are also investigating novel techniques to use the n-gram statistics for ASR or SMT N-best re-ranking.

5. ACKNOWLEDGEMENTS

This work was partially supported by the Defense Advanced Research Projects Agency under contract No. HR0011-06-2-0001. We would also like to thank Brian Kingsbury for providing us with the lattices used in the ASR N-best re-ranking experiments.

6. REFERENCES

[1] Lalit R. Bahl, Frederick Jelinek, and Robert L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 2, pp. 179–190, 1983.

[2] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin, "A Statistical Approach to Machine Translation," *Computational Linguistics*, vol. 16, no. 2, pp. 79–85, 1990.

[3] A. Stolcke, "Entropy-based pruning of backoff language models," in *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, Lansdowne, VA, 1998, pp. 270–274.

[4] Ying Zhang, Almut Silja Hildebrand, and Stephan Vogel, "Distributed language modeling for n-best list re-ranking," in *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, Sydney, Australia, July 2006, pp. 216–223, Association for Computational Linguistics.

[5] William Gropp, Ewing Lusk, and Anthony Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA.

[6] Udi Manber and Gene Myers, "Suffix arrays: a new method for on-line string searches," *SIAM J. Comput.*, vol. 22, no. 5, pp. 935–948, 1993.

[7] Jelinek Frederick and Robert L. Mercer, "Interpolated estimation of markov source parameters from sparse data," in *Workshop on Pattern Recognition in Practice*, Amsterdam, The Netherlands, May 1980, North-Holland.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society*, vol. 39, pp. 1–38, 1977.

[9] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Proc. Interspeech*, 2005.

[10] Christoph Tillmann and Hermann Ney, "Word Re-ordering and a DP Beam Search Algorithm for Statistical Machine Translation," *Computational Linguistics*, vol. 29, no. 1, pp. 97–133, 2003.

[11] Yaser Al-Onaizan, "IBM Arabic-to-English MT Submission," *Presentation given at DARPA/TIDES NIST MT Evaluation workshop*, 2005.

[12] Abraham Ittycheriah and Salim Roukos, "A direct translation model," Tech. Rep., IBM TJ Watson Research Center, Yorktown Heights, NY, To appear.