Fast Collapsed Gibbs Sampling For Latent Dirichlet Allocation

Ian Porteous
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
iporteou@ics.uci.edu

David Newman
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
newman@uci.edu

Alexander Ihler
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
ihler@ics.uci.edu

Arthur Asuncion
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
asuncion@ics.uci.edu

Padhraic Smyth
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
smyth@ics.uci.edu

Max Welling
Dept. of Computer Science
University of California, Irvine
Irvine, CA 92697-3425
welling@ics.uci.edu

ABSTRACT

In this paper we introduce a novel collapsed Gibbs sampling method for the widely used latent Dirichlet allocation (LDA) model. Our new method results in significant speedups on real world text corpora. Conventional Gibbs sampling schemes for LDA require O(K) operations per sample where K is the number of topics in the model. Our proposed method draws equivalent samples but requires on average significantly less then K operations per sample. On real-word corpora FastLDA can be as much as 8 times faster than the standard collapsed Gibbs sampler for LDA. No approximations are necessary, and we show that our fast sampling scheme produces exactly the same results as the standard (but slower) sampling scheme. Experiments on four real world data sets demonstrate speedups for a wide range of collection sizes. For the PubMed collection of over 8 million documents with a required computation time of 6 CPU months for LDA, our speedup of 5.7 can save 5 CPU months of computation.

Categories and Subject Descriptors

G.3 [**Probabilistic algorithms**]: Miscellaneous

General Terms

Experimentation, Performance, Algorithms

Keywords

Latent Dirichlet Allocation, Sampling

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

1. INTRODUCTION

The latent Dirichlet allocation (LDA) model (or "topic model") is a general probabilistic framework for modeling sparse vectors of count data, such as bags of words for text, bags of features for images, or ratings of items by customers. The key idea behind the LDA model (for text data for example) is to assume that the words in each document were generated by a mixture of topics, where a topic is represented as a multinomial probability distribution over words. The mixing coefficients for each document and the word-topic distributions are unobserved (hidden) and are learned from data using unsupervised learning methods. Blei et al [3] introduced the LDA model within a general Bayesian framework and developed a variational algorithm for learning the model from data. Griffiths and Steyvers [6] subsequently proposed a learning algorithm based on collapsed Gibbs sampling. Both the variational and Gibbs sampling approaches have their advantages: the variational approach is arguably faster computationally, but the Gibbs sampling approach is in principal more accurate since it asymptotically approaches the correct distribution.

Since the original introduction of the LDA model, the technique has been broadly applied in machine learning and data mining, particularly in text analysis and computer vision, with the Gibbs sampling algorithm in common use. For example, Wei and Croft [19] and Chemudugunta, Smyth, and Steyvers [5] have successfully applied the LDA model to information retrieval and shown that it can significantly outperform – in terms of precision-recall – alternative methods such as latent semantic analysis. LDA models have also been increasingly applied to problems involving very large text corpora: Buntine [4], Mimno and McCallum [12] and Newman et al [15] have all used the LDA model to automatically generate topic models for millions of documents and used these models as the basis for automated indexing and faceted Web browsing.

In this general context there is significant motivation to speed-up the learning of topic models, both to reduce the time taken to learn topic models for very large text collections, as well as moving towards "real-time" topic modeling (e.g., for a few thousand documents returned by a search engine). The collapsed Gibbs sampling algorithm of Griffiths and Steyvers involves repeatedly sampling a topic assign-

ment for each word in the corpus, where a single iteration of the Gibbs sampler consists of sampling a topic for each word. Each sampled topic assignment is generated from a conditional multinomial distribution over the K topics, which in turn requires the computation of K conditional probabilities. As an example, consider learning a topic model with one million documents, each with 1000 words on average, $K = 1000$ topics, and performing 500 Gibbs iterations (a typical number in practice). This would require generating a total of 5×10^{11} word-topic assignments via sampling, where each sampling operation itself involves $K = 1000$ computations.

The key idea of our paper is to reduce the time taken for the "inner-loop" sampling operation, reducing it from K to significantly less then K on average; we observe speedups up to a factor of 8 in our experiments. Furthermore, the speedup usually increases as K increases. In our proposed approach we exploit the fact that, for any particular word and document, the sampling distributions of interest are frequently skewed such that most of the probability mass is concentrated on a small fraction of the total number of topics K. This allows us to order the sampling operations such that on average only a fraction of the K topic probabilities need to be calculated. Our proposed algorithm is exact, i.e., no approximation is made and the fast algorithm correctly and exactly samples from the same true posterior distribution as the slower standard Gibbs sampling algorithm.

2. RELATED WORK

The problem of rapidly evaluating or approximating probabilities and drawing samples arises in a great many domains. However, most existing solutions are characterized by the data being embedded in a metric space, so that geometric relationships can be exploited to rapidly evaluate the total probability of large sets of potential states. Mixture modeling problems provide a typical example: a data structure which clusters data by spatial similarity, such as a KD-tree [2], stores statistics of the data in a hierarchical fashion and uses these statistics to compute upper and lower bounds on the association probabilities for any data within those sets. Using these bounds, one may determine whether the current estimates are sufficiently accurate, or whether they need to be improved by refining the clusters further (moving to the next level of the data structure).

Accelerated algorithms of this type exist for many common probabilistic models. In some cases, such as k-means, it is possible to accelerate the computation of an exact solution [1, 16, 17]. For other algorithms, such as expectation–maximization for Gaussian mixtures, the evaluations are only approximate but can be controlled by tuning a quality parameter [13, 10, 9]. In [8], a similar branch-and-bound method is used to compute approximate probabilities and draw approximate samples from the product of several Gaussian mixture distributions.

Unfortunately, the categorical nature of LDA makes it difficult to apply any of these techniques directly. Instead, although we apply a similar "bound and refine" procedure, both the bound and the sequence of refinement operations must be matched to the expected behavior of the data in topic modeling. We describe the details of this bound along with our algorithm in Section 4, after first reviewing the standard LDA model and Gibbs sampling.

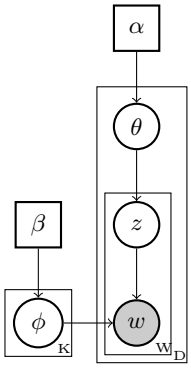

Figure 1: Graphical model for LDA.

3. LDA

LDA models each of D documents as a mixture over K latent topics, each of which describes a multinomial distribution over a W word vocabulary. Figure 1 shows the graphical model representation of the LDA model.

The LDA model is equivalent to the following generative process for words and documents:

For each of N_j words in document j

1. sample a topic $z_{ij} \sim \text{Multinomial}(\theta_j)$
2. sample a word $x_{ij} \sim \text{Multinomial}(\phi_{z_{ij}})$

where the parameters of the multinomials for topics in a document θ_j and words in a topic ϕ_k have Dirichlet priors. Intuitively we can interpret the multinomial parameter ϕ_k as indicating which words are important in topic k and the parameter θ_j as indicating which topics appear in document j [6]. Given the observed words $\mathbf{x} = \{x_{ij}\}$, the task of Bayesian inference is to compute the posterior distribution over the latent topic indices $\mathbf{z} = \{z_{ij}\}$, the mixing proportions θ_j, and the topics ϕ_k. An efficient inference procedure is to use collapsed Gibbs sampling [6], where θ and ϕ are marginalized out, and only the latent variables $\mathbf{z}$ are sampled. After the sampler has burned-in we can calculate an estimate of θ and ϕ given $\mathbf{z}$.

We define summations of the data by $N_{wkj} = \#\{i : x_{ij} = w, z_{ij} = k\}$, and use the convention that missing indices are summed out, so that $N_{kj} = \sum_w N_{wkj}$ and $N_{wk} = \sum_j N_{wkj}$. In words, N_{wk} is the number of times the word w is assigned to the topic k and N_{kj} is the number of times a word in document j has been assigned to topic k. Given the current state of all but one variable z_{ij}, the conditional probability of z_{ij} is then

$$p(z_{ij} = k | \mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta) = \frac{1}{Z} a_{kj} b_{wk} \tag{1}$$

where

$$a_{kj} = N_{kj}^{\neg ij} + \alpha \qquad b_{wk} = \frac{N_{wk}^{\neg ij} + \beta}{N_k^{\neg ij} + W\beta},$$

Z is the normalization constant

$$Z = \sum_k a_{kj} b_{wk},$$

and the superscript $\neg ij$ indicates that the corresponding datum has been excluded in the count summations N_{wkj}.

Algorithm 3.1: LDA GIBBS SAMPLING($\mathbf{z}, \mathbf{x}$)

for $i \leftarrow 1$ **to** N
do
$u \leftarrow$ draw from Uniform$[0, 1]$
for $k \leftarrow 1$ **to** K
do
$P[k] \leftarrow P[k-1] + \frac{(N_{kj}^{\neg ij}+\alpha)(N_{x_{ij}k}^{\neg ij}+\beta)}{(N_k^{\neg ij}+W\beta)}$
for $k \leftarrow 1$ **to** K
do
if $u < P[k]/P[K]$
then $z_{ij} = k, stop$

An iteration of Gibbs sampling proceeds by drawing a sample for z_{ij} according to (1) for each word i in each document j. A sample is typically accomplished by first calculating the normalization constant Z, then sampling z_{ij} according to its normalized probability; see Algorithm 3.1. Given the value sampled for z_{ij} the counts N_{kj}, N_k, N_{wk} are updated. The time complexity for each iteration of Gibbs sampling is then $O(NK)$.

Given a sample we can then get an estimate for $\hat{\theta}_j$ and $\hat{\phi}_k$ given $\mathbf{z}$:

$$\hat{\phi}_{wk} = \frac{N_{wk} + \beta}{N_k + W\beta}$$
$$\hat{\theta}_{kj} = \frac{N_{kj} + \alpha}{N_j + K\alpha}$$

4. FAST LDA

For most real data sets after several iterations of the Gibbs sampler, the probability mass of the distribution $p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta)$ becomes concentrated on only a small set of the topics as in Figure 4. FastLDA takes advantage of this concentration of probability mass by only checking a subset of topics before drawing a correct sample. After calculating the unnormalized probability in (1) of a subset of topics, FastLDA determines that the sampled value does not depend on the probability of the remaining topics. To describe how FastLDA works, it is useful to introduce a graphical depiction of how a sample for z_{ij} is conventionally drawn. We begin by segmenting a line of unit length into K sections, with the k^{th} section having length equal to $p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta)$. We then draw a sample for z_{ij} by drawing a value uniformly from the interval, $u \sim$ Uniform[0,1], and selecting the value of z_{ij} based on the segment into which u falls; see Figure 2.

As an alternative, suppose that we have a sequence of bounds on the normalization constant Z, denoted $Z_1 \ldots Z_K$, such that $Z_1 \geq Z_2 \geq \ldots \geq Z_K = Z$. Then, we can graphically depict the sampling procedure for FastLDA in a similar way, seen in Figure 3. Instead of having a single segment for topic k, of length $p_k/Z = p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta)$, we instead have *several* segments $s_l^k \ldots s_K^k$ associated with each topic. The first segment for a topic k, s_k^k, describes a conservative estimate of the probability of the topic given the upper bound Z_k on the true normalization factor Z. Each of the subsequent segments associated with topic k, namely s_l^k for $l > k$, are the corrections for the missing probability mass for topic k given the improved bound Z_l. Mathematically,

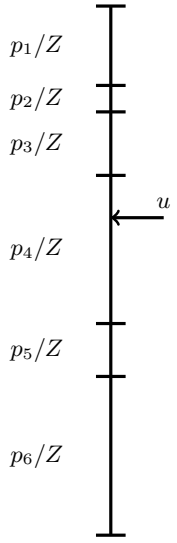

Figure 2: Draw from $p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta)$. $p_k = a_{jk}b_{wk}$. $u \sim$ Uniform$[0, 1]$. A topic k is sampled by finding which segment (p_k) contains the draw u. Here the total number of topics $K = 6$.

the lengths of these segments are given by

$$s_k^k = \frac{a_{jk}b_{wk}}{Z_k}$$
$$\forall l > k \qquad s_l^k = (a_{jk}b_{wk})(\frac{1}{Z_l} - \frac{1}{Z_{l-1}})$$

Since the final bound $Z_K = Z$, the total sum of the segment lengths for topic k is equal to the true, normalized probability of that topic:

$$p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta) = \sum_{l=k}^{K} s_l^k$$

Therefore, as in the conventional sampling method, we can draw z_{ij} from the correct distribution by first drawing $u \sim$ Uniform$[0, 1]$, then determining the segment in which it falls.

By organizing the segments in this way, we can obtain a substantial advantage: we can check each segments in order, knowing only $p_1 \ldots p_k$ and Z_k, and if we find that u falls within a particular segment s_l^k, the remaining segments are irrelevant. Importantly, if for our sequence of bounds $Z_1 \ldots Z_K$, an intermediate bound Z_l depends only on the values of a_{jk} and b_{jk} for $k \leq l$, then we may be able to draw the sample after only examining topics $1 \ldots l$. Given that in LDA, the probability mass is typically concentrated on a small subset of topics for a given word and document, in practice we may have to do far fewer operations per sample on average.

4.1 Upper Bounds for Z

FastLDA depends on finding a sequence of improving bounds on the normalization constant, $Z_1 \geq Z_2 \geq \ldots \geq Z_K = Z$.

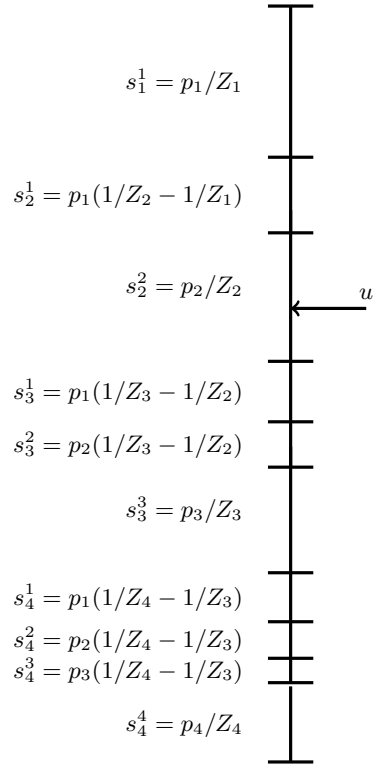

Figure 3: Draw from $p(z_{ij} = k|\mathbf{z}^{\neg ij}, \mathbf{x}, \alpha, \beta)$. $p_k = a_{jk}b_{wk}$. $u \sim$ Uniform$[0,1]$. A topic k is sampled by finding which segment (s_j^k) contains the draw u. Here the total number of topics $K = 4$.

We first define Z in terms of component vectors $\vec{a}, \vec{b}, \vec{c}$:

$$\vec{a} = [N_{1j}^{\neg ij} + \alpha, .., N_{Kj}^{\neg ij} + \alpha]$$
$$\vec{b} = [N_{w1}^{\neg ij} + \beta, .., N_{wK}^{\neg ij} + \beta]$$
$$\vec{c} = [1/(N_1^{\neg ij} + W\beta), .., 1/(N_K^{\neg ij} + W\beta)]$$

Then, the normalization constant is given by

$$Z = \sum_k \vec{a}_k \vec{b}_k \vec{c}_k$$

To construct an initial upper bound Z_0 on Z, we turn to the generalized version of Hölder's inequality [7], which states

$$Z_0 = \|\vec{a}\|_p \|\vec{b}\|_q \|\vec{c}\|_r \geq Z$$
$$\text{where} \qquad 1/p + 1/q + 1/r = 1$$

Notice that, as we examine topics in order, we learn the actual value of the product $\vec{a}_k\vec{b}_k\vec{c}_k$. We can use these calculations to improve the bound at each step. We then have a bound at step l given by:

$$Z_l = \sum_{i=1}^{l} \left(\vec{a}_i \vec{b}_i \vec{c}_i\right) + \|\vec{a}_{l+1:K}\|_p \|\vec{b}_{l+1:K}\|_q \|\vec{c}_{l+1:K}\|_r \geq Z$$

This sequence of bounds satisfies our requirements: it is decreasing, and if $l = K$ we recover the exact value of Z. In this way the bound improves incrementally at each iteration until we eventually obtain the correct normalization factor.

Hölder's inequality describes a class of bounds, for any valid choice of (p, q, r); these values are a design choice of the algorithm. A critical aspect in the choice of bounds is that it must be computationally efficient to maintain. In particular we want to be able to calculate Z_0 and update

$$\|\vec{a}_{l+1:K}\|_p \ \|\vec{b}_{l+1:K}\|_q \ \|\vec{c}_{l+1:K}\|_r$$

in constant time. We focus our attention on two natural choices of values which lead to computationally efficient implementations: $(p, q, r) = (2, 2, \infty)$ and $(3, 3, 3)$. For $p, q, r < \infty$, the norms can be updated in constant time, while for $r = \infty$, we have $\|\vec{c}_{l+1:K}\|_r = \min_k N_k$ which is also relatively efficient to maintain. Section 4.4 provides more detail on how these values are maintained. Empirically, we found that the first choice typically results in a better bound (see Figure 10 in Section 6.4).

FastLDA maintains the norms $\|\vec{a}\|_p, \|\vec{b}\|_q, \|\vec{c}\|_r$ separately and then uses their product to bound Z. One might consider maintaining the norm $\|\vec{a}\vec{b}\|$, $\|\vec{b}\vec{c}\|$ or even Z instead, improving on or eliminating the bound for Z. The problem with maintaining any combination of the vectors $\vec{a}, \vec{b}$ or $\vec{c}$ is that the update of one z_{ij} will cause many separate norms to change because they depend on the counts of z_{ij} variables, N_{wkj}. For example, if we maintain $d_{wk} = b_{wk}c_k$, then a change of the value of z_{ij} from k to k' requires changes to $d_{wk}, d_{wk'} \forall_w$ resulting in $O(2W)$ operations. However without $\vec{d}$, only $b_{wk}, b_{wk'}, c_k, c_{k'}$ change.

Algorithm 4.1: FastLDA$(\vec{a}, \vec{b}, \vec{c})$

```
sump_k ← 0
u ←~Uniform[0,1]
for k ← 1 to K
  do
{ sump_k ← sump_{k-1} + a_k b_k c_k
{ Z_k ← sump_k + ||a_{l+1:K}|| ||b_{l+1:K}|| 1/(min_k N_k^{¬ij} + Wβ)
{ if u × Z_k > sump_k
  then continue to next k
  else { if (k = 1) or (uZ_k > sump_{k-1})
       { then return (k)
       { else { u ← ((uZ_{k-1} − sump_{k-1}) Z_k) / (Z_{k-1} − Z_k)
              { for t ← 1 to k
                do
              { if (sump_t >= u)
                then return (t)
```

4.2 Refinement Sequence

Finally, we must also consider the order in which the topics are evaluated. Execution time improves as the number of topics considered before we find the segment s_l^k containing u decreases. We thus would like the algorithm to consider the longest segments first, and only check the short segments if necessary. Two factors affect the segment length: p_k, the unnormalized probability, and Z_l, the bound on Z at step l. Specifically, we want to check the topics with the largest p_k early, and similarly the topics which will improve (decrease) the bound Z_l.

Those topics which fall into the former category are those with (relatively) large values for the product $\vec{a}_k\vec{b}_k\vec{c}_k$, while

those falling into the latter category are those with large values for at least one of $\vec{a}_k$, $\vec{b}_k$, and $\vec{c}_k$. Thus it is natural to seek out those topics k which have large values in one or more of these vectors.

Another factor which must be balanced is the computational effort to find and maintain an order for refinement. Clearly, to be useful a method must be faster than a direct search over topics. To greedily select a good refinement order while ensuring that we maintain computational efficiency, we consider topics in descending order of N_{kj}, the frequency of word assignments to a topic in the current document (equivalent to descending order on the elements of $\vec{b}$). This order is both efficient to maintain (see Section 4.4) and appears effective in practice.

4.3 Fast LDA Algorithm

The sampling step for FastLDA begins with a sorted list of topics in descending order by N_{jk}, the most popular topic for a document to the least popular. A random value u is sampled $u \sim \text{Uniform}[0, 1]$. The algorithm then considers topics in order, calculating the length of segments s_l^k as it goes. Each time the next topic is considered the bound Z_k is improved. As soon as the sum of segments calculated so far is greater then u, the algorithm can stop and return the topic associated with the segment u falls on. Graphically, the algorithm scans down the line in Figure 3 calculating only s_l^k and Z_k for the k topics visited so far. When the algorithm finds a segment whose end point is past u it stops and returns the associated topic. By intelligently ordering the comparisons as to whether u is within a segment, we need to do $2K$ comparisons in the worst case.

4.4 Complexity of the Algorithm

To improve over the conventional algorithm, FastLDA must maintain the sorted order of N_{kj} and the norms of each component: $min_k Nk$, $\|\vec{a}_{l:K}\|$ and $\|\vec{b}_{l:K}\|$, more efficiently then the K steps required for the calculation of Z. The strategy used is to calculate the values initially and then update only the affected values after each sample of z_{ij}. Maintaining the descending sort order of N_{kj} or the minimum element of N_k can be done inexpensively, and in practice much faster than the worst case $O(\log K)$ required for a delete/insert operation into a sorted array. We start by performing an initial sort of these integer arrays, which takes $O(K \log K)$ time. During an update, one element of N_{kj} is incremented by one, and another element of N_{kj} is decremented by one (likewise for N_k). Given that we have integer arrays, this update will render the array in almost sorted order, and we expect that only a few swaps are required to restore sorted order. Using a simple bubble sort, the amortized time for this maintain-sort operation is very small, and in practice much faster than $O(\log K)$.

Maintaining the value of the finite norms, $\|\vec{a}\|$ and $\|\vec{b}\|$, from iteration to iteration can be done by calculating the values once during initialization and then updating the value when an associated z_{ij} is sampled. Two norms need to be updated when z_{ij} is updated, the value of $\|\vec{a}\|$ for document j and the value of $\|\vec{b}\|$ for word w, where $x_{ij} = w$. These updates can be done in $O(1)$ time.

In addition, we require the incremental improvements at each step of the sampling process, i.e., at topic $k-1$ we require $\|\vec{a}_{k:K}\|$ and $\|\vec{b}_{k:K}\|$, the norms of the remaining topics from k to K. (We upper-bound $\|\vec{c}_{k:K}\|$ by its initial value, $\|\vec{c}\|$.) For finite p-norms, given $\|\vec{a}_{k:K}\|_p$ it is an $O(1)$ update from $\|\vec{a}_{k:K}\|_p \rightarrow \|\vec{a}_{k+1:K}\|_p$, and equivalently for $\|\vec{b}_{k:K}\|_q$.

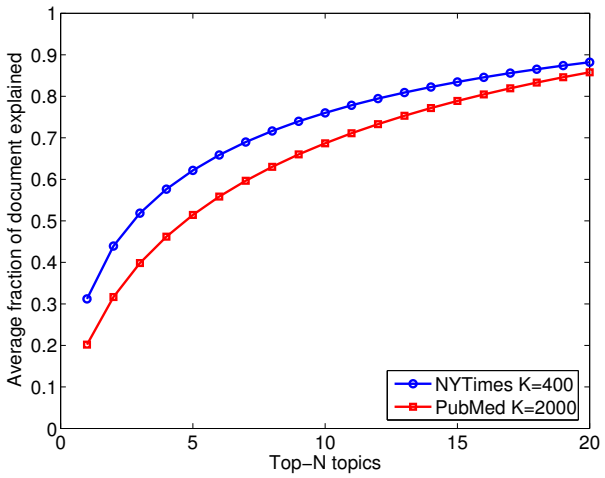

Figure 4: Average fraction of a document explained by top-20 topics, for NYTimes (K=400 topics) and PubMed (K=2000 topics). We see that, on average, the top-20 topics in any document account for approximately 90% of the words in the document.

	D	N	W
NIPS	$1,500$	1.9×10^6	$12,419$
Enron	$39,861$	6.4×10^6	$28,102$
NYTimes	$300,000$	100×10^6	$102,660$
PubMed	$8,200,000$	730×10^6	$141,043$

Table 1: Size parameters for the four data sets used in experiments. D is number of documents, N is total number of words in the collection, and W is size of vocabulary.

5. DATA SETS

We compared execution times of LDA and FastLDA using four data sets: NIPS full papers (from books.nips.cc), Enron emails (from www.cs.cmu.edu/∼enron), NYTimes news articles (from ldc.upenn.edu), and PubMed abstracts (from www.pubmed.gov). These four data sets span a wide range of collection size, content, and average document length. The NYTimes and PubMed collections are relatively large, and therefore useful for demonstrating the potential benefits of FastLDA. For each collection, after tokenization and removal of stopwords, the vocabulary of unique words was truncated by only keeping words that occurred more than ten times. The size parameters for these four data sets are shown in Table 1.

While the NIPS and Enron data sets are moderately sized, and thus useful for conducting parameter studies, the NYTimes and PubMed data sets are relatively large. Running LDA on the NYTimes data set using $K = 1600$ topics can take more than a week on a typical high-end desktop computer, and running LDA on the PubMed data set using $K = 4000$ topics would take months, and would require

memory well beyond typical desktop computers. Consequently, these larger data sets are ideal candidates for showing the reduction in computation time from our FastLDA method, and measuring speedup on real-life large-scale corpora.

6. EXPERIMENTS

Before describing and explaining our experiments, we point the reader to the Appendix, which lists the exact parameter specifications used to run our experiments. With the goal of repeatability, we have made our LDA and FastLDA code publicly available at http:// www.ics.uci.edu/ ∼iporteou/ fastlda and the four data sets at the UCI Machine Learning Repository, http:// archive.ics.uci.edu/ml/ machine-learning-databases/ bag-of-words/.

The purpose of our experiments was to measure actual reduction in execution time of FastLDA relative to LDA. Consequently, we setup a highly-controlled compute environment to perform timing tests. All speedup experiments were performed in pairs, with LDA and FastLDA being run on the same computer, compiler and environment to allow a fair comparison of execution times. Most computations were run on workstations with dual Xeon 3.0GHz processors with code compiled by gcc version 3.4 using -O3 optimization.

While equivalence of FastLDA to LDA is guaranteed by construction, we performed additional tests to verify that our implementation of FastLDA produced results identical to LDA. In the first test we verified that the implementations of LDA and FastLDA drew samples for z_{ij} from the same distribution. To do this, we kept the assignment variables $\mathbf{z}^{\neg ij}$ constant, and sampled a value for, but did not update, z_{ij}. We did this for 1000 iterations and then verified that the histograms of sampled values were the same between LDA and FastLDA. In the second test, using 100 runs on the NIPS corpus, we confirmed that the perplexity for FastLDA was the same as the perplexity for LDA. This double checking affirmed that FastLDA was indeed correctly coded, and therefore timings produced by FastLDA would be valid and comparable to those produced by LDA.

6.1 Measuring Speedup

For the NIPS and Enron data sets, we timed the execution of LDA and FastLDA for 500 iterations of the Gibbs sampler, i.e., 500 sweeps through the entire corpus. This number of iterations was chosen to be large enough to guarantee that burn-in had occurred, and that samples were being drawn from the posterior distribution. This number of iterations also meant that the measurement of execution time was relatively accurate. Each separate case was run twice using different random initializations to estimate variation in timings. These repeat timings of runs showed that the variation in CPU time for any given run was approximately 1%. We do not show error bars in the figures because this variation in timings was negligible.

For the NYTimes and PubMed data sets, we used a slightly different method to measure speedup, because of the considerably larger size of these data sets compared to NIPS and Enron. Instead of measuring CPU time for an entire run, we measured CPU time per iteration. To produce an accurate estimate, we estimated this per-iteration CPU time by timing 20 consecutive iterations. FastLDA was initialized with

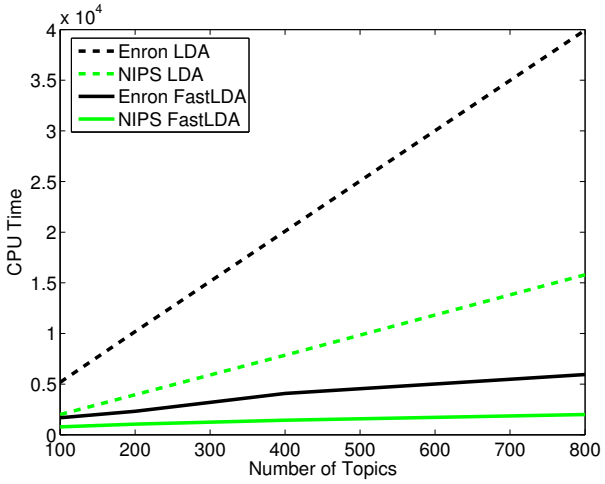

Figure 5: CPU time for LDA and FastLDA, as a function of the number of topics K for NIPS and Enron data sets.

parameters from an already burned-in model for NYTimes and PubMed. The $K = 2000$ and $K = 4000$ topic models of PubMed were computed on a supercomputer using 256 processors using the parallel AD-LDA algorithm [14].

Because of its large size, PubMed presented further challenges. Running LDA or FastLDA on PubMed with $K = 2000$ and $K = 4000$ topics requires on the order of 100-200 GB of memory, well beyond the limit of typical workstations. Therefore, we estimated speedup on PubMed using a 250,000 document subset of the entire collection, but running LDA and FastLDA initialized with the parameters from the aforementioned burned-in model that was computed using the entire PubMed corpus of 8.2 million documents. While the measured CPU times were for a subset of PubMed, the speedup results we show hold for FastLDA running on the entire collection, since the topics used were those learned for the entire 8.2 million documents.

6.2 Experimental Setup

For all experiments, we set Dirichlet parameter $\beta = 0.01$ (prior on word given topic) and Dirichlet parameter $\alpha = 2/K$ (prior on topic given document), except where noted. Setting α this way ensured that the total added probability mass was constant. These settings of Dirichlet hyperparameters are typical for those used for topic modeling these data sets, and similar to values that one may learn by sampling or optimization. We also investigated the sensitivity of speedup to the Dirichlet parameter α.

The bound presented in Section 4.1 was expressed in the more general form of Hölder's inequality. For all experiments, except where noted, we used the general form of Hölder's inequality with $p = 2$, $q = 2$, $r = \infty$. Section 6.4 examines the effect of different choices of p, q and r. As is shown and discussed in that section, the choice of $p = 2$, $q = 2$, $r = \infty$ is the better one to use in practice.

6.3 Speedup Results

CPU time for LDA increases linearly with the number of topics K (Figure 5), an expected experimental result given

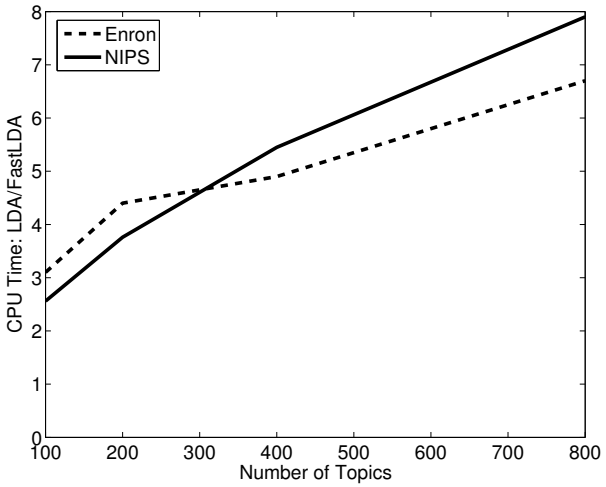

Figure 6: Speedup factor over LDA, as a function of the number of topics K for NIPS and Enron data sets.

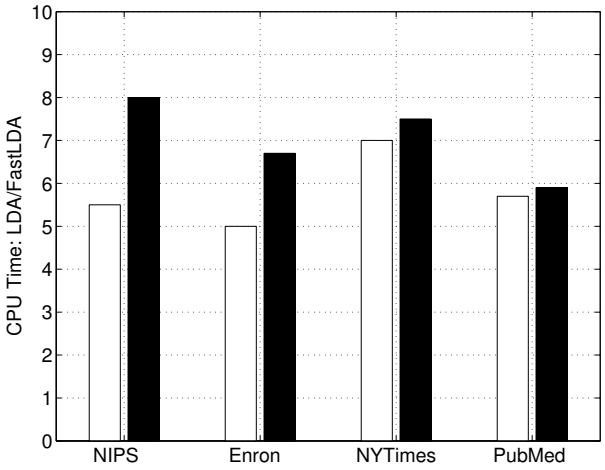

Figure 7: Speedup of FastLDA over LDA for the four corpora. Bars show: NIPS $K = 400, 800$, Enron $K = 400, 800$, NYTimes $K = 800, 1600$ and PubMed $K = 2000, 4000$. $\alpha = 2/K$ for all runs.

the **for** loop over K topics in algorithm 3.1. The CPU time for FastLDA is significantly less than the CPU time for LDA for both the NIPS and Enron data sets. Furthermore, we see that FastLDA CPU time increases slower than linearly with increasing topics, indicating a greater speedup with increasing number of topics. Figure 6 shows the same results, this time displayed as speedup, i.e. the y-axis is the CPU Time for LDA divided by the CPU Time for FastLDA. For these data sets, we see speedups between $3\times$ and $8\times$, with speedup increasing with higher number of topics. The fraction of topics FastLDA must consider on average per sample is related to the fraction of topics used by documents on average. This in turn depends on other factors such as the latent structure of the data and the Dirichlet parameters α and β. Consequently, in experiments using a reasonable number of topics the speedup of FastLDA increases as the number of topics increase.

Our summary of the speedup results for all four data sets are shown in Figure 7. Each pair of bars shows the speedup of FastLDA relative to LDA, for two different topic settings per corpus. The number of topics are: NIPS $K = 400, 800$, Enron $K = 400, 800$, NYTimes $K = 800, 1600$ and PubMed $K = 2000, 4000$, with the speedup for the larger number of topics shown in the black bar on the right of each pair. We see a range of $5\times$ to $8\times$ speedup for this wide variety of data sets and topic settings. On the two huge data sets, NYTimes and PubMed, FastLDA shows a consistent $5.7\times$ to $7.5\times$ speedup. This speedup is non-trivial for these larger computations. For example, FastLDA reduces the computation time for NYTimes from over one week to less than one day, for $K = 1600$ topics.

The speedup is relatively insensitive to the number of documents in a corpus, assuming that as the number of documents increases the content stays consistent. Figure 8 shows the speedup for the NIPS collection versus number of topics. The three different curves respectively show the entire NIPS collection of $D = 1500$ documents, and two sub-collections made up of $D = 800$ and $D = 400$ documents (where the sub-collections are made up from random subsamples of the full 1500-document collection). The figure shows that speedup is not significantly effected by corpus size, but predominantly dependent on number of topics, as observed earlier. The choice of Dirichlet parameter α more directly affects speedup, as shown in Figure 9. This is because using a larger Dirichlet parameter smooths the distribution of topics within a document, and gives higher probability to topics that may be irrelevant to any particular document. The resulting effect of increasing α is that FastLDA needs to visit and compute more topics before drawing a sample. Conversely, setting α to a low value further concentrates the topic probabilities, and produces more than an $18\times$ speedup on the NIPS corpus using $K = 800$ topics.

6.4 Choice of Bound

We experimented with two different bounds for Z, corresponding to particular choices of p, q and r in Hölder's inequality. The first was setting $p = q = 2$ and $r = \infty$, i.e. using $\min_k N_k$. We also used the symmetric setting of $p = q = r = 3$. In all comparisons so far we found the $p = q = r = 3$ setting resulted in slower execution times than $p = q = 2$ and $r = \infty$.

Figure 10 shows given two choices for p, q, r, how quickly the bound Z_k converges to Z as a function of the number of topics evaluated. This plot shows the average ratio Z_k/Z for the k_{th} topic evaluated before drawing a sample. The faster Z_k/Z converges to 1, the fewer calculations are needed on average. Using the NIPS data set, four runs are compared using the two different choices of p, q, r and $K = 400$ versus $K = 4000$ topics. Here as well, we see that the bound produced by $p = q = r = 3$ tends to give much higher ratios on average, forcing the algorithm to evaluate many topics before the probabilities approach their true values.

7. CONCLUSIONS

Topic modeling of text collections is rapidly gaining importance for a wide variety of applications including information retrieval and automatic subject indexing. Among

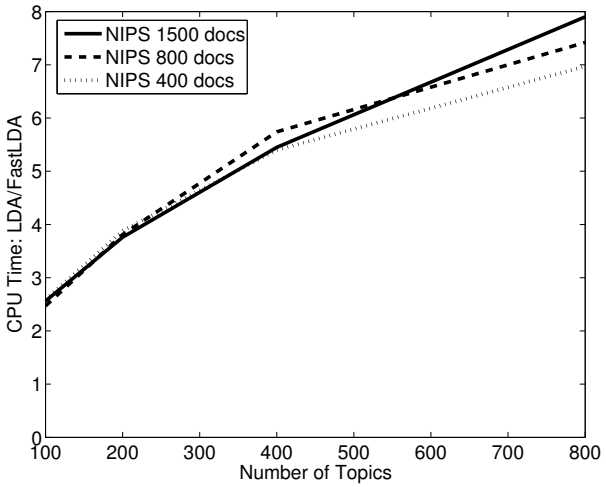

Figure 8: Speedup over LDA as a function of the number of topics K, for different collection size (D) versions of the NIPS data set.

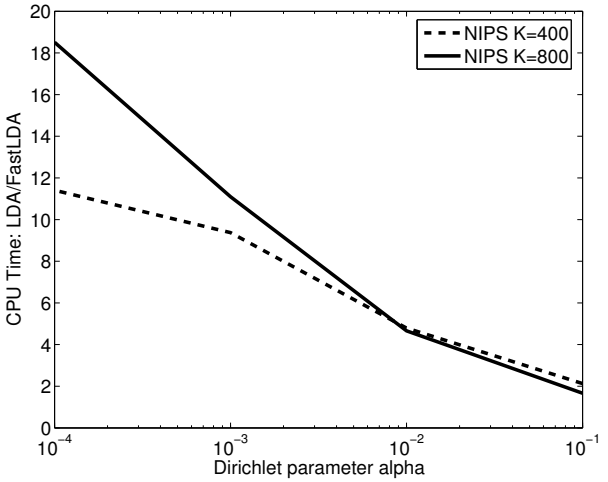

Figure 9: Speedup over LDA as a function of the Dirichlet parameter α, using the NIPS data set. Decreasing α encourages sparse, concentrated topic probabilities, increasing the speed of our method.

these, Latent Dirichlet Allocation and Gibbs sampling are perhaps the most widely used model and inference algorithm. However, as the size of both the individual documents and the total corpus grows, it becomes increasingly important to be as computationally efficient as possible.

In this paper, we have described a method for increasing the speed of LDA Gibbs sampling while providing exactly equivalent samples, thus retaining all the optimality guarantees associated with the original LDA algorithm. By organizing the computations in a better way, and constructing an adaptive upper bound on the true normalization constant, we can take advantage of the sparse and predictable nature of the topic association probabilities. This ensures both rapid improvement of the adaptive bound and that high-probability topics are visited early, allowing the sampling process to stop as soon as the sample value is located. We find that this process gives a 3–8× factor of improvement in speed, with this factor increasing with greater numbers of topics. These speed-ups are in addition to improvements gained through other means (such as the parallelization technique of Newman et al. [14]), and can be used in conjunction to make topic modeling of extremely large corpora practical.

The general method we describe, to avoid having to consider all possibilities when sampling from a discrete distribution, should be applicable to other models as well. In particular we expect the method to work well for other varieties of topic model, such as the Hierarchical Dirichlet Process [18] and Pachinko allocation [11], which have a sampling step similar to LDA. However, how to maintain an efficient upper bound for Z, the accuracy of the bound, and an efficient–to–maintain ordering in which to consider topics, remain model specific problems.

Additionally, our bound–and–refine algorithm used one particular class of bounds based on Hölder's inequality, and a refinement schedule based on the document statistics. Whether other choices of bounds or schedules could further improve the performance of FastLDA is an open question.

8. ACKNOWLEDGMENTS

We thank John Winn for his discussions on the method. Computations were performed at San Diego Supercomputing Center using an MRAC Allocation. This material is based upon work supported by the National Science Foundation under Grant No. 0447903 and No. 0535278 and by ONR under Grant No. 00014-06-1-073. The work of PS, AA, and DN is supported in part by the National Science Foundation under Award Number IIS-0083489 and also by a National Science Foundation Graduate Fellowship (AA) and by a Google Research Award (PS).

9. REFERENCES

[1] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. Workshop on High-Performance Data Mining at IPPS/SPDP, Mar. 1998.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, Sept. 1975.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] W. Buntine, J. Löfström, J. Perkiö, S. Perttu, V. Poroshin, T. S. H. Tirri, and V. T. A. Tuominen. A scalable topic-based open source search engine. In *Proceedings of the IEEE/WIC/ACM Conference on Web Intelligence (WI 2004)*, pages 228–234, 2004.

[5] C. Chemudugunta, P. Smyth, , and M. Steyvers. Modeling general and specific aspects of documents with a probabilistic topic model. In *Neural Information Processing Systems 19*. MIT Press, 2006.

[6] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proc Natl Acad Sci U S A*, 101 Suppl 1:5228–5235, April 2004.

[7] G. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1989.

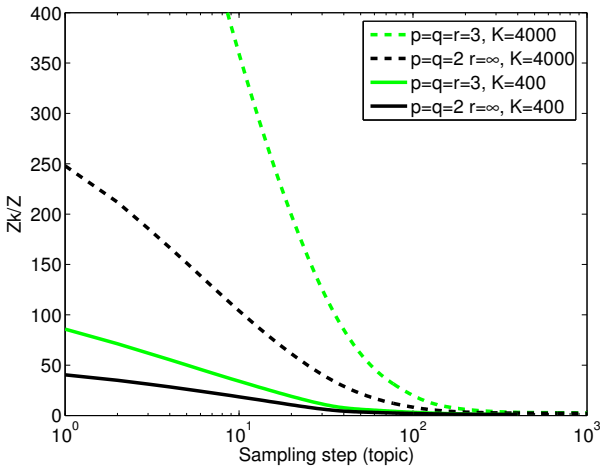

Figure 10: The average accuracy of the bound Zk/Z, as a function of the number of topics visited, for two possible choices of (p, q, r). The norm choices $(2, 2, \infty)$ appears to be considerably tighter, on average, than the symmetric choice $(3, 3, 3)$.

[8] A. T. Ihler, E. B. Sudderth, W. T. Freeman, and A. S. Willsky. Efficient multiscale sampling from products of Gaussian mixtures. In *Proc. Neural Information Processing Systems (NIPS) 17*, Dec. 2003.

[9] K. Kurihara and M. Welling. Bayesian k-means as a maximization-expectation. In *Neural Computation*, accepted.

[10] K. Kurihara, M. Welling, and N. Vlassis. Accelerated variational dirichlet process mixtures. In *NIPS*, volume 19, 2006.

[11] W. Li and A. McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584, 2006.

[12] D. Mimno and A. McCallum. Organizing the OCA: learning faceted subjects from a library of digital books. In *JCDL '07: Proceedings of the 2007 conference on Digital libraries*, pages 376–385, New York, NY, USA, 2007. ACM.

[13] A. Moore. Very fast EM-based mixture model clustering using multiresolution kd-trees. In *NIPS*, volume 10, 1998.

[14] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed inference for latent Dirichlet allocation. In *Proc. Neural Information Processing Systems (NIPS) 22*, dec 2007.

[15] D. Newman, K. Hagedorn, C. Chemudugunta, and P. Smyth. Subject metadata enrichment using statistical topic models. In *JCDL '07: Proceedings of the 2007 conference on Digital libraries*, pages 366–375, New York, NY, USA, 2007. ACM.

[16] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. of the 5th Int'l Conf. on Knowledge Discovery in Databases*, pages 277–281, 1999.

[17] D. Pelleg and A. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *ICML*, volume 17, pages 727–734, 2000.

[18] Y. Teh, M. Jordan, M. Beal, and D. Blei. Hierarchical Dirichlet processes. In *NIPS*, volume 17, 2004.

[19] X. Wei and W. B. Croft. Lda-based document models for ad-hoc retrieval. In *SIGIR*, pages 178–185, 2006.

APPENDIX

We describe here the parameter specifications used to run our experiments. We have made our LDA and FastLDA code publicly available at http:// www.ics.uci.edu/ ∼iporteou/ fastlda and the four data sets at the UCI Machine Learning Repository, http:// archive.ics.uci.edu/ ml/ machine-learning-databases/ bag-of-words/.

For all NIPS and Enron runs:

1. $\alpha = 2/K$, $\beta = 0.01$, except for experiments versus α
2. CPU times measured over 500 iterations, including burn-in
3. Speedup computed over entire run
4. Runs repeated with different random initializations

For NYTimes and PubMed runs:

1. $\alpha = 2/K$, $\beta = 0.01$
2. CPU times measured over 20 iterations
3. Speedup computed on a per-iteration basis
4. LDA and FastLDA runs initialized with model parameters from already burned-in run (1000 iterations, $\alpha = 2/K$,$\beta = 0.01$)