Distributed Gibbs Sampling of Latent Topic Models: The Gritty Details

THIS IS AN EARLY DRAFT. YOUR FEEDBACKS ARE HIGHLY APPRECIATED.

Yi Wang
yi.wang.2005@gmail.com

August 2008

Contents

Chapter 1

Preface

In 2003, Blei, Ng and Jordan [3] presented the Latent Dirichlet Allocation (LDA) model and a Variational Expectation-Maximization algorithm for training the model. In 2004, Griffiths and Steyvers [7] derived a Gibbs sampling algorithm for learning LDA. Since then, Gibbs sampling was shown more efficient than other LDA training algorithms including variational EM and Expectation-Propagation [10]. This efficiency is due to a instrisic property of LDA – the conjugacy between the Dirichlet prior the multinomial likelihood. For this reason, Gibbs sampling algorithms were derived for inference in many models that extends LDA [14] [1] [4] [2] [9].

To further improve the efficiency of the Gibbs sampling algorithm for LDA, researchers tried to distribute the computation on multiple computers [11] or to optimize the Gibbs sampling speed on each computer [12].

Since November 2007, I started to work on developing distributed computing solutions of topic models. Industrial solutions are often required to train models using massive data sets, so I need to express training algorithms using modern distributed computing models, in particular, MapReduce, BSP and MPI.

This document is my learning and working note. If you are interested with large scale computing of latent topic models, I hope this document could be helpful in the first stage of your work.

Chapter 2

Latent Dirichlet Allocation

2.1 Introduction

I started to write this chapter since November 2007, right after my first MapReduce implementation of the AD-LDA algorithm[11]. I had worked so hard to understand LDA, but cannot find any document that were comprehensive, complete, and contain all necessary details. It is true that I can find some open source implementations of LDA, for example, LDA Gibbs sampler in Java[1] and GibbsLDA++[2], but code does not reveal math derivations. I read Griffiths' paper [7] and technical report [6], but realized most derivations are skipped. It was lucky to me that in the paper about Topic-over-time[14], an extension of LDA, the authors show part of the derivation. The derivation is helpful to understand LDA but is too short and not self-contained. A primer [8] by Gregor Heinrich contains more details than documents mentioned above. Indeed, most of my initial understanding on LDA comes from [8]. As [8] focuses more on the Bayesian background of latent variable modeling, I wrote this article focusing on more on the derivation of the Gibbs sampling algorithm for LDA. You may notice that the Gibbs sampling updating rule we derived in (2.38) differs slightly from what was presented in [6] and [8], but matches that in [14].

2.2 LDA and Its Learning Problem

In the literature, the training documents of LDA are often denoted by a long and segmented vector W:

$$W = \begin{bmatrix} \{w_1, \ldots, w_{N_1}\}, & \text{words in the 1st document} \\ \{w_{N_1+1}, \ldots, w_{N_1+N_2}\}, & \text{words in the 2nd document} \\ \cdots & \cdots \\ \{w_{1+\sum_{j=1}^{D-1} N_j}, \ldots, w_{\sum_{j=1}^{D} N_j}\} & \text{words in the } D\text{-th document} \end{bmatrix}, \tag{2.1}$$

where N_d denotes the number of words in the d-th document. [3]

[1] http://arbylon.net/projects/LdaGibbsSampler.java

[2] http://gibbslda++.sourceforge.net

[3] Another representation of the training documents are two vectors: $\boldsymbol{d}$ and W:

$$\begin{bmatrix} \boldsymbol{d} \\ W \end{bmatrix} = \begin{bmatrix} d_1, & \ldots, & d_N \\ w_1, & \ldots, & w_N \end{bmatrix}, \tag{2.2}$$

where $N = \sum_{j=1}^{D} N_j$, d_i indices in $\mathcal{D}$, w_i indices in $\mathcal{W}$, and the tuple $[d_i, w_i]^T$ denotes an edge between the two vertices indexed by d_i and w_i respectively. Such representation is comprehensive

In rest of this article, we consider hidden variables

$$Z = \begin{bmatrix} \{z_1, \dots, z_{N_1}\}, & \text{topics of words in the 1st document} \\ \{z_{N_1+1}, \dots, z_{N_1+N_2}\}, & \text{topics of words in the 2nd document} \\ \dots & \dots \\ \{z_{1+\sum_{j=1}^{D-1} N_j}, \dots, z_{\sum_{j=1}^{D} N_j}\} & \text{topics of words in the } D\text{-th document} \end{bmatrix} , \quad (2.3)$$

where each $z_i \in Z$ corresponds to a word $w_i \in W$ in (2.1).

2.3 Dirichlet and Multinomial

To derive the Gibbs sampling algorithm with LDA, it is important to be familiar with the conjugacy between Dirichlet distribution and multinomial distribution.

The Dirichlet distribution is defined as:

$$\mathrm{Dir}(\boldsymbol{p}; \boldsymbol{\alpha}) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{v=1}^{|\boldsymbol{\alpha}|} p_t^{\alpha_t - 1} , \quad (2.4)$$

where the normalizing constant is the multinomial beta function, which can be expressed in terms of gamma function:

$$\mathrm{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{|\boldsymbol{\alpha}|} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{|\boldsymbol{\alpha}|} \alpha_i)} . \quad (2.5)$$

When Dirichlet distribution is used in LDA to model the prior, α_i's are positive integers. In this case, the gamma function degenerates to the factorial function:

$$\Gamma(n) = (n-1)! \quad (2.6)$$

The multinomial distribution is defined as:

$$\mathrm{Mult}(\boldsymbol{x}; \boldsymbol{p}) = \frac{n!}{\prod_{i=1}^{K} x_i!} \prod_{i=1}^{K} p_i^{x_i} , \quad (2.7)$$

where x_i denotes the number of times that value i appears in the samples drawn from the discrete distribution $\boldsymbol{p}$, $K = |\boldsymbol{x}| = |\boldsymbol{p}| = |\boldsymbol{\alpha}|$ and $n = \sum_{i=1}^{K} x_i$. The normalization constant comes from the product of combinatorial numbers.

We say the Dirichlet distribution is the conjugate distribution of the multinomial distribution, because if the prior of $\boldsymbol{p}$ is $\mathrm{Dir}(\boldsymbol{p}; \boldsymbol{\alpha})$ and $\boldsymbol{x}$ is generated by $\mathrm{Mult}(\boldsymbol{x}; \boldsymbol{p})$, then the posterior distribution of $\boldsymbol{p}$, $p(\boldsymbol{p}|\boldsymbol{x}, \boldsymbol{\alpha})$, is a Dirichlet distribution:

$$\begin{aligned} p(\boldsymbol{p}|\boldsymbol{x}, \boldsymbol{\alpha}) &= \mathrm{Dir}(\boldsymbol{p}; \boldsymbol{x} + \boldsymbol{\alpha}) \\ &= \frac{1}{\mathrm{B}(\boldsymbol{x} + \boldsymbol{\alpha})} \prod_{v=1}^{|\boldsymbol{\alpha}|} p_t^{x_t + \alpha_t - 1} . \end{aligned} \quad (2.8)$$

and is used in many implementations of LDA, e.g, [13].

However, many papers (including this article) do not use this representation, because it misleads readers to consider two sets of random variables, W and $\boldsymbol{d}$. The fact is that $\boldsymbol{d}$ is the structure of W and is highly dependent with W. $\boldsymbol{d}$ is also the structure of the hidden variables Z, as shown by (2.3).

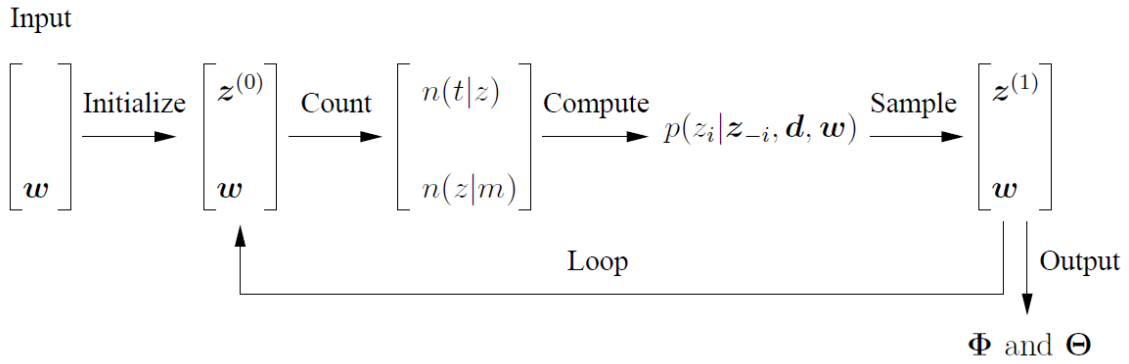

Figure 2.1: The procedure of learning LDA by Gibbs sampling.

Because (2.8) is a probability distribution function, integrating it over $\boldsymbol{p}$ should result in 1:

$$\begin{aligned} 1 &= \int \frac{1}{\mathrm{B}(\boldsymbol{x}+\boldsymbol{\alpha})} \prod_{v=1}^{|\boldsymbol{\alpha}|} p_t^{x_t+\alpha_t-1} \mathrm{d}\boldsymbol{p} \\ &= \frac{1}{\mathrm{B}(\boldsymbol{x}+\boldsymbol{\alpha})} \int \prod_{v=1}^{|\boldsymbol{\alpha}|} p_t^{x_t+\alpha_t-1} \mathrm{d}\boldsymbol{p} \ . \end{aligned} \tag{2.9}$$

This implies

$$\int \prod_{v=1}^{|\boldsymbol{\alpha}|} p_t^{x_t+\alpha_t-1} \mathrm{d}\boldsymbol{p} = \mathrm{B}(\boldsymbol{x}+\boldsymbol{\alpha}) \ . \tag{2.10}$$

This property will be used in the following derivations.

2.4 Learning LDA by Gibbs Sampling

There have been three strategies to learn LDA: EM with variational inference [3], EM with expectation propagation [10], and Gibbs sampling [7]. In this article, we focus on the Gibbs sampling method, whose performance is comparable with the other two but is tolerant better to local optima.

Gibbs sampling is one of the class of samplings methods known as Markov Chain Monte Carlo. We use it to sample from the posterior distribution, $p(Z|W)$, given the training data W represented in the form of (2.1). As will be shown by the following text, given the sample of Z we can infer model parameters $\boldsymbol{\Phi}$ and $\boldsymbol{\Theta}$. This forms a learning algorithm with its general framework shown in Fig. 2.1.

In order to sample from $p(Z|W)$ using the Gibbs sampling method, we need the full conditional posterior distribution $p(z_i|Z^{\neg i}, W)$, where $Z^{\neg i}$ denotes all z_j's with $j \neq i$. In particular, Gibbs sampling does not require knowing the exact form of $p(z_i|Z^{\neg i}, W)$; instead, it is enough to have a function $f(\cdot)$, where

$$f(z_i|Z^{\neg i}, W) \propto p(z_i|Z^{\neg i}, W) \tag{2.11}$$

The following mathematical derivation is for such a function $f(\cdot)$.

N	the number of words in the corpus
$1 \leq i, j \leq N$	index of words in the corpus
$W = \{w_i\}$	the corpus, and w_i denotes a word
$Z = \{z_i\}$	latent topics assigned to words in W
$W^{\neg i} = W \setminus w_i$	the corpus excluding w_i
$Z^{\neg i} = Z \setminus z_i$	latent topics excluding z_i
K	the number of topics specified as a parameter
V	the number of unique words in the vocabulary
$\boldsymbol{\alpha}$	the parameters of topic Dirichlet prior
$\boldsymbol{\beta}$	the parameters of word Dirichlet prior
$\Omega_{d,k}$	count of words in d assigned topic k; Ω_d denotes the d-th row of matrix Ω.
$\Psi_{k,v}$	count of word v in corpus assigned k Ψ_k denotes the k-th row of matrxi Ψ.
$\Omega_{d,k}^{\neg i}$	like $\Omega_{d,k}$ but excludes w_i and z_i
$\Psi_{k,v}^{\neg i}$	like $\Psi_{k,v}$ but excludes w_i and z_i
$\boldsymbol{\Theta} = \{\theta_{d,k}\}$	$\theta_{d,k} = P(z = k\|d)$, $\boldsymbol{\theta}_d = P(z\|d)$.
$\boldsymbol{\Phi} = \{\phi_{k,v}\}$	$\phi_{k,v} = P(w = v\|z = k)$, $\boldsymbol{\phi}_k = P(w\|z = k)$.

Table 2.1: Symbols used in the derivation of LDA Gibbs sampling rule.

The Joint Distribution of LDA We start from deriving the joint distribution [4],

$$p(Z, W|\boldsymbol{\alpha}, \boldsymbol{\beta}) = p(W|Z, \boldsymbol{\beta})p(Z|\boldsymbol{\alpha}) \ , \tag{2.12}$$

which is the basis of the derivation of the Gibbs updating rule and the parameter estimation rule. As $p(W|Z, \boldsymbol{\beta})$ and $p(Z|\boldsymbol{\alpha})$ depend on $\boldsymbol{\Phi}$ and $\boldsymbol{\Theta}$ respectively, we derive them separately.

According to the definition of LDA, we have

$$p(W|Z, \boldsymbol{\beta}) = \int p(W|Z, \boldsymbol{\Phi})p(\boldsymbol{\Phi}|\boldsymbol{\beta})\mathrm{d}\boldsymbol{\Phi} \ , \tag{2.13}$$

where $p(\boldsymbol{\Phi}|\boldsymbol{\beta})$ has Dirichlet distribution:

$$p(\boldsymbol{\Phi}|\boldsymbol{\beta}) = \prod_{k=1}^{K} p(\boldsymbol{\phi}_k|\boldsymbol{\beta}) = \prod_{k=1}^{K} \frac{1}{\mathrm{B}(\boldsymbol{\beta})} \prod_{v=1}^{V} \phi_{k,v}^{\beta_v - 1} \ , \tag{2.14}$$

and $p(W|Z, \boldsymbol{\Phi})$ has multinomial distribution:

$$p(W|Z, \boldsymbol{\Phi}) = \prod_{i=1}^{N} \phi_{z_i, w_i} = \prod_{k=1}^{K} \prod_{v=1}^{V} \phi_{k,v}^{\Psi_{k,v}} \ , \tag{2.15}$$

where Ψ is a $K \times V$ count matrix and $\Psi_{k,v}$ is the number of times that topic k is assigned to word v. With W and Z defined by (2.1) and (2.3) respectively, we can represent $\Psi_{k,v}$ mathematically by

$$\Psi_{k,v} = \sum_{i=1}^{N} \mathrm{I}\{w_i = v \wedge z_i = k\} \ , \tag{2.16}$$

[4] Notations used in this article are identical with those used in [8].

where N is the corpus size in number of words. In later sections, we use Ψ_k to denote the k-th row of the matrix Ψ.

Given (2.15) and (2.14), (2.13) becomes

$$p(W|Z,\boldsymbol{\beta}) = \int \prod_{k=1}^{K} \frac{1}{\mathrm{B}(\boldsymbol{\beta})} \prod_{v=1}^{V} \phi_{k,v}^{\Psi_{k,v}+\beta_v-1} \mathrm{d}\boldsymbol{\phi}_k \ . \tag{2.17}$$

Using the property of integration of a product, which we learned in our colleague time,

$$\int \prod_{k=1}^{K} f_k(\phi_k)\ \mathrm{d}\phi_1 \ldots \mathrm{d}\phi_K = \prod_{k=1}^{K} \int f_k(\phi_k)\ \mathrm{d}\phi_k \ , \tag{2.18}$$

we have

$$\begin{aligned} p(W|Z,\boldsymbol{\beta}) &= \prod_{k=1}^{K} \left(\int \frac{1}{\mathrm{B}(\boldsymbol{\beta})} \prod_{v=1}^{V} \phi_{k,v}^{\Psi_{k,v}+\beta_v-1} \mathrm{d}\boldsymbol{\phi}_k \right) \\ &= \prod_{k=1}^{K} \left(\frac{1}{\mathrm{B}(\boldsymbol{\beta})} \int \prod_{v=1}^{V} \phi_{k,v}^{\Psi_{k,v}+\beta_v-1} \mathrm{d}\boldsymbol{\phi}_k \right) \ . \end{aligned} \tag{2.19}$$

Using property (2.10), we can compute the integratation over $\boldsymbol{\phi}_k$ in a close form, so

$$p(W|Z,\boldsymbol{\beta}) = \prod_{k=1}^{K} \frac{\mathrm{B}(\Psi_k+\boldsymbol{\beta})}{\mathrm{B}(\boldsymbol{\beta})} \ , \tag{2.20}$$

where Ψ_k denotes the k-th row of matrix Ψ.

Now we derive $p(Z|\boldsymbol{\alpha})$ analogous to $p(W|Z,\boldsymbol{\beta})$. Similar with (2.14), we have

$$p(\boldsymbol{\Theta}|\boldsymbol{\alpha}) = \prod_{d=1}^{D} p(\boldsymbol{\theta}_m|\boldsymbol{\alpha}) = \prod_{d=1}^{D} \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{k=1}^{K} \theta_{d,k}^{\alpha_k-1} \ ; \tag{2.21}$$

similar with (2.15), we have

$$p(Z|\boldsymbol{\Theta}) = \prod_{i=1}^{N} \theta_{d_i,z_i} = \prod_{d=1}^{D} \prod_{k=1}^{K} \theta_{d,k}^{\Omega_{d,k}} \ ; \tag{2.22}$$

and similar with (2.20) we have

$$\begin{aligned} p(Z|\boldsymbol{\alpha}) &= \int p(Z|\boldsymbol{\Theta}) p(\boldsymbol{\Theta}|\boldsymbol{\alpha}) \mathrm{d}\boldsymbol{\Theta} \\ &= \prod_{d=1}^{D} \left(\int \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{k=1}^{K} \theta_{d,k}^{\Omega_{d,k}+\alpha_k-1} \mathrm{d}\boldsymbol{\theta}_d \right) \\ &= \prod_{d=1}^{D} \frac{\mathrm{B}(\Omega_d+\boldsymbol{\alpha})}{\mathrm{B}(\boldsymbol{\alpha})} \ , \end{aligned} \tag{2.23}$$

where Ω is a count matrix and $\Omega_{d,k}$ is the number of times that topic k is assigned to words in document d; Ω_d denotes the d-th row of Ω. With input data defined by (2.2), $\Omega_{d,k}$ can be represented mathematically

$$\Omega_{d,k} = \sum_{i=1}^{N} \mathrm{I}\{d_i = d \wedge z_i = k\} \ , \tag{2.24}$$

where N is the corpus size by words. In later sections, we use Ω_d to denote the d-th row of the matrix Ω.

Given (2.20) and (2.23), the joint distribution (2.12) becomes:

$$\begin{aligned} p(Z, W|\boldsymbol{\alpha}, \boldsymbol{\beta}) &= p(W|Z, \boldsymbol{\beta})p(Z|\boldsymbol{\alpha}) \\ &= \prod_{k=1}^{K} \frac{\mathrm{B}(\Psi_k + \boldsymbol{\beta})}{\mathrm{B}(\boldsymbol{\beta})} \cdot \prod_{d=1}^{D} \frac{\mathrm{B}(\Omega_d + \boldsymbol{\alpha})}{\mathrm{B}(\boldsymbol{\alpha})} \end{aligned} \tag{2.25}$$

The Gibbs Updating Rule With (2.25), we can derive the Gibbs updating rule for LDA:

$$p(z_i = k|Z^{\neg i}, W, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{p(z_i = k, Z^{\neg i}, W|\boldsymbol{\alpha}, \boldsymbol{\beta})}{p(Z^{\neg i}, W|\boldsymbol{\alpha}, \boldsymbol{\beta})} \ . \tag{2.26}$$

Because z_i depends only on w_i, (2.26) becomes

$$p(z_i|Z^{\neg i}, W, \boldsymbol{\alpha}, \boldsymbol{\beta}) \propto \frac{p(Z, W|\boldsymbol{\alpha}, \boldsymbol{\beta})}{p(Z^{\neg i}, W^{\neg i}|\boldsymbol{\alpha}, \boldsymbol{\beta})} \qquad \text{where } z_i = k \ . \tag{2.27}$$

The numerator in (2.27) is (2.25); whereas the denominator has a similar form:

$$\begin{aligned} p(Z^{\neg i}, W^{\neg i}|\boldsymbol{\alpha}, \boldsymbol{\beta}) &= p(W^{\neg i}|Z^{\neg i}, \boldsymbol{\beta})p(Z^{\neg i}|\boldsymbol{\alpha}) \\ &= \prod_{k=1}^{K} \frac{\mathrm{B}(\Psi_k^{\neg i} + \boldsymbol{\beta})}{\mathrm{B}(\boldsymbol{\beta})} \cdot \prod_{d=1}^{D} \frac{\mathrm{B}(\Omega_d^{\neg i} + \boldsymbol{\alpha})}{\mathrm{B}(\boldsymbol{\alpha})} \ , \end{aligned} \tag{2.28}$$

where $\Psi_k^{\neg i}$ denotes the k-th row of the count $K \times V$ count matrix $\Psi^{\neg i}$, and $\Psi_{k,v}^{\neg i}$ is the number of times that topic k is assigned to word w, but with the i-th word and its topic assignment excluded. Similarly, $\Omega_d^{\neg i}$ is the d-th row of count matrix $\Omega^{\neg i}$, and $\Omega_{d,k}^{\neg i}$ is the number of words in document d that are assigned topic k, but with the i-th word and its topic assignment excluded. In more details,

$$\begin{aligned} \Psi_{k,v}^{\neg i} &= \sum_{\substack{1 \le j \le N \\ j \neq i}} \mathrm{I}\{w_j = v \wedge z_j = k\} \ , \\ \Omega_{d,k}^{\neg i} &= \sum_{\substack{1 \le j \le N \\ j \neq i}} \mathrm{I}\{d_j = d \wedge z_j = k\} \ , \end{aligned} \tag{2.29}$$

where d_j denotes the document to which the i-th word in the corpus belongs. Some properties can be derived from the definitions directly:

$$\begin{aligned} \Psi_{k,v} &= \begin{cases} \Psi_{k,v}^{\neg i} + 1 & \text{if } v = w_i \text{ and } k = z_i; \\ \Psi_{k,v}^{\neg i} & \text{all other cases.} \end{cases} \\ \Omega_{d,k} &= \begin{cases} \Omega_{d,k}^{\neg i} + 1 & \text{if } d = d_i \text{ and } k = z_i; \\ \Omega_{d,k}^{\neg i} & \text{all other cases.} \end{cases} \end{aligned} \tag{2.30}$$

From (2.30) we have

$$\begin{aligned} \sum_{v=1}^{V} \Psi_{z_i,v} &= 1 + \sum_{v=1}^{V} \Psi_{z_i,v}^{\neg i} \\ \sum_{k=1}^{K} \Omega_{d_i,z} &= 1 + \sum_{k=1}^{K} \Omega_{d_i,z}^{\neg i} \end{aligned} \tag{2.31}$$

Using (2.30) and (2.31), we can simplify (2.28):

$$p(z_i = k|Z^{\neg i}, W, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\mathrm{B}(\Psi_k + \boldsymbol{\beta})}{\mathrm{B}(\Psi_k^{\neg i} + \boldsymbol{\beta})} \cdot \frac{\mathrm{B}(\Omega_d + \boldsymbol{\alpha})}{\mathrm{B}(\Omega_d^{\neg i} + \boldsymbol{\alpha})} \quad . \tag{2.32}$$

where d denotes the document that contains w_i.

Further simplification can be achieved by substituting

$$\mathrm{B}(\boldsymbol{x}) = \frac{\prod_{k=1}^{\dim \boldsymbol{x}} \Gamma(x_k)}{\Gamma(\sum_{k=1}^{\dim \boldsymbol{x}} x_k)} \quad , \tag{2.33}$$

and we have

$$p(z_i = k|Z^{\neg i}, w_i = v, W^{\neg i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\frac{\prod_{v=1}^{V} \Gamma(\Psi_{k,v}+\beta_v)}{\Gamma(\sum_{v=1}^{V} \Psi_{k,v}+\beta_v)}}{\frac{\prod_{v=1}^{V} \Gamma(\Psi_{k,v}^{\neg i}+\beta_v)}{\Gamma(\sum_{v=1}^{V} \Psi_{k,v}^{\neg i}+\beta_v)}} \cdot \frac{\frac{\prod_{k=1}^{K} \Gamma(\Omega_{d,k}+\alpha_k)}{\Gamma(\sum_{k=1}^{K} \Omega_{d,k}+\alpha_k)}}{\frac{\prod_{k=1}^{K} \Gamma(\Omega_{d,k}^{\neg i}+\alpha_k)}{\Gamma(\sum_{k=1}^{K} \Omega_{d,k}^{\neg i}+\alpha_k)}} \quad . \tag{2.34}$$

Using (2.30), we can remove most factors in the products of (2.34) and get

$$p(z_i = k|Z^{\neg i}, w_i = v, W^{\neg i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\frac{\Gamma(\Psi_{k,v}+\beta_v)}{\Gamma(\sum_{v=1}^{V} \Psi_{k,v}+\beta_v)}}{\frac{\Gamma(\Psi_{k,v}^{\neg i}+\beta_v)}{\Gamma(\sum_{v=1}^{V} \Psi_{k,v}^{\neg i}+\beta_v)}} \cdot \frac{\frac{\Gamma(\Omega_{d,k}+\alpha_k)}{\Gamma(\sum_{k=1}^{K} \Omega_{d,k}+\alpha_k)}}{\frac{\Gamma(\Omega_{d,k}^{\neg i}+\alpha_k)}{\Gamma(\sum_{k=1}^{K} \Omega_{d,k}^{\neg i}+\alpha_k)}} \quad . \tag{2.35}$$

Here it is important to know that

$$\Gamma(y) = (y-1)! \qquad \text{if } y \text{ is a positive integer} \quad , \tag{2.36}$$

so we expand (2.35) and using (2.30) to remove most factors in the factorials. This leads us to the Gibbs updating rule for LDA:

$$p(z_i = k|Z^{\neg i}, w = v, W^{\neg i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\Psi_{k,v} + \beta_{w_i} - 1}{\left[\sum_{v=1}^{V} \Psi_{k,v} + \beta_t\right] - 1} \cdot \frac{\Omega_{d,k} + \alpha_k - 1}{\left[\sum_{k=1}^{K} \Omega_{d,k} + \alpha_z\right] - 1} \quad . \tag{2.37}$$

Note that the denominator of the second factor at the right hand side of (2.38) does not depend on z_i, the parameter of the function $p(z_i|Z^{\neg i}, W, \boldsymbol{\alpha}, \boldsymbol{\beta})$. Because the Gibbs sampling method requires only a function $f(z_i) \propto p(z_i)$, c.f. (2.11), we can use the following updating rule in practice:

$$p(z_i = k|Z^{\neg i}, w = v, W^{\neg i}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\Psi_{k,v} + \beta_{w_i} - 1}{\left[\sum_{v=1}^{V} \Psi_{k,v} + \beta_t\right] - 1} \cdot \left[\Omega_{d,k} + \alpha_k - 1\right] \quad . \tag{2.38}$$

Parameter Estimation By the definition of $\phi_{k,v}$ and $\theta_{d,k}$, we have

$$\begin{aligned} \phi_{k,v} &= \frac{\Psi_{k,v} + \beta_t}{\left(\sum_{v'=1}^{V} \Psi_{k,v'} + \beta_{v'}\right)} \quad , \\ \theta_{m,k} &= \frac{\Omega_{d,k} + \alpha_k}{\left(\sum_{k=1}^{K} \Omega_{d,k} + \alpha_z\right)} \quad . \end{aligned} \tag{2.39}$$

zero all count variables `NWZ`, `NZM`, `NZ` ;
foreach *document* $m \in [1, D]$ **do**
  foreach *word* $n \in [1, N_m]$ *in document* m **do**
    sample topic index $z_{m,n} \sim \text{Mult}(1/K)$ for word $w_{m,n}$;
    increment document-topic count: `NZM[`$z_{m,n}, m$`]++` ;
    increment topic-term count: `NWZ[`$w_{m,n}, z_{m,n}$`]++` ;
    increment topic-term sum: `NZ[`$z_{m,n}$`]++` ;
  end
end
while *not finished* **do**
  foreach *document* $m \in [1, D]$ **do**
    foreach *word* $n \in [1, N_m]$ *in document* m **do**
      `NWZ[`$w_{m,n}, z_{m,n}$`]--`, `NZ[`$z_{m,n}$`]--`, `NZM[`$z_{m,n}, m$`]--` ;
      sample topic index $\tilde{z}_{m,n}$ according to (2.40) ;
      `NWZ[`$w_{m,n}, \tilde{z}_{m,n}$`]++`, `NZ[`$\tilde{z}_{m,n}$`]++`, `NZM[`$\tilde{z}_{m,n}, m$`]++` ;
    end
  end
  if *converged and* L *sampling iterations since last read out* **then**
    read out parameter set $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$ according to (2.39) ;
  end
end

Algorithm 1: The Gibbs sampling algorithm that learns LDA.

The Algorithm With (2.38), we can realize the learning procedure shown in Fig. 2.1 using Algorithm. 1.

In this algorithm, we use a 2D array `NWZ[w,z]` to maintain Ψ and `NZM[z,m]` for Ω. Also, to accelerate the computation, we use a 1D array `NZ[z]` to maintain $n(z) = \sum_{v=1}^{|\mathcal{W}|} n(t; z)$. Note that we do not need an array `NM[d]` for $n(z) = \sum_{k=1}^{|\mathcal{Z}|} n(z, d)$. Although it appears in (2.37), but it is not necessary in (2.38).

The input of this algorithm is not in the form of (2.1); instead, it is in a more convenient form: with each document represented by a set of words it includes, the input is given as a set of documents. Denote the n-th word in the m-th document by $w_{m,n}$, the corresponding topic is denoted by $z_{m,n}$ and corresponding document is m. With these correspondence known, the above derivations can be easily implemented in the algorithm.

The sampling operation in this algorithm is not identical with the full conditional posterior distribution (2.38); instead, it does not contain those "-1" terms:

$$p(z_i) = \frac{\Psi_{z_i,w_i} + \beta_{w_i}}{\left[\sum_{v=1}^{V} \Psi_{k,v} + \beta_t\right]} \cdot \left[\Omega_{d_i,z_i} + \alpha_{z_i}\right] \ . \tag{2.40}$$

This makes it elegant to maintain the consistency of sampling new topics and updating the counts (`NWZ`, `NZ` and `NZM`) using three lines of code: decreasing the counts, sampling and increasing the counts. [5]

[5] This trick illustrates the physical meaning of those "-1" terms in (2.38). I learn this trick from Heinrich's code at http://arbylon.net/projects/LdaGibbsSampler.java.

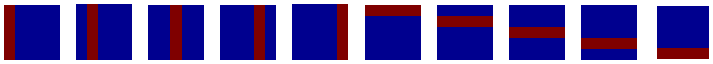

Figure 2.2: The ground-truth $\mathbf{\Phi}$ used to synthesize our testing data.

2.5 Experiments

The synthetic image data mentioned in [7] is useful to test the correctness of any implementation of LDA learning algorithm. To synthesize this data set, we fix $\mathbf{\Phi} = \{\boldsymbol{\phi}_k\}_{k=1}^{K=10}$ as visualized in Fig. 2.2 (also, Fig. 1 in [7]), set $\boldsymbol{\alpha} = [1]$, the number of words/pixels in each document/image $d = 100$. Because every image has 5×5 pixels, the vocabulary size is 25.[6]

Fig. 2.3 shows the result of running Gibbs sampling to learn an LDA from this testing data set, where the left pane shows the convergence of the likelihood, and the right pane shows the updating process of $\mathbf{\Phi}$ along the iterations of the algorithm. The 20 rows of images in the right pane are visualizations of $\mathbf{\Phi}$ estimated at the iterations of 1, 2, 3, 5, 7, , 10, 15, 20, 25, 50, 75, 100, 125, 150, 175, , 200, 250, 300, 400, 499. From this figure we can see that since iteration 100, the estimated $\mathbf{\Phi}$ is almost identical to Fig. 1(a) of [7].

2.6 Acknowledgement

Thanks go to Gregor Heinrich and Xuerui Wang for their gentle explanation of some math details.

[6] All these settings are identical with those described in [7].

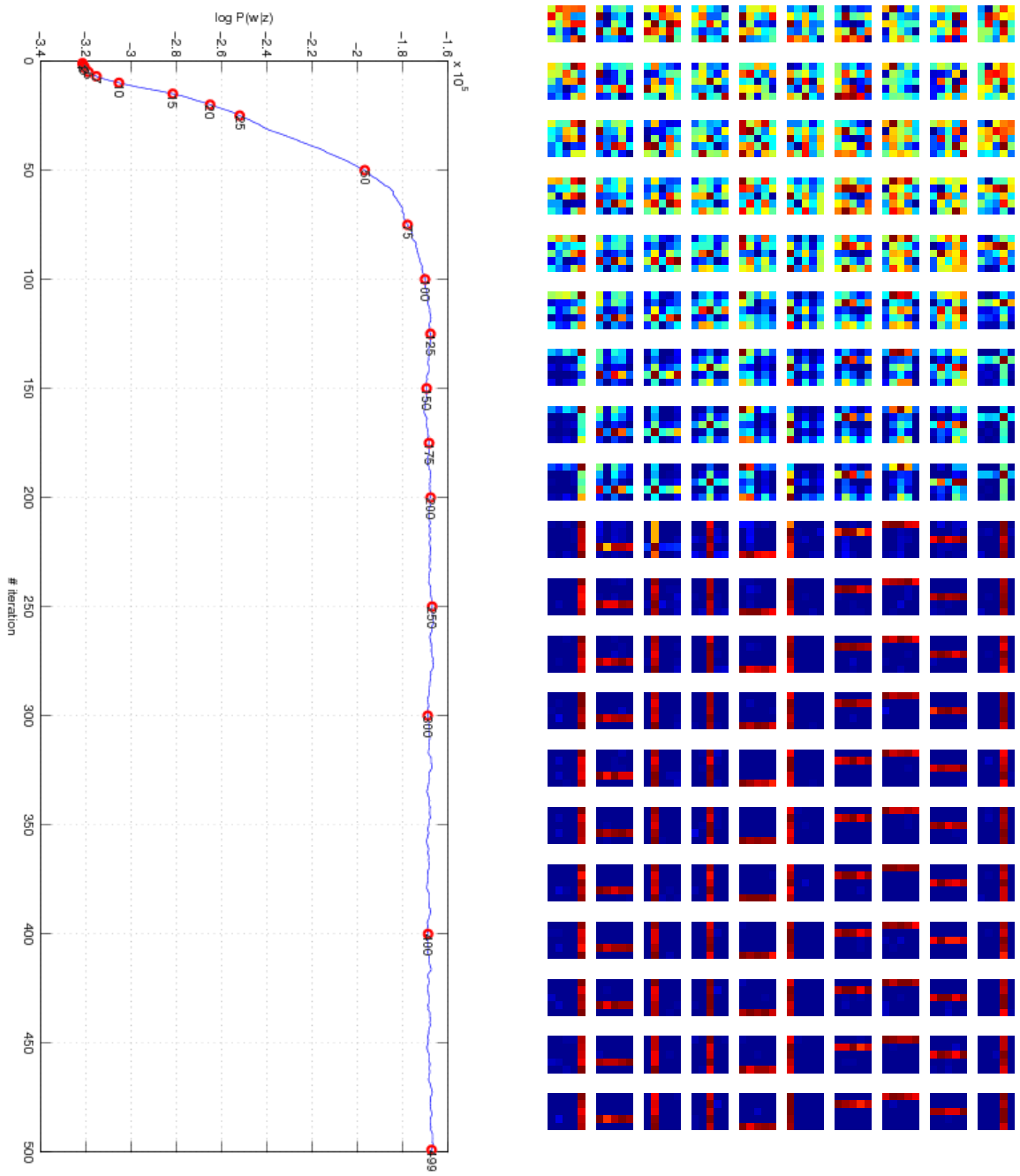

Figure 2.3: The result of running the Gibbs sampling algorithm to learn an LDA from synthetic data. Each row of 10 images in the right pane visualizes the $\mathbf{\Phi} = \{\boldsymbol{\phi}_k\}_{k=1}^{K=10}$ estimated at those iterations indicated by red circles in the left pane.

Chapter 3

Distributed Training of LDA

3.1 Introduction

Considering the generation process described by LDA, it is notable that, the documents are independent given the LDA model. This property makes it possible to design a distributed Gibbs sampling algorithm for learning LDA [11]. Also, this makes it possible to develop this algorithm using MapReduce.

3.2 Scalable Training

In order to support large-scale image annotation, we adopt a distributed Gibbs sampling algorithm, AD-LDA, proposed in [11]. The basic idea is to divide the training corpus into P parts, each part is saved on one computer. Each computer executes one iteration of the Gibbs sampling algorithm [7] to update its local model using its local data, and then the P local models are summed up to form the global model, which is replicated to the P computers to support the next iteration.

Usually, the AD-LDA algorithm can be developed using the MPI programming model, which is flexible and highly efficient, but does not support auto fault recovery — as long as one computer fails during computation, all computers have to restart their tasks. This is not a problem when we use tens of computers. But to support real large-scale data like Web images, we need magnitudes more computers. During the hours long training process, the probability that none of them fails is close to 0. Without auto fault recovery, the training will restart again and again, and statistically, never ends.

We solve this problem by modeling the AD-LDA algorithm by the MapReduce programming model[5], which has been supporting most Web-scale computations in Google©. MapReduce takes input from a set of tuples and outputs tuples. A set of

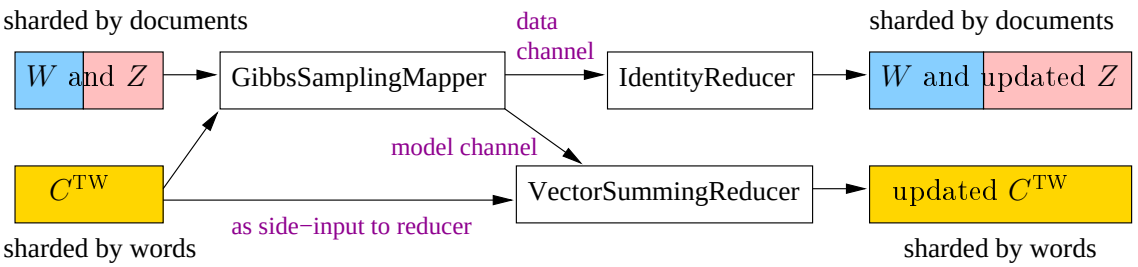

Figure 3.1: The framework of MapReduce-LDA

tuples is usually distributed stored as subsets known as a "shards". A MapReduce computation job consists of three stages — mapping, shuffling and reducing. The mapping and shuffling stages are programmable. To program the mapping stage, users provide three functions `Start()`, `Map()` and `Flush`. For every shard, a thread known as "map worker" is created to invoke `Start` once, and then `Map()` multiple times for each tuple in the shard, and then `Reduce()` once. These functions can output one or more key-value pairs, which are collected and aggregated by the shuffling stage. Key-value pairs that come from various map workers but share the same key are aggregated into a set known as a "reduce input", and this common key is assigned the key of the reduce input. Each reduce input will be processed by an invocation of the `Reduce` function, which is provided by the user.

We model each iteration of AD-LDA by the following MapReduce job:

`Start()` loads the model updated by the previous iteration;

`Map()` updates topic assignments of words in a document using the model, and records how the model should be updated according to the new topic assignments;

`Flush()` outputs the recorded update opinion.

Note that both the model as its update opinion are $V \times K$ sparse matrices, where V is the number of unique words in the training corpus, and K is the number of topics specified by the user. `Flush()` outputs each row of the update opinion matrix with the key of the corresponding word[1]. So the shuffling stage aggregates update opinion rows corresponding to a certain word and coming from various map workers into a reduce input. So `Reduce()` sums the rows element-wise to get the aggregated update opinion row for a word. The update opinion row should be added to the corresponding row of the old model estimated by the previous iteration. In a very restrictive MapReduce model, this has to be done in a separate MapReduce job. However, most MapReduce implementations now support multiple types of mappers. So rows of the old model can be loaded by an IdentityMapper and sent to `Reducer()`. It is also notable that we need to save the updated topic assignments for use in the next iteration. This can be done in `Map()`, because each document is processed once in each iteration.

3.3 Scalable Model Selection

We have mentioned two parameters of the above training algorithm: the vocabulary importance factor γ and the number of topics K. Values of these parameters can be determined by cross-validation — given any pair of $\langle \gamma, K \rangle$, we train a two-vocabulary LDA using part of data (known as "training data") and then compute the perplexity of the rest data (known as "testing data") given the trained model [8]. The smaller the perplexity value, the better that the model explains the data. To support large-scale testing data, we model the computation of perplexity by MapReduce:

`Start()` loads the model;

[1]This document is based on Google MapReduce implementation. Another well known implementation of the MapReduce model is Hadoop (`http://hadoop.apache.org`), which does not expose shard boundaries to programmers via `Start()` and `Flush()`.

Figure 3.2: The ground-truth model parameters visualized as 12 512×512 images.

Figure 3.3: The model parameters estimated in the 281-th iteration.

`Map()` computes the log-likelihood $\mathcal{L}$ of the current document given the model, and outputs a key-value pair, where key is a constant value, which makes all outputs being packed into one reduce input; value is a pair $\langle \mathcal{L}_d, N(d) \rangle$, where $N(d)$ is the length of document d;

`Reduce()` output perplexity $perp = \exp\left[\sum_d \mathcal{L}_d / \sum_d N(d)\right]$.

3.4 Experiments Using Synthetic Data

Bibliography

[1] A. C.-E. Andrew McCallum and X. Wang. Topic and role discovery in social networks. In *IJCAI*, 2005.

[2] D. Blei and J. McAuliffe. Supervised topic models. In *NIPS*, 2007.

[3] D. Blei, A. Ng, M. Jordan, and J. Lafferty. Latent dirichlet allocation. *JMLR*, 3:993–1022, 20003.

[4] D. M. Blei, T. L. Griffiths, M. I. Jordan, and J. B. Tenenbaum. Hierarchical topic models and the nested chinese restaurant process. In *NIPS*, 2003.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[6] T. Griffith. Gibbs sampling in the generative model of latent dirichlet allocation. Technical report, Stanford University, 2004.

[7] T. Griffiths and M. Steyvers. Finding scientific topics. In *PNAS*, volume 101, pages 5228–5235, 2004.

[8] G. Heinrich. Parameter estimation for text analysis. Technical report, vsonix GmbH and University of Leipzig, Germany, 2009.

[9] W. Li and A. McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *ICML*, 2006.

[10] T. Minka and J. Lafferty. Expectation propagation for the generative aspect model. In *UAI*, 2002. https://research.microsoft.com/ minka/papers/aspect/minka-aspect.pdf.

[11] D. Newman, A. Asuncion, P. Smyth, and MaxWelling. Distributed inference for latent dirichlet allocation. In *NIPS*, 2007.

[12] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *KDD*, 2008.

[13] M. Steyvers and T. Griffiths. Matlab topic modeling toolbox. Technical report, University of California, Berkeley, 2007.

[14] X. Wang and A. McCallum. Topics over time: A non-markov continuous-time model of topical trends. In *KDD*, 2006.