Spectral Latent Dirichlet Allocation model on Spark

Furong Huang[*1], Jencir Lee[†2], and Anima Anandkumar[‡2]

[1]Microsoft Research NYC
[2]University of California Irvine

March 1, 2017

Abstract

Learning latent variable mixture models in high-dimension is applicable in numerous domains where low rank factorization is desired. Popular likelihood based methods optimize over a non-convex likelihood which is computationally challenging to obtain optimal solution and therefore is usually not guaranteed to converge to a global or even local optima without additional assumptions. We propose a framework to overcome the problem of unscalable and non-convex likelihood by taking the power of inverse method of moments. By matching the moments, the problem of learning latent variable mixture models is reduced to a low-rank tensor decomposition problem. This framework incorporates dimensionality reduction. We show that it is highly scalable and implemented a distributed version on Apache Spark.

1 Introduction

Latent varaible graphical models are versatile to model complicated or mixed data. The popular excahngeable model and multi-view model fall under the category of single latent variable models. The simplest latent varaible graphical model is the exchangeble model as in Fig 1(left), where there exists only one latent variable and conditional independence of observed nodes under the latent variable model is posit. Additionally the conditional probabilities

[*]furongh@microsoft.com
[†]jencirl@uci.edu
[‡]a.anandkumar@uci.edu

$P(x_i|h)$ are the same for all i under the exchangeble model. The simple exchangeble model is extended to model data with more structure. Hidden markov model (HMM) as in Fig 1(middle) is an extension to multiple latent variables but maintains a flat structure. The hierarchical latent variable models such as latent tree as in Fig 1(right) are flexible to model data with complicated hiearchical structure. The multi latent variable (either hierarchical or not) models can be viewed as compositions of local single latent variable models.

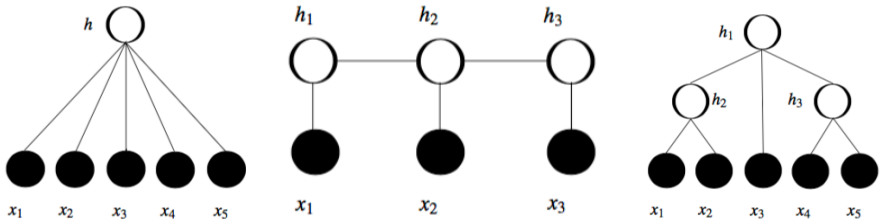

Figure 1: Examples of mixture models

A prior on the latent variable model incorporates our prior knowledge on the latent varaible. With the prior knowledge, we estimate the model parameter by matching lower order population moments (computed under model parameters) with the empirical moments. First three orders of moments are sufficient statistics to learning Latent Dirichlet Allocation (LDA) model [2], which is an admixture model with a Dirichelt distributed latent variable and has been versatile in numerous fields since its introduction over a decade ago. The core of our method is *tensor decomposition*, and the learned eigenvectors correspond to the desired model parameters of the latent varaible model. In the context of topic model using LDA, it is computationally challenging to learn large scale topic models due to high dimensional data with large vocabulary size and large number of documents. Using the inverse method of moments, learning LDA model reduces to CANDECOMP/PARAFAC (CP) decomposition of a low-dimensional tensor. CP tensor decomposition has been studied in the literature [9, 1], but no distributed implementation exists.

Dimensionality reduction is the key for the scalability of our method. Decomposing the second order moment, we find a subspace of the model parameters. We then we project raw data onto this subspace, and implicitly compute the third order empirical moment of the projected data which is a low-dimensional tensor T. We finally find the model parameters via CP tensor decomposition of T. There can be various methods for CP tensor

decomposition: Kolda et al. proposed Alternating Least Square (ALS) [9]; Anandkumar et al. proposed Power Method (PM) [1]; and Ge et al. introduced stochastic gradient descent with noise (SGDwN) [4]. The latter is guaranteed to converge to a global optima. As a starting point, we will design a distributed framework for ALS and implement our proposed distributed ALS on Apache Spark, as it is an in-memory platform which is perfect for iterative updates.

Our tensor decomposition framework is a unified one which is easily amendable to other mixture models including the multi-view model, hidden Markov model and hierarchical mixture models. Our method is different from variational inference [2, 8] which is susceptible to local optima [12] and Markov chain Monte Carlo [5, 13, 15, 14] which is subject to exponential mixing time.

2 Overview of the Inverse Method of Moments

Fig 2 exhibts the overview of our unsupervised learnign of mixed data using tensor decomposition. First we model the data using an appropriate admixture model and compute the population moments (a function of unknown model parameters) corresponding to the admixture model; we then estimate the empirically moments of the data from the training data (Note that third order moments are sufficient statistics for a wide class of linear mixture models); therefore moment matching between population moments and empirical moments is reduced to decomposition of the third order empirical moments (which uniquely identifies the model parameters); and the model parameters we learned are used in test time for inference and prediction tasks.

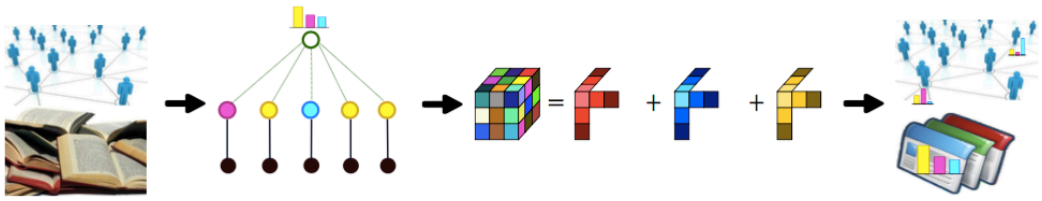

Figure 2: Overall framework of the spectral tensor method

Before we introduce formally the data moments of LDA, let us first re-

view the generative LDA model. LDA models is a bag of words model where word orders are ignored. Each of n documents is considered as a mixture of words generated over k latent topics. In each topic i, a *multinomial distribution* β_i over V words/tokens is defined, and we call this *topic-word conditional probability* of component i. Since there are k topics in total, we have k such components (i.e. k such topic-word conditional probability vectors). We stack them in a matrix as $\beta = [\beta_1, \ldots, \beta_k] \in \mathbf{R}^{V \times k}$ and call β the *topic-word matrix*.

In each document, a topic proportion vector is first drawn from a Dirichlet distributed prior $\mathrm{Dir}(\alpha)$ where $\alpha = [\alpha_1, \ldots, \alpha_k] \in \mathbf{R}^k$ and $\alpha_0 = \sum_{i=1}^{k} \alpha_i$. The topic proportion vector is also known as the mixing distribution of latent topics per document. To generate a word, a topic i is first determined according to the topic proportion for this document. Then a word is drawn from the i-th column of the topic-word matrix. Our algorithm takes α_0 as a user defined parameter, and will estimate the topic-word matrix β and the Dirichlet prior α.

2.1 Data moments

Theorem 3.5 in [1] first defines the shifted moments of a rank-k LDA model from the word counts,

$$M_1 = \mathbf{E}[x_1] \tag{1}$$

$$M_2 = \mathbf{E}[x_1 \otimes x_2] - \frac{\alpha_0}{\alpha_0 + 1} M_1 \otimes M_1 \tag{2}$$

$$\begin{aligned} M_3 =& \mathbf{E}[x_1 \otimes x_2 \otimes x_3] \\ & - \frac{\alpha_0}{\alpha_0 + 2} \left(\mathbf{E}[x_1 \otimes x_2 \otimes M_1] + \mathbf{E}[x_1 \otimes M_1 \otimes x_2] + \mathbf{E}[M_1 \otimes x_1 \otimes x_2] \right) \\ & + \frac{2\alpha_0^2}{(\alpha_0 + 1)(\alpha_0 + 2)} M_1 \otimes M_1 \otimes M_1 \end{aligned} \tag{3}$$

where $\mathbf{E}[x_1]$ is the probability of any word in the vocabulary by the model specification, $\mathbf{E}[x_1 \otimes x_2 \otimes x_3]$ any word pair, $\mathbf{E}[x_1 \otimes x_2 \otimes x_3]$ any word triplet.

2.2 From data moments to model parameters

Under Dirichlet prior the population moments in Equation (1) (2) (3) are related to model parameters α_i, β_i, $1 \leq i \leq k$ through the following

$$M_2 = \frac{1}{\alpha_0 (\alpha_0 + 1)} \sum_{i=1}^{k} \alpha_i \beta_i \otimes \beta_i \tag{4}$$

$$M_3 = \frac{2}{\alpha_0 (\alpha_0 + 1) (\alpha_0 + 2)} \sum_{i=1}^{k} \alpha_i \beta_i \otimes \beta_i \otimes \beta_i \tag{5}$$

Therefore a unique decomposition of empirical estimations of M_3 recovers the model parameters α_i and β_i for $1 \leq i \leq k$.

3 CP Decomposition with Whitening

In general, tensor CP decomposition is not necessarily unique. In fact, a tensor CP decomposition may not exist. However, according to the Theorem 5.1 in [1], a tensor that has a orthogonal decomposition admits a unique decomposition. If a symmetric tensor T is orthogonal, i.e. $T = \sum_i \lambda_i \nu_i \otimes \nu_i \otimes \nu_i$, $\langle \nu_i, \nu_j \rangle = 0$, $\forall i \neq j$, the decomposition is unique.

Our strategy is therefore: first orthogonalize the tensor in Equation (5) then decompose it. We call the orthogonalization step "whitening".

3.1 Whitening with Dimension Reduction

We slightly transform Equation (4) (5) for convenience of notation

$$\tilde{M_2} = \alpha_0 (\alpha_0 + 1) M_2 = \sum_{i=1}^{k} \alpha_i \beta_i \otimes \beta_i, \tag{6}$$

$$\tilde{M_3} = \frac{\alpha_0 (\alpha_0 + 1) (\alpha_0 + 2)}{2} M_3 = \sum_{i=1}^{k} \alpha_i \beta_i \otimes \beta_i \otimes \beta_i, \tag{7}$$

which we could write as

$$\tilde{M_2} = \sum_{i=1}^{k} \left(\alpha_i^{1/2} \beta_i \right) \otimes \left(\alpha_i^{1/2} \beta_i \right), \tag{8}$$

$$\tilde{M_3} = \sum_{i=1}^{k} \alpha_i^{-1/2} \left(\alpha_i^{1/2} \beta_i \right) \otimes \left(\alpha_i^{1/2} \beta_i \right) \otimes \left(\alpha_i^{1/2} \beta_i \right). \tag{9}$$

We perform an eigendecomposition on $\tilde{M}_2$ and find $\tilde{M}_2 = U\Sigma U^\top$. As $\tilde{M}_2$ has at most rank k, it follows that $U \in \mathbf{R}^{V\times k}$, $\Sigma \in \mathbf{R}^{k\times k}$. If we denote the whitening matrix

$$W = U\Sigma^{-1/2} \in \mathbf{R}^{V\times k}, \tag{10}$$

we have

$$\begin{aligned} W^\top \tilde{M}_2 W =& \sum_{i=1}^{k} \left(\alpha_i^{1/2} W^\top \beta_i\right) \otimes \left(\alpha_i^{1/2} W^\top \beta_i\right) \\ =& \Sigma^{-1/2} U^\top U \Sigma U^\top U \Sigma^{-1/2} = I \end{aligned} \tag{11}$$

therefore the vectors $\left\{\alpha_i^{1/2} W^\top \beta_i\right\}$, $1 \le i \le k$ are orthogonal now.

3.2 Orthogonal Tensor Decomposition

We proceed to whiten $\tilde{M}_3$ with W,

$$\tilde{M}_3(W,W,W) = \sum_{i=1}^{k} \alpha_i^{-1/2} \left(\alpha_i^{1/2} W^\top \beta_i\right) \otimes \left(\alpha_i^{1/2} W^\top \beta_i\right) \otimes \left(\alpha_i^{1/2} W^\top \beta_i\right). \tag{12}$$

If the orthogonal decomposition gives

$$\tilde{M}_3(W,W,W) = \sum_{i=1}^{k} \lambda_i \nu_i \otimes \nu_i \otimes \nu_i \tag{13}$$

it is straightforward to recover

$$\alpha_i = \lambda_i^{-2} \tag{14}$$

$$\beta_i = U\Sigma^{1/2} \lambda_i \nu_i \tag{15}$$

for $1 \le i \le k$.

Note that before whitening, $\tilde{M}_3 \in \mathbf{R}^{V\times V\times V}$; after whitening we significantly brought down the dimension, $\tilde{M}_3(W,W,W) \in \mathbf{R}^{k\times k\times k}$ as $W \in \mathbf{R}^{V\times k}$.

4 Implementation

We implemented the Spectral LDA algorithm on Apache Spark. The repository is at `https://github.com/FurongHuang/SpectralLDA-TensorSpark`.

As of October 2016, the code is implemented against Apache Spark v2.0.0, compiled with the support for native BLAS/CBLAS/LAPACK. Apache

Spark is a distributed framework that provides an easy means to distribute data and functional programming. In terms of linear algebra routines, we rely on the Breeze package, which is the internal linear algebra engine for Apache Spark as well.

In this section we discuss the design choices and implementation details of the major components of the algorithm, how we achieve efficiency on Apache Spark, and computational complexity analysis.

4.1 Overview of the Architecture

Apache Spark In order to understand how the components of our algorithm are distributed, we first take a look at the node architecture of an Apache Spark cluster in Fig 3 [10].

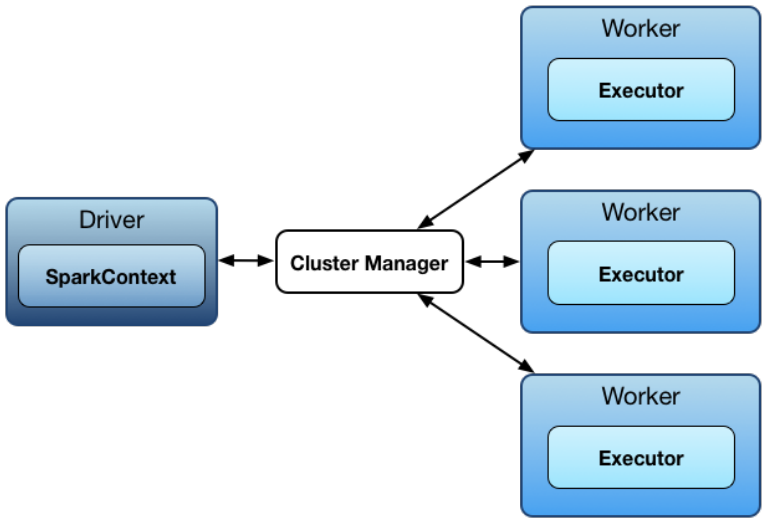

Figure 3: Node architecture of an Apache Spark cluster

The cluster consists of Cluster Manager and Worker. The Cluster Manager receives Code Execution and Resource Allocation requests, and it subsequently tries to allocate requested resources on Workers and distribute the code for execution onto the Workers.

The Driver is the "client" in this architecture that sends out requests and collects results. When the Code is deployed in 'client' mode, the Driver usually sits on the Developer's machine and allows interactive use of the cluster; when the Code is deployed in 'cluster' mode, the Driver will be

created inside one Worker node and performs the role of client till the end of execution of the Code.

Spectral LDA A global view of the Spectral LDA algorithm, as described in Section 3, is plotted in Fig 4.

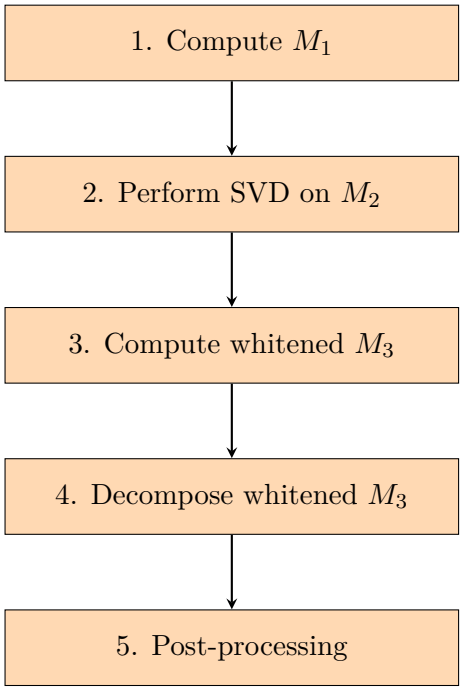

Figure 4: Diagram of the Spectral LDA algorithm

The steps 1-3 are distributed across the cluster as we need to compute the (whitened) moments from the empirical word counts data and each of these steps is *embarrassingly parallel* as we shall see.

The steps 4-5 are evaluated locally (on the Driver node alone) as the data dimension has been scaled down to $O\left(k^3\right)$ or $O(V)$, that a machine today can perfectly handle.

In the rest of this section, each step will be explained and analyzed in more detail.

4.2 Randomized SVD of M_2

$\tilde{M_2}$ is of size $V \times V$, where V is the vocabulary size, which is usually in the order of 100k. In order to accelerate the SVD, we employ the Power

Method [11] as in Algorithm 1.

Algorithm 1 Simultaneous Iteration

function RANDSVD(A,ϵ,k) ▷ $A \in \mathbf{R}^{d\times d}$, error $\epsilon \in (0,1)$, $k \ll d$
  $q \leftarrow \Theta\left(\frac{\log d}{\epsilon}\right)$, $\Pi \leftarrow \mathcal{N}(0,1)^{d\times k}$
  $K \leftarrow A\Pi$
  $K \leftarrow \left(AA^\top\right)^q K$ ▷ $K = \left(AA^\top\right)^q A\Pi$
  Orthogonalize the columns of K to obtain $Q \in \mathbf{R}^{d\times k}$
  $M \leftarrow Q^\top AA^\top Q \in \mathbf{R}^{k\times k}$
  $U_k \leftarrow$ the eigenvectors of M, $S_k \leftarrow$ the eigenvalues of M
  return $Z \leftarrow QU_k$ as top k eigenvectors of A and $S_k^{\frac{1}{2}}$ as top k eigenvalues of A
end function

The computational complexity of Algorithm 1)is $O(k^2 d)$, and it takes $q \sim O(\log d/\epsilon)$ iterations to bound the error with probability 99/100

$$\|A - ZZ^\top A\|_{\text{norm}} \leq (1+\epsilon)\|A - A_k\|_{\text{norm}}, \tag{16}$$

where A_k is the projection of A onto the space spanned by its top k eigenvectors, $\|\cdot\|_{\text{norm}}$ is either the Frobenius norm or Spectral norm. Refer to [6] for the analysis of the convergence properties of the Randomized SVD algorithms.

4.3 Orthogonal Tensor Decomposition of Whitened M_3

With regards to the orthogonal tensor decomposition in Equation (13) [1] proves that the Power Method will recover all the robust eigenvectors of an orthogonal tensor with high probability.

For practical implementation however, we use the Alternating Least-Squares algorithm in [9] to solve for $\Lambda = [\lambda_1, \ldots, \lambda_k] \in \mathbf{R}^k$, $A = [a_1, \cdots, a_k]$, $B = [b_1, \cdots, b_k]$, $C = [c_1, \cdots, c_k] \in \mathbf{R}^{k\times k}$ such that

$$\tilde{M_3}(W,W,W) = \sum_{i=1}^{k} \lambda_i a_i \otimes b_i \otimes c_i. \tag{17}$$

Equivalently, we could write Eq 17 with notion

$$\tilde{M_3}(W,W,W) = [\Lambda; A, B, C]\,. \tag{18}$$

Algorithm 2 Alternative Least-Squares for CP Decomposition

1: **function** ALS(T,ϵ,N,trials) ▷ $T \in \mathbf{R}^{k\times k\times k}$, N: max runs
2: **for** $s \leftarrow$ range(trials) **do**
3: initialize Λ, A, B, C randomly ▷ $\Lambda \in \mathbf{R}^k$,$A, B, C \in \mathbf{R}^{k\times k}$
4: $i \leftarrow 0$
5: **while** $i < N$ and not converged on Λ, A, B, C **do**
6: ▷ $T^{(i)}$ is the mode-i unfolding of T, $T^{(i)} \in \mathbf{R}^{k\times k^2}$
7: $\Lambda, A \leftarrow \texttt{ALSUpdate}(T^{(1)}, B, C)$
8: $\Lambda, B \leftarrow \texttt{ALSUpdate}(T^{(2)}, C, A)$
9: $\Lambda, C \leftarrow \texttt{ALSUpdate}(T^{(3)}, A, B)$
10: $i \leftarrow i + 1$
11: **end while**
12: reconstruction error $e \leftarrow \|T - [\Lambda; A, B, C]\|_2$
13: retain Λ, A, B, C if they achieve the lowest e so far
14: **end for**
15: **return** optimal Λ, A, B, C
16: **end function**
17:
18: **function** ALSUpdate($\mathcal{T}$,B,C) ▷ $\mathcal{T} \in \mathbf{R}^{k\times k^2}$,$B, C \in \mathbf{R}^{k\times k}$
19: $\hat{A} \leftarrow \mathcal{T}(C \odot B)(C^TC * B^TB)^\dagger$ ▷ $\odot$ for Khatri-rao product
20: ▷ $*$ for Hadamard product
21: ▷ $\dagger$ for Moore-Penrose pseudoinverse
22: Normalize $\hat{A}$ column-wise into A, store the norms of the columns into Λ
23: **return** Λ, A
24: **end function**

The algorithm is listed in Algorithm 2. It starts with random initial eigenvalues Λ, eigenvector matrices A, B, C, and update them iteratively. At every iteration, we update the eigenvector matrices one by one. Matrix $T^{(i)}$ is the mode-i unfolding of the tensor T, such that the p-th row of $T^{(i)}$ is the flattened version of the slice

$$T(:, \cdots, :, \underbrace{p}_{i\text{-th dim}}, :, \cdots, :) \in \mathbf{R}^{k\times k},$$

where $p = 1, \ldots, k$, and : denotes that we collect all elements along that particular dimension. Please refer to [9] for more details on tensor notions and the update equation at Line 19. Usually it can converge within 50 iterations and seconds time-wise for our problem.

We also implemented the multi-starts of initial values outside each ALS loop in hope to minimize the decomposition error, following the Power Method algorithm in [1].

We retain the results that minimize the decomposition error over all runs and return Λ, A out of that run as the solution to the orthogonal tensor decomposition.

4.4 Building Whitened M_3 from Empirical Counts

Given the collection of n documents, for document d, $1 \leq d \leq n$, denote its term count vector by $n_d \in \mathbf{R}^V$, it has m_d total tokens, $m_d = \sum_{t=1}^{V} (n_d)_t$.

Denote the contribution of document d to the empirically estimated $\hat{\mathbf{E}}[x_1 \otimes x_2 \otimes x_3]$ by tensor T_d. From [1] [16]

$$(T_d)_{i,j,k} = \frac{1}{n} \frac{1}{m_d (m_d - 1)(m_d - 2)} \begin{cases} (n_d)_i \left[(n_d)_j - 1\right] \left[(n_d)_k - 2\right], & i = j = k; \\ (n_d)_i \left[(n_d)_j - 1\right] (n_d)_k, & i = j, j \neq k; \\ (n_d)_i (n_d)_j \left[(n_d)_k - 1\right], & j = k, i \neq j; \\ \left[(n_d)_i - 1\right] (n_d)_j (n_d)_k, & i = k, i \neq j; \\ (n_d)_i (n_d)_j (n_d)_k, & \text{otherwise.} \end{cases} \tag{19}$$

It is possible to represent the above tensor by equation,

$$T_d = \frac{1}{n} \frac{1}{\binom{m_d}{3} \cdot 3!} \Bigg\{ n_d^{\otimes 3} - \sum_{t=1}^{V} (n_d)_t \left(e_t \otimes e_t \otimes n_d + e_t \otimes n_d \otimes e_t + n_d \otimes e_t \otimes e_t\right) + \sum_{t=1}^{V} 2 (n_d)_t e_t^{\otimes 3} \Bigg\}, \tag{20}$$

where e_t is the basis vector in $\mathbf{R}^V$ with only the t-th element set to one, the rest to zero.

When whitened,

$$T_d(W, W, W) = \frac{1}{n} \frac{1}{\binom{m_d}{3} \cdot 3!} \Bigg\{ p_d^{\otimes 3} - \sum_{t=1}^{V} (n_d)_t \left(w_t \otimes w_t \otimes p_d + w_t \otimes p_d \otimes w_t + p_d \otimes w_t \otimes w_t\right) + \sum_{t=1}^{V} 2 (n_d)_t w_t^{\otimes 3} \Bigg\}, \tag{21}$$

where $p_d = W^\top n_d$, $w_t = W^\top e_t$, i.e. the t-th row of W.

Similarly denote the contribution of document d to $\hat{\mathbf{E}}\left[x_1 \otimes x_2\right]$ by S_d,

$$S_d = \frac{1}{n} \frac{1}{\binom{m_d}{2} \cdot 2!} \left\{ n_d^{\otimes 2} - \sum_{t=1}^{V} \left(n_d\right)_t e_t \otimes e_t \right\}, \tag{22}$$

$$S_d(W, W) = \frac{1}{n} \frac{1}{\binom{m_d}{2} \cdot 2!} \left\{ p_d^{\otimes 2} - \sum_{t=1}^{V} \left(n_d\right)_t w_t \otimes w_t \right\}. \tag{23}$$

Its contribution to the whitened $\hat{\mathbf{E}}\left[x_1 \otimes x_2 \otimes M_1\right]$ is then

$$\left[S_d \otimes M_1\right](W, W, W) = \frac{1}{n} \frac{1}{\binom{m_d}{2} \cdot 2!} \left\{ p_d \otimes p_d \otimes q_d - \sum_{t=1}^{V} \left(n_d\right)_t w_t \otimes w_t \otimes q_d \right\}, \tag{24}$$

where q_d is the whitened M_1, $q_d = W^\top M_1$.

Once we have Equation (21) (24), it is straightforward to piece them together and derive the contribution of document d to the whitened M_3 based on Equation (3). We leave the reader to deduce the long formula.

4.5 Distributed Evaluation of Whitened M_3

Evaluating the whitened M_3 is the matter of summing the contributions of all documents to it. An efficient implementation would be critical to the success of the algorithm.

When implemented on Apache Spark, we exploit the MapReduce capability to synthesize intermediate results and evaluate separately the following terms ($c_{d,t}$ denotes a scalar coefficient specific to document d and term t, $x_{d,t}$ denotes a general vector specific to d and t, $\cdot$ denotes the value associated to term t):

- $p_d^{\otimes 3}$, $p_d \otimes p_d \otimes q_d$, $p_d \otimes q_d \otimes p_d$, $q_d \otimes p_d \otimes p_d$: we simply map out p_d per document, compute all these tensor product parallelly then sum over d;
- $\sum_{t=1}^{V} c_{d,t} \left(w_t \otimes w_t \otimes x_{d,t} + w_t \otimes x_{d,t} \otimes w_t + x_{d,t} \otimes w_t \otimes w_t\right)$: we flat map out all $(t, c_{d,t} x_{d,t})$ per document, reduce by the key t to get $(t, \sum_d c_{d,t} x_{d,t})$ parallelly, finally compute $w_t \otimes w_t \otimes \cdot + w_t \otimes \cdot \otimes w_t + \cdot \otimes w_t \otimes w_t$ parallelly and sum over all t;

- $\sum_{t=1}^{V} c_{d,t} w_t^{\otimes 3}$: we flat map out all $(t, c_{d,t})$ per document, reduce by the key t to get $(t, \sum_d c_{d,t})$ parallelly, finally compute $\cdot\, w_t^{\otimes 3}$ parallelly and sum over all t.

In a stark contrast to other approaches to LDA, the workhorse function is *nothing but the tensor product*, which only takes two lines to implement.

Algorithm 3 Third-Order Tensor Product

```
function TensorProduct3(x, y, z)          ▷ x ∈ R^d, y ∈ R^d, z ∈ R^d
    p ← yz^⊤
    return x(flatten(p))^⊤
end function
```

When Apache Spark is compiled with support for native BLAS, the above function could be evaluated at FORTRAN-level efficiency, despite being run on a Java-based platform.

4.6 ℓ_1-Simplex Projection of the Word Distribution per Topic

The Spectral LDA algorithm does not enforce that the columns β_i are simplex (i.e., sum up to one), $1 \leq i \leq k$. As a post-processing step, we project the solution of Equation (15) into $l1$-simplex using [3].

We observe that for a given solution of Equation (17) and a fixed i, $1 \leq i \leq k$, if we reverse the signs of any two of $\{a_i, b_i, c_i\}$, the equation still holds. This means that for any i, $1 \leq i \leq k$, we need to do projection of both a_i and $-a_i$ in the solution A to Equation (17), and decide to retain one of the two. The question is how to tell which is the bette one.

We notice that the algorithm in [3] would give the same result to any input vector w subject to a parallel shift. We thus run $\text{proj}\,(w - \min(w))$ and $\text{proj}\,((-w) - \min(-w))$. The algorithm would compute the shift θ (a Lagrange multiplier used in [3]) for either input. As $w - \min(w)$ and $(-w) - \min(-w)$ have the same maximums and minimums, only that the shape of one is the mirror to the other, we could compare the shift θ for both and retain the projection with smaller θ.

4.7 Complexity Analysis

We analyze the computational complexity of the Spectral LDA algorithm and compare it with the Online Variational Inference [7].

As the word count vector for each document is usually sparse relative to $\mathbf{R}^V$, for the convenience of analysis of matrix calculations, we suppose that

averagely each document has m unique words, and there are n documents in the collection. We have naturally that $nnz(\text{all word count vectors}) \approx mn$.

For the Online Variational Inference algorithm, we denote the average number of iterations in the inner loop to get the variational priors ϕ_{dwk}, γ_{dk} to converge by h — we cannot bound h in priori, which depends on the input data. Furthermore we suppose that it takes $O(1)$ to compute the exp and digamma functions.

We list the computational complexities in Table 1 2 for both algorithms.

Step	Complexity
Compute M_1	$O(mn)$
Perform Randomized SVD on M_2	$O\left(mnk + k^2V\right)$
Compute Whitened M_3	$O\left(mnk + k^3n\right)$
Decompose Whitened M_3	$O\left(k^3\right)$
Post-processing	$O\left(k^2V\right)$
Overall	$O\left(mnk + k^2V + k^3n\right)$

Table 1: Computational complexity by step of the Spectral LDA algorithm

Update to Parameter	Complexity
Topic Assignment ϕ_{dwk}	$O(hmnk + kV)$
Prior of Topic Distribution γ_{dk}	$O(hmnk)$
Prior of Word Distribution λ_{kw}	$O(mnk)$
Overall	$O(hmnk + kV)$

Table 2: Computational complexity of the updates to each variational parameter in the Online Variational Inference algorithm

For applications on large collection of documents, $n \gg V$. For the Spectral LDA algorithm, the complexity could simplify to $O\left(mnk + k^3n\right)$, which corresponds to that of the step of computing whitened M_3. For the Online Variational Inference algorithm, the complexity simplifies to $O(hmnk)$.

5 Experiments

We ran experiments for log-perplexity and scalability on AWS EC2. We set up a cluster of `c4.8xlarge` instances running Spark v2.0.0. Each instance dedicates all 36 vCPUs and 57G memory to the cluster.

Every instance has a copy of the vectorized document data on its local EBS storage so that we didn't have to set up an HDFS cluster for the

Instance	vCPUs	Memory
`x4.8xlarge`	36	60

Table 3: AWS EC2 Instance

distributed file service. The Spark cluster is launched in the 'Stand-alone' mode so that the Spark Master manages resource allocation alone.

We're going to report the log-perplexities, compare them vs the Online Variational Inference algorithm, as well as the runtime when scaling up the cluster.

5.1 Datasets

We ran experiments on the PubMed and Wikipedia datasets. The PubMed dataset is hosted by UCI Machine Learning Repository, the Wikipedia set is extracted from the daily dump of English Wikipedia dated 2016-06-01. As the Wikipedia set contains many short documents which serve as reference to other articles (the median document length is only 6), we only keep those with at least 100 words. Table 4 summarizes the characteristics of the datasets.

Dataset	Articles	Vocab.	Document Length						
			mean	stdev	5%	25%	50%	75%	95%
PubMed	8.2 mil	140k	89	38	35	63	88	119	167
Wikipedia (all)	16.4 mil	25k	99	494	1	2	6	100	112862
Wikipedia (filt.)	3.1 mil	25k	482	1062	111	155	258	493	1476

Table 4: Characteristics of the datasets. We detail the the number of articles, vocabulary size, and statistics of document length (the sum of the word count vector for every document). The statistics are mean, standard deviation, 5%-, 25%-, 50%-, 75%-, 95%-quantiles. For Wikipedia, we list the case of full dataset and that with only documents with at least 100 words. We see that the full Wikipedia dataset has a majority of very short documents.

5.2 Log-perplexity

We evaluate the log-perplexity using the same variational approach as in [7]. It essentially computes the lower bound of the log-likelihood and therefore an upper bound to the log-perplexity.

Methodology We randomly split the dataset into 90%/10% train/test sets. As the log-perplexity calculation is orders longer than the model fitting itself, we ran on 3 independently random splits and report the average log-perplexity. We allow the Spectral LDA algorithm runs to the end, and let the Online Variational Inference algorithm run within the same length of time.

Parameters The number of topics $k = 20$. The total prior for topic distribution $\alpha_0 = \sum_{i=1}^{k} \alpha_i$ takes values 1.0 or 5.0.

Additionally, for the Spectral LDA algorithm, we set the tolerance parameter for checking the convergence of ALS to be 10^{-9}, that is, if the dot product of two eigenvectors in successive steps is above $1 - 10^{-9}$ they're considered to have converged.

For the Online Variational Inference, We use uniform prior for both the topic distribution $\{\alpha_i\}$, $1 \leq i \leq k$ and the word distribution η. In the tests we vary $\eta = 10^{-5}, 1.0, 10.0$.

Smoothing for log-perplexity For the Spectral LDA algorithm, as it is possible that certain elements in the fitted β are zero, we augment the fitted α, β with one dummy topic that corresponds to the average word distribution of the training set. For the Online Variational Inference, there's no need for smoothing or augmenting, as the latent topic-word distributions are guaranteed to be all positive.

Results The results are plotted in Fig 5. The log-perplexity of the Spectral LDA algorithm is in par with the Online Variational Inference with $\eta = 1.0$. If we run for higher k and wider range of α_0 we would have similar pattern.

If now we let the Online Variational Inference run more iterations, we list in Table 5 the running times and the log-perplexities. The Online Variational Inference achieves slightly lower log-perplexity for much longer runtime.

5.3 Scalability

On the same PubMed datast, we plot in Fig 6 the total runtime (in seconds) when the number of Worker nodes increases from 4, 8, 16 to 32. We see that the Spectral LDA algorithm is linearly scalable.

5.4 Choice of q for RandSVD

We wanted to see the relation between the choice of q (iteration steps) in RandSVD (Alg 1) and the computed log-perplexity. We discovered that the

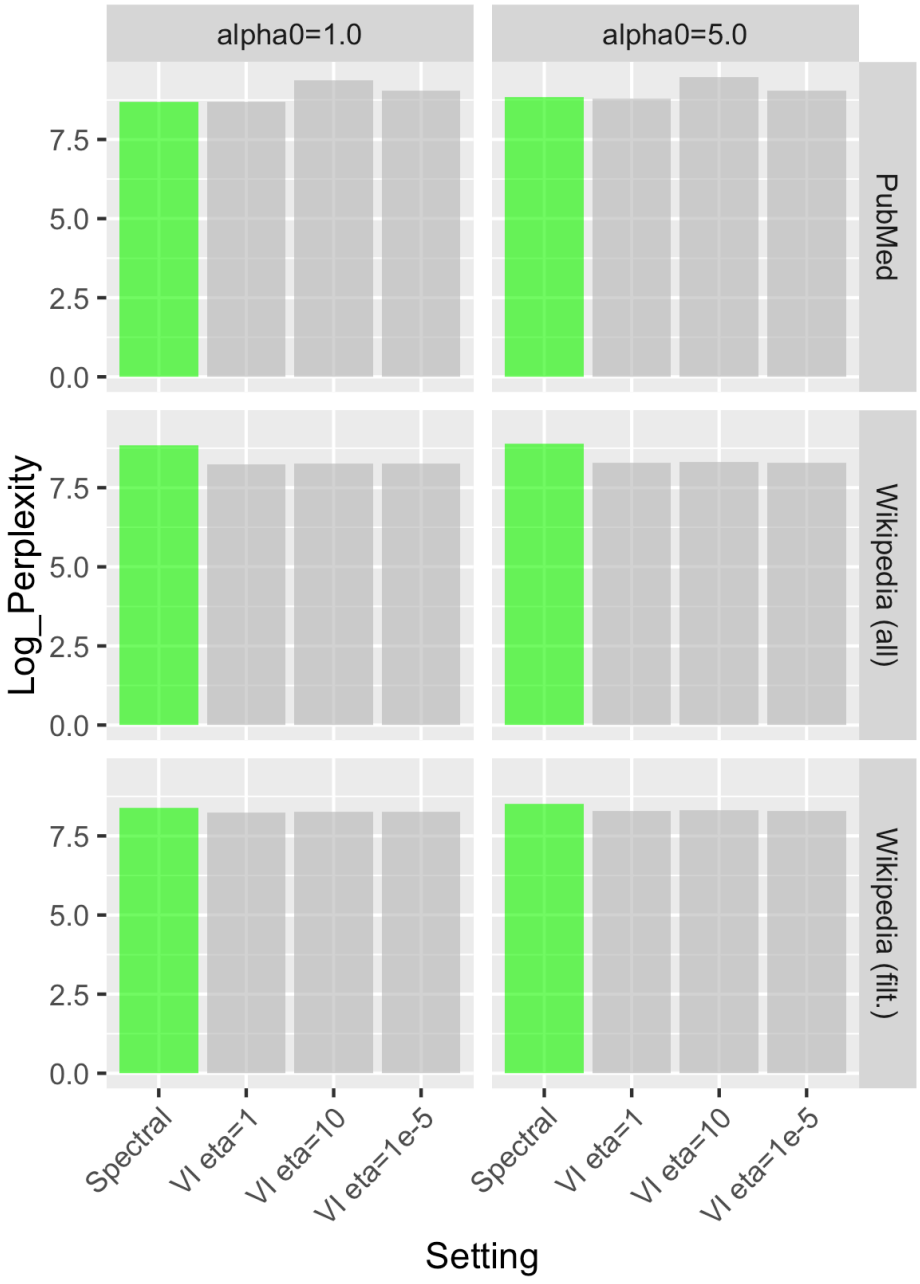

Figure 5: Log-perplexity of the Spectral LDA algorithm vs the Online Variational Inference on the PubMed and Wikipedia datasets. "VI" stands for Online Variational Inference. "eta" is the Dirichlet parameter for the prior of the topic-word distribution. The number of topics $k = 20$. We grouped the results by $\alpha_0 = \sum_{i=1}^{k} \alpha_i$. For the Online Variational Inference, we varied the β's prior $\eta = 10^{-5}, 1.0, 10.0$ and reported results respectively.

Setting	Runtime (min)	Log-Perplexity
Spectral LDA	2	8.68
VI 10 iterations	10	8.70
VI 200 iterations	95	8.60
VI 400 iterations	184	8.58

Table 5: Log-perplexity of the Spectral LDA algorithm vs the Online Variational Inference on the PubMed dataset, run on 32 'c4.8xlarge' Worker nodes. "VI" stands for Online Variational Inference. The number of topics $k = 20$, the sum of prior for the topic distribution $\alpha_0 = 1.0$. For the Online Variational Inference, we assume uniform prior for the topic-word distributions $\eta = 1.0$; the minibatch size is 5% of the corpus.

Figure 6: Runtime (in seconds) vs number of workers for the Spectral LDA algorithm

different sparsity of input corpus as in Tab 4 has both a significant impact on the final result and runtime of log-perplexity calculation.

The default $q = 1$, as the previous results were computed with. We also experimented with $q = 0, 5, 10$ on the PubMed, full and filtered Wikipedia sets.

The runtimes of log-perplexity computation on the full vs filtered Wikipedia sets are 30:1. To understand the reason, the filtered set represents much lower sparsity with respect to the full set, which makes it far easier for the latent topic attribution for every document.

We list the results of log-perplexity on the PubMed and filtered Wikipedia sets in Tab 6 below for $k = 20$, $\alpha_0 = 1.0$.

q	PubMed	Wikipedia (filt.)
0	8.91	8.48
1	8.69	8.43
5	8.67	8.48
10	8.68	8.49

Table 6: Log-perplexity of the Spectral LDA algorithm in function of the number of iterations q of the power method for RandSVD. $k = 20$, $\alpha_0 = 1.0$. We report results on the PubMed and the filtered Wikipedia sets (document length ≥ 100).

We see that q taking a small positive number improves over $q = 0$. For the PubMed set with high sparsity, increasing q improves the log-perplexity marginally; for the filtered Wikipedia set, which is one with reasonable sparsity, $q = 1$ largely suffices. Actually, a dataset with lower sparsity will contribute more "evidence" to M_2 therefore reduce the need for any further q to distinguish between eigenvectors.

In the code by default we set $q = 1$. We also have a command line/API option to adjust it.

5.5 Remarks

We noticed that the Spectral LDA algorithm scales well with the number of topics k. With Apache Spark v2.0.0, if we run on 4 'c4.8xlarge' worker nodes for the PubMed dataset, the built-in collapsed Gibbs sampling algorithm will crash for $k = 20$ after 6 iterations; the Online Variational Inference algorithm will crash for $k = 50$ after 250 iterations (with each minibatch to be 5% of the entire corpus), probably due to memory problems in both cases. We'd look forward to these problems being fixed in later releases.

6 Conclusion

We discussed the algorithm design and implementation details of the Spectral LDA algorithm. With a carefully crafted implementation we could achieve linear scalability and rivalling log-perplexity vs the Online Variational Inference.

The Spectral LDA algorithm has strong advantage in that the convergence is guaranteed by theory whilst it would take the Online Variational Inference 50x more time to compute log-perplexity and check convergence. The basis part that takes the majority of runtime in the Spectral LDA algorithm is nothing but the tensor product which could be efficiently executed by native BLAS. Lastly it has clear memory usage and eases memory management, whilst its counterparts, the collapsed Gibbs sampling or Online Variational Inference, by their iterative nature, tend to crash when we run for higher $k \geq 50$ and number of iterations on Apache Spark v2.0.0.

We would argue therefore for the inclusion of the Spectral LDA algorithm in the Apache Spark MLlib and we wish for wider use of the tensor methods in more applications.

References

[1] Anandkumar, A., Ge, R., Hsu, D., Kakade, S. M., and Telgarsky, M. Tensor decompositions for learning latent variable models. *J. Mach. Learn. Res. 15*, 1 (Jan. 2014), 2773–2832.

[2] Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res. 3* (Mar. 2003), 993–1022.

[3] Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning* (New York, NY, USA, 2008), ICML '08, ACM, pp. 272–279.

[4] Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points — online stochastic gradient for tensor decomposition. In *Proceedings of The 28th Conference on Learning Theory* (2015), pp. 797– –842.

[5] Griffiths, T. L., and Steyvers, M. Finding scientific topics. *Proceedings of the National Academy of Sciences 101*, suppl 1 (2004), 5228–5235.

[6] Halko, N., Martinsson, P. G., and Tropp, J. A. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review 53*, 2 (2011), 217–288.

[7] Hoffman, M., Bach, F. R., and Blei, D. M. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 856–864.

[8] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *J. Mach. Learn. Res. 14*, 1 (May 2013), 1303–1347.

[9] Kolda, T. G., and Bader, B. W. Tensor decompositions and applications. *SIAM Review 51*, 3 (2009), 455–500.

[10] Laskowski, J. Mastering Apache Spark 2.0. `https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details`.

[11] Musco, C., and Musco, C. Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1396–1404.

[12] Nallapati, R., Cohen, W., and Lafferty, J. Parallelized variational em for latent dirichlet allocation: An experimental evaluation of speed and scalability. In *Seventh IEEE International Conference on Data Mining Workshops (ICDMW 2007)* (Oct 2007), pp. 349–354.

[13] Newman, D., Smyth, P., Welling, M., and Asuncion, A. U. Distributed inference for latent dirichlet allocation. In *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 1081–1088.

[14] Smola, A., and Narayanamurthy, S. An architecture for parallel topic models. *Proc. VLDB Endow. 3*, 1-2 (Sept. 2010), 703–710.

[15] Wang, Y., Bai, H., Stanton, M., Chen, W.-Y., and Chang, E. Y. Plda: Parallel latent dirichlet allocation for large-scale applications. In *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management* (Berlin, Heidelberg, 2009), AAIM '09, Springer-Verlag, pp. 301–314.

[16] Wang, Y., Tung, H.-Y., Smola, A., and Anandkumar, A. Fast and guaranteed tensor decomposition via sketching. In *Proceedings of the 28th International Conference on Neural Information Processing Systems* (Cambridge, MA, USA, 2015), NIPS'15, MIT Press, pp. 991–999.