Overcoming the Memory Bottleneck in Distributed Training of Latent Variable Models of Text

Yi Yang
Northwestern University
Evanston, IL
yiyang@eecs.northwestern.edu

Alexander Yates
Temple University
Philadelphia, PA
yates@temple.edu

Doug Downey
Northwestern University
Evanston, IL
ddowney@eecs.northwestern.edu

Abstract

Large unsupervised latent variable models (LVMs) of text, such as Latent Dirichlet Allocation models or Hidden Markov Models (HMMs), are constructed using parallel training algorithms on computational clusters. The memory required to hold LVM parameters forms a bottleneck in training more powerful models. In this paper, we show how the memory required for parallel LVM training can be reduced by partitioning the training corpus to minimize the number of unique words on any computational node. We present a greedy document partitioning technique for the task. For large corpora, our approach reduces memory consumption by over 50%, and trains the same models up to three times faster, when compared with existing approaches for parallel LVM training.

1 Introduction

Unsupervised latent variable models (LVMs) of text are utilized extensively in natural language processing (Griffiths and Steyvers, 2004; Ritter et al., 2010; Downey et al., 2007; Huang and Yates, 2009; Li and McCallum, 2005). LVM techniques include Latent Dirichlet Allocation (LDA) (Blei et al., 2003), Hidden Markov Models (HMMs) (Rabiner, 1989), and Probabilistic Latent Semantic Analysis (Hofmann, 1999), among others.

LVMs become more predictive as they are trained on more text. However, training LVMs on massive corpora introduces computational challenges, in terms of both time and space complexity. The time complexity of LVM training has been addressed through parallel training algorithms (Wolfe et al., 2008; Chu et al., 2006; Das et al., 2007; Newman et al., 2009; Ahmed et al., 2012; Asuncion et al., 2011), which reduce LVM training time through the use of large computational clusters.

However, the memory cost for training LVMs remains a bottleneck. While LVM training makes sequential scans of the corpus (which can be stored on disk), it requires consistent random access to model parameters. Thus, the model parameters must be stored in memory on each node. Because LVMs include a multinomial distribution over words for each latent variable value, the model parameter space increases with the number of latent variable values times the vocabulary size. For large models (i.e., with many latent variable values) and large corpora (with large vocabularies), the memory required for training can exceed the limits of the commodity servers comprising modern computational clusters. Because model accuracy tends to increase with both corpus size and model size (Ahuja and Downey, 2010; Huang and Yates, 2010), training accurate language models requires that we overcome the memory bottleneck.

We present a simple technique for mitigating the memory bottleneck in parallel LVM training. Existing parallelization schemes begin by partitioning the training corpus *arbitrarily* across computational nodes. In this paper, we show how to reduce memory footprint by instead partitioning the corpus to *minimize the number of unique words* on each node (and thereby minimize the number of parameters the node must store). Because corpus partitioning is a pre-processing step in parallel LVM training, our

technique can be applied to reduce the memory footprint of essentially any existing LVM or training approach. The accuracy of LVM training for a *fixed* model size and corpus remains unchanged, but intelligent corpus partitioning allows us to train larger and typically more accurate models using the same memory capacity.

While the general minimization problem we encounter is NP-hard, we develop greedy approximations that work well. In experiments with both HMM and LDA models, we show that our technique offers large advantages over existing approaches in terms of both memory footprint and execution time. On a large corpus using 50 nodes in parallel, our best partitioning method can reduce the memory required per node to less than 1/10th that when training without corpus partitioning, and to half that of a random partitioning. Further, our approach reduces the training time of an existing parallel HMM codebase by 3x. Our work includes the release of our partitioning codebase, and an associated codebase for the parallel training of HMMs.[1]

2 Problem Formulation

In a distributed LVM system, a training corpus $D = \{d_1, d_2, \ldots, d_N\}$ of documents is distributed across T computational nodes. We first formalize the memory footprint on each node n_t, where $t = \{1, ..., T\}$. Let $D_t \subset D$ denote the document collection on node n_t, and V_t be the number of word types (i.e., the number of unique words) in D_t. Let K be the number of latent variable values in the LVM.

With these quantities, we can express how many parameters must be held in memory on each computational node for training LVMs in a distributed environment. In practice, the LVM parameter space is dominated by an *observation model*: a conditional distribution over words given the latent variable value. Thus, the observation model includes $K(V_t - 1)$ parameters. Different LVMs include various other parameters to specify the complete model. For example, a first-order **HMM** includes additional distributions for the initial latent variable and latent variable transitions, for a total of $K(V_t - 1) + K^2$ parameters. **LDA**, on the other hand, includes just a single multinomial over the latent variables, making a total of $K(V_t - 1) + K - 1$ parameters.

The LVM parameters comprise almost all of the memory footprint for LVM training. Further, as the examples above illustrate, the number of parameters on each node tends to vary almost linearly with V_t (in practice, V_t is typically larger than K by an order of magnitude or more). Thus, in this paper we attempt to minimize memory footprint by limiting V_t on each computational node. We assume the typical case in a distributed environment where nodes are homogeneous, and thus our goal is to partition the corpus such that the *maximum* vocabulary size $V_{max} = max_{t=1}^{T} V_t$ on any single node is minimized. We define this task formally as follows.

Definition $\mathcal{CORPUSPART}$: Given a corpus of N documents $D = \{d_1, d_2, \ldots, d_N\}$, and T nodes, partition D into T subsets $D_1, D_2, \ldots, D_T$, such that V_{max} is minimized.

For illustration, consider the following small example. Let corpus C contain three short documents {c_1="I live in Chicago", c_2="I am studying physics", c_3="Chicago is a city in Illinois"}, and consider partitioning C into 2 non-empty subsets, i.e., $T = 2$. There are a total of three possibilities:

- $\{\{c_1, c_2\}, \{c_3\}\}$. $V_{max} = 7$
- $\{\{c_1, c_3\}, \{c_2\}\}$. $V_{max} = 8$
- $\{\{c_2, c_3\}, \{c_1\}\}$. $V_{max} = 10$

The decision problem version of $\mathcal{CORPUSPART}$ is NP-Complete, by a reduction from independent task scheduling (Zhu and Ibarra, 1999). In this paper, we develop greedy algorithms for the task that are effective in practice.

We note that $\mathcal{CORPUSPART}$ has a *submodular* problem structure, where greedy algorithms are often effective. Specifically, let $|S|$ denote the vocabulary size of a set of documents S, and let $S' \subseteq S$. Then for any document c the following inequality holds.

$$|S' \cup c| - |S'| \geq |S \cup c| - |S|$$

That is, adding a document c to the subset S' increases vocabulary size at least as much as adding c to S; the vocabulary size function is submodular. The $\mathcal{CORPUSPART}$ task thus seeks a partition of the data that minimizes the maximum of a set of submodular functions. While formal approximation

[1] https://code.google.com/p/corpus-partition/

guarantees exist for similar problems, to our knowledge none apply directly in our case. For example, (Krause et al., 2007) considers maximizing the minimum over a set of monotonic submodular functions, which is the opposite of our problem. The distinct task of minimizing a single submodular function has been investigated in e.g. (Iwata et al., 2001).

It is important to emphasize that data partitioning is a *pre-processing* step, after which we can employ precisely the same Expectation-Maximization (EM), sampling, or variational parameter learning techniques as utilized in previous work. In fact, for popular learning techniques including EM for HMMs (Rabiner, 1989) and variational EM for LDA (Wolfe et al., 2008), it can be shown that the parameter updates are *independent* of how the corpus is partitioned. Thus, for those approaches our partitioning is guaranteed to produce the same models as any other partitioning method; i.e., model accuracy is unchanged.

Lastly, we note that we target *synchronized* LVM training, in which all nodes must finish each training iteration before any node can proceed to the next iteration. Thus, we desire balanced partitions to help ensure iterations have similar durations across nodes. We achieve this in practice by constraining each node to hold at most 3% more than Z/T tokens, where Z is the corpus size in tokens.

3 Corpus Partitioning Methods

Our high-level greedy partitioning framework is given in Algorithm 1. The algorithm requires answering two key questions: How do we select which document to allocate next? And, given a document, on which node should it be placed? We present alternative approaches to each question below.

Algorithm 1 Greedy Partitioning Framework

INPUT: $\{D, T\}$
OUTPUT: $\{D_1, \ldots, D_T\}$
Objective: Minimize V_{max}
Initialize each subset $D_t = \emptyset$ for T nodes
repeat
  document selection:Select document d from D
  node selection: Select node n_t, and add d to D_t
  Remove d from D
until all documents are allocated

A **baseline** partitioning method commonly used in practice simply distributes documents across nodes randomly. As our experiments show, this baseline approach can be improved significantly.

In the following, set operations are interpreted as applying to the set of unique words in a document. For example, $|d \cup D_t|$ indicates the number of unique word types in node n_t after document d is added to its document collection D_t.

3.1 Document Selection

For document selection, previous work (Zhu and Ibarra, 1999) proposed a heuristic **DISSIMILARITY** method that selects the document d that is *least similar* to *any* of the node document collections D_t, where the similarity of d and D_t is calculated as: $Sim(d, D_T) = |d \cap D_t|$. The intuition behind the heuristic is that dissimilar documents are more likely to impact future node selection decisions. Assigning the dissimilar documents earlier helps ensure that more greedy node selections are informed by these impactful assignments.

However, DISSIMILARITY has a prohibitive time complexity of $\mathcal{O}(TN^2)$, because we must compare T nodes to an order of N documents for a total of N iterations. To scale to large corpora, we propose a novel BATCH DISSIMILARITY method. In BATCH DISSIMILARITY, we select the top L most dissimilar documents in each iteration, instead of just the most dissimilar. Importantly, L is altered dynamically: we begin with $L = 1$, and then increase L by one for iteration $i+1$ *iff* using a batch size of $L+1$ in iteration i would not have altered the algorithm's ultimate selections (that is, if the most dissimilar document in iteration $i + 1$ is in fact the $L + 1$st most dissimilar in iteration i). In the ideal case where L is incremented each iteration, BATCH DISSIMILAR will have a reduced time complexity of $O(TN^{3/2})$.

Our experiments revealed two key findings regarding document selection. First, BATCH DISSIMILARITY provides a memory reduction within 0.1% of that of DISSIMILARITY (on small corpora where running DISSIMILARITY is tractable), but partitions an estimated 2,600 times faster on our largest evaluation corpus. Second, we found that document selection has *relatively minor* impact on memory footprint, providing a roughly 5% incremental benefit over random document selection. Thus, although

we utilize BATCH DISSIMILARITY in the final system we evaluate, simple random document selection may be preferable in some practical settings.

3.2 Node Selection

Given a selected document d, the **MINIMUM** method proposed in previous work selects node n_t having the minimum number of word types after allocation of d to n_t (Zhu and Ibarra, 1999). That is, MINIMUM minimizes $|d \cup D_t|$. Here, we introduce an alternative node selection method **JACCARD** that selects node n_t maximizing the Jaccard index, defined here as $|d \cap D_t|/|d \cup D_t|$.

Our experiments showed that our JACCARD node selection method outperforms the MINIMUM selection method. In fact, for the largest corpora used in our experiments, JACCARD offered an 12.9% larger reduction in V_{max} than MINIMUM. Our proposed system, referred to as BJAC, utilizes our best-performing strategies for document selection (BATCH DISSIMILARITY) and node selection (JACCARD).

4 Evaluation of Partitioning Methods

We evaluate our partitioning method against the baseline and **Z&I**, the best performing scalable method from previous work, which uses random document selection and MINIMUM node selection (Zhu and Ibarra, 1999). We evaluate on three corpora (Table 1): the *Brown* corpus of newswire text (Kucera and Francis, 1967), the Reuters Corpus Volume1 (*RCV1*) (Lewis et al., 2004), and a larger *Web-Sent* corpus of sentences gathered from the Web (Downey et al., 2007).

Corpus	N	V	Z
Brown	57339	56058	1161183
RCV1	804414	288062	99702278
Web-Sent	2747282	214588	58666983

Table 1: Characteristics of the three corpora. N = # of documents, V = # of word types, Z = # of tokens. We treat each sentence as a document in the Brown and Web-Sent corpora.

Table 2 shows how the maximum word type size V_{max} varies for each method and corpus, for $T = 50$ nodes. BJAC significantly decreases V_{max} over the random partitioning baseline typically employed in practice. Furthermore, the advantage of BJAC over the baseline is maintained as more computational nodes are utilized, as illustrated in Figure 1. BJac reduces V_{max} by a larger factor over the baseline as more computational nodes are employed.

Corpus	baseline	Z&I	BJAC
Brown	6368	5714	4369
RCV1	49344	32136	24923
Web-Sent	72626	45989	34754

Table 2: Maximum word type size V_{max} for each partitioning method, for each corpus. For the larger corpora, BJAC reduces V_{max} by over 50% compared to the baseline, and by 23% compared to Z&I.

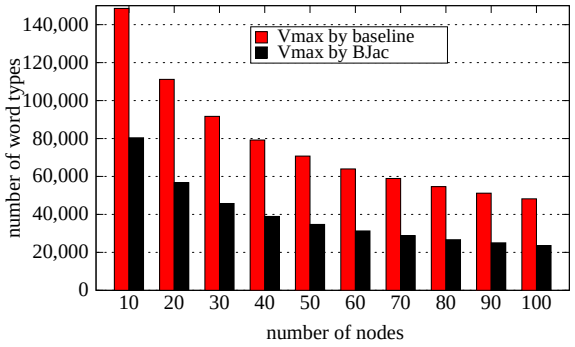

Figure 1: Effects of partitioning as the number of computational nodes increases (Web-Sent corpus). With 100 nodes, BJac's V_{max} is half that of the baseline, and 1/10th of the full corpus vocabulary size.

5 Evaluation in Parallel LVM Systems

We now turn to an evaluation of our corpus partitioning within parallel LVM training systems.

Table 3 shows the memory footprint required for HMM and LDA training for three different partitioning methods. We compare BJAC with the random partitioning baseline, Zhu's method, and with *all-words*, the straightforward approach of simply storing parameters for the entire corpus vocabulary on every node (Ahuja and Downey, 2010; Asuncion et al., 2011). All-words has the same memory footprint as when training on a single node.

For large corpora, BJAC reduces memory size per node by approximately a factor of two over the random baseline, and by a factor of 8-11 over all-

LVM	Corpus	all-words	baseline	BJAC
HMM	Brown	435.3	56.2	40.9
	RCV1	2205.4	384.1	197.8
	Web-Sent	1644.8	561.7	269.7
LDA	Brown	427.7	48.6	33.3
	RCV1	2197.7	376.5	190.1
	Web-Sent	1637.2	554.1	262.1

Table 3: Memory footprint of computational nodes in megabytes(MB), using 50 computational nodes. Both models utilize 1000 latent variable values.

T	all-words	baseline	BJAC
25	4510	1295	1289
50	2248	740	735
100	1104	365	364
200	394	196	192

Table 4: Average iteration time(sec) for training an HMM with 50 hidden states on Web-Sent. Partitioning with BJAC outperforms all-words, which stores parameters for all word types on each node.

words. The results demonstrate that in addition to the well-known savings in computation time offered by parallel LVM training, distributed computation *also* significantly reduces the memory footprint on each node. In fact, for the RCV1 corpus, BJAC reduces memory footprint to less than 1/10th that of training with all words on each computational node.

We next evaluate the *execution time* for an iteration of model training. Here, we use a parallel implementation of HMMs, and measure iteration time for training on the Web-sent corpus with 50 hidden states as the number of computational nodes varies. We compare against the random baseline and against the all-words approach utilized in an existing parallel HMM codebase (Ahuja and Downey, 2010). The results are shown in Table 4. Moving beyond the all-words method to exploit corpus partitioning reduces training iteration time, by a factor of two to three. However, differences in partitioning methods have only small effects in iteration time: BJAC has essentially the same iteration time as the random baseline in this experiment.

It is also important to consider the additional time required to execute the partitioning methods themselves. However, in practice this additional time is negligible. For example, BJAC can partition the Web-sent corpus in 368 seconds, using a single computational node. By contrast, training a 200-state HMM on the same corpus requires over a hundred CPU-days. Thus, BJAC's time to partition has a negligible impact on total training time.

6 Related Work

The $\mathcal{CORPUSPART}$ task has some similarities to the graph partitioning task investigated in other parallelization research (Hendrickson and Kolda, 2000). However, our LVM training task differs significantly from those in which graph partitioning is typically employed. Specifically, graph partitioning tends to be used for scientific computing applications where communication is the bottleneck. The graph algorithms focus on creating balanced partitions that minimize the cut edge weight, because edge weights represent communication costs to be minimized. By contrast, in our LVM training task, memory consumption is the bottleneck and communication costs are less significant.

Zhu & Ibarra (1999) present theoretical results and propose techniques for the general partitioning task we address. In contrast to that work, we focus on the case where the data to be partitioned is a large corpus of text. In this setting, we show that our heuristics partition faster and provide smaller memory footprint than those of (Zhu and Ibarra, 1999).

7 Conclusion

We presented a general corpus partitioning technique which can be exploited in LVM training to reduce memory footprint and training time. We evaluated the partitioning method's performance, and showed that for large corpora, our approach reduces memory consumption by over 50% and learns models up to three times faster when compared with existing implementations for parallel LVM training.

Acknowledgments

This work was supported in part by NSF Grants IIS-101675 and IIS-1065397, and DARPA contract D11AP00268.

References

Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J. Smola. 2012. Scalable inference in latent variable models. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 123–132, New York, NY, USA. ACM.

Arun Ahuja and Doug Downey. 2010. Improved extraction assessment through better language models. In *Human Language Technologies: Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*.

Arthur U. Asuncion, Padhraic Smyth, and Max Welling. 2011. Asynchronous distributed estimation of topic models for document analysis. *Statistical Methodology*, 8(1):3 – 17. Advances in Data Mining and Statistical Learning.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March.

Cheng T. Chu, Sang K. Kim, Yi A. Lin, Yuanyuan Yu, Gary R. Bradski, Andrew Y. Ng, and Kunle Olukotun. 2006. Map-Reduce for machine learning on multicore. In Bernhard Schölkopf, John C. Platt, and Thomas Hoffman, editors, *NIPS*, pages 281–288. MIT Press.

Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA. ACM.

D. Downey, S. Schoenmackers, and O. Etzioni. 2007. Sparse information extraction: Unsupervised language models to the rescue. In *Proc. of ACL*.

T. L. Griffiths and M. Steyvers. 2004. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(Suppl. 1):5228–5235, April.

Bruce Hendrickson and Tamara G Kolda. 2000. Graph partitioning models for parallel computing. *Parallel computing*, 26(12):1519–1534.

Thomas Hofmann. 1999. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 50–57, New York, NY, USA. ACM.

Fei Huang and Alexander Yates. 2009. Distributional representations for handling sparsity in supervised sequence labeling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Fei Huang and Alexander Yates. 2010. Exploring representation-learning approaches to domain adaptation. In *Proceedings of the ACL 2010 Workshop on Domain Adaptation for Natural Language Processing (DANLP)*.

Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. 2001. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48:761–777.

Andreas Krause, H. Brendan Mcmahan, Google Inc, Carlos Guestrin, and Anupam Gupta. 2007. Selecting observations against adversarial objectives. Technical report, In NIPS, 2007a.

H. Kucera and W. N. Francis. 1967. *Computational analysis of present-day American English*. Brown University Press, Providence, RI.

David D. Lewis, Yiming Yang, Tony G. Rose, Fan Li, G. Dietterich, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397.

Wei Li and Andrew McCallum. 2005. Semi-supervised sequence modeling with syntactic topic models. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 2*, AAAI'05, pages 813–818. AAAI Press.

David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. 2009. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10:1801–1828.

L. R. Rabiner. 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Alan Ritter, Colin Cherry, and Bill Dolan. 2010. Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 172–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Jason Wolfe, Aria Haghighi, and Dan Klein. 2008. Fully distributed EM for very large datasets. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1184–1191, New York, NY, USA. ACM.

Huican Zhu and Oscar H. Ibarra. 1999. On some approximation algorithms for the set partition problem. In *Proceedings of the 15th Triennial Conf. of Int. Federation of Operations Research Society*.