UNIVERSITY OF CALIFORNIA,
IRVINE

Stochastic Gradient MCMC: Algorithms and Applications

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Sungjin Ahn

2015

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Max Welling, for his excellent guidance and persistent encouragement throughout my doctoral program. His attitude, passion, and insight as a researcher and also his kindness and warmth as a person have greatly influenced my life. I will always be very proud that Max Welling was my advisor.

I would like to thank my defense committee members, Babak Shahbaba and Charless Fowlkes, for their invaluable advices and guidance. I am particularly grateful to Babak Shahbaba also for his insightful advices and discussion on my research projects.

I am also grateful to my research collaborators and fellow students: Anoop Korattikara, Yutian Chen, Dilan Gorur, Wenzhe Li, Nathan Liu, Suju Rajan, and Andrew Gelfand. I particularly spent many times with Anoop. He was always kind and supportive. It was alway a pleasure to discuss with him and I learned a lot from him.

Finally, I would like to thank my parents for a lifetime of love and support. Also, I am very grateful to my wife, Nayoung Sohn. None of this would have been possible without her love and support.

This material is based upon work supported by the National Science Foundation under Grant No.0447903, 0914783, 0928427, and IIS-1216045.

CURRICULUM VITAE

Sungjin Ahn

EDUCATION

Doctor of Philosophy in Computer Science **2015**
University of California, Irvine *Irvine, CA*

Master of Science in Computer Science **2006**
Korea Advanced Institute of Science and Technology *South Korea*

Bachelor of Science in Computer Engineering **2004**
Korea Aerospace University *South Korea*

RESEARCH EXPERIENCE

Graduate Research Assistant **09/2010–08/2015**
University of California, Irvine *Irvine, California*

Research Intern **07/2014–01/2015**
Yahoo Labs *Sunnyvale, California*

Research Intern **06/2013–09/2013**
Bosch Research & Technology Center *Palo Alto, California*

Researcher **12/2007–06/2010**
Agency for Defense Development *South Korea*

REFEREED CONFERENCE PUBLICATIONS

Large-Scale Distributed Bayesian Matrix Factorization using Stochastic Gradient MCMC **2015**
Sungjin Ahn, Anoop Korattikara, Nathan Liu, Suju Rajan, and Max Welling
ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)

Distributed Stochastic Gradient MCMC **2014**
Sungjin Ahn, Babak Shahbaba, Max Welling
International Conference on Machine Learning (ICML)

Distributed and Adaptive Darting Monte Carlo through Regenerations **2013**
Sungjin Ahn, Yutian Chen, and Max Welling
Artificial Intelligence and Statistics (AISTATS)

Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring **2012**
Sungjin Ahn, Anoop Korattikara Balan, and Max Welling
International Conference on Machine Learning (ICML)

WORKS UNDER REVIEW

Scalable Markov chain Monte Carlo for Overlapping Community Detection 2015
Wenzhe Li, Sungjin Ahn, and Max Welling
Neural Information Processing Systems (NIPS)

ABSTRACT OF THE DISSERTATION

Stochastic Gradient MCMC: Algorithms and Applications

By

Sungjin Ahn

Doctor of Philosophy in Computer Science

University of California, Irvine, 2015

Professor Max Welling, Chair

Despite the powerful advantages of Bayesian inference such as quantifying uncertainty, accurate averaged prediction, and preventing overfitting, the traditional Markov chain Monte Carlo (MCMC) method has been regarded unsuitable for large-scale problems because it required processing the entire dataset per iteration rather than using a small random minibatch as performed in the stochastic gradient optimization. The first attempt toward the scalable MCMC method based on stochastic gradients is the stochastic gradient Langevin dynamics (SGLD) proposed by Welling and Teh [2011]. Originated from the Langevin Monte Carlo method, SGLD achieved $\mathcal{O}(n)$ computation per iteration (here, n is the size of a minibatch) by using stochastic gradients estimated using minibatches and skipping the Metropolis-Hastings accept-reject test.

In this thesis, we introduce recent advances in the stochastic gradient MCMC method since the advent of SGLD. Our contributions are two-fold: algorithms and applications. In the algorithm part, we first propose the stochastic gradient Fisher scoring algorithm (SGFS) which resolves two drawbacks of SGLD: the poor mixing rate and the arbitrarily large bias occurred when using large step sizes. Then, we also propose the distributed SGLD (D-SGLD) algorithm which makes it possible to extend the power of stochastic gradient MCMC to the distributed computing systems. In the second part, we apply the developed SG-MCMC algorithms to the most popular large-scale problems: the topic modeling using the latent Dirichlet allocation model, recommender systems using matrix factorization, and community modeling in social networks using mixed membership stochastic blockmodels. By resolving the unique challenges raised by each of the applications, which make it difficult to directly use the existing SG-MCMC methods, we obtain the-state-of-the-art results outperforming existing approaches using collapsed Gibbs sampling, stochastic variational inference, or distributed stochastic gradient descent.

Chapter 1

Introduction

We are living in the era of Big data. The number of mobile devices, cameras, microphones, wireless sensor networks, and software logs has been exponentially increasing the volume of data. In 2012, 2.5 exabytes (2.5×10^{18}) of data were created every day and 90% of the data in the world today has been created in the last two years[1]. We expect that this data deluge will be spurred even further by the rise of the Internet of Things (IoT) technology [Gubbi et al., 2013] which tries to connect objects in our daily life to the Internet.

Big data is a big opportunity. By distilling useful information such as trends, causality, and associations, from a vast sea of data, we will be better able to understand and predict the behaviors and properties of individuals, communities, society, and various systems surrounding them. This will also make us solve many problems which were difficult in the past. For example, by analyzing activity records of thousands of millions of users, companies like Google, Facebook, and Amazon, provide high-quality recommendations for books, news articles, restaurants, and even persons who we may be interested in [Koren et al., 2009]. By learning from enormous amounts of images and speech records, these companies have also succeeded in increasing recognition accuracies of some tasks such as face and speech

[1] "IBM What is big data? – Bringing big data to the enterprise", www.ibm.com.

recognition close to the level of humans [Krizhevsky et al., 2012, Taigman et al., 2014]. Furthermore, some startups like Enlitic[2] are also working to develop disease diagnosis systems which can predict diseases or estimate its risk by learning latent factors from various medical records of a large number of patients.

However, the fruits of Big data are not free. We have to resolve many challenges raised by the enormous amount of data. Among others, the biggest challenge is the computational overhead in learning algorithms provided the large-scale datasets. Traditionally, algorithms of linear computation complexity $\mathcal{O}(N)$ per iteration have been considered as an efficient class of algorithms because the number of observations N was not as large as to be a computational challenge. However, the scale that we are facing today easily exceeds the level which the traditional learning algorithm can return a meaningful result within a reasonable time-budget for training. That is, these traditional algorithms became *practically intractable* for modern large-scale problems.

One way to scale up these algorithms is to use a small number of samples (called the minibatch) from the original dataset at every iteration instead of using the entire dataset, for instance, as is performed in the stochastic gradient descent (SGD) algorithm [Bishop, 2006]. In fact, it turned out that this type of algorithms, which requires only a random minibatch of size $n \ll N$ for an update, is almost optimally efficient for large-scale problems [Bottou and Bousquet, 2008]. Another problem regarding the computational inefficiency of the traditional learning algorithms is that they were not designed for distributed computing. Because most of the large-scale datasets being used in industry these days significantly exceed the storage capacity of a single machine, efficient operability in distributed computing systems is not an option anymore but an indispensable component of any large-scale learning algorithm.

In this thesis, we aim to introduce recent advances in these directions, particularly focusing on the Bayesian learning algorithms.

[2]http://www.enlitic.com/

1.1 Bayesian Inference for Machine Learning

There exist two main approaches to formulating machine learning problems. In the *frequentist* approach, one defines a loss function $\mathcal{L}_f(\theta; \mathcal{D})$ by a model parameter θ and the observed dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ in such a way that predictions made by the model $\hat{y}_i(\theta) = f(x_i; \theta)$ can be close to the observed label y_i. Here x_i denotes the feature and y_i the label of i^{th} data point $d_i \in \mathcal{D}$. The mean squared error (MSE) function $\frac{1}{N}\sum_{i=1}^N (y_i - \hat{y}_i(\theta))^2$ is an example of such loss functions which is often used for regression problems.

Then, we run an optimization algorithm (typically iterative) to find a single optimal model parameter θ^* that minimizes the loss function. Because minimizing solely this loss function can cause overfitting, usually an additional term $\Omega(\theta)$ which penalizes complex models is added. As a result, in the frequentist approach, we solve an optimization problem of the following form

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_f(\theta; \mathcal{D}) + \lambda\Omega(\theta). \tag{1.1}$$

Here, λ is the regularization hyperparameter for which a good value has to be set by the user, e.g., using cross-validation. At test time, given a test point x', we make a prediction by $y'(\theta^*) = f(x'; \theta^*)$.

Unlike the frequentist approach where the output of a learning algorithm is a single model θ^*, in the *Bayesian* approach we are interested in obtaining a "distribution" of the model parameter θ given the observation $\mathcal{D}$. This is called the posterior distribution $p(\theta|\mathcal{D})$ and we use the Bayes' rule to obtain it

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{1.2}$$

Here, $p(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(d_i|\theta)$ is the likelihood function of the observation $\mathcal{D}$ parameterized by θ, and $p(\theta)$ is the prior distribution which encodes our prior knowledge on the parameter θ. The denominator, called the evidence or the marginal likelihood, is obtained by $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta)\mathrm{d}\theta$. At test time, the prediction is obtained by integrating out predictions $p(y'|\theta, x')$ w.r.t the posterior distribution $p(\theta|\mathcal{D})$, resulting in the following posterior predictive distribution

$$p(y'|\mathcal{D}) = \int p(y'|\theta, x')p(\theta|\mathcal{D})\mathrm{d}\theta. \tag{1.3}$$

Bayesian inference is usually regarded as computationally more demanding than the frequentist approach because, except some trivial cases, the posterior distribution in Equation (1.2) is intractable since the marginal likelihood $p(\mathcal{D})$ is difficult to compute.

Approximate Inference

To resolve this problem, modern Bayesian inference methods for machine learning rely on the approximate inference methods such as Markov chain Monte Carlo (MCMC) and variational inference. The general idea behind MCMC is that, if we can draw samples from the posterior distribution only by evaluating the numerator[3] $p(\mathcal{D}|\theta)p(\theta)$ in Equation (1.2), the integration in Equation (1.3) can be approximated by the Monte Carlo method (introduced in the next chapter) using a finite number of posterior samples. In the next chapter, we provide a more detail introduction to MCMC.

Another powerful approximate inference algorithm is the variational inference. In variational inference, we introduce a distribution $q(\theta|\phi)$, called the variational distribution, which is easy to evaluate (unlike the true posterior distribution $p(\theta|\mathcal{D})$) and parameterized by ϕ. For example, we can use a normal distribution $q(\theta|\phi) = \mathcal{N}(\theta; \mu, \sigma^2)$ where $\phi = (\mu, \sigma^2)$. The goal

[3]Traditionally, it was assumed that the numerator term is easy to evaluate. However, as we will see in the following chapter, this will not be true for large-scale problems where $|\mathcal{D}|$ is very large.

is then to find an optimal variational parameter ϕ^* which minimizes the discrepancy between the true posterior distribution $p(\theta|\mathcal{D})$ and the variational distribution $q(\theta|\phi)$. Then, we use the optimized variational distribution $q(\theta|\phi^*)$ as a tractable alternative to the true posterior distribution. Using the Kullback-Leibler (KL) divergence as a metric of the discrepancy between two distributions, the problem of obtaining an approximate posterior distribution is converted to an optimization problem of the following objective function

$$\phi^* = \operatorname*{argmin}_{\phi} \mathbb{KL}[q(\theta|\phi)|p(\theta|\mathcal{D})]. \tag{1.4}$$

Note that the variational distribution q has to be flexible/complex enough to approximate the true posterior distribution as close as possible while at the same time remaining computationally tractable, e.g. for the analytic integration of the Equation (1.3).

MCMC and variational inference have advantages and disadvantages which are exclusive each other. MCMC usually provides better performance than variational inference because the MCMC estimate converges to the true value as the sample size increases, whereas in variational inference even the best parameter of a variational distribution cannot exactly model the true posterior distribution because the expressiveness of the variational distribution parameterized by ϕ is usually restricted to a specific family of distributions which may not include the true posterior. However, MCMC can be computationally more demanding than variational inference because collecting a large number of samples can be computational expensive. Besides, when the sample size is not large enough, the MCMC estimates usually show large variance while variational inference is a deterministic optimization process. In this thesis, we only focus on MCMC.

1.2 Advantages of Bayesian Inference

Compared to the frequentist approach, the Bayesian method in general has the following advantages.

1. *Quantifying uncertainty in principled way.* Unlike the loss function minimization of the frequentist method where the goal is to find a single model parameter, the output of Bayesian inference is a probability distribution over the model parameter. Therefore, uncertainty under the model parameter is quantified under the framework of probability theory.

2. *Accurate prediction and overfitting prevention.* In Bayesian inference, the final prediction is based on an (weighted) average of (infinitely) many predictions made by (infinitely) many models drawn from the posterior distribution. In contrast, predictions by frequentist methods rely only on a single model. Thus, Bayesian inference not only improves the prediction accuracy in a similar way as ensemble learning [Dietterich, 2000], but also prevents overfitting because the averaged prediction generalizes well (not settling down on a single model).

3. *Avoiding hyperparameter tuning.* In the Bayesian framework, we can learn also the hyperparameters by placing a hyper-prior distribution on the prior parameters, resulting in a hierarchical Bayesian model. Therefore, the posterior distribution of the model parameter is obtained automatically based on the most probable hyperparameters. Note that this removes the necessity of the time-consuming hyperparameter tuning procedure required in most of the frequentist methods. This is a particularly attractive property for large-scale datasets because each experiment of the cross-validation becomes expensive.

4. *Prior knowledge and side information.* In some applications, we often have useful prior knowledge of the model parameter. Bayesian inference makes us easily incorporate prior knowledge to the model. For example, we can help a learning algorithm avoid visiting some problematic or unrealistic area of the model parameter space by designing a proper prior distribution (e.g., Laplace prior for sparsity). Some useful side-information (e.g., demographic information of users in recommender systems) can also be incorporated in the inference procedure in principled way using the prior distribution [Porteous et al., 2010].

5. *Hierarchical modeling.* A large dataset is often divided into several groups each of which has different statistical properties. For example, in recommender systems, users may have different average values of ratings (e.g., one may give two stars on average while others like to give four stars). Likewise, subsets of the data associated with each user may have different characteristics. Bayesian hierarchical modeling naturally fits well in modeling such phenomena which are prevalent in large-scale problems.

1.3 Large-Scale Machine Learning

Despite such advantages of Bayesian inference, most of the large-scale problems in machine learning have been dominated so far by the frequentist approach. This is because an efficient minibatch based training algorithm, i.e., the stochastic gradient descent (SGD), is easily applicable to frequentist approaches, whereas Bayesian approaches use the entire dataset per iteration.

Specifically, assuming that the data points in $\mathcal{D}$ are i.i.d., we can easily obtain an unbiased estimator of the gradient of the objective function in Equation (1.1) using a minibatch $\mathcal{D}_n$

of size n. That is,

$$\nabla_\theta \mathcal{L}_f(\theta; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f(x_i; \theta), y_i) = \mathbb{E}_{\mathcal{D}_n} \left[\frac{1}{n} \sum_{(x,y) \in \mathcal{D}_n} \nabla_\theta \ell(f(x; \theta), y) \right]. \tag{1.5}$$

The stochastic approximation theory [Robbins and Monro, 1951] says that we can find a local minimum of the objective function using the following update rule

$$\theta_{t+1} \leftarrow \theta_t + \epsilon_t g(\theta_t; \mathcal{D}_n). \tag{1.6}$$

Here, $g(\theta; \mathcal{D}_n) = \frac{1}{n} \sum_{(x,y) \in \mathcal{D}_n} \nabla_\theta \ell(f(x; \theta), y)$ and the step size ϵ_t is required to satisfy the following conditions

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \qquad\qquad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty. \tag{1.7}$$

The success of SGD on the single machine setting has also led to recent advances in parallel/distributed SGD algorithms [Hall et al., 2010, McDonald et al., 2010, Mann et al., 2009, Zinkevich et al., 2010].

On the other hand, the assumption used in most of the Bayesian methods, that the likelihood term $p(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(d_i|\theta)$ is easy to compute, became problematic as N becomes very large. It is only very recent that we have the minibatch based algorithms for Bayesian inference. For variational inference, the stochastic variational Bayes (SVB) [Hoffman et al., 2013] and the stochastic gradient variational Bayes (SGVB) [Kingma and Welling, 2014] are proposed recently. And for MCMC, to the best of our knowledge there has been no minibatch based algorithm before the class of algorithms introduced in this thesis. Therefore, the toolbox of large-scale machine learning has been limited to those of frequentist methods and consequently the powerful advantages of Bayesian inference have not been accessible.

When it comes to scalable Bayesian inference methods for large N, it is particularly interesting to note one specific misconception regarding the relationship between uncertainty and the dataset size. One might argue that,

1. Bayesian inference is all about dealing with uncertainty.
2. But uncertainty will be negligible when a dataset is very large.
3. Thus, the frequentist approach should be good enough for large-scale problems.

The problem of this argument is that it fails to relate the model flexibility (complexity) to the dataset size. That is, when we obtain more observations, we are also likely to have more information which can be modeled by using a more flexible model. Therefore, a proper approach in the case of increasing dataset size is to increase accordingly the model complexity as well so as to capture more information. Otherwise, our model will underfit. That is, a simple model cannot model some important information provided by the increased dataset.

A good example is given by a rule of thumb in choosing a proper number of hidden layers in deep neural networks (or the width of a hidden layer): "increase the number of hidden layers until you see some overfitting and then begin to regularize from that model". As such, in many real-world problems, the need for dealing with uncertainty and the risk of overfitting will still exist regardless of the dataset size, unless the user is satisfied with settling down on sub-optimal performance of a simple model, thus giving up the full power of the available dataset.

Also, in many cases the term "large-scale" is about a situation where the dimension of the model parameter is very large rather than the number of observations. For this type of models, Bayesian inference has traditionally outperformed the frequentist approach [Neal and Zhang, 2006].

1.4 Research Challenges

While the breakthrough by the stochastic gradient Langevin dynamics (SGLD) [Welling and Teh, 2011] has opened a way towards scalable MCMC, in order to be more scalable and practical, further innovations are still required for some of the core challenges. We itemize such important research challenges (RC) below which are also the contributions of the thesis.

- **[RC1] Improving mixing rate**: in SGLD, the stepsize plays the role of a knob which controls trade-off between the mixing rate and the approximation level. One drawback of SGLD here is that we are limited to use relatively small stepsizes to obtain reasonably accurate samples and thus the mixing rate becomes low. Using large step-sizes to increase the mixing rate can result in a large discrepancy between the stationary distribution and the true posterior distribution. Therefore, improving the mixing rate while keeping the stationary distribution not diverge too far from the true posterior distribution is an important problem.

- **[RC2] Efficient distributed SG-MCMC algorithm**: datasets in most of the industrial large-scale problems cannot be stored in a single machine. Besides, it is also desired to reduce the training time by providing multiple machines. Although SGLD is a scalable sampler in the sense that an update iteration can be completed using only a minibatch, it is still limited to the single machine setting. Therefore, in order to be a practical tool for industry sized large-scale problems, developing an efficient distributed SG-MCMC algorithm is important.

- **[RC3] Applications to industry sized large-scale problems**: Although SGLD has opened a way to the minibatch based MCMC, making it practically applicable to industry sized large-scale problems, each of which has its own unique challenges, is not straightforward. For example, in matrix factorization for recommender systems, the parameters have special dependency to only some subset of the observations, and in some social network analysis, we have to deal with a very large number of communities, etc. Among many, (i) latent Dirichlet allocation for topic modeling, (ii) matrix factorization for recommender systems, and (ii) community detection in social networks are among the most important problems on which successful application of the SG-MCMC method can make a large impact.

1.5 Outline of the Thesis

The remainder of the thesis is outlined as follows. As a preliminary chapter, we first start by reviewing the theory and principles of MCMC in Chapter 2.

The following chapters are then divided into two parts. In the first part, we cover the stochastic gradient MCMC algorithms (SG-MCMC). Specifically, in Chapter 3 we introduce the first SG-MCMC algorithm, called the stochastic gradient Langevin dynamics (SGLD), and in Chapter 4 the stochastic gradient Fisher scoring (SGFS) which improves some drawbacks of SGLD. We close the first part in Chapter 5 by introducing an important extension of SGLD toward distributed computing, called D-SGLD.

The second part of the thesis introduces practical large-scale applications of the algorithms introduced in the first part. We first show in Chapter 6 how we obtained the state-of-the-art result using D-SGLD on the distributed inference problem for the latent Dirichlet allocation model which has been one of the most important large-scale problems in machine learning. The matrix factorization for recommender systems which is another important problem is followed in Chapter 7. In particular, this chapter shows that the Bayesian approach can significantly outperform the state-of-the-art frequentist method by adopting our stochastic gradient MCMC method. In Chapter 8, we introduce our last application, a scalable Bayesian inference method for overlapping community detection of very large networks. We then conclude the thesis in Chapter 9.

Chapter 2

Markov Chain Monte Carlo

As the name implies, the Markov chain Monte Carlo (MCMC) method combines the theory of Markov chain and Monte Carlo method: a Markov chain is used to sample from a distribution of our interest (e.g., the posterior distribution), and then the Monte Carlo method uses the samples to approximate an expectation whose exact solution is expensive to compute. In this chapter, we first introduce the theory of the Monte Carlo method and then move to the Markov chain. Then, we review some important traditional MCMC algorithms such as the Metropolis-Hastings algorithm, and discuss why traditional MCMC is not scalable for large-scale problems.

2.1 Monte Carlo Method

In many applications of machine learning, we are interested in computing an expectation $f^* = \mathbb{E}[f(\theta)]$ w.r.t. a distribution $p(\theta)$. The Monte Carlo method is used to approximate the expectation (integration) when computing the exact value is intractable. The general idea of the Monte Carlo method is to approximate the intractable integration as a sum of finite samples drawn from the distribution $p(\theta)$. More specifically, given L samples $\{\theta^{(l)}\}_{l=1}^{L}$ from $p(\theta)$, we estimate the expectation by

$$f^* \approx f_L = \frac{1}{L}\sum_{l=1}^{L} f(\theta^{(l)}). \tag{2.1}$$

For example, in Bayesian inference we can use samples from the posterior distribution to approximate the posterior predictive distribution. That is,

$$\begin{aligned} p(y'|x', \mathcal{D}) = \mathbb{E}_{\theta|\mathcal{D}}[p(y'|x', \theta)] &= \int p(y'|x', \theta)p(\theta|\mathcal{D})\mathrm{d}\theta \\ &\approx \frac{1}{L}\sum_{l=1}^{L} p(y'|x', \theta^{(l)}). \end{aligned} \tag{2.2}$$

Here $\theta^{(l)} \sim p(\theta|\mathcal{D})$.

The Monte Carlo estimator is unbiased because $\mathbb{E}[f_L] = f^*$, and consistent because by the strong law of large numbers it converges almost surely to the true value as the number of samples increases

$$f_L \xrightarrow[L\to\infty]{a.s.} f^*. \tag{2.3}$$

Also, by central limit theorem, the variance of the Monte Carlo estimator is:

$$\mathrm{var}[f_L] = \frac{\mathrm{var}[f]}{L}. \tag{2.4}$$

Thus, the more samples we have, the better accuracy we obtain.

Now the problem is how to obtain samples from a target distribution, especially when we can evaluate only the unnormalized part $\tilde{p}(\theta)$ of the distribution $p(\theta)$. There have been many sampling methods such as the inverse cdf method, rejection sampling, and importance sampling [Bishop, 2006]. However, they either are applicable only to some specific cases (e.g., computing the inverse cdf is not always possible) or perform very inefficiently in high-dimensional spaces (e.g., it is difficult to obtain a good proposal distribution in rejection sampling and importance sampling). The goal of Markov chain Monte Carlo is to resolve these problems.

2.2 Markov Chains

A series of random variables $\theta^{(1)}, \ldots, \theta^{(T)}$ is a first-order Markov chain if the following conditional independence holds for $t = 1, \ldots, T$

$$p(\theta^{(t+1)}|\theta^{(t)}, \ldots, \theta^{(1)}) = p(\theta^{(t+1)}|\theta^{(t)}). \tag{2.5}$$

We can specify a Markov chain by defining the initial distribution $p(\theta^{(0)})$ and the *transition probabilities* $p_t(\theta^{(t+1)}|\theta^{(t)})$ for all t. A Markov chain is called *homogeneous* or *time invariant* when the transition probabilities are the same for all t. A distribution is said to be *invariant* w.r.t. a Markov chain if the transition probabilities do not change the distribution. For a homogeneous Markov chain, a distribution $p^*(\theta)$ is invariant if

$$p^*(\theta') = \sum_{\theta} p(\theta'|\theta)p^*(\theta). \tag{2.6}$$

In MCMC, we want a Markov chain to have a specific invariant distribution, i.e., the distribution whose samples we use for the Monte Carlo estimation (e.g., the posterior distribution). One way to ensure this is to design a transition probability in such a way that it satisfies the *detailed balance* condition defined by

$$p(\theta'|\theta)p^*(\theta) = p(\theta|\theta')p^*(\theta'). \tag{2.7}$$

What the detailed balance condition says is that the rates of flows between two states θ and θ' are always the same in both directions, $\theta \to \theta'$ and $\theta' \to \theta$. Thus, a Markov chain that satisfies the detailed balance is also called *reversible*. It is easy to show that this is a sufficient (but not necessary) condition to have $p^*(\theta)$ as the invariant distribution of a Markov chain

$$\sum_{\theta} p(\theta'|\theta)p^*(\theta) = \sum_{\theta} p(\theta|\theta')p^*(\theta') = p^*(\theta') \sum_{\theta} p(\theta|\theta') = p^*(\theta'). \tag{2.8}$$

Another important property is *ergodicity*. A Markov chain is called ergodic if it converges to the invariant distribution as $t \to \infty$ regardless of the choice of the initial distribution $p(\theta^{(0)})$. An ergodic Markov chain has only one invariant distribution, called *equilibrium distribution*. A homogeneous Markov chain is ergodic subject to some weak restrictions on the invariant distribution and the transition probabilities [Neal93].

Technically, samples generated by an MCMC algorithm are not independent because every state (except the initial state) is dependent on its previous state. Because highly correlated samples can increase the variance of the MCMC estimate (provided a finite number of samples), a good MCMC sampler is required to generate samples of low correlation. One way to measure how much the samples are correlated to each other is to use the following auto-correlation function (ACF) [Murphy, 2012]:

$$\rho_t = \frac{\frac{1}{S-t}\sum_{s=1}^{S-t}(f_s - \bar{f})(f_{s+t} - \bar{f})}{\frac{1}{S-1}\sum_{s=1}^{S}(f_s - \bar{f})^2}. \tag{2.9}$$

Here, $\bar{f} = \frac{1}{S}\sum_{s=1}^{S} f_s$ and t is the time lag. The lower the ACF value is, the more independent the samples are. Also, when the samples of a MCMC method become more independent on their previous states, we say that the MCMC algorithm *mixes* better or has better mixing rate.

Algorithm 1 Metropolis-Hastings algorithm

```
Input: proposal distribution $q(\theta'|\theta)$
    Draw initial state $\theta^{(0)}$ from $p(\theta^{(0)})$
    for $t = 0, 1, \dots, T$ do
        Draw a proposal state $\theta'$ from a proposal distribution $q(\theta^{(t+1)}|\theta^{(t)})$
        Compute the accept-reject probability $\alpha(\theta'|\theta^{(t)})$
        Draw $u$ from a uniform distribution of a range $[0, 1]$.
        if $\alpha_t > u$ then
            Accept: $\theta^{(t+1)} \leftarrow \theta'$
        else
            Reject: $\theta^{(t+1)} \leftarrow \theta^{(t)}$
        end if
    end for
```

2.3 MCMC Algorithms

2.3.1 The Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm is the most generally applicable type of MCMC algorithm. The algorithm works as follows. At iteration t where the current state is $\theta^{(t)}$, we draw a proposal sample θ' from a proposal distribution $q(\theta'|\theta^{(t)})$ which is introduced by the user. Then, the proposed sample θ' is passed to the accept-reject test in which the acceptance probability is computed by

$$\alpha(\theta'|\theta^{(t)}) = \min\left(1, \frac{q(\theta^{(t)}|\theta')\tilde{p}(\theta')}{q(\theta'|\theta^{(t)})\tilde{p}(\theta^{(t)})}\right). \tag{2.10}$$

Note here that we only need to evaluate the unnormalized probability density $\tilde{p}(\theta)$. Given the acceptance probability, we accept the proposed sample θ' as the next state of the chain if a sample from a uniform distribution is less than the acceptance probability α. Otherwise, the current state $\theta^{(t)}$ is copied to the next state. The pseudo-code is described in the Algorithm 1.

It is easy to show that the Metropolis-Hastings algorithm ensures $p(\theta)$ to be the invariant distribution

$$\begin{aligned} p(\theta)q(\theta'|\theta)\alpha(\theta'|\theta) &= \min(p(\theta)q(\theta'|\theta), p(\theta')q(\theta|\theta')) \\ &= \min(p(\theta')q(\theta|\theta'), p(\theta)q(\theta'|\theta)) \\ &= p(\theta')q(\theta|\theta')\alpha(\theta|\theta'). \end{aligned} \tag{2.11}$$

For continuous space, a simple choice for the proposal distribution is to use a Gaussian distribution where the mean is set to the current state and the variance is chosen by the user. This specific algorithm is called the random walk Metropolis (RWM) algorithm. In RWM, it is important to set the variance parameter of the Gaussian proposal distribution properly. Using a small value may give us high acceptance rate but the mixing rate can be poor. Conversely, by using a large variance, the chain can move by a large step when accepted. However, the acceptance rate can be low.

One problem of using a Gaussian proposal distribution is that it does not consider to give a preference to an obvious direction of high acceptance probability which we can actually identify using the gradient of the target probability distribution. We will see an example of this type of algorithm in Section 2.3.3.

2.3.2 Gibbs Sampling

Gibbs sampling is a powerful MCMC algorithm which we can use when it is possible to sample efficiently from a subset (often only a single component) of the posterior random variables while conditioning on the other random variables. Thus, we can see Gibbs sampling as a coordinate-wise sampling method.

More specifically, consider a distribution $p(\boldsymbol{\theta}) = p(\theta_1, \ldots, \theta_D)$ from which we want to sample. We use θ_i to denote i^{th} component of $\boldsymbol{\theta}$, and $\boldsymbol{\theta}_{\setminus i}$ to denote all $\theta_1, \ldots, \theta_D$ except θ_i. Gibbs sampling uses the conditional distribution $p(\theta_i|\boldsymbol{\theta}_{\setminus i})$ as a proposal distribution while alternating the component index i. For example, when $D = 3$, at each iteration t, we iterate sampling from $\theta_1^{(t+1)} \sim p(\theta_1|\theta_2^{(t)}, \theta_3^{(t)})$, $\theta_2^{(t+1)} \sim p(\theta_2|\theta_1^{(t+1)}, \theta_3^{(t)})$, and $\theta_3^{(t+1)} \sim p(\theta_3|\theta_1^{(t+1)}, \theta_2^{(t+1)})$. The pseudo-code of the algorithm is described in Algorithm 2.

Each update of Gibbs sampling preserves the distribution invariant. When we update i^{th} component, the marginal distribution $p(\boldsymbol{\theta}_{\setminus i})$ does not change because we do not update the remaining variables $\boldsymbol{\theta}_{\setminus i}$. And by definition we sample exactly from the conditional distribution $p(\theta_i|\boldsymbol{\theta}_{\setminus i})$. Thus, the joint distribution is invariant.

Gibbs sampling can also be seen as a special case of the Metropolis-Hastings algorithm where the proposal distribution is set to the conditional distribution, i.e., $q(\theta_i|\boldsymbol{\theta}) = p(\theta_i|\boldsymbol{\theta}_{\setminus i})$. Then, the ratio r in the $\min(1, r)$ of the acceptance probability is always 1 and thus we always accept the proposal[1]

$$\frac{p(\theta_i'|\boldsymbol{\theta}_{\setminus i})p(\boldsymbol{\theta}_{\setminus i})}{p(\theta_i|\boldsymbol{\theta}_{\setminus i})p(\boldsymbol{\theta}_{\setminus i})} \times \frac{p(\theta_i|\boldsymbol{\theta}_{\setminus i})}{p(\theta_i'|\boldsymbol{\theta}_{\setminus i})} = 1. \tag{2.12}$$

Here, the second term in the LHS is the ratio of the proposal distributions.

The main limitation of Gibbs sampling is that it must be easy to sample from the conditional posterior distribution. This is often not the case in many applications of practical interest.

[1]Note that, if the parameters are updated in a fixed order, this argument is not satisfied because in this case the reversibility is not satisfied.

Algorithm 2 Gibbs sampling

```
Initialize $\boldsymbol{\theta}$
for $t = 1, \ldots, T$ do
    Sample $\theta_1 \sim p(\theta_1|\boldsymbol{\theta}_{\setminus 1})$
    Sample $\theta_2 \sim p(\theta_2|\boldsymbol{\theta}_{\setminus 2})$
    $\cdots$
    Sample $\theta_D \sim p(\theta_D|\boldsymbol{\theta}_{\setminus D})$
    $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}$
end for
```

2.3.3 Langevin Monte Carlo

One way to design an efficient proposal distribution for the Metropolis-Hastings algorithm is to use the gradient of the target distribution. Langevin Monte Carlo (LMC), a.k.a. the Metropolis-Adjusted Langevin Algorithm (MALA), uses the Langevin dynamics to obtain an efficient proposal distribution. The Langevin dynamics is modeled by the following stochastic differential equation called Langevin equation:

$$\mathrm{d}\theta_t = \frac{1}{2}\nabla_\theta \log p(\theta_t) + \mathrm{d}\eta_t. \tag{2.13}$$

Here $\theta \in \mathbb{R}^D$ and η_t is the D-dimension standard Brownian motion. The Langevin equation is used to model the dynamics of molecular systems whose equilibrium distribution conforms to the distribution $p(\theta)$.

Because this continuous-time dynamics cannot be simulated exactly in digital computers, the following discretized simulation is used for update:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon}{2}\nabla_{\theta_t} \log p(\theta_t) + \eta_t, \qquad \text{where} \quad \eta_t \sim \mathcal{N}(0, \epsilon). \tag{2.14}$$

However, the above discretization leads to some drift (error) from exact simulation, and therefore the equilibrium distribution of the discrete simulation results in a distribution $p_\epsilon(\theta)$ which is not the same as the target distribution $p(\theta)$. To mitigate the discrepancy

Algorithm 3 Langevin Monte Carlo

```
Initialize $\boldsymbol{\theta}$
for $t = 1, \dots, T$ do
    $\eta_t \sim \mathcal{N}(0, \epsilon)$
    Propose $\theta' \leftarrow \theta_t + \frac{\epsilon}{2}\nabla_{\theta_t} \log P(\theta_t) + \eta_t$
    Compute the accept-reject probability $\alpha(\theta'|\theta_t) = \frac{p(\theta')\mathcal{N}\left(\theta_t|\theta' + \frac{\epsilon}{2}\nabla_{\theta'} \log P(\theta'), \epsilon\right)}{p(\theta_t)\mathcal{N}\left(\theta'|\theta_t + \frac{\epsilon}{2}\nabla_{\theta_t} \log P(\theta_t), \epsilon\right)}$
    Draw $u$ from a uniform distribution of a range $[0, 1]$.
    if $\alpha_t > u$ then
        Accept: $\theta^{(t+1)} \leftarrow \theta'$
    else
        Reject: $\theta^{(t+1)} \leftarrow \theta^{(t)}$
    end if
end for
```

between $p_\epsilon(\theta)$ and $p(\theta)$, the Langevin Monte Carlo thus performs the Metropolis-Hastings accept-reject test with the following proposal distribution

$$q(\theta_{t+1}|\theta_t) = \mathcal{N}\left(\theta_{t+1}|\theta_t + \frac{\epsilon}{2}\nabla_{\theta_t} \log p(\theta_t), \epsilon\right). \tag{2.15}$$

Note that as the step size $\epsilon \to 0$, the discretization error becomes negligible and thus the acceptance probability approaches to 1. The pseudo-code is given in Algorithm 3.

2.4 Scalability of Traditional MCMC

The main problem of traditional MCMC algorithms for large-scale problems is that it requires $\mathcal{O}(N)$ computations per iteration. For instance, the Metropolis-Hastings method requires to compute an acceptance probability at every iteration which involves evaluating the posterior distribution $p(\theta)\prod_{i=1}^{N} p(x_i|\theta)$ (up to a normalization constant) in both the numerator and the denominator. Note that here the N will be 1.4 billion if it is the number of Facebook users in 2015 and 253 millions if it is the number of products sold in Amazon USA in 2014. Thus, computing the product (or summation in the log domain) of the likelihood for all the observations will be prohibitively expensive computation. (The accept-reject test is also statistically inefficient because the computation used for the acceptance probability becomes wasteful if a proposal is rejected.) Furthermore, in the Langevin Monte Carlo method, the computation of the gradient is another bottleneck requiring another $\mathcal{O}(N)$ computations.

Although, unlike the Metropolis-Hastings algorithm, Gibbs sampling is free from the expensive accept-reject tests, it has a limitation in its general applicability because in many applications exact sampling from the conditional posterior distribution is not easy.

According to Korattikara [2014], we can analyze the error of traditional MCMC estimators using the variance-bias decomposition. Consider f_T an estimate obtained by using T samples from an MCMC algorithm, i.e., $f_T = \frac{1}{T}\sum_{t=1}^{T} f(\theta^{(t)})$, and f^* the true value, i.e., $f^* = \int p(\theta)f(\theta)\mathrm{d}\theta$. Then, the expected squared error can be decomposed as follows:

$$\begin{aligned}
\mathbb{E}[(f_T - f^*)^2] &= \mathbb{E}[(f_T - \bar{f}_T + \bar{f}_T - f^*)^2] \\
&= \mathbb{E}[(f_T - \bar{f}_T)^2] + 2\mathbb{E}[(f_T - \bar{f}_T)(\bar{f}_T - f^*)] + \mathbb{E}[(\bar{f}_T - f^*)^2] \\
&= \mathbb{E}[(f_T - \bar{f}_T)^2] + (\bar{f}_T - f^*)^2 \\
&= \mathrm{var}[f_T] + \mathrm{bias}[f_T]^2. \qquad (2.16)
\end{aligned}$$

Here, we used $\bar{f}_T$ to denote $\mathbb{E}[f_T]$.

Assuming that collected samples are unbiased samples of the true target distribution $p(\theta)$, we have $\bar{f}_T = f^*$. Therefore, the error of the traditional MCMC depends only on the variance of f_T which collapses to zero when $T \to \infty$ because of $\mathrm{var}[f_T] = \mathrm{var}[f]/T$. However, the problem is that given a finite amount of time it is difficult in practice to collect such large amount of samples for the scale of modern industrial/web datasets. Consequently, the traditional MCMC estimator will have a large error due to the large variance (i.e. small number of samples) despite the asymptotic unbiasedness.

Error
Variance
Bias^2
Error
Variance
Bias^2

Figure 2.1: (left, traditional MCMC): the error solely comes from the variance error because the samples are unbiased, (right, SG-MCMC): large variance error is reduced by sampling much faster at the expense of allowing some bias error.

The key idea behind the SG-MCMC method introduced in this thesis can be understood through this variance-bias analysis. That is, observing that in large-scale problems variance is a dominant factor of the overall error, we propose a class of algorithms which focus more on reducing large variance by significantly speeding-up the sampling process at the cost of allowing some (small) level of bias (that is, the samples are collected from a biased distribution). Consequently, the overall accuracy is expected to improve as illustrated in Figure 2.1.

Chapter 3

Stochastic Gradient Langevin Dynamics

When a dataset has a billion data-cases (as is not uncommon these days) the traditional MCMC algorithms will not even have generated a single (burn-in) sample when a clever learning algorithm based on stochastic gradients may already be making fairly good predictions. In fact, the intriguing results of Bottou and Bousquet [2008] seem to indicate that in terms of "number of bits learned per unit of computation", an algorithm as simple as stochastic gradient descent is almost optimally efficient. We therefore argue that for Bayesian methods to remain useful in an age when the datasets grow at an exponential rate, they need to embrace the ideas of the stochastic optimization literature.

In this chapter, we review the first attempt in this direction, the stochastic gradient Langevin dynamics (SGLD), proposed by Welling and Teh [2011].

3.1 Stochastic Gradient Langevin Dynamics

The stochastic gradient Langevin dynamics (SGLD) is related to the Langevin Monte Carlo (LMC) algorithm which we briefly reviewed in Section 2.3.3. The LMC uses the following discretized simulation of the Langevin equation as a proposal distribution of the Metropolis-Hastings algorithm

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon}{2}\nabla_{\theta_t} \log p(\theta_t|X) + \eta_t, \qquad \text{where } \ \eta_t \sim \mathcal{N}(0, \epsilon). \tag{3.1}$$

where the gradient of the log posterior is

$$\nabla_{\theta_t} \log p(\theta_t|X) = \nabla_{\theta_t} \log p(\theta_t) + \sum_{i=1}^{N} \nabla_{\theta_t} \log p(x_i|\theta_t). \tag{3.2}$$

Although the equilibrium distribution of the Langevin equation in Equation (2.13) is equal to the target distribution $p(\theta|X)$, we cannot exactly simulate this continuous-time dynamics due to the discretization required in the digital computer. To resolve this problem, LMC uses the following discretized Langevin dynamics

$$q(\theta'|\theta) = \mathcal{N}\left(\theta'|\theta + \frac{\epsilon}{2}\nabla_{\theta} \log p(\theta|X), \epsilon\right). \tag{3.3}$$

Then, the Metropolis-Masting accept-reject test is followed to correct the discrepancy between the discretized simulation and the exact continuous-time simulation. As a result, LMC remains as an asymptotically unbiased MCMC algorithm at the expense of the $\mathcal{O}(N)$ computations both in the proposal stage and the accept-reject stage.

SGLD applies two major changes to the LMC in order to obtain scalability desired for the large-scale MCMC. First, exact gradients used in the proposal stage is replaced by stochastic gradients which can be computed using only a small mini-batch of size $n \ll N$. A mini-batch

X_t^n is randomly sampled from the entire dataset at every iteration t. The resulting update equation becomes:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon_t}{2}\left\{\nabla_{\theta_t}\log p(\theta_t) + \frac{N}{n}\sum_{x\in X_t^n}\nabla_{\theta_t}\log p(x|\theta_t)\right\} + \eta_t, \quad \text{where} \quad \eta_t \sim \mathcal{N}(0,\epsilon_t). \tag{3.4}$$

The second modification from LMC to SGLD is to omit the accept-reject test, another source of the $\mathcal{O}(N)$ computations. In other words, in SGLD all states obtained by applying the above update equation are simply accepted. Therefore, the overall computation complexity becomes $\mathcal{O}(n)$. Welling and Teh [2011] showed that, as $t \to \infty$ and the step-size ϵ_t goes to zero at a certain rate satisfying the following condition[1]

$$\sum_{t=1}^{\infty}\epsilon_t = \infty, \qquad \sum_{t=1}^{\infty}\epsilon_t^2 < \infty, \tag{3.5}$$

the update rule in Equation (3.4) generates samples from the posterior distribution despite the absence of the accept-reject test. Sato and Nakagawa [2014] later showed more detailed proof on the convergence properties of SGLD.

The basic idea behind the justification of SGLD as a posterior sampler is as follows. Assuming that the stochastic gradients are unbiased, we can see a stochastic gradient as a sum of the true gradient and the subsampling noise. Then, because the order of the subsampling noise is $\mathcal{O}(\epsilon^2)$ and that of the injected noise is $\mathcal{O}(\epsilon)$ (by the variance of the Gaussian noise), as $\epsilon_t \to 0$ the injected Gaussian noise dominates the subsampling noise of the stochastic gradient. As a result, the remaining true gradient and injected Gaussian noise make SGLD imitate the discrete simulation of the Langevin dynamics in Eqn. (3.1). Noticing that the discretization error in turn disappears as $\epsilon_t \to 0$, the acceptance probability approaches 1 and thus SGLD generates samples from the posterior distribution without the accept-reject test.

[1]A practical rule of setting the step-size is to use a function $\epsilon_t = a(b+t)^{-\gamma}$ where $\gamma \in (0.5, 1]$.

3.1.1 SGLD with a constant step-size

Although in theory SGLD samples from the posterior distribution as $\epsilon_t \to \infty$, such small step-sizes can make the mixing very poor. Therefore, in practice we often stop the annealing procedure at a certain constant step-size ϵ_0 which is large enough to obtain reasonable mixing performance but also small enough to ignore the accept-reject test. Because of this and the absence of the accept-reject test, SGLD allows some discretization error (the larger the constant step-size ϵ_0, the larger the error) and thus remains as an approximate MCMC sampler which samples from an approximate posterior $p_{\epsilon_0}(\theta|X)$. One particularly interesting property of SGLD analyzed by Sato and Nakagawa [2014] is that, subject to some weak conditions, we can use SGLD with a small constant stepsize for the purpose of the approximate Bayesian prediction using posterior averaging.

Also, as we have analyzed in Section 2.4 using the variance-bias decomposition, in practice we are expected to obtain better overall accuracy using SGLD than the traditional asymptotically unbiased sampler because, under the constrained training time-budget in the large-scale data analysis tasks, SGLD can collect a larger number of samples, consequently leading to a great reduction in the variance error. Furthermore, given more time-budget available, we can reduce the constant step-size further to increase the accuracy. Therefore, using the step-size as a knob trading off between mixing performance and asymptotic accuracy, SGLD provides a mechanism to strategically obtain the best performance according to the allowed time-budget.

3.1.2 SGLD during burn-in

MCMC starts from a random initial state and thus requires some iterations until it approaches where significant probability density of the target distribution (e.g., a mode) is located. These iterations are called the *burn-in* period. Because during burn-in period an

MCMC sampler is not sampling from the stationary distribution, we simply discard the samples collected during burn-in. The burn-in in the traditional MCMC is particularly regarded as an inefficient part due to the accept-reject test and the random-walk behavior (not all updates move toward the target distribution). Thus, to resolve this inefficiency of the traditional MCMC algorithms, often a separate optimization algorithm is applied as a preliminary phase to find an important area to start sampling. However, this also raises additional overheads, e.g., because we have to find another set of hyper-parameters for the extra preliminary optimization algorithm. Note that this can be a severe problem when it comes to large-scale datasets because the execution of a single experiment can require a large computation time.

Efficiency during burn-in is another advantage of SGLD. By the above argument of the previous section, when the step-size ϵ_t is large, we can ignore the injected Gaussian noise $\mathcal{O}(\epsilon_t)$ because it is dominated by the subsampling noise $\mathcal{O}(\epsilon_t^2)$ of the stochastic gradient. As a result, during burn-in SGLD behaves like the efficient stochastic gradient ascent optimizer and removes the necessity of running a preliminary optimization algorithm separately.

Chapter 4

Stochastic Gradient Fisher Scoring

Having a good mixing rate is the most important factor of an efficient MCMC algorithm. Especially, when the target distribution has high correlation among the parameters, MCMC usually shows a poor mixing rate and takes a long time to converge. Similar to the Newton method in optimization, incorporating the curvature information of the target distribution is crucial for improving the mixing rate of an MCMC algorithm [Girolami and Calderhead, 2011].

This chapter is based on results from Ahn et al. [2012]. Max Welling proposed the initial version of the algorithm, and Anoop Korattikara and I helped elaborating it. I performed the experiments in section 4.5.1 and 4.5.2. Anoop Korattikara performed the experiment in section 4.5.3.

4.1 Motivation

While SGLD succeeds in (asymptotically) generating samples from the posterior at $O(n)$ computational cost with $(n \ll N)$ it's mixing rate is unnecessarily slow. This can be traced back to its lack of a proper pre-conditioner: SGLD takes large steps in directions of small variance and conversely, small steps in directions of large variance which hinders convergence of the Markov chain. Our work builds on top of Welling and Teh [2011]. We leverage the "Bayesian Central Limit Theorem" which states that when N is large (and under certain conditions) the posterior will be well approximated by a normal distribution. Our algorithm is designed so that for large stepsizes (and thus at high mixing rates) it will sample from this approximate normal distribution, while at smaller stepsizes (and thus at slower mixing rates) it will generate samples from an increasingly accurate (non-Gaussian) approximation of the posterior. Our main claim is therefore that we can trade-in a usually small bias in our estimate of the posterior distribution against a potentially very large computational gain, which could in turn be used to draw more samples and reduce sampling variance.

From an optimization perspective one may view this algorithm as a Fisher scoring method based on stochastic gradients (see e.g. [Schraudolph et al., 2007]) but in such a way that the randomness introduced in the subsampling process is used to sample from the posterior distribution when we arrive at its mode. Hence, it is an efficient optimization algorithm that smoothly turns into a sampler when the correct (statistical) scale of precision is reached.

4.2 Preliminaries and Notation

We will start with some notation, definitions and preliminaries. We have a large dataset X_N consisting of N i.i.d. data-points $\{x_1 ... x_N\}$ and we use a family of distributions parametrized by $\theta \in \mathbb{R}^D$ to model the distribution of the x_i's. We choose a prior distribution $p(\theta)$ and are interested in obtaining samples from the posterior distribution, $p(\theta|X_N) \propto p(X_N|\theta)p(\theta)$.

As is common in Bayesian asymptotic theory, we will also make use of some frequentist concepts in the development of our method. We assume that the true data generating distribution is in our family of models and denote the true parameter which generated the dataset X_N by θ_0. We denote the score or the gradient of the log likelihood w.r.t. data-point x_i by $g_i(\theta) = g(\theta; x_i) = \nabla_\theta \log p(\theta; x_i)$. We denote the sum of scores of a batch of n data-points $X_r = \{x_{r_1} ... x_{r_n}\}$ by $G_n(\theta; X_r) = \sum_{i=1}^{n} g(\theta; x_{r_i})$ and the average by $\overline{g}_n(\theta; X_r) = \frac{1}{n} G_n(\theta; X_r)$. Sometimes we will drop the argument X_r and instead simply write $G_n(\theta)$ and $\overline{g}_n(\theta)$ for convenience.

The covariance of the gradients is called the Fisher information defined as

$$I(\theta) = \mathbb{E}_x[g(\theta; x) g(\theta; x)^T],$$

where $\mathbb{E}_x$ denotes expectation w.r.t the distribution $p(x; \theta)$ and we have used the fact that $\mathbb{E}_x[g(\theta; x)] = 0$. It can also be shown that $I(\theta) = -\mathbb{E}_x[H(\theta; x)]$, where H is the Hessian of the log likelihood.

Since we are dealing with a dataset with samples only from $p(x; \theta_0)$ we will henceforth be interested only in $I(\theta_0)$ which we will denote by I_1. It is easy to see that the Fisher information of n data-points I_n is equal to nI_1. The empirical covariance of the scores computed from a batch of n data-points is called the *empirical* Fisher information [Scott, 2002],

$$V(\theta; X_r) = \frac{1}{n-1} \sum_{i=1}^{n} \left(g_{r_i}(\theta) - \overline{g}_n(\theta) \right) \left(g_{r_i}(\theta) - \overline{g}_n(\theta) \right)^T .$$

Also, it can be shown that $V(\theta_0)$ is a consistent estimator of $I_1 = I(\theta_0)$.

We now introduce an important result in Bayesian asymptotic theory. As N becomes large, the posterior distribution becomes concentrated in a small neighbourhood around θ_0 and becomes asymptotically Gaussian. This is formalized by the Bernstein-von Mises theorem, a.k.a the Bayesian Central Limit Theorem, [Le Cam, 1986], which states that under suitable regularity conditions, $p(\theta | \{x_1 ... x_N\})$ approximately equals $\mathcal{N}(\theta_0, I_N^{-1})$ as $N \to \infty$.

4.3 Stochastic Gradient Fisher Scoring

We are now ready to derive our Stochastic Gradient Fisher Scoring (SGFS) algorithm. The starting point in the derivation of our method is the Stochastic Gradient Langevin Dynamics (SGLD) which has the following update equation:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon C}{2}\left\{\nabla \log p(\theta_t) + N\bar{g}_n(\theta_t; X_n^t)\right\} + \nu, \qquad \text{where} \quad \nu \sim \mathcal{N}(0, \epsilon C). \tag{4.1}$$

Here C is called the preconditioning matrix [Girolami and Calderhead, 2010]. SGLD can sample accurately from the posterior but suffers from a low mixing rate. In Section 4.3.1, we show that it is easy to construct a Markov chain that can sample from a normal approximation of the posterior at any mixing rate. We will then combine these methods to develop our Stochastic Gradient Fisher Scoring (SGFS) algorithm in Section 4.3.2.

4.3.1 Sampling from the Gaussian Approximate Posterior

Since it is not clear how to use Equation (4.1) at high step sizes, we will move away from Langevin dynamics and explore a different approach. As mentioned in Section 4.2, the posterior distribution can be shown to approach a normal distribution, $\mathcal{N}(\theta_0, I_N^{-1})$, as the size of the dataset becomes very large. It is easy to construct a Markov chain which will sample from this approximation of the posterior at any step size. We will now show that the following update equation achieves this:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon C}{2}\left\{-I_N(\theta_t - \theta_0)\right\} + \omega, \qquad \text{where} \quad \omega \sim \mathcal{N}\left(0, \epsilon C - \frac{\epsilon^2}{4} C I_N C\right). \tag{4.2}$$

The update is an affine transformation of θ_t plus injected independent Gaussian noise, ω. Thus if θ_t has a Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$, θ_{t+1} will also have a Gaussian distribution, which we will denote as $\mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$. These distributions are related by:

$$\begin{aligned} \mu_{t+1} &= \left(I - \frac{\epsilon C}{2} I_N\right) \mu_t + \frac{\epsilon C}{2} I_N \theta_0, \\ \Sigma_{t+1} &= \left(I - \frac{\epsilon C}{2} I_N\right) \Sigma_t \left(I - \frac{\epsilon C}{2} I_N\right)^T + \epsilon C - \frac{\epsilon^2}{4} C I_N C. \end{aligned} \tag{4.3}$$

If we choose C to be symmetric, it is easy to see that the approximate posterior distribution, $\mathcal{N}(\theta_0, I_N^{-1})$, is an invariant distribution of this Markov chain. Since Equation (4.2) is not a Langevin equation, it samples from the approximate posterior at large step-size and does not require any MH accept/reject steps. The only requirement is that C should be symmetric and should be chosen so that the covariance matrix of the injected noise in Equation (4.2) is positive-definite.

4.3.2 Stochastic Gradient Fisher Scoring

In practical problems both sampling accuracy and mixing rate are important, and the extreme regimes dictated by both the above methods are very limiting. If the posterior is close to Gaussian (as is usually the case), we would like to take advantage of the high mixing rate. However, if we need to capture a highly non-Gaussian posterior, we should be able to trade-off mixing rate for sampling accuracy. One could also think about doing this in an “anytime” fashion where if the posterior is somewhat close to Gaussian, we can start by sampling from a Gaussian approximation at high mixing rates, but slow down the mixing rate to capture the non-Gaussian structure if more computation becomes available. In other words, one should have the freedom to manage the right trade off between sampling accuracy and mixing rate depending on the problem at hand.

With this goal in mind, we combine the above methods to develop our Stochastic Gradient Fisher Scoring (SGFS) algorithm. We accomplish this using a Markov chain with the following update equation:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon C}{2}\left\{\nabla \log p(\theta_t) + N\overline{g}_n(\theta_t; X_n^t)\right\} + \tau \qquad \text{where} \quad \tau \sim \mathcal{N}(0, Q). \tag{4.4}$$

When the step size is small, we want to choose $Q = \epsilon C$ so that it behaves like the Markov chain in Eqn (4.1). Now we will see how to choose Q so that when the step size is large and the posterior is approximately Gaussian, our algorithm behaves like the Markov chain in Equation (4.2). First, note that if n is large enough for the central limit theorem to hold, we have:

$$\overline{g}_n(\theta_t; X_n^t) \sim \mathcal{N}\left(\mathbb{E}_x[g(\theta_t; x)], \frac{1}{n}\mathrm{Cov}\left[g(\theta_t; x)\right]\right). \tag{4.5}$$

Here $\mathrm{Cov}\left[g(\theta_t; x)\right]$ is the covariance of the scores at θ_t. Using $N\mathrm{Cov}\left[g(\theta_t; x)\right] \approx I_N$ and $N\mathbb{E}_x[g(\theta_t; x)] \approx G_N(\theta_t; X_N)$, we have:

$$\begin{aligned}
&\nabla \log p(\theta_t) + N\overline{g}_n(\theta_t; X_n^t) \\
&\quad \approx \nabla \log p(\theta_t) + G_N(\theta_t; X_N) + \phi \\
&\quad \text{where} \quad \phi \sim \mathcal{N}\left(0, \frac{N I_N}{n}\right).
\end{aligned} \tag{4.6}$$

Now, $\nabla \log p(\theta_t) + G_N(\theta_t; X_N) = \nabla \log p(\theta_t | X_N)$, the gradient of the log posterior. If we assume that the posterior is close to its Bernstein-von Mises approximation, we have $\nabla \log p(\theta_t | X_N) = -I_N(\theta_t - \theta_0)$. Using this in Equation (4.6) and then substituting in Equation (4.4), we have:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon C}{2}\left\{-I_N(\theta_t - \theta_0)\right\} + \psi + \tau \tag{4.7}$$

where,

$$\psi \sim \mathcal{N}\left(0, \frac{\epsilon^2}{4}\frac{N}{n} C I_N C\right) \quad \text{and} \quad \tau \sim \mathcal{N}(0, Q).$$

Comparing Equation (4.7) and Equation (4.2), we see that at high step sizes, we need:

$$\begin{aligned} Q + \frac{\epsilon^2}{4}\frac{N}{n} C I_N C = \epsilon C - \frac{\epsilon^2}{4} C I_N C &\Rightarrow \\ Q = \epsilon C - \frac{\epsilon^2}{4}\frac{N+n}{n} C I_N C. \end{aligned} \tag{4.8}$$

Thus, we should choose Q such that:

$$Q = \begin{cases} \epsilon C & \text{for small } \epsilon \\ \epsilon C - \frac{\epsilon^2}{4}\gamma C I_N C & \text{for large } \epsilon \end{cases}$$

where we have defined $\gamma = \frac{N+n}{n}$. Since ϵ dominates ϵ^2 when ϵ is small, we can choose $Q = \epsilon C - \frac{\epsilon^2}{4}\gamma C I_N C$ for both the cases above. With this, our update equation becomes:

$$\begin{aligned} \theta_{t+1} \leftarrow \theta_t + \frac{\epsilon C}{2}\left\{\nabla \log p(\theta_t) + N\overline{g}_n(\theta_t; X_n^t)\right\} + \tau \\ \text{where} \quad \tau \sim \mathcal{N}\left(0, \epsilon C - \frac{\epsilon^2}{4}\gamma C I_N C\right). \end{aligned} \tag{4.9}$$

Now, we have to choose C so that the covariance matrix of the injected noise in Equation (4.9) is positive-definite. One way to enforce this, is by setting:

$$\epsilon C - \frac{\epsilon^2}{4}\gamma C I_N C = \epsilon C B C \Rightarrow C = 4\left[\epsilon\gamma I_N + 4B\right]^{-1} \tag{4.10}$$

where B is any symmetric positive-definite matrix. Plugging in this choice of C in Equation 4.9, we get:

$$\begin{aligned} \theta_{t+1} \leftarrow \theta_t + 2 & \left[\gamma I_N + \frac{4B}{\epsilon}\right]^{-1} \times \\ & \{\nabla \log p(\theta_t) + N\overline{g}_n(\theta_t; X_t) + \eta\} \\ & \text{where} \quad \eta \sim \mathcal{N}\left(0, \frac{4B}{\epsilon}\right). \end{aligned} \tag{4.11}$$

However, the above method considers I_N to be a known constant. In practice, we use $N\hat{I}_{1,t}$ as an estimate of I_N, where $\hat{I}_{1,t}$ is an online average of the empirical covariance of gradients (empirical Fisher information) computed at each θ_t.

$$\hat{I}_{1,t} = (1 - \kappa_t)\hat{I}_{1,t-1} + \kappa_t V(\theta_t; X_n^t) \tag{4.12}$$

where $\kappa_t = 1/t$. In the supplementary material we prove that this online average converges to I_1 plus $\mathcal{O}(1/N)$ corrections if we assume that the samples are actually drawn from the posterior:

Theorem 4.3.1. *Consider a sampling algorithm which generates a sample θ_t from the posterior distribution of the model parameters $p(\theta|X_N)$ in each iteration t. In each iteration, we draw a random mini-batch of size n, $X_n^t = \{x_{t_1} ... x_{t_n}\}$, and compute the empirical covariance of the scores $V(\theta_t; X_n^t) = \frac{1}{n-1}\sum_{i=1}^{n} \{g(\theta_t; x_{t_i}) - \overline{g}_n(\theta_t)\}\{g(\theta_t; x_{t_i}) - \overline{g}_n(\theta_t)\}^T$. Let V_T be the average of $V(\theta_t)$ across T iterations. For large N, as $T \to \infty$, V_T converges to the Fisher information $I(\theta_0)$ plus $\mathcal{O}(\frac{1}{N})$ corrections, i.e.*

$$\lim_{T\to\infty} \left[V_T \triangleq \frac{1}{T}\sum_{t=1}^{T} V(\theta_t; X_n^t)\right] = I(\theta_0) + \mathcal{O}(\frac{1}{N}). \tag{4.13}$$

Note that this is not a proof of convergence of the Markov chain to the correct distribution. Rather, assuming that the samples are from the posterior, it shows that the online average of the covariance of the gradients converges to the Fisher information (as desired). Thus, it strengthens our confidence that if the samples are almost from the posterior, the learned pre-conditioner converges to something sensible. What we do know is that if we anneal the stepsizes according to a certain polynomial schedule, and we keep the pre-conditioner fixed, then SGFS is a version of SGLD which was shown to converge to the correct equilibrium distribution [Welling and Teh, 2011]. We believe the adaptation of the Fisher information through an online average is slow enough for the resulting Markov chain to still be valid, but a proof is currently lacking. The theory of adaptive MCMC [Andrieu and Thoms, 2008] or two time scale stochastic approximations [Borkar, 1997] might hold the key to such a proof which we leave for future work. Putting it all together, we arrive at Algorithm 4 below.

Algorithm 4 Stochastic Gradient Fisher Scoring (SGFS)

```
Input: $n$, $B$, $\{\kappa_t\}_{t=1:T}$
Output: $\{\theta_t\}_{t=1:T}$
 1: Initialize $\theta_1$, $\hat{I}_{1,0}$
 2: $\gamma \leftarrow \frac{n+N}{n}$
 3: for $t = 1 : T$ do
 4:     Choose random minibatch $X_n^t = \{x_{t_1} ... x_{t_n}\}$
 5:     $\overline{g}_n(\theta_t) \leftarrow \frac{1}{n}\sum_{i=1}^{n} g_{t_i}(\theta_t)$
 6:     $V(\theta_t) \leftarrow$
        $\frac{1}{n-1}\sum_{i=1}^{n}\{g_{t_i}(\theta_t) - \overline{g}_n(\theta_t)\}\{g_{t_i}(\theta_t) - \overline{g}_n(\theta_t)\}^T$
 7:     $\hat{I}_{1,t} \leftarrow (1-\kappa_t)\hat{I}_{1,t-1} + \kappa_t V(\theta_t)$
 8:     Draw $\eta \sim \mathcal{N}[0, \frac{4B}{\epsilon}]$
 9:     $\theta_{t+1} \leftarrow \theta_t +$
        $2\left(\gamma N \hat{I}_{1,t} + \frac{4B}{\epsilon}\right)^{-1}\{\nabla \log p(\theta_t) + N\overline{g}_n(\theta_t) + \eta\}$
10: end for
```

The general method still has a free symmetric positive-definite matrix, B, which may be chosen according to our convenience. Examine the limit $\epsilon \to 0$. In this case our method becomes SGLD with preconditioning matrix B^{-1} and step size ϵ.

If the posterior is Gaussian, as is usually the case when N is large, the proposed SGFS algorithm will sample correctly for arbitrary choice of B even when the step size ϵ is large. However, for some models the conditions of the Bernstein-von Mises theorem are violated and the posterior may not be well approximated by a Gaussian. This is the case for e.g. neural networks and discriminative RBMs, where the identifiability condition of the parameters do not hold. In this case, we have to choose a small ϵ to achieve accurate sampling (see Section 4.5). These two extremes can be combined in a single "anytime" algorithm by slowly annealing the stepsize. For a non-adaptive version of our algorithm (i.e. where we would stop changing $\hat{I}_1$) after a fixed number of iterations) this would according to the results from Welling and Teh [2011] lead to a valid Markov chain for posterior sampling.

We recommend choosing $B \propto I_N$. With this choice, our method is highly reminiscent of "Fisher scoring" which is why we named it "Stochastic Gradient Fisher Scoring" (SGFS). In fact we can think of the proposed updates as a stochastic version of Fisher scoring based on small mini-batches of gradients. But remarkably, the proposed algorithm is not only much faster than Fisher scoring (because it only requires small mini-batches to compute an update), it also samples approximately from the posterior distribution. So the knife cuts on both sides: SGFS is a faster optimization algorithm but also doesn't overfit due to the fact that it switches to sampling when the right statistical scale of precision is reached.

4.4 Computational Efficiency

Clearly, the main computational benefit relative to standard MCMC algorithms comes from the fact that we use stochastic mini-batches instead of the entire dataset at every iteration. However, for a model with a large number of parameters another source of significant computational effort is the computation of the $D \times D$ matrix $\gamma N \hat{I}_{1,t} + \frac{4B}{\epsilon}$ and multiplying its inverse with the mean gradient resulting in a total computational complexity of $\mathcal{O}(D^3)$ per iteration. In the case $n < D$ the computational complexity per iteration can be brought down to $\mathcal{O}(nD^2)$ by using the Sherman-Morrison-Woodbury equation. A more numerically stable alternative is to update Cholesky factors [Seeger, 2004].

In case even this is infeasible one can factor the Fisher information into k independent blocks of variables of, say size d, in which case we have brought down the complexity to $\mathcal{O}(kd^3)$. The extreme case of this is when we treat every parameter as independent which boils down to replacing the Fisher information by a diagonal matrix with the variances of the individual parameters populating the diagonal. While for a large stepsize this algorithm will not sample from the correct Gaussian approximation, it will still sample correctly from the posterior for very small stepsize. In fact, it is expected to do this more efficiently than SGLD which does not rescale its stepsizes at all. We have used the full covariance algorithm (SGFS-f) and the diagonal covariance algorithm (SGFS-d) in the experiment section.

4.5 Experiments

Below we report experimental results where we test SGFS-f, SGFS-d, SGLD, SGD and HMC on three different models: logistic regression, neural networks and discriminative RBMs. The experiments share the following practice in common. Stepsizes for SGD and SGLD are always selected through cross-validation for at least five settings. The minibatch size n is set to either 300 or 500, but the results are not sensitive to the precise value as long as it is large enough for the central limit theorem to hold (typically, $n > 100$ is recommended). Also, we used $\kappa_t = \frac{1}{t}$.

4.5.1 Logistic Regression

A logistic regression model (LR) was trained on the MNIST dataset for binary classification of two digits 7 and 9 using a total of 10,000 data-items. We used a 50 dimensional random projection of the original features and ran SGFS with $\lambda = 1$. We used $B = \gamma I_N$ and tested the algorithm for a number of α values (where $\alpha = \frac{2}{\sqrt{\epsilon}}$). We ran the algorithm for 3,000 burn-

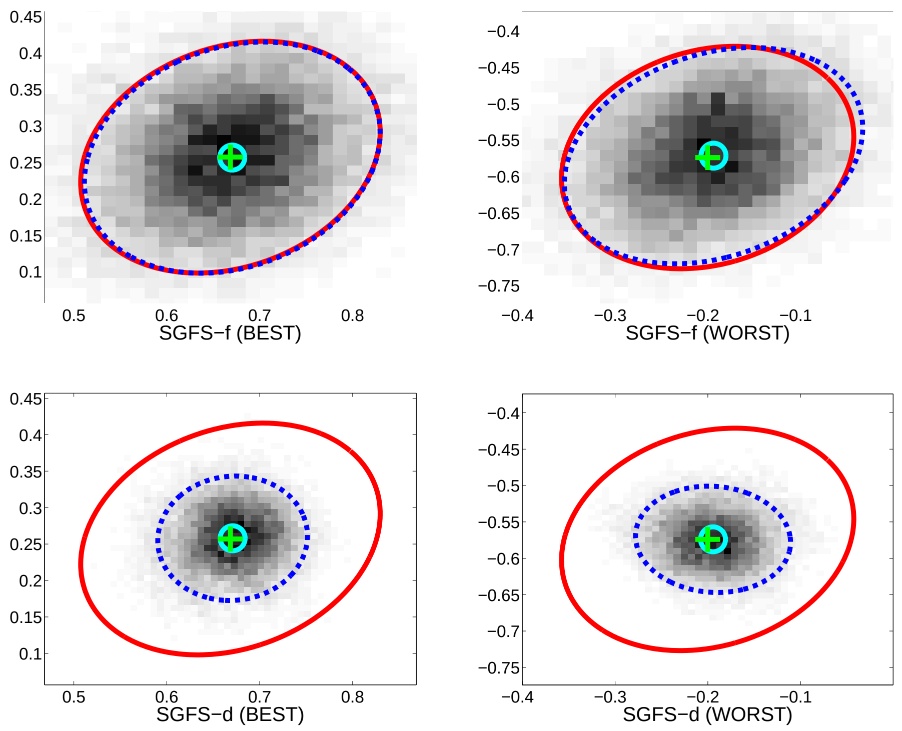


Figure 4.1: 2-d marginal posterior distributions for logistic regression. Grey colors correspond to samples from SGFS. Red solid and blue dotted ellipses represent iso-probability contours at two standard deviations away from the mean computed from HMC and SGFS, respectively. Top plots are the results for SGFS-f and bottom plots represent SGFS-d. Plots on the left represent the 2-d marginals with the smallest difference between HMC and SGFS while the plots on the right represent the 2-d marginals with the largest difference. Value for α is 0 meaning that no additional noise was added.

in iterations and then collected 100,000 samples. We compare the algorithm to Hamilton Monte Carlo sampling [Neal, 1993] and to SGLD [Welling and Teh, 2011]. For HMC, the "leapfrogstep" size was adapted during burn-in so that the acceptance ratio was around 0.8. For SGLD we also used a range of fixed stepsizes.

In figure 4.1 we show 2-d marginal distributions of SGFS compared to the ground truth from a long HMC run where we used $\alpha = 0$ for SGFS. From this we conclude that even for the largest possible stepsize the fit for SGFS-f is almost perfect while SGFS-d underestimates the variance in this case (note however that for smaller stepsizes (larger α) SGFS-d becomes very similar to SGLD and is thus guaranteed to sample correctly albeit with a low mixing rate).

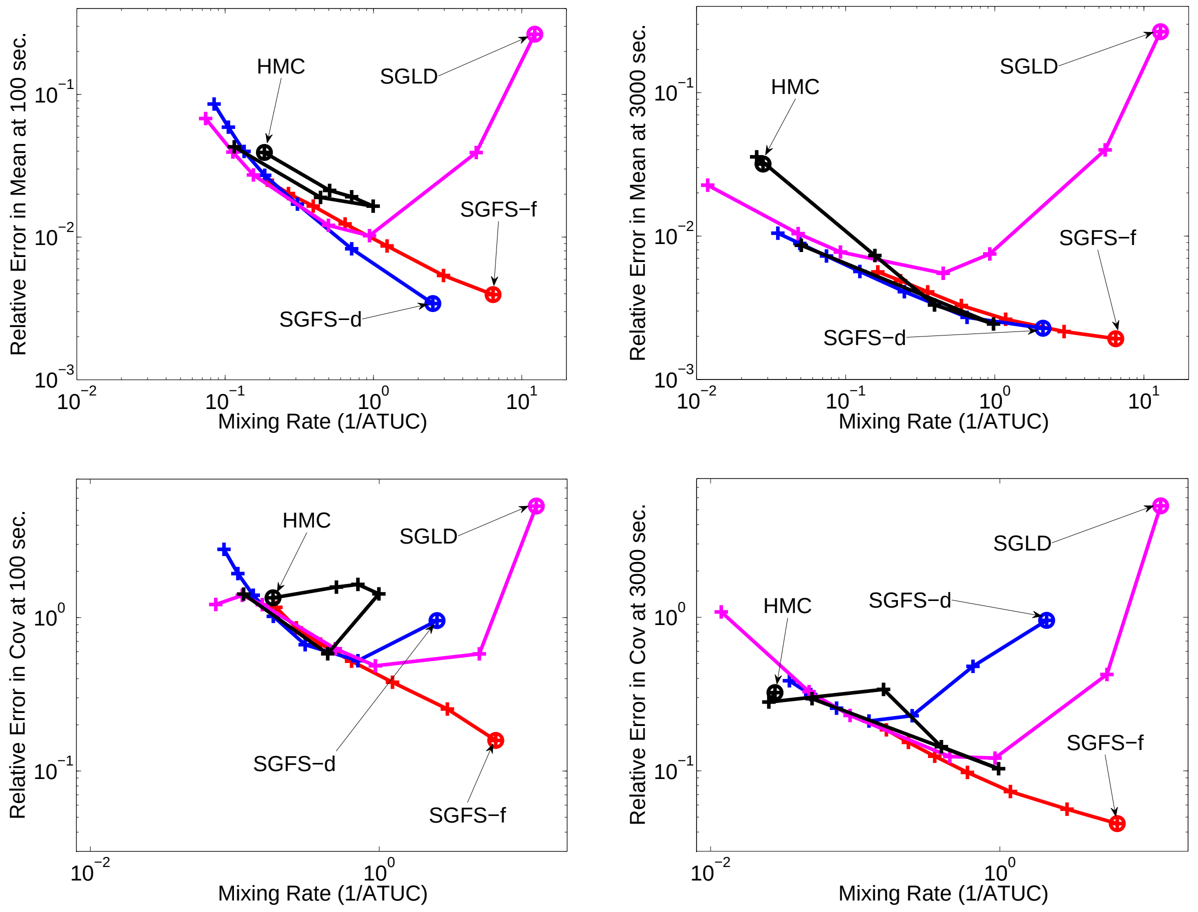


Figure 4.2: Final error of logistic regression at time T versus mixing rate for the mean (top) and covariance (bottom) estimates after 100 (left) and 3000 (right) seconds of computation. See main text for detailed explanation.

Next, we studied the inverse autocorrelation time per unit computation (ATUC)[1] averaged over the 51 parameters and compared this with the relative error after a fixed amount of computation time. The relative error is computed as follows: first compute the mean and covariance of the parameter samples up to time t : $\overline{\theta^t} = \frac{1}{t}\sum_{t'=1}^{t}\theta_{t'}$ and $C^t = \frac{1}{t}\sum_{t'=1}^{t}(\theta_{t'} - \overline{\theta^t})(\theta_{t'} - \overline{\theta^t})^T$. We do the same for the long HMC run which we indicate with $\overline{\theta^\infty}$ and C^∞. Finally we compute

$$E_{1t} = \frac{\sum_i |\overline{\theta_i^t} - \overline{\theta_i^\infty}|}{\sum_i |\overline{\theta_i^\infty}|}, \quad E_{2t} = \frac{\sum_{ij} |C_{ij}^t - C_{ij}^\infty|}{\sum_{ij} |C_{ij}^\infty|}. \tag{4.14}$$

[1]ATUC = Autocorrelation Time × Time per Sample. Autocorrelation time is defined as $1 + 2\sum_{s=1}^{\infty}\rho(s)$ with $\rho(s)$ the autocorrelation at lag s Neal [1993].

In Figure 4.2 we plot the "Error at time T" for two values of T (T=100, T=3000) as a function of the *inverse* ATUC, which is a measure of the mixing rate. Top plots show the results for the mean and bottom plots for the covariance. Each point denoted by a cross is obtained from a different setting of parameters that control the mixing rate: $\alpha = [0, 1, 2, 3, 4, 5, 6]$ for SGFS, stepsizes $\epsilon = [1e-3, 5e-4, 1e-4, 5e-5, 1e-5, 5e-6, 1e-6]$ for SGLD, and number of leapfrog steps $s = [50, 40, 30, 20, 10, 1]$ for HMC. The circle is the result for the fastest mixing chain.

For SGFS and SGLD, if the slope of the curve is negative (downward trend) then the corresponding algorithm was still in the phase of reducing error by reducing sampling variance at time T. However, when the curve bends upwards and develops a positive slope the algorithm has reached its error floor corresponding to the approximation bias. The situation is different for HMC, (which has no bias) but where the bending occurs because the number of leapfrog steps has become so large that it is turning back on itself. HMC is not faring well because it is computationally expensive to run (which hurts both its mixing rate and error at time T). We also observe that in the allowed running time SGFS-f has not reached its error floor (both for the mean and the covariance). SGFS-d is reaching its error floor only for the covariance (which is consistent with Figure 4.1 bottom) but still fares well in terms of the mean. Finally, for SGLD we clearly see that in order to obtain a high mixing rate (low ATUC) it has to pay the price of a large bias. These plots clearly illustrate the advantage of SGFS over both HMC as well as SGLD.

4.5.2 SGFS on Neural Networks

We also applied our methods to a 3 layer neural network (NN) with logistic activation functions. Below we describe classification results for two datasets.

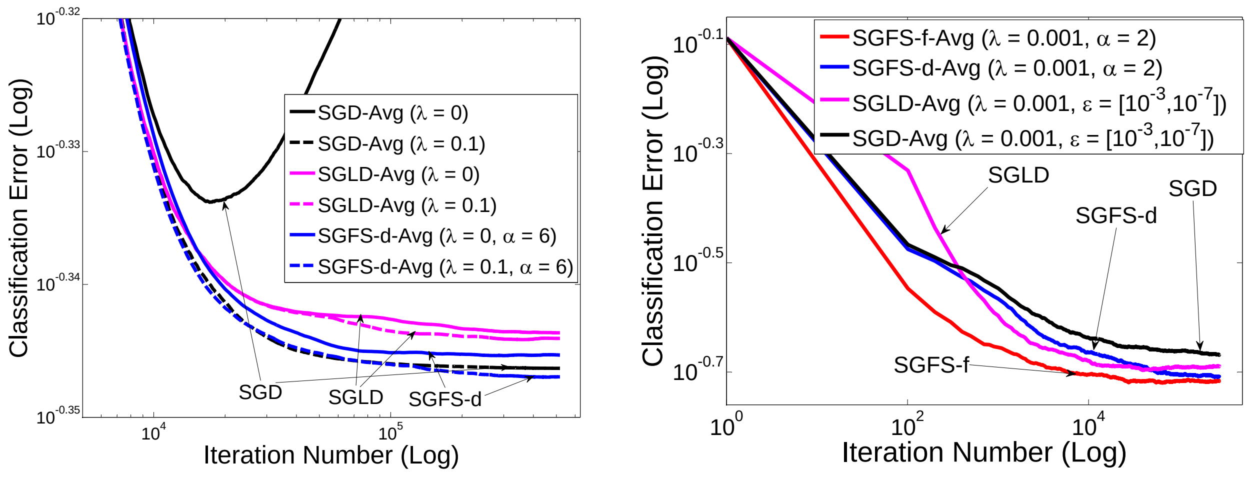


Figure 4.3: Test-set classification error of neural networks trained with SGFS-f, SGFS-d, SGLD and SGD on the HHP dataset (left) and the MNIST dataset (right)

Heritage Health Prize (HHP)

The goal of this competition is to predict how many days between $[0-15]$ a person will stay in a hospital given his/her past three years of hospitalization records[2]. We used the same features as the team *market makers* that won the first milestone prize. Integrating the first and second year data, we obtained 147,473 data-items with 139 feature dimensions and then used a randomly selected 70% for training and the remainder for testing. NNs with 30 hidden units were used because more hidden units did not noticeably improved the results. Although we used $\alpha = 6$ for SGFS-d, there was no significant difference for values in the range $3 \leq \alpha \leq 6$. However, $\alpha < 3$ didn't work for this dataset due to the fact that many features had values 0.

For SGD, we used stepsizes from a polynomial annealing schedule $a(b+t)^{-\delta}$. Because the training error decreased slowly in a valid range $\delta = [0.5, 1]$, we used $\delta = 3$, $a = 10^{14}$, $b = 2.2 \times 10^5$ instead which was found optimal through cross-validation. (This setting reduced the stepsize from 10^{-2} to 10^{-6} during 1e+7 iterations). For SGLD, $a = 1$, $b = 10^4$, and $\delta = 1$ reducing the step size from 10^{-4} to 10^{-6} was used. Figure 4.3 (left) shows the

[2]http://www.heritagehealthprize.com

classification errors averaged over the posterior samples for two regularizer values, $\lambda = 0$ and the best regularizer value λ found through cross-validation. First, we clearly see that SGD severely overfits without a regularizer while SGLD and SGFS prevent it because they average predictions over samples from a posterior mode. Furthermore, we see that when the best regularizer is used, SGFS (marginally) outperforms both SGD and SGLD. The result from SGFS-d submitted to the actual competition leaderboard gave us an error of 0.4635 which is comparable to 0.4632 obtained by the milestone winner with a fine-tuned Gradient Boosting Machine.

Character Recognition

We also tested our methods on the MNIST dataset for 10 digit classification which has 60,000 training instances and 10,000 test instances. In order to test with SGFS-f, we used inputs from 20 dimensional random projections and 30 hidden units so that the number of parameters equals 940. Moreover, we increased the mini-batch size to 2,000 to reduce the time required to reach a good approximation of the 940×940 covariance matrix. The classification error averaged over the samples is shown in Figure 4.3 (right). Here, we used a small regularization parameter of $\lambda = 0.001$ for all methods as overfitting was not an issue. For SGFS, $\alpha = 2$ is used while for both SGD and SGLD the stepsizes were annealed from 10^{-3} to 10^{-7} using $a = 1$, $b = 1000$, and $\gamma = 1$.

4.5.3 Discriminative Restricted Boltzmann Machine (DRBM)

We trained a DRBM [Larochelle and Bengio, 2008] on the KDD99 dataset which consists of 4,898,430 datapoints with 40 features, belonging to a total of 23 classes. We first tested the classification performance by training the DRBM using SGLD, SGFS-f, SGFS-d and SGD. For this experiment the dataset was divided into a 90% training set, 5% validation and 5%

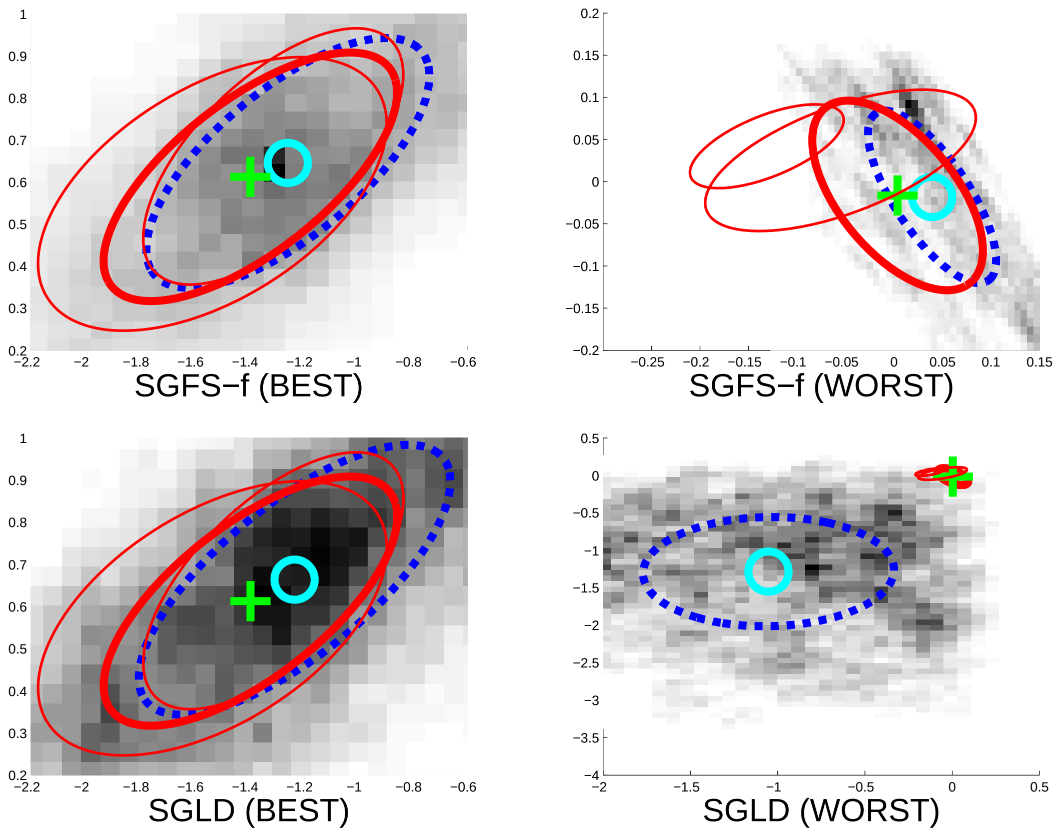


Figure 4.4: 2-d marginal posterior distributions of DRBM. Grey colors correspond to samples from SGFS/SGLD. Thick red solid lines correspond to iso-probability contours at two standard deviations away from the mean computed from HMC samples. Thin red solid lines correspond to HMC results based on subsets of the samples. The thick blue dashed lines correspond to SGFS-f (top) and SGLD (bottom) runs. Plots on the left represent the 2-d marginals with the smallest difference between HMC and SGFS/SGLD while the plots on the right represent the 2-d marginals with the largest difference.

test set. We used 41 hidden units giving us a total of 2647 parameters in the model. We used $\lambda = 10$ and $B = \gamma I_N$. We tried 6 different (α, ϵ) combinations for SGFS-f and SGFS-d and tried 18 annealing schedules for SGD and SGLD, and used the validation set to pick the best one. The best results were obtained with an α value of 8.95 for SGFS-f and SGFS-d, and $[a = 0.1,\ b = 100000,\ \delta = 0.9]$ for SGD and SGLD. We ran all algorithms for 100,000 iterations. Although we experimented with different burn-in iterations, the algorithms were insensitive to this choice. The final error rates are given in table 4.1 from which we conclude that the samplers based on stochastic gradients can act as effective optimizers whereas HMC

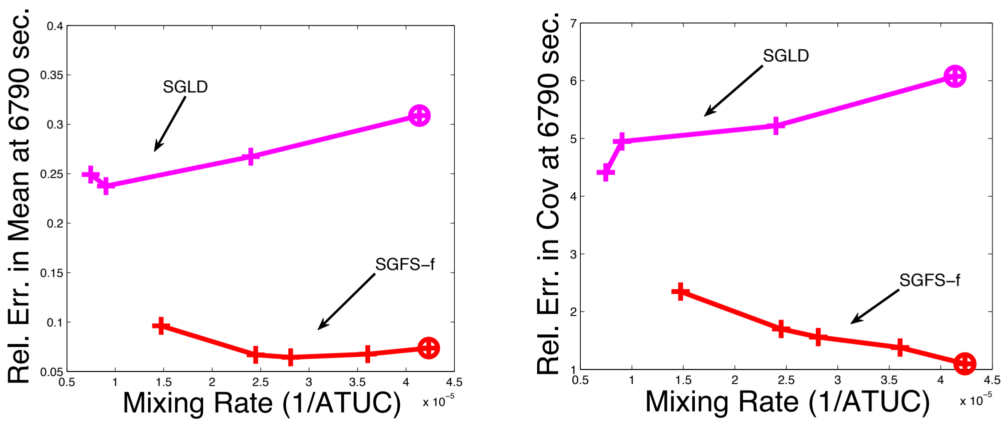


Figure 4.5: Final error for DRBM at time T versus mixing rate for the mean (left) and covariance (right) estimates after 6790 seconds of computation on a subset of KDD99.

SGD	SGLD	SGFS-d	SGFS-f
8.010^{-4}	6.610^{-4}	4.210^{-4}	4.410^{-4}

Table 4.1: Final test error rate on the KDD99 dataset.

on the full dataset becomes completely impractical because it has to compute 11.7 billion gradients per iteration which takes around 7.5 minutes per sample (4408587 datapoints $\times$ 2647 parameters).

To compare the quality of the samples drawn after burn-in, we created a 10% subset of the original dataset. This time we picked only the 6 most populous classes. We tested all algorithms with 41, 10 and 5 hidden units, but since the posterior is highly multi-modal, the different algorithms ended up sampling from different modes. In an attempt to get a meaningful comparison, we therefore reduced the number of hidden units to 2. This improved the situation to some degree, but did not entirely get rid of the multi-modal and non-Gaussian structure of the posterior. We compare results of SGFS-f/SGLD with 30 independent HMC runs, each providing 4000 samples for a total of 120,000 samples. Since HMC was very slow (even on the reduced set) we initialized at a mode and used the Fisher information at the mode as a pre-conditioner. We used 1 leapfrog step and tuned the step-size to get an acceptance rate of 0.8. We ran SGFS-f with $\alpha = [2, 3, 4, 5, 10]$ and SGLD with fixed step sizes of [5e-4, 1e-4, 5e-5, 1e-5, 5e-6]. Both algorithms were initialized at the same mode and ran for 1 million iterations. We looked at the marginal distributions of the top 25 pairs of variables which had the highest correlation coefficient. In Figure 4.4 (top-left and bottom-left) we show a set of parameters where both SGFS-f and SGLD obtained an accurate estimate of the marginal posterior. In Figure 4.4 (top-right and bottom-right) we show an example where SGLD failed. The thin solid red lines correspond to HMC runs computed from various subsets of the samples, whereas the thick solid red line is computed using the all samples from all HMC runs. We have shown marginal posterior estimates of the SGFS-f/SGLD algorithms with a thick dashed blue ellipse. After inspection, it seemed that the posterior structure was highly non-Gaussian with regions where the probability very sharply decreased. SGLD regularly stepped into these regions and then got catapulted away due to the large gradients there. SGFS-f presumably avoided those regions by adapting to the local covariance structure. We found that in this region even the HMC runs are not consistent with one another. Note that the SGFS-f contours seem to agree with the HMC contours as much as the HMC contours agree with the results of its own subsets, in both the easy and the hard case.

Finally, we plot the error after 6790 seconds of computation versus the mixing rate. Figure 4.5-left shows the results for the mean and the right for the covariance (for an explanation of the various quantities see discussion in Section 4.5.1). We note again that SGLD incurs a significantly larger approximation bias at the same mixing rate as SGFS-f.

4.6 Discussion

In this chapter, we have introduced a novel method, “Stochastic Gradient Fisher Scoring” (SGFS) for approximate Bayesian learning. The main idea is to use stochastic gradients in the Langevin equation and leverage the central limit theorem to estimate the noise induced by the subsampling process. This subsampling noise is combined with artificially injected noise and multiplied by the estimated inverse Fisher information matrix to approximately sample from the posterior. This leads to the following desirable properties.

- Unlike regular MCMC methods, SGFS is fast because it uses only stochastic gradients based on small mini-batches to draw samples.
- Unlike stochastic gradient descent, SGFS samples (approximately) from the posterior distribution.
- Unlike SGLD, SGFS samples from a Gaussian approximation of the posterior distribution (that is correct for $N \to \infty$) for large step-sizes.
- By annealing the stepsize, SGFS becomes an anytime method capturing more non-Gaussian structure with smaller step-sizes but at the cost of slower mixing.
- During its burn-in phase, SGFS is an efficient optimizer because like Fisher scoring and Gauss-Newton methods, it is based on the natural gradient.

For an appropriate annealing schedule, SGFS thus goes through three distinct phases: 1) during burn-in we use a large stepsize and the method is similar to a stochastic gradient version of Fisher scoring, 2) when the stepsize is still large, but when we have reached the mode of the distribution, SGFS samples from the asymptotic Gaussian approximation of the posterior, and 3) when the stepsize is further annealed, SGFS will behave like SGLD with a pre-conditioning matrix and generate increasingly accurate samples from the true posterior.

Chapter 5

Distributed Stochastic Gradient Langevin Dynamics

In the previous two chapters, we introduced stochastic gradient MCMC methods which make it possible to complete each iteration only using a random mini-batch of the dataset. However, considering that many large-scale datasets cannot be stored in a single machine, it is an important direction to develop an efficient "distributed" stochastic gradient MCMC method.

This chapter introduces the first fully distributed MCMC algorithm based on stochastic gradients. We argue that stochastic gradient MCMC algorithms are particularly suited for distributed inference because individual chains can draw minibatches from their local pool of data for a flexible amount of time before jumping to or syncing with other chains. This greatly reduces communication overhead and allows adaptive load balancing.

In this work, under the guidance of Max Welling, I developed the algorithm and performed the experiments . Babak Shahbaba provided important comments and discussions and helped writing.

5.1 Motivation

The most straightforward, embarrassingly parallel implementation would be to copy the full dataset to each worker, run separate Markov chains and use their results as independent samples (see e.g. Wilkinson [2006], Laskey and Myers [2003], Ahn et al. [2013]). However, the size of modern day datasets can be so large that a single machine cannot store the full dataset. In this case, one can still parallelize most MCMC algorithms by performing data-specific computations (e.g. the gradient of the log-probability for one data-case) locally on each relevant worker and combining these computations in a master server. These methods however lead to very high communication costs.

We argue that MCMC algorithms based on stochastic mini-batches have a key property that make them ideally suited for parallelization, *namely that each Markov chain can independently generate samples for a variable amount of time*, which can later be combined. The reason is that each chain can draw mini-batches from its local pool of data in order to generate samples. Chains must jump to other machines (synchronously or asynchronously) in order to generate unbiased estimates of the posterior in the limit, but the time spend on each worker is flexible provided that the chain's hyper-parameters are properly adjusted to remove potential bias. This flexibility leads to less communication (because chains can run longer on individual workers) and entirely removes the problem that fast workers are blocked by slow workers because they depend on their results in order to proceed.

5.2 Preliminaries and Notations

Let $X = \{x_1, \ldots, x_N\}$ be a dataset of N i.i.d. data points assumed to be sampled from a parameterized distribution $p(x|\theta)$ where $\theta \in \mathbb{R}^d$ has a prior distribution $p(\theta)$. We are interested in collecting samples from the posterior distribution $p(\theta|X) \propto p(X|\theta)p(\theta)$. As discussed above, we assume that the dataset X is too large to reside in a single machine. Therefore, it is partitioned into S subsets, called *shards*: $X_1, \ldots, X_S$ such that $X = \cup_s X_s$ and $N = \sum_s N_s$. We assign shard $X_s = \{x_1^s, \ldots, x_{N_s}^s\}$ to worker s, where $s = 1, \ldots, S$. We refer to the posterior distribution based on a specific shard as *local posterior*: $p(\theta|X_s) \propto p(X_s|\theta)p(\theta)$.

The score function or the gradient of the log likelihood given a data point x is denoted by $g(\theta; x) = \nabla_\theta \log p(\theta; x)$. We also denote a mini-batch of n data points by X^n when sampled from X and by X_s^n when sampled from shard X_s. Additional time index t is used sometimes to distinguish mini-batches sampled over iterations: $X_{s,t}^n$. The sum and mean of scores over all elements of a set, X, are denoted by $G(\theta; X) = \sum_{x \in X} g(\theta; x)$ and $\bar{g}(\theta; X) = \frac{1}{|X|} G(\theta; X)$ respectively. We now review two approaches to scale up MCMC algorithms; one by using mini-batches and the other by using distributed computational resources.

The stochastic gradient Langevin dynamics (SGLD) proposed by Welling and Teh [2011] is the first sequential mini-batch-based MCMC algorithm. In SGLD, the parameters are updated as follows:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon_t}{2}\{\nabla \log p(\theta_t) + N\bar{g}(\theta_t; X_t^n)\} + \nu_t. \tag{5.1}$$

We can generalize the SGLD update rule in Eqn. (5.1) by replacing the mean score $\bar{g}(\theta_t; X_t^n)$ to a general form of score estimator $f(\theta_t, Z; X)$, where Z is a set of auxiliary random variables associated with the estimator. According to Welling and Teh [2011], an estimator $f(\theta_t, Z; X)$ is guaranteed to converge to the correct posterior if (i) $f(\theta_t, Z; X)$ is an unbiased estimator of $\bar{g}(\theta_t; X) = \frac{1}{N}\sum_{x \in X} g(\theta_t; x)$ (assuming the variance of f is finite) and (ii) the step size is annealed to zero by a schedule satisfying $\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$.

Definition 1. We define an estimator $f(\theta, Z; X)$ as a *valid SGLD estimator* if it is an unbiased estimator of $\bar{g}(\theta; X)$, i.e., $\mathbb{E}_Z[f(\theta, Z; X)] = \bar{g}(\theta; X)$, where $\mathbb{E}_Z$ denotes expectation w.r.t. the distribution $p(Z; X)$, and it has finite variance $\mathbb{V}_Z[f(\theta, Z; X)] < \infty$.

5.3 SGLD on Partitioned Datasets

We begin the exposition of our algorithm with the following question: *"Is an SGLD algorithm that samples mini-batches from randomly chosen local shards valid?"* The number of possible combinations of mini-batches that can be generated by this procedure is significantly smaller set than that of the standard SGLD. The answer will clearly depend on quantities like the shard sizes and shard selection probabilities. We now introduce an estimator $\bar{g}_d$ in the proposition below as an answer to the above question (the proof is provided in the Appendix).

Proposition 5.3.1. *For each shard $s = 1, \dots, S$, given the shard size, N_s, and the normalized shard selection frequency, q_s, such that $N_s > 0$, $\sum_{s=1}^{S} N_s = N$, $q_s \in (0, 1)$, and $\sum_{s=1}^{S} q_s = 1$, the following estimator is a valid SGLD estimator,*

$$\bar{g}_d(\theta; X_s^n) \overset{def}{=} \frac{N_s}{N q_s} \bar{g}(\theta; X_s^n) \tag{5.2}$$

where shard s is sampled by a scheduler $h(\mathcal{Q})$ with frequencies $\mathcal{Q} = \{q_1, \dots, q_S\}$.

For example, we can (1) choose a shard by sampling $s \sim h(\mathcal{Q}) = \text{Category}(q_1, \dots, q_S)$, (2) sample a mini-batch X_s^n from the selected shard, (3) compute mean score $\bar{g}(\theta; X_s^n)$ using that mini-batch, and then (4) multiply the mean score by $\frac{N_s}{N q_s}$ to correct the bias. Then, the resulting SGLD update rule becomes

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon_t}{2} \left\{ \nabla \log p(\theta_t) + \frac{N_{s_t}}{q_{s_t}} \bar{g}(\theta_t; X_{s_t}^n) \right\} + \nu_t. \tag{5.3}$$

We can interpret this as a correction to the step sizes for the $\bar{g}(\theta_t; X^n_{s_t})$ term. That is, the algorithm takes larger steps for shards that are relatively larger in size and/or used less frequently than others. This implies that every data-case contributes equally to the mixing of the chain. Note that $\mathcal{Q}$ represent free parameters that we can choose depending on the system properties.

5.4 Distributed Stochastic Gradient Langevin Dynamics

5.4.1 Traveling Worker Parallel Chains

Now assume that the shards are distributed between the workers, so from now on selecting shard s is equivalent to choosing worker s. We note that running the above algorithm occupies only a single worker at a time. Therefore, assuming single-core workers, it is possible to run C $(\leq S)$ independent and valid SGLD chains *in parallel*, i.e., one chain per worker.

This approach, however, has some shortcomings. First, the communication cycle is still short $\mathcal{O}(n)$ because each chain is required to jump to a new worker at every iteration. Second, it can suffer from the block-by-the-slowest problem if its next scheduled worker is still occupied by another chain due to workers' imbalanced response delays. The *response delay*, denoted by d_s, is defined as the elapsed time that worker s spends to process a $\mathcal{O}(n)$ workload. In the following sections, we present our method to address these issues.

5.4.2 Distributed Trajectory Sampling

To deal with the "short-communication-cycle" problem, we propose to use *trajectory sampling*: instead of jumping to another worker at every iteration, each chain c takes τ consecutive updates in each visit to a worker. Then, after τ updates, only the last (τth) state is passed to the next worker of the chain. Trajectory sampling reduces communication overhead by increasing the communication cycle from $\mathcal{O}(n)$ to $\mathcal{O}(\tau n)$. Furthermore, instead of transferring all samples collected over a trajectory to the master, we can store them in a distributed way by caching each trajectory at its corresponding worker. This keeps the packet size at $\mathcal{O}(1)$ regardless of the trajectory length, and mitigates the memory problem caused by storing many high-dimension samples at a single machine.

In trajectory sampling for parallel chains, we employ a scheduler $h_c(\mathcal{Q})$ for each chain c to choose the *next worker* from which the next trajectory is sampled. Note here that the scheduler is now called with an interval τ. Because there are a total of C such schedulers (one per chain), the schedulers should avoid two situations in order to be efficient: (1) *collision* (i.e., multiple chains visit a worker at the same time), and (2) *jump-in-place* (i.e., jumping to the current worker) can negatively affect mixing across shards. One way to avoid these issues is to set $\mathcal{Q}$ uniform, and simply use a random permutation (or, cyclic rotation) to assign chains to workers. That is, we can sample the chain-to-worker assignments by $(s^1, \ldots, s^C) \sim (h_1(\mathcal{Q}), \ldots, h_C(\mathcal{Q})) = \textsc{randperm}(S)$. Here, s^c denotes a worker that chain c is scheduled to visit; we assume $C = S$ for simplicity.

Similar to the effect of step sizes in standard SGLD, trajectory lengths can also be used to control the level of approximation by trading off computation time with asymptotic accuracy. As both the trajectory length and the annealed step sizes $\{\epsilon_t\}$ can affect the equilibrium distribution of the chain, we consider first that ϵ is fixed. Then, with a long trajectory, we can reduce the communication overhead at the cost of some loss in asymptotic accuracy.

In fact, it is not difficult to see that in this case our method samples from a mixture of local posteriors, $\frac{1}{S}\sum_{s=1}^{S} p_\epsilon(\theta|X_s)$ at one end of the spectrum where long trajectory lengths are used, and it approaches the true posterior at the other end of the spectrum with short trajectory lengths (ϵ is small enough).

Note that this is indeed the desired behavior when dealing with massive datasets. That is, as $N \to \infty$, the local posteriors become close to the true posterior and thus the error decreases by the central limit theorem (provided X_s is a uniform random partition of X): $\bar{g}(\theta; X_s) \sim \mathcal{N}(\mathbb{E}[g(\theta;x)], \mathrm{Cov}[g(\theta;x)]/N_s)$. Therefore, as the dataset increases, we can increase the trajectory length accordingly without a significant loss in asymptotic accuracy. The following Corollary 5.4.1 states that for any finite τ, trajectory sampling is a valid SGLD (assuming the step sizes decrease to zero over time).

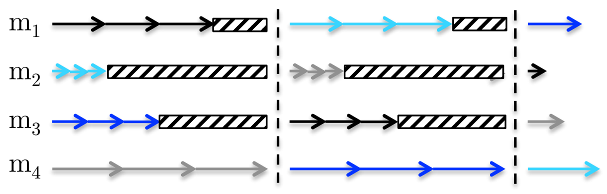
Corollary 5.4.1. *A trajectory sampler with a finite $\tau \geq 1$, obtained by redefining the worker (shard) selection process $h(\mathcal{Q})$ in Proposition 5.3.1 by the process $h(\mathcal{Q},\tau)$ below, is a valid SGLD sampler. $h(\mathcal{Q},\tau)$: for chain c at iteration t, choose the next worker s^c_{t+1} by*

$$s^c_{t+1} = \begin{cases} \tilde{h}(\mathcal{Q}), & \text{if } t = k\tau \text{ for } k = 0,1,2,\ldots \\ s^c_t, & \text{otherwise,} \end{cases} \tag{5.4}$$

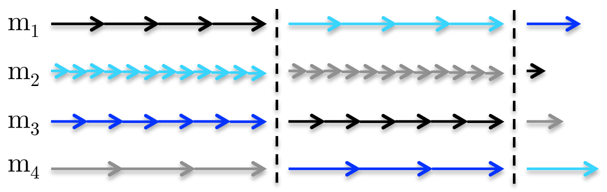
where $\tilde{h}(\mathcal{Q})$ is an arbitrary scheduler with selection probabilities $\mathcal{Q}$.

5.4.3 Adaptive Load Balancing

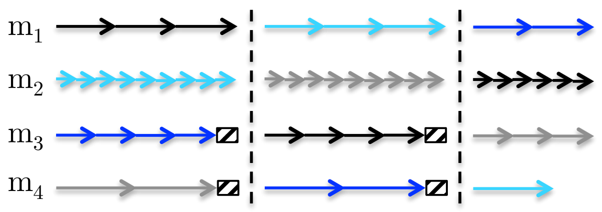
Using trajectory sampling, we can mitigate the short-communication-cycle problem. Moreover, if response delays are balanced, we can set $\mathcal{Q}$ to be uniform and use a random permutation scheduler to keep the block-by-the-slowest delay small. However, for imbalanced response delays, using uniform $\mathcal{Q}$ would lead to long block-by-the-slowest delays (See, Fig.



(a) Without load balancing, $\tau = 3$



(b) With load balancing, $\bar{\tau} = 25/4$



(c) With load balancing, $\bar{\tau} = 75/16$

Figure 5.1: Illustration of adaptive load balancing. Each row represents a worker and the chains are represented by different colors. The box filled by diagonal lines are block-time, and at the vertical dotted lines represent chains jumping to other workers. A sample is collected at each arrow whose length represents the time required to collect the sample. In the present example, four workers have different response delays, 3, 1, 2, and 4, respectively. In (a) τ is set to a constant $\tau = 3$ for all workers, and in (b) with $\bar{\tau} = \frac{25}{4}$, the trajectory plan becomes $\mathcal{T} = (4, 12, 6, 3)$, and in (c), $\mathcal{T} = (3, 9, 4.5, 2.25)$ with $\bar{\tau} = \frac{75}{16}$.

5.1 (a)). In this section, we propose a solution to balance the workloads by adapting $\mathcal{Q}$ to the worker response delays.

The basic idea is to make the faster workers work longer until the slower workers finish their tasks so that the overall response times of the workers become as balanced as possible. For instance, twice longer trajectories can be used for a worker that is twice as fast. More specifically, we achieve this by (1) having uniform worker selection and (2) setting the trajectory length τ_s of worker s to $\tau_s = q_s \bar{\tau} S$; here, q_s is set to $d_s^{-1} / \sum_{z=1}^{S} d_z^{-1}$ (i.e., the relative speed of worker s), and $\bar{\tau}$ is a user-defined mean trajectory length: $\mathbb{E}[\tau_s] = \sum_s \frac{1}{S} q_s \bar{\tau} S = \bar{\tau}$ (the expectation is w.r.t. the worker selection probability $1/S$).

In other words, we select a worker uniformly and perform trajectory sampling of length τ_s, which is proportional to the relative speed of the worker, q_s. (If τ_s is not an integer, we can either adjust $\bar{\tau}$ to make it integer or take simply the closest integer.) Note that using unequal trajectory lengths across the workers remains a valid SGLD because the step sizes are properly corrected by Eqn. (5.2) where $q_s \propto \tau_s$.

This is illustrated in Figure 5.1 and stated in Corollary 5.4.2.

Corollary 5.4.2. *Given τ_s, where $1 \leq \tau_s < \infty$ for $s = 1, \ldots, S$, the adaptive trajectory sampler, obtained by redefining the worker (shard) selection process $h(\mathcal{Q})$ in Proposition 5.3.1 by the process $h(\mathcal{Q}, \{\tau_s\})$ below, is a valid SGLD sampler. $h(\mathcal{Q}, \{\tau_s\})$: for chain c at iteration t, choose the next worker s^c_{t+1} by*

$$s^c_{t+1} = \begin{cases} \tilde{h}(1/S), & \text{if } t = k\tau_{s^c_t} \text{ for } k = 0, 1, 2, \ldots \\ s^c_t, & \text{otherwise,} \end{cases} \tag{5.5}$$

where $\tilde{h}(1/S)$ is a scheduler with uniform selection probabilities.

Our method can deal with temporal imbalances as well. To this end, the master needs to monitor the changes in response delays; when a substantial change is detected, a new trajectory plan can replace the old one. Note that although this online adaptation affects the Markov property, it can still converge to correct target distribution assuming that the adaptation satisfies the Corollary 5.4.2 and the response delays converge fast enough. Refer to Andrieu and Thoms [2008] for the details of the “fast enough” condition. Pseudo code for the proposed D-SGLD method is presented in Algorithm 5.

Algorithm 5 D-SGLD Pseudo Code

```
1: function MASTER($S, C, \bar{\tau}$)
2:     while sampling do
3:         Monitor response delays $\{d_s\}$
4:         if $\{d_s\}$ are changed enough then
5:             Adapt $\tau_s \leftarrow \bar{\tau} S d_s^{-1} / \sum_{z=1}^{S} d_z^{-1}, \forall s$
6:         end if
7:         Assign workers $(s^1, \ldots, s^C) \sim$ RANDPERM($S$)
8:         for each chain $c$ parallel do
9:             $\theta_c \leftarrow$ SAMPLE_TRAJ($s^c, c, \theta_c, \tau_s$) (line 13)
10:        end for
11:    end while
12: end function
13: function SAMPLE_TRAJ($c, \theta, \tau_s$)
14:    Initialize $\theta_1 \leftarrow \theta$
15:    for $t = 1 : \tau_s$ do
16:        Sample a mini-batch $X_s^n$ and noise $\nu \sim \mathcal{N}(0, \epsilon)$
17:        Obtain $\theta_{t+1}$ by Eqn. (5.3) (set $q_s = \frac{\tau_s}{\bar{\tau} S}$)
18:    end for
19:    Set trajectory, $T_c \leftarrow (\theta_1, \ldots, \theta_{\tau_s - 1})$
20:    Store (append) trajectory, $\Theta_s^c \leftarrow [\Theta_s^c, T_c]$
21:    Send the last state $\theta_{\tau_s}$ to the master
22: end function
```

5.4.4 Variance Reduction by Chain Coupling

Here, we introduce one approach that could reduce the variance of the gradient estimator in Eqn (5.2) by having some interactions among the chains. The basic idea is to "tie" a *group* of chains by averaging their corresponding samples. More specifically, consider $R \leq S$ chains forming a group and staying at a state θ_t at time t, i.e., $\theta_t^r = \theta_t$ for $r = 1, \ldots, R$. After an update using the standard SGLD update rule in Eqn. (5.1), we have R different states $\theta_{t+1}^1, \ldots, \theta_{t+1}^R$. By averaging the new states, we have $\theta_{t+1} = \frac{1}{R} \sum_{r=1}^{R} \theta_{t+1}^r$, which is

$$\theta_t + \frac{\epsilon_t}{2} \left\{ \nabla \log p(\theta_t) + \frac{N}{nR} \sum_{x \in \cup_r X_{t,r}^n} g(\theta_t; x) \right\} + \bar{\nu}_t \tag{5.6}$$

Here we used $\frac{1}{R}\sum_{r=1}^{R}\nabla\log p(\theta_t^r) = \nabla\log p(\theta_t)$ and $\frac{1}{R}\sum_{r=1}^{R}\theta_t^r = \theta_t$ noting that $\theta_t^r = \theta_t$ for all r. Note that although the averaged noise $\bar{\nu}_t = \frac{1}{R}\sum_{r=1}^{R}\nu_t^r$ has smaller variance $\mathcal{N}(0, \frac{\epsilon_t}{R})$ than the standard SGLD, we can recover a valid SGLD with additional noise[1] $\eta_t \sim \mathcal{N}(0, \frac{R-1}{R}\epsilon_t)$ so that $\bar{\nu}_t + \eta_t \sim \mathcal{N}(0, \epsilon_t)$. By the central limit theorem, the variance of the estimator of $\bar{g}(\theta_t; X)$ reduces from $\frac{\mathrm{Cov}[g(\theta;x)]}{n}$ to $\frac{\mathrm{Cov}[g(\theta;x)]}{Rn}$.

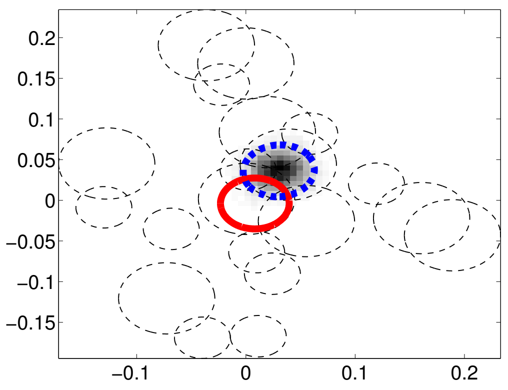
Although this approach leads to a valid SGLD with reduced variance in the gradient estimation, unfortunately it is difficult to perform the trajectory sampling in this case since it requires communication among chains at each update. One way to bypass this issue is to employ the averaging strategy only at the end of each trajectory during the burn-in period. Alternatively, we can also gradually reduce the number of chains being coupled. Note that although coupling chains imposes algorithmic dependency, it is weaker than other algorithms (e.g., AD-LDA) that require synchronizations for *all* workers since (i) the number of dependent chains, R, in our method is relatively small ($R < S$), and (ii) the response delays are already balanced by the adaptive trajectory sampling.

5.5 Experiments

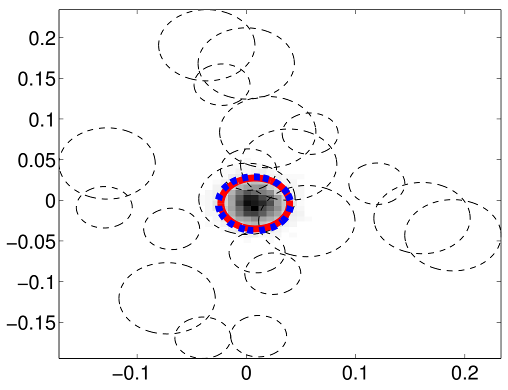
5.5.1 Simple Demonstration

We illustrate our proposed method based on sampling from a multivariate normal posterior distribution obtained by assuming a normal prior $\mathcal{N}(\mu_x; \mu_0, \Sigma_0)$ on the d-dimension mean μ_x of a normal distribution $\mathcal{N}(x; \mu_x, \Sigma_x)$ from which we have observed N samples. Because this is a conjugate prior, the posterior distribution is also a normal distribution.

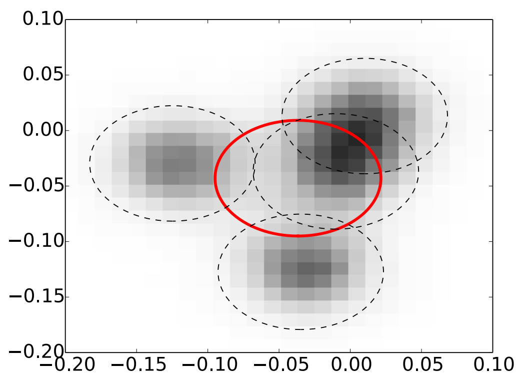
[1]The correction cannot be used for the LDA experiments because for SGRLD the noise term depends on current state θ_t^r.



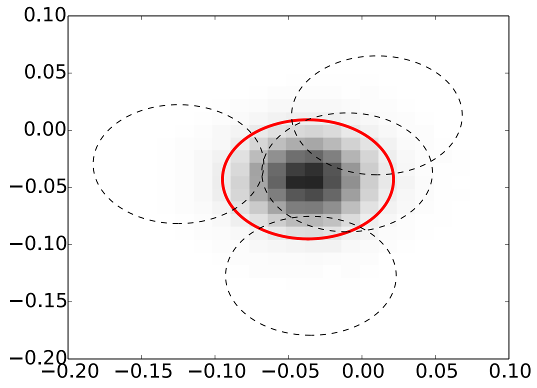
(a) Without correction



(b) With correction



(c) τ = 10,000



(d) τ = 200

Figure 5.2: Bias correction and trajectory length effects.

To examine the bias correction effect, we allocated a total of 20,000 data points to a cluster of 20 workers. Furthermore, we made the shard sizes $\{N_s\}$ highly imbalanced by setting $N_s = 500$ for 10 workers and setting $N_s = 1500$ for the remaining 10 workers. Then, to impose a higher level of imbalance, we also used the small shards 7 times more often than the large shards by setting the trajectory lengths for the small shards to 70 and those for the large shards to 10. We set the step size ϵ to 10^{-7} and the mini-batch size to $n = 300$.

In Fig. 5.2 (a) and (b), the black dotted circles represent the 2-d marginal covariance centered at the mean of the 20 local posteriors. Note that these are rescaled such that small circles represent the local posteriors based on small shards, whereas the large circles represent the local posteriors based on large shards. Also, the red circle represents the true posterior, and the dotted blue circle represents the empirical distribution based on our samples. As we can see, our algorithm corrects the bias. We have evaluated our method for various dimensions (up to $d = 100$) and found similar results.

The effect of trajectory lengths is also tested in Fig. 5.2 (c) and (d) using two different trajectory lengths, $\tau = 10,000$ and $\tau = 200$, for a cluster of 4 workers. Here, the shard size was set to 2,000 for each worker, the trajectory lengths were kept the same for all workers, and the step size, ϵ, was set to 2×10^{-6}. As described in section 5.4.2, we can see that D-SGLD samples from a mixture of the local posteriors with long trajectory lengths and becomes close to the standard SGLD posterior as the length decreases.

5.6 Discussion

We have introduced a novel algorithm, "distributed stochastic gradient Langevin dynamics (D-SGLD)". Using D-SGLD, the advantages of the sequential mini-batch-based MCMC are extended to distributed computing environments. We showed that (i) by adding a proper correction term, our algorithm prevents the local-subset-bias while (ii) reducing communication overhead through trajectory sampling and adaptive load balancing. Furthermore, (iii) it improved convergence speed using a variance reduction strategy. Finally, in several experiments for LDA, we have shown at least an order of magnitude faster convergence speed of D-SGLD over the state of the art both in sequential mini-batch-based MCMC and distributed MCMC. We believe that D-SGLD is just one example of a much larger class of powerful MCMC algorithms that combine sampling updates based on mini-batches with distributed computation.

Chapter 6

Large-Scale Distributed Inference for Latent Dirichlet Allocation

In this chapter, we apply D-SGLD to the distributed inference of latent Dirichlet allocation (LDA). The LDA model used most frequently for topic modeling has been one of the most popular large-scale problems in machine learning and data mining.

The results in this chapter is based on Ahn et al. [2014]. Max Welling and I developed the D-SGLD algorithm using SGRLD as the local sampler. I performed the experiments, and Max Welling and Babak Shahbaba provided important comments and helped writing.

6.1 Motivation

The scale of the document corpus used in topic modeling easily exceeds millions of documents and hundreds of millions of tokens. For example, considering that the New York Times alone produces more than 1,500 articles and around 300 blog posts every day[1] and 113-million blogs

[1] https://www.quora.com/How-many-articles-does-nytimes-com-publish-every-day

are posted on Tumblr every day[2], a personalized news/blog recommendation system, that needs to handle many of such content providers, would be required to analyze the topics of millions of articles every day or week. Therefore, an efficient scalable and distributed inference algorithm is indispensable component in successful application of topic modeling to the industry problems of practical interest.

For single machine settings, a variant of SGLD, called stochastic gradient Riemannian Langevin dynamics (SGRLD) [Patterson and Teh, 2013], has recently been proposed and showed the-state-of-the-art performance outperforming the online variational inference [Hoffman et al., 2010b]. For distributed computing environment, a number of MCMC algorithms have also been developed. In particular, in the approximate distributed LDA (AD-LDA) by Newman et al. [2007], the computation cost per sample is reduced to $\mathcal{O}(N/S)$, where S the number of workers, by allowing each worker to perform collapsed Gibbs sampling only on its local shard. AD-LDA also corrects (approximately) the biases in the local copies of the global states by performing global synchronization regularly.

AD-LDA, however, suffers from some shortcomings. First, due to the $\mathcal{O}(N/S)$ computation complexity, it becomes slower as the dataset size N increases, unless additional workers are provided. Second, due to the global synchronization, it suffers from the "block-by-the-slowest" problem, meaning that some workers are blocked until the slowest worker finishes its task. Lastly, running parallel chains usually adds a large overhead. The Yahoo-LDA (Y-LDA) algorithm [Ahmed et al., 2012] has improved upon the AD-LDA in such a way to make the updates asynchronous to resolve the block-by-the-slowest problem. However, it is also shown that the asynchronous updates could severely deteriorate performance if it is not tightly bounded [Ho et al., 2013].

In this chapter, we introduce a distributed inference algorithm for LDA using D-SGLD. In particular, we extend D-SGLD to use the SGRLD as the local sampler. Our experiments

[2]http://expandedramblings.com/index.php/tumblr-user-stats-fact/

for LDA on Wikipedia and Pubmed datasets show that, relative to both the current fastest sequential MCMC sampler [Patterson and Teh, 2013] and the fastest distributed MCMC samplers [Newman et al., 2007, Ahmed et al., 2012, Smola and Narayanamurthy, 2010], D-SGLD reduces the computation-time from 27 hours to half an hour in order to reach the same level of perplexity.

6.2 LDA and SGRLD

In LDA, given D documents and a vocabulary of W words, the goal is to obtain K topic distributions π_k, and a mixing proportion η_d of the topics for each document d. A topic π_k is a W-dimension distribution over the vocabulary words drawn from a symmetric Dirichlet distribution with hyper-parameter β. The mixing proportion η_d is modeled by a K-dimension Dirichlet distribution with hyper-parameter α. Then, the generative process of a document d, represented as bag-of-words, is modeled by drawing a latent topic assignment z_{di} for each word w_{di} in document d and then drawing a word w_{di} from the topic distribution $\pi_{z_{di}}$. Then we have the posterior distribution:

$$p(\eta, \pi, z|W, \alpha, \beta) \propto \prod_{d=1}^{D} p(\eta_d|\alpha) \prod_{k=1}^{K} p(\pi_k|\beta) \prod_{d=1}^{D} \prod_{i=1}^{N_d} p(w_{di}|\pi_{z_{di}}) p(z_{di}|\eta_d). \tag{6.1}$$

SGRLD is a sampler for probability simplex ϕ with the following constraints

$$\phi = \{(\phi_1, \ldots, \phi_K) : \phi_k \geq 0, \sum_k \phi_k = 1\} \in \mathbb{R}^K.$$

To satisfy this constraint, SGRLD for LDA uses a reparameterization technique, called expand-mean parameterization: for each topic π_k, introduces W-dimension unnormalized parameter θ_k from a Gamma distribution $p(\theta_k) \propto \prod_{w=1}^{W} \theta_{kw}^{\beta_w - 1} \exp^{-\theta_{kw}}$ and set $\pi_{kw} = \frac{\theta_{kw}}{\sum_{w'=1}^{W} \theta_{kw'}}$. One advantage of the expand-mean parameterization, in addition to satisfying the probability simplex contition, is that by converting the sampling space to θ_{kw} we can obtain a computationally efficient Riemannian metric $G(\theta) = \mathrm{diag}(\theta_{11}, \ldots, \theta_{1W}, \ldots, \theta_{K1}, \ldots, \theta_{KW})$.

Using this reparameterization and the Riemannian Langevin dynamics [Girolami and Calderhead, 2010], Patterson and Teh [2013] derived the following SGRLD update equation

$$\theta^*_{kw} \leftarrow \left| \theta_{kw} + \frac{\epsilon}{2} \left(\beta - \theta_{kw} + \frac{|D|}{|M_t|} \sum_{d \in M_t} \mathbb{E}_{z_d|w_d,\theta,\alpha}[n_{dkw} - \pi_{kw}\eta_{kd\cdot}] + (\theta_{kw})^{\frac{1}{2}} \xi_{kw} \right) \right| \tag{6.2}$$

where $\xi_{kw} \sim \mathcal{N}(0, \epsilon)$ and M_t is the size of the mini-batch documents. We use n_{dkw} to denote the appearance count of word w assigned to topic k in document d. To compute the expectation in the above equation, we use collapsed Gibbs sampling with the following conditional distribution on the latent topic assignment,

$$p(z_{di} = k | w_d, \theta, \alpha) = \frac{\left(\alpha + n^{\backslash i}_{dk\cdot}\right) \theta_{kw_{di}}}{\sum_{k'=1}^{K} \left(\alpha + n^{\backslash i}_{dk'\cdot}\right) \theta_{k'w_{di}}}. \tag{6.3}$$

Here, we use $n^{\backslash i}_{dk\cdot}$ to denote the appearance count of words assigned to topic k in document d but excluding word i that we are currently updating. Refer to Patterson and Teh [2013] for the detail derivation of the update equation.

6.3 D-SGLD for LDA

Using SGRLD as the local sampler of D-SGLD is straightforward. By applying the D-SGLD estimator in Eqn. (5.3) in Chapter 5, we obtain the following D-SGLD update equation for LDA

$$\theta^*_{kw} \leftarrow \left| \theta_{kw} + \frac{\epsilon}{2} \left(\beta - \theta_{kw} + \frac{|D_{s_t}|}{q_{s_t}|M_{s_t}|} \sum_{d \in M_{s_t}} \mathbb{E}_{z_d|w_d,\theta,\alpha}[n_{dkw} - \pi_{kw}\eta_{kd\cdot}] + (\theta_{kw})^{\frac{1}{2}} \xi_{kw} \right) \right|. \tag{6.4}$$

Here, M_{s_t} is a mini-batch sampled from a shard s_t and D_{s_t} is the total size of the local shard s_t. We use q_{s_t} to denote the shard selection frequency.

6.4 Experiments

In the experiments below, we use the following datasets: (i) *Wikipedia* corpus, which contains 4.6M articles of approximately 811M tokens in total. We used the same vocabulary of 7702 words as used by Hoffman et al. [2010a]. (ii) *PubMed Abstract* corpus contains 8.2M articles of approximately 730M tokens in total. After removing stopwords and low occurrence (less than 300) words, we obtained a vocabulary of 39,987 words. For our Python implementation, each of the datasets has 47GB memory footprint.

The algorithms we compare to the D-SGLD are as follows:

1. *AD-LDA*: In AD-LDA, to obtain a single sample, each worker s performs collapsed Gibbs iterations only on the full *local* shard (and is thus approximate), and then synchronizes the local topic assignments n^s_{kw} of shard s at the master to obtain the global state n_{kw} based on the update rule $n_{kw} \leftarrow n_{kw} + \sum_{s=1}^{S}(n_{kw} - n^s_{kw})$. Then, the local states are globally updated by the new global state, $n^s_{kw} \leftarrow n_{kw}$ for all s. It is shown that in practice AD-LDA shows comparable perplexities to the standard collapsed Gibbs sampling in a single machine but at much fast speed.

2. *Async-LDA* (Y-LDA): Unlike AD-LDA, Y-LDA [Ahmed et al., 2012, Smola and Narayanamurthy, 2010] performs asynchronous updates for the global state by executing the following update rule, $n_{k,w} \leftarrow n_{k,w} + \sum_{s=1}^{S}(\tilde{n}^s_{k,w} - n^s_{k,w})$. Here, $\tilde{n}^s_{k,w}$ is a copy of the old local state at the time of previous synchronization. Because the original Y-LDA proposed in Ahmed et al. [2012], Smola and Narayanamurthy [2010] is a specific implementation optimized along with many other dimensions, we implemented an algorithm

called Async-LDA which replaces the update of AD-LDA with the asynchronous update of Y-LDA. Async-LDA was used to compare to D-SGLD in terms of the load balancing ability (i.e. when each worker has different processing speed).

3. *SGRLD*: The above two algorithms perform distributed inference on many workers but are not mini-batch based algorithm. Because there has been no mini-batch based distributed algorithm, we also use SGRLD as the state-of-the-art mini-batch method running on a single machine. Following Patterson and Teh [2013], we set the mini-batch size to 50 documents, and to compute the expectation in the computation of the gradient, we ran 100 Gibbs iterations for each document in the mini-batch. The step-sizes were annealed by a schedule $\epsilon_t = a(1 + t/b)^{-c}$. As we fixed $b = 1000$ and $c = 0.6$, the entire schedule was set by a which we choose by running parallel chains with different a's and then choosing the best. Note that in SGRLD there is no communication overhead.

4. *D-SGLD*: We used cyclic rotation as the chain-to-worker scheduler and set the trajectory length $\tau = 10$ for all workers while we kept other parameters the same as for SGRLD by default.

In particular, to see the effect of the variance reduction (i.e., sample averaging), we implemented three different versions of D-SGLD, (i) *Complete Coupling* (D-CC), (ii) *Complete Independent* (D-CI), and (iii) *Hybrid* (D-Hybrid). D-CC couples *all* chains; whereas, D-CI runs independent chains without any interaction among them. D-Hybrid partitions the chains into groups and the averaging is performed only for the chains in the same group. When the variance reduction is used, it was performed at the end of each trajectory; we did not inject any additional noise for correction.

Additionally, we used the following settings for all algorithms. The predictive perplexities were computed on 1000 separate holdout set, with a 90/10 (training/test) split, and LDA's

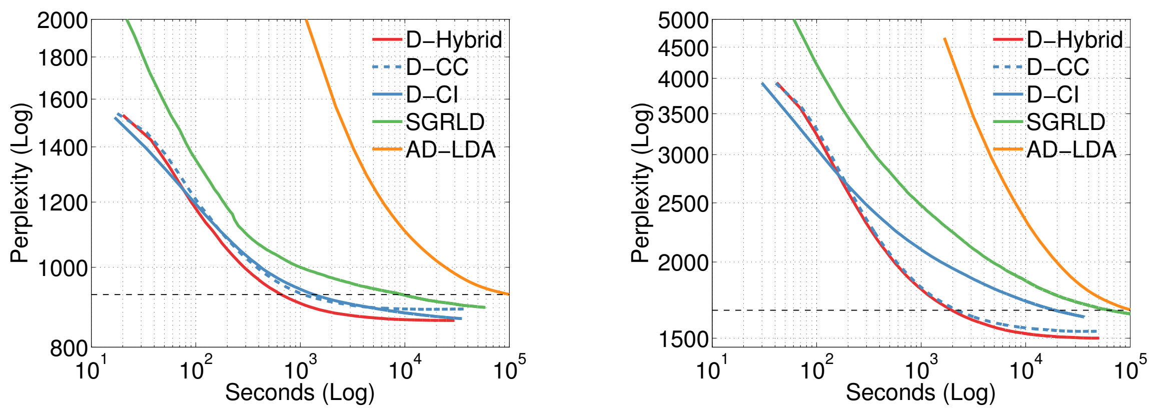


Figure 6.1: Perplexity. Left: Wikipedia, Right: Pubmed.

	D-SGLD	SGRLD	AD-LDA
Wikipedia	10 min.	2.6 hr.	27.7 hr.
Pubmed	33 min.	16.7 hr.	27.7 hr.

Table 6.1: Required time to reach the perplexity that AD-LDA obtains after running 10^5 seconds (27.7 hours).

hyper-parameters were set to $\alpha = 0.01$ and $\beta = 0.0001$ following Patterson and Teh [2013]. The number of topics K was set to 100. Parallelism within a worker is not considered, although D-SGLD can be easily parallelized within a worker.

6.4.1 Perplexity

We first compare the above algorithms in terms of the convergence in perplexity over wall-clock time on 20 homogeneous workers dedicated to the given task only. For D-Hybrid, we set the number of groups, G, to 5 and 3 for Wikipedia and Pubmed respectively. For Wikipedia, we set the group size to $R = 4$. For Pubmed, we set the sizes of the three groups to 7,7, and 6. To examine the effect of the variance reduction strategy, it was continued until the end of the experiment, as opposed to stopping at some point. The step size parameter a was set to 0.0001 for Wikipedia and to 0.0005 for Pubmed.

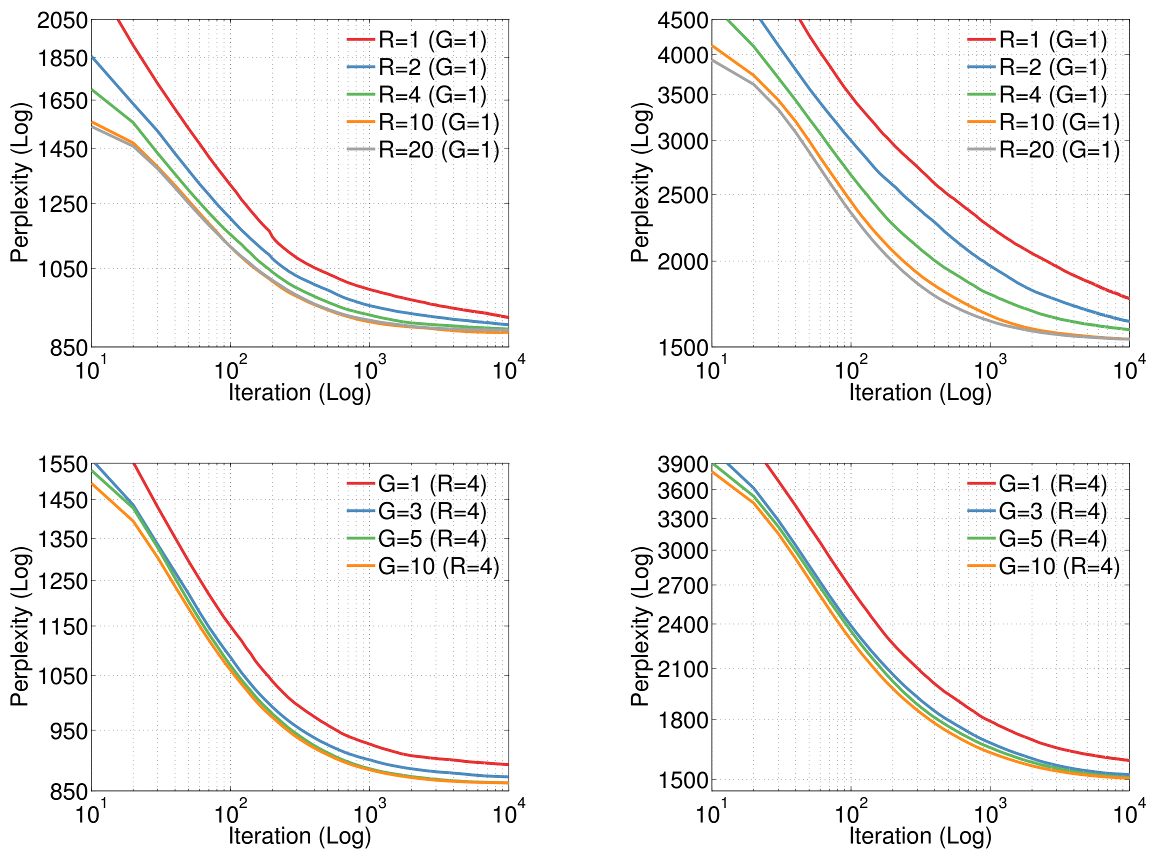


Figure 6.2: Group size and number of groups. Top: group size, Bottom: number of groups, Left: Wikipedia, Right: Pubmed.

In Fig. 6.1 (a) and (b), we first see that all the variants of D-SGLD significantly outperform both AD-LDA and SGRLD. Note that AD-LDA ran in an ideal setting where each worker has equal workloads (in terms of shard size) resulting in negligible block-by-the-slowest delays. As shown in Table 6.1, D-SGLD required substantially shorter times than AD-LDA and SGRLD to reach the same perplexity level that AD-LDA achieves after running 10^5 seconds (27.7 hours) indicated by the black horizontal dotted line. Throughout the experiments, Async-LDA always performed worse than AD-LDA given balanced workloads.

For the three different versions of D-SGLD, we see that D-CC and D-Hybrid (which use the sample averaging) converge faster than D-CI (which uses independent chains). However, when we couple too many chains as shown in D-CC, it could lead to some lose of accuracy (possibly, due to the bias by the coupling). Hence, in the following experiments, we only use hybrid D-SGLD; a proper group configuration is chosen by cross-validation. Fig. 6.2 shows other effects of the group configuration by increasing group size (R) and number of groups (G).

6.4.2 Dataset size

In D-SGLD the computation cost per sample $\mathcal{O}(n)$ is independent of N. AD-LDA, on the other hand, becomes slower as N increases. To see the effect of N, we examined the algorithms on random subsets of the full dataset with different sizes, 100K, 1000K, and full, using 20 homogeneous workers. For N=[100K, 1000K, full], the initial step sizes a were set to respectively a=[0.005, 0.0005, 0.0001] for Wikipedia and a=[0.01, 0.005, 0.0005] for Pubmed.

As shown in Fig. 6.3, for Wikipedia, D-SGLD showed similar convergence in perplexity (they increase slightly as the size of datasets decreases) while providing better results than AD-LDA in all settings. However, for Pubmed, which has a larger vocabulary and is expected to have a larger number of topics, D-SGLD was not better than AD-LDA for the small (100K) dataset while still had better performance for larger datasets. In fact, SGRLD seemed to work less efficiently (and so does D-SGLD) for rather small datasets as shown by Patterson and Teh [2013] based on the NIPS corpus. Nevertheless, we found that (results not shown here) D-SGLD outperforms a single SGRLD based on a 100K dataset.

6.4.3 Number of workers

We also varied the number of workers while fixing the dataset size to the full. In Fig. 6.4, we show the results for three cluster sizes, $S = [20, 40, 60]$. As expected, AD-LDA improves linearly by increasing the number of workers (i.e., by reducing local shard sizes). For D-

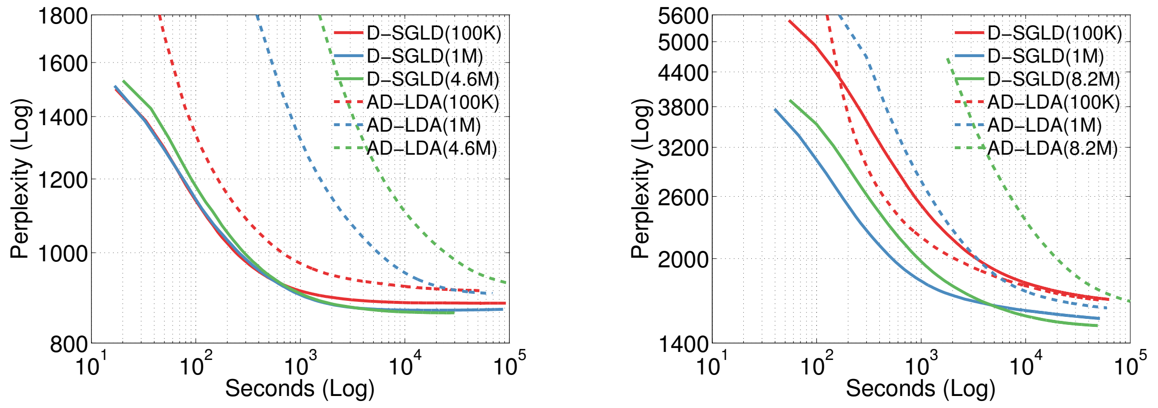


Figure 6.3: Dataset size. Left: Wikipedia, Right: Pubmed

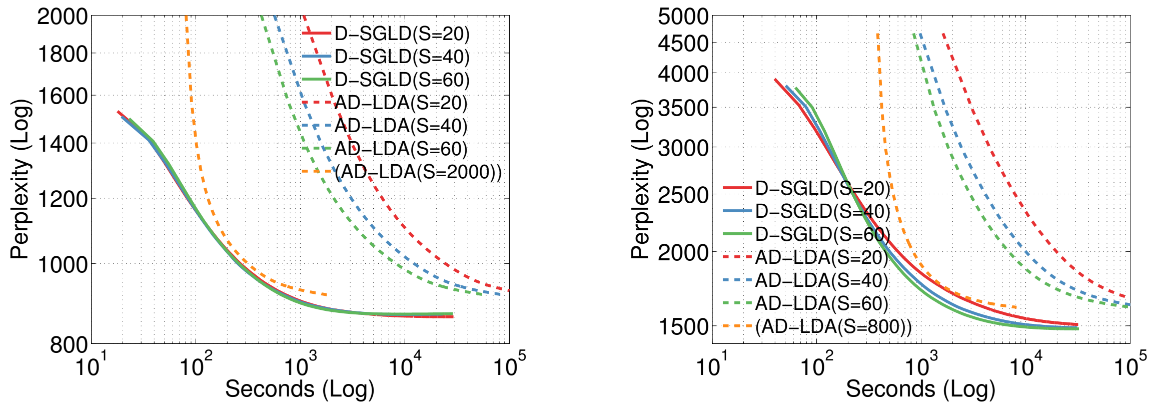


Figure 6.4: Number of workers. Left: Wikipedia, Right: Pubmed

SGLD, we fixed G to 5 and increased only the group size R to $4, 8, 12$. Although more workers imposed more communication overhead during sample averaging, D-SGLD showed its scalability by keeping the performance at a similar level (for Pubmed, it is improved). From this result, we calculated the number of workers required by AD-LDA to show a similar speed as D-SGLD with 20 workers. As shown in the Fig. 6.4, AD-LDA needs 2000 workers for Wikipedia and 800 workers for Pubmed to obtain a similar speed as D-SGLD. (This simple calculation does not include the communication overhead.)

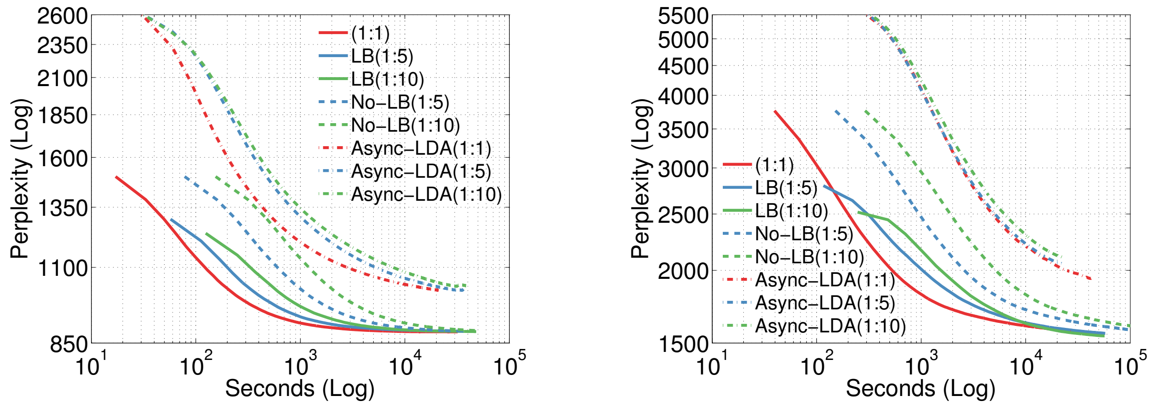


Figure 6.5: Load balancing. Left: Wikipedia, Right: Pubmed.

6.4.4 Load balancing

We also examined D-SGLD's ability to balance the workloads and thus mitigate the block-by-the-slowest problem on 20 workers. To do this, we added dummy delays to half of the workers to make them D times slower. We denote this setting by (1:D) and used three settings: $D = [1, 5, 10]$. The actual response delays then became equal, for example, by setting the trajectory length to 10 for slow workers and to $D \times 10$ for fast ones. The initial step size a was set to 0.005 for all settings of Wikipedia and to 0.001 for all settings of Pubmed. Here, we used 100K Wikipedia and 1000K Pubmed corpus because the Async-LDAs (as well as AD-LDA) were too slow for the full datasets. As shown in Fig. 6.5, D-SGLD with load-balancing through adaptive trajectory sampling converges much faster than those without load-balancing; it also converges faster than Async-LDA.

6.4.5 Number of topics

We tested the effect of the number of topics K by examining K=[100,200,300,400,500] on 20 homogeneous workers. As shown in Fig.6.6, although the packet size increases for large K, D-SGLD consistently outperforms SGRLD for all K.

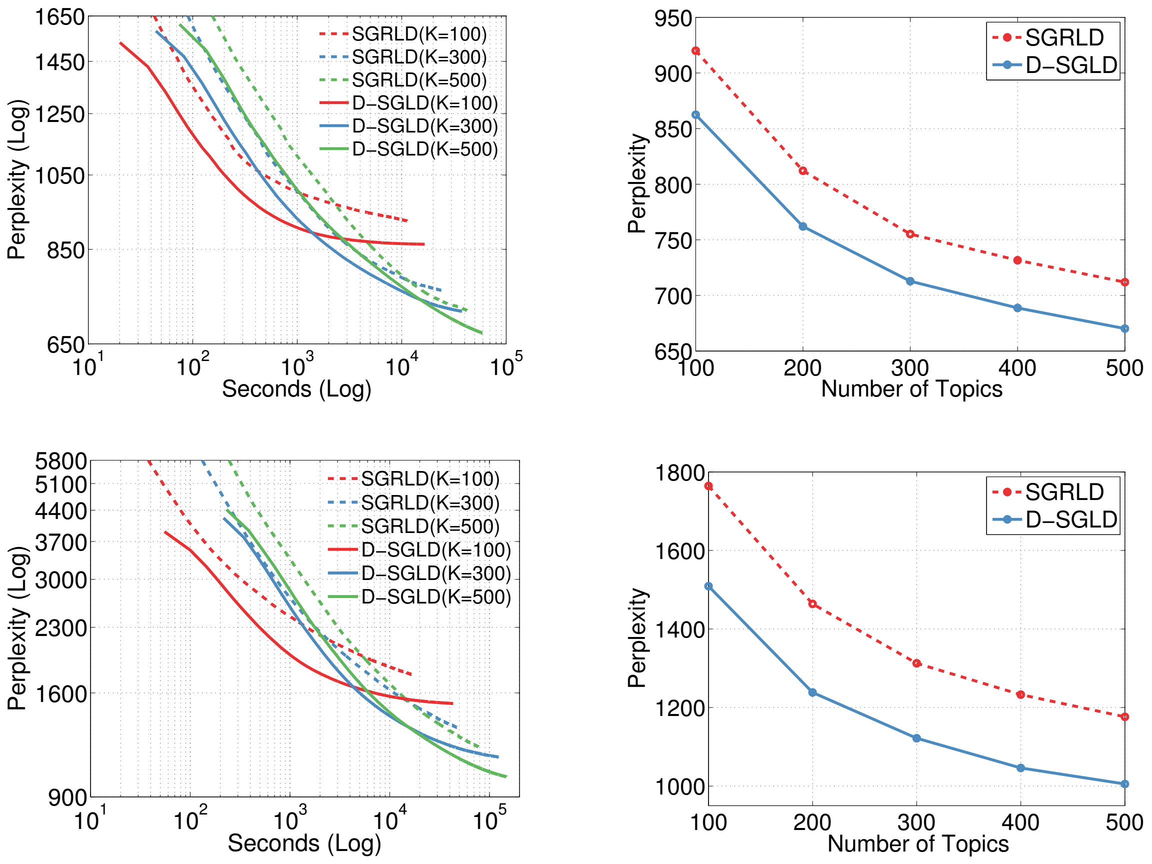


Figure 6.6: Number of topics. Top: Wikipedia, Bottom: Pubmed. Right: Perplexity after 10^4 updates (that is, the end points of each line in the left plots).

Chapter 7

Large-Scale Distributed Bayesian Matrix Factorization

Despite having various attractive qualities such as high prediction accuracy, the ability to quantify uncertainty, and avoiding over-fitting, Bayesian Matrix Factorization has not been widely adopted because of the prohibitive cost of inference. In this chapter, we propose a scalable distributed Bayesian matrix factorization algorithm using stochastic gradient MCMC. Our algorithm, based on Distributed Stochastic Gradient Langevin Dynamics (D-SGLD), can not only match the prediction accuracy of standard MCMC methods like Gibbs sampling, but at the same time is as fast and simple as stochastic gradient descent. In our experiments, we show that our algorithm can achieve the same level of prediction accuracy as Gibbs sampling an order of magnitude faster. We also show that our method reduces the prediction error as fast as distributed stochastic gradient descent, achieving a 4.1% improvement in RMSE for the Netflix dataset and an 1.8% for the Yahoo music dataset.

This work is a joint work with UC Irvine (Max Welling, Anoop Korattikara, and I) and Yahoo Labs (Nathan Liu and Suju Rajan) and performed partly while I was an intern researcher at

Yahoo labs. I developed the algorithm and performed the experiments. Anoop Korattikara helped developing the single machine version of the algorithm. And, Max Welling, Nathan Liu and Suju Rajan provided important comments on the algorithm and the experiments.

7.1 Motivation

Recommender systems have become a pervasive tool in industry to understand customers and their interests in products. Examples range between music recommendation (Pandora), book recommendation (Amazon), movie recommendation (Netflix), news recommendation (Yahoo) to partner recommendation (eHarmony). Recommender systems represent a personalized technology that can help filter at an individual level the enormous amounts of information that is available to us. Given the exponential growth of data, recommender systems are likely to play an increasingly important role to manage our information streams.

During 2006-2011 Netflix [Bennett and Lanning, 2007] ran a competition where teams around the world could develop and test new recommender technology on Netflix movie rating data. A few valuable lessons were learnt from that exercise. First, matrix factorization methods work very well compared to nearest neighbor type models. Second, averaging over many different models pays off in terms of prediction accuracy. One particularly effective model was Bayesian probabilistic matrix factorization (BPMF) [Salakhutdinov and Mnih, 2008] where predictions are averaged over samples from the posterior distribution. Besides improved prediction accuracy, a full Bayesian analysis also comes with additional advantages such as probabilities over models, confidence intervals, robustness against overfitting, and incorporating prior knowledge and side-information [Adams et al., 2010, Porteous et al., 2010].

Unfortunately, since the number of user-product interactions can easily run into the billions, posterior inference is usually too expensive to be practical. Learning at that scale requires data and computation to be distributed over many machines and learning updates to only depend on small minibatches of the data. Effective distributed learning algorithms have been devised for alternating least squares (ALS) and stochastic gradient descent (SGD) [Gemulla et al., 2011, Recht and Re, 2013, Zhuang et al., 2013, Teflioudi et al., 2012, Niu et al., 2011, Hall et al., 2010, McDonald et al., 2010, Mann et al., 2009, Zinkevich et al., 2010]. In particular, Distributed Stochastic Gradient Descent (DSGD) [Gemulla et al., 2011] has achieved a significant speed-up by assigning partitioned rating matrix blocks to workers and then by updating some "orthogonal" blocks in parallel using "stratified" SGD. DSGD outperformed other parallel SGD approaches such as PSGD [Hall et al., 2010, McDonald et al., 2010] and ISGD [Mann et al., 2009, Zinkevich et al., 2010] where SGD is applied also on some subsets of the ratings while synchronizing globally after each sub-epoch (PSGD) or once at the end of the training (ISGD). Unfortunately, so far it has proven difficult to apply these advances in distributed learning to posterior sampling in Bayesian matrix factorization models. For instance, for BPMF which requires $\mathcal{O}((L+M)D^3)$ computation per iteration (with L and M are number of users and items, and D is latent feature dimension), distributed computation has not nearly been as effective.

In this chapter, we propose a scalable and distributed Bayesian matrix factorization method which combines the predictive accuracy of Bayesian inference and the learning efficiency of stochastic gradient updates. To this end, we extend a recently developed distributed MCMC method, called Distributed Stochastic Gradient Langevin Dynamics (D-SGLD) [Ahn et al., 2014], so that the updates become efficient in the setting of distributed, large-scale matrix factorization. We adapt the SGLD updates to make them suitable for distributed learning on subsets of users and products (or blocks). Each worker manages only a small block of the rating matrix, and updates and communicates only a small subset of the parameters in a fully-asynchronous or weakly-synchronous fashion. Unlike distributed SGD where a single model is learnt, our method deploys multiple parallel chains over workers. Consequently, samples are collected at a much faster rate than ordinary MCMC and the multiple parallel chains can explore different modes of parameter space. Both features contribute to reducing the variance and increasing the accuracies of our predictions.

In the experiments on the Netflix and Yahoo music datasets (the latter being one of the largest publicly available dataset for recommendation problems), we show that our method achieves the same level of accuracy as BPMF but an order of magnitude faster. Reversely, at almost the same efficiency as distributed SGD, our method achieves much better accuracy (4.1% RMSE improvement for the Netflix dataset and 1.8% for Yahoo music dataset). As such we believe that the method proposed in this chapter is currently the most competitive matrix factorization method for industry scale problems.

7.2 Bayesian Matrix Factorization

Suppose we have L users and M items. Our goal is to learn latent feature vectors $U_i, V_j \in \mathbb{R}^D$ such that the rating R_{ij} for item j by user i can be predicted as $R_{ij} \approx U_i^\top V_j$. We denote the entire rating matrix by $\mathbf{R} \in \mathbb{R}^{L\times M}$, and the latent feature matrices by $\mathbf{U} \in \mathbb{R}^{D\times L}$ and $\mathbf{V} \in \mathbb{R}^{D\times M}$, so that $\mathbf{R} \approx \mathbf{U}^\top \mathbf{V}$. Assuming a Gaussian error model, the likelihood of the parameters $\mathbf{U}$ and $\mathbf{V}$ can be written as:

$$p(\mathbf{R}|\mathbf{U},\mathbf{V},\tau) = \prod_{i=1}^{L}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|U_i^\top V_j, \tau^{-1})\right]^{I_{ij}}. \qquad (7.1)$$

where I_{ij} is equal to 1 if user i rated item j and 0 otherwise. Throughout the chapter, we fixed $\tau = 1$ for simplicity[1]. Although, in theory, $\mathbf{U}$ and $\mathbf{V}$ can be learned by maximizing the

[1]All update equations are derived with $\tau = 1$.

likelihood above, this results in severe over-fitting because only a few ratings are known (i.e. $\mathbf{R}$ is very sparse).

Therefore, a Bayesian Probabilistic Matrix Factorization (BPMF) model was proposed to overcome this problem [Salakhutdinov and Mnih, 2008]. In addition to controlling over-fitting through posterior averaging, BPMF also provides estimates of uncertainty through the posterior predictive distribution. The BPMF model as proposed in [Salakhutdinov and Mnih, 2008] is as follows. We place priors on $\mathbf{U}$ and $\mathbf{V}$ as:

$$\begin{aligned} p(\mathbf{U}|\mu_U, \Lambda_U) &= \prod_{i=1}^{L} \mathcal{N}(U_i|\mu_U, \Lambda_U^{-1}), & (7.2) \\ p(\mathbf{V}|\mu_V, \Lambda_V) &= \prod_{j=1}^{M} \mathcal{N}(V_j|\mu_V, \Lambda_V^{-1}). & (7.3) \end{aligned}$$

We further place Gaussian-Wishart hyper-priors on the user and item hyperparameters $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$:

$$\begin{aligned} p(\Theta_U|\Theta_0) &= \mathcal{N}(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0), & (7.4) \\ p(\Theta_V|\Theta_0) &= \mathcal{N}(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_V|W_0, \nu_0), & (7.5) \end{aligned}$$

where ν_0 is the number of degrees of freedom and W_0 is a $D \times D$ scale matrix. We collectively denote the parameters of the hyper-prior by $\Theta_0 = \{\mu_0, \beta_0, \nu_0, W_0\}$.

At test time, the predictive distribution of an unknown rating R^*_{ij} can be obtained by marginalizing over both model parameters $\mathbf{U}, \mathbf{V}$ and hyper-parameters Θ_U, Θ_V,

$$\begin{aligned} p(R^*_{ij}|\mathbf{R}, \Theta_0) &= \int\int p(R^*_{ij}|U_i, V_j)p(\mathbf{U}, \mathbf{V}|\mathbf{R}, \Theta_U, \Theta_V) \\ &\quad p(\Theta_U, \Theta_V|\Theta_0)\mathrm{d}\{\mathbf{U}, \mathbf{V}\}\mathrm{d}\{\Theta_U, \Theta_V\} & (7.6) \end{aligned}$$

Algorithm 6 Gibbs Sampling for BPMF

```
1: Initialize model parameters $\mathbf{U}^{(1)}, \mathbf{V}^{(1)}$
2: for $t = 1 : T$ do
3:     // Sample hyperparameters
       $\Theta_U^{(t)} \sim p(\Theta_U|\mathbf{U}^{(t)}, \Theta_0), \quad \Theta_V^{(t)} \sim p(\Theta_V|\mathbf{V}^{(t)}, \Theta_0)$
4:     for $i = 1 : L$ in parallel do
5:         $U_i^{(t+1)} \sim p(U_i|\mathbf{R}, \mathbf{V}^{(t)}, \Theta_U^{(t)})$ // sample user features
6:     end for
7:     for $j = 1 : M$ in parallel do
8:         $V_j^{(t+1)} \sim p(V_j|\mathbf{R}, \mathbf{U}^{(t)}, \Theta_V^{(t)})$ // sample item features
9:     end for
10: end for
```

We can estimate this using a Monte Carlo approximation:

$$p(R_{ij}^*|\mathbf{R}, \Theta_0) \;\approx\; \frac{1}{T}\sum_{t=1}^{T} p(R_{ij}^*|U_i^{(t)}, V_j^{(t)}). \tag{7.7}$$

where $\left\{U_i^{(t)}, V_j^{(t)}\right\}$ is the t-th sample from the posterior distribution:

$$p(\mathbf{U}, \mathbf{V}, \Theta_U, \Theta_V|\mathbf{R}, \Theta_0). \tag{7.8}$$

These samples can be generated using Gibbs sampling (Algorithm 6), since by conjugacy the conditional distributions of U_i and V_j are Gaussian, and those of Θ_U and Θ_V are Gaussian-Wishart. However, sampling from the conditional distribution of U_i or V_j involves $\mathcal{O}(D^3)$ computations (for inverting a $D \times D$ precision matrix) and since this has to be done for each user and item, results in a total of $\mathcal{O}((L+M)D^3)$ computations per iteration. Thus, BPMF using Gibbs sampling cannot scale up to real world recommender systems with millions of users and / or items.

Although it is possible to parallelize BPMF using MapReduce style synchronous global updates, the cubic order complexity still limits its applicability to small D. Also, we require a large number of workers to effectively distribute the $L + M$ cubic-order computations. Furthermore, since running the Gibbs sampler from scratch is too expensive, a separate

SGD optimizer is usually deployed to reach near the Maximum-a-Posteriori (MAP) state before starting the Gibbs sampler. However, running two different large-scale distributed algorithms, each of which requires different optimal settings for the distribution of data and parameters, as well as cluster architectures, adds another considerable level of complexity.

7.3 Bayesian Matrix Factorization using SGLD

We will now show how DSGLD can be used for BPMF. Instead of the model described in Section 7.2, we will use a slightly simplified model [Mnih and Salakhutdinov, 2007, Chen et al., 2014]. We use the same likelihood as in eqn. 7.1, but choose simpler priors:

$$p(\mathbf{U}|\Lambda_U) = \prod_{i=1}^{L} \mathcal{N}(U_i|0, \Lambda_U^{-1}), \tag{7.9}$$

$$p(\mathbf{V}|\Lambda_V) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \Lambda_V^{-1}). \tag{7.10}$$

Here, Λ_U and Λ_V are D-dimension diagonal matrices whose d-th diagonal elements are λ_{U_d} and λ_{V_d} respectively. We also choose the following hyper-priors:

$$\lambda_{U_d}, \lambda_{V_d} \sim \text{Gamma}(\alpha_0, \beta_0). \tag{7.11}$$

We choose this simplified model because the proposed method benefits mainly from performing a large number of inexpensive updates (i.e. collecting many samples) per unit time rather than very expensive but high quality updates. The above model is well suited for this because each latent vector can be updated in linear $\mathcal{O}(D)$ time. At the same time, we still benefit from the power of Bayesian inference through marginalization of the important regularization parameters $\Lambda = \{\Lambda_U, \Lambda_V\}$ as well as $\mathbf{U}$ and $\mathbf{V}$.

Although it is possible to apply our method to the model in Section 7.2, updating the full covariance matrix is more expensive ($\mathcal{O}(D^2)$ time per update) and therefore requires more time to converge without significant gain in accuracy (as per our pilot experiments).

In the following section, we first present our algorithm in a single machine setting and later extend it for distributed inference. We alternate between sampling from $p(\mathbf{U}, \mathbf{V}|\mathbf{R}, \Lambda)$ using SGLD and sampling from $p(\Lambda|\mathbf{R}, \mathbf{U}, \mathbf{V})$ using Gibbs.

7.3.1 Sampling from $P(\mathbf{U}, \mathbf{V}|\mathbf{R}, \Lambda)$ using SGLD

Since, usually only $N \ll M \times L$ ratings are observed, the rating matrix $\mathbf{R}$ is stored using a sparse representation as $\mathcal{X} = \{x_n = (p_n, q_n, r_n)\}_{n=1}^{N}$, where each x_n is a (user, item, rating) tuple and N is the number of observed ratings. The gradient of the log-posterior w.r.t.[2] U_i is:

$$G(\mathcal{X}) = \sum_{n=1}^{N} g_n(U_i; \mathcal{X}) - \Lambda_U U_i \tag{7.12}$$

where

$$g_n(U_i; \mathcal{X}) = \mathbb{I}[p_n = i|\mathcal{X}](r_n - U_{p_n}^{\top} V_{q_n}) V_{q_n} \tag{7.13}$$

Here $\mathbb{I}[p_n = i|\mathcal{X}]$ is an indicator function that equals 1 if the n-th tuple in $\mathcal{X}$ pertains to user i and 0 otherwise. To use SGLD, we need an unbiased estimate of this gradient that can be computed cheaply from a mini-batch.

[2]We derive only w.r.t. U_i. Update rules for other parameters can be obtained by the same procedure.

One way to obtain this is by subsampling a mini-batch $\mathcal{M} = \{(p_n, q_n, r_n)\}_{n=1}^{m}$ of m tuples from $\mathcal{X}$ and computing the following stochastic approximation of the gradient:

$$G_1(\mathcal{M}) = N\bar{g}(U_i; \mathcal{M}) - \Lambda_U U_i \tag{7.14}$$

where, $\bar{g}(U_i; \mathcal{M}) = \frac{1}{m}\sum_{n=1}^{m} g_n(U_i; \mathcal{M})$. Note that the mini-batch is subsampled from the complete dataset $\mathcal{X}$ and not just from the tuples associated with user i. The expectation of G_1 over all possible mini-batches is:

$$\begin{aligned} \mathbb{E}_{\mathcal{M}}[G_1(\mathcal{M})] &= \mathbb{E}_{\mathcal{M}}\left[N\bar{g}(U_i; \mathcal{M})\right] - \Lambda_U U_i \\ &= \sum_{n=1}^{N} g_n(U_i; \mathcal{X}) - \Lambda_U U_i \\ &= G(\mathcal{X}). \end{aligned}$$

Since G_1 is an unbiased estimator of the true gradient, we can use it for computing SGLD updates. However, note that G_1 is non-zero even for users that are not in the mini-batch $\mathcal{M}$, because of the prior gradient term $-\Lambda_U U_i$. Therefore, we have to update the parameters for all users in every iteration, which is very expensive.

If we were to update only the parameters of users who have ratings in the mini-batch $\mathcal{M}$, the estimator can be written as:

$$G_2(\mathcal{M}) = N\bar{g}(U_i; \mathcal{M}) - \mathbb{I}[i \in \mathcal{M}_p]\Lambda_U U_i \tag{7.15}$$

where $\mathbb{I}[i \in \mathcal{M}_p]$ is equal to 1 if $\mathcal{M}$ contains a tuple associated with user i and 0 otherwise. However, G_2 is not an unbiased estimator of the true gradient:

$$\mathbb{E}_{\mathcal{M}}[G_2(\mathcal{M})] = \sum_{n=1}^{N} g_n(U_i; \mathcal{X}) - h_{i*}\Lambda_U U_i. \tag{7.16}$$

where $h_{i*} = \mathbb{E}_{\mathcal{M}}[\mathbb{I}[i \in \mathcal{M}_p]]$, i.e. the fraction of mini-batches that contains at least one tuple associated with user i (among all possible mini-batches). If the mini-batches are sampled with replacement, we can compute this as:

$$h_{i*} = 1 - \left(1 - \frac{N_{i*}}{N}\right)^m \tag{7.17}$$

where $N_{i*} = \sum_{n=1}^{N} \mathbb{I}[p_n = i|\mathcal{X}]$, the number of ratings by user i in the complete dataset $\mathcal{X}$. Thus, we can remove the bias in G_2 by multiplying the gradient of the prior term with h_{i*}^{-1} as follows:

$$G_3(\mathcal{M}) = N\bar{g}(U_i; \mathcal{M}) - \mathbb{I}[i \in \mathcal{M}_p] h_{i*}^{-1} \Lambda_U U_i. \tag{7.18}$$

G_3 is an unbiased estimator of the true gradient G and is non-zero only for users that have at least one rating in $\mathcal{M}$. Thus we need to update only a subset of user features in each iteration. The SGLD update rule (for users with ratings in $\mathcal{M}_t$) is:

$$U_{i,t+1} \leftarrow U_{i,t} + \frac{\epsilon_t}{2}\left\{N\bar{g}(U_{i,t}; \mathcal{M}_t) - \frac{\Lambda_U U_{i,t}}{h_{i*}}\right\} + \nu_t \tag{7.19}$$

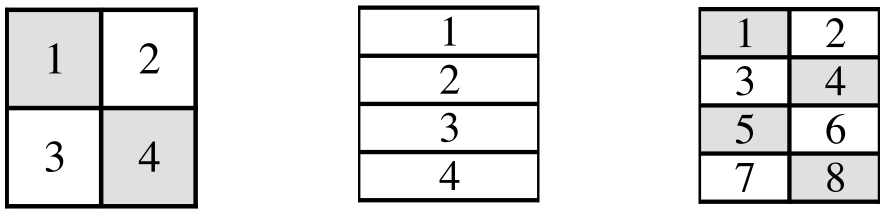
where $\nu_t \sim \mathcal{N}(0, \epsilon_t)$.

7.3.2 Sampling from $P(\Lambda|\mathbf{U}, \mathbf{V})$ using Gibbs sampling

Due to the conditional conjugacy, we can easily sample from the conditional distribution $p(\Lambda|\mathbf{U}, \mathbf{V})$ using Gibbs sampling:

$$\lambda_{U_d}|\mathbf{U}, \mathbf{V} \sim \text{Gamma}\left(\alpha_0 + \frac{L}{2}, \beta_0 + \frac{1}{2}\sum_{i=1}^{L} U_{di}^2\right), \tag{7.20}$$

$$\lambda_{V_d}|\mathbf{U}, \mathbf{V} \sim \text{Gamma}\left(\alpha_0 + \frac{M}{2}, \beta_0 + \frac{1}{2}\sum_{i=1}^{M} V_{dj}^2\right). \tag{7.21}$$



(a) square (b) column (c) hybrid

Figure 7.1: Block split schemes.

Note that, if this is also computationally demanding, we can consider updating Λ using SGLD or the mini-batch Metropolis-Hastings algorithm [Korattikara et al., 2014, Bardenet et al., 2014].

7.4 Distributed Inference

For distributed inference, we partition the rating matrix $\mathbf{R}$ into a number of blocks. Fig. 7.1 shows a few different ways of partitioning $\mathbf{R}$. Two blocks are said to be *orthogonal* to each other if the users and items in one block do not appear in the other block. A set of two or more mutually orthogonal blocks is called an *orthogonal block group* (or simply, orthogonal group). For example, the two gray-colored blocks (1 and 4) in Fig. 7.1 (a) are orthogonal to each other and thus form an orthogonal group. In Fig. 7.1 (b), the blocks are not orthogonal because all columns are shared. In this case, we say that each block by itself is an orthogonal group.

The blocks are then distributed to workers in such a way that all blocks are assigned and a worker has at least one block. In the following, we assume for simplicity that each worker is a single-core machine. However, it is easy to generalize our algorithm to take advantage of multi-core (or threads) workers with shared memory support.

We will now describe our distributed algorithm for BPMF. First, imagine that there is only one Markov chain c (but the dataset is distributed across multiple workers). A central parameter server holds the *global* parameters $\mathbf{U}^c$ and $\mathbf{V}^c$ of chain c. Since Λ depends only on $\mathbf{U}^c$ and $\mathbf{V}^c$, it is easy to update Λ at the parameter server using Gibbs as per Eqns. 7.21 and 7.20. Thus, we will focus on the DSGLD part of the chain that samples from $p(\mathbf{U}, \mathbf{V}|\mathbf{R}, \Lambda)$.

Each sampling round consists of the following steps: (1) The parameter server picks a block s via a block-scheduler and sends the corresponding sub-parameter $\mathbf{U}^{(c,s)}$ and $\mathbf{V}^{(c,s)}$ to the block's worker. (2) The worker updates the sub-parameter by running DSGLD (see section 7.4.1 for update equations) for a number of iterations using its local block of ratings. (3) The worker sends the final sub-parameter state back to the parameter server. (4) The parameter server updates its global copy to the new sub-parameter state.

Thus, the Markov chain jumps among the distributed blocks through the corresponding workers and updates the sub-parameters associated with the block chosen in each round. Since each iteration of local DSGLD updates requires only a mini-batch of data, sampling is very fast. Also, communication overhead is low because a) the multiple local updates (iterations) performed within a round do not require any communication b) only a small sub-parameter associated with a specific block is transferred in each round. There are two levels of parallelization that we use to further speed up sampling.

1. **Parallel updates within a chain**: a chain can update sub-parameters $\mathbf{U}^{(c,s_1)}$ and $\mathbf{U}^{(c,s_2)}$ in parallel if the blocks s_1 and s_2 are orthogonal to each other. For example, in Fig. 7.1 (a), updating block 1 *and then* block 4 produces the same result as updating both in parallel. This makes the algorithm progress faster in terms of number of updated parameters per round. The actual performance improvement is dependent on the size of the orthogonal group. For instance, with a 4×4 split, the algorithm will update the parameters faster than with a 2×2 split because more parameter blocks can

be updated in parallel. However, updates in smaller blocks can be noisier, because the gradients computed from smaller blocks will have higher variance. Therefore, at some point the loss in performance caused by noisier updates on small blocks can exceed the gain obtained by faster updating of the parameters.

2. **Multiple parallel chains**: we can run as many chains in parallel as we like, subject to only computational resource constraints. Each chain can update its parameters in parallel independent of other chains. Hence, the chains are asynchronous in the sense that the status of a chain does not block other chains unless the chains conflict for computation resources. For the split in Fig. 7.1 (a), one chain can update using the gray block group while another chain is using the white block group. Or both chains can use the same block if we assume a shared memory multi-threaded implementation. By running multiple chains in parallel, we effectively multiply the number of collected samples by the number of parallel chains. Since the variance of an MCMC estimator is inversely proportional to the number of samples, fast sample generation will compensate for the low mixing rate of SGLD. Also, by initializing the different chains in different places of parameter space, we can explore multiple local minima. This is especially important for large-scale high dimensional problems where the time budget is usually not enough for a single chain to mix between different local minima.

An illustration of these ideas is given in Fig. 7.2. Algorithms 7 and 8 describe the operations at the parameter server and workers respectively.

A proper block splitting scheme can be chosen according to the characteristics of the problem and available resources. In other words, we can trade-off *within-chain* parallelization and *between-chain* parallelization. For example, given S workers, by using a squared split as in Fig. 7.1 (a), we can run $\sqrt{S}$ chains in parallel where each chain updates $\sqrt{S}$ blocks in parallel. This way we maximize the within-chain parallelism. On the other hand, by reducing the size of orthogonal groups, we can decrease the within-chain parallelism in order

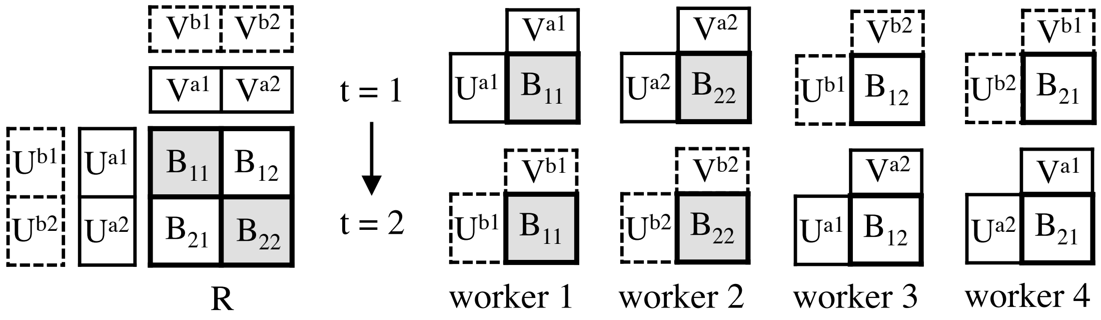


Figure 7.2: An example illustration. On the left, a matrix $\mathbf{R}$ is partitioned into 2×2 blocks, $\mathbf{B}_{11}, \cdots, \mathbf{B}_{22}$. There are two orthogonal groups (the gray $(\mathbf{B}_{11}, \mathbf{B}_{22})$ group and the white $(\mathbf{B}_{12}, \mathbf{B}_{21})$ group). We run two independent chains, chain a with parameters $\mathbf{U}^a$ and $\mathbf{V}^a$ (solid-line rectangles) and chain b, with parameters $\mathbf{U}^b$ and $\mathbf{V}^b$ (dotted-line rectangles). Given four workers, we assign a block to each worker. At round $t = 1$, chain a updates using the gray orthogonal group and chain b using the white orthogonal group. Note that the entire $\mathbf{U}$ and $\mathbf{V}$ matrices of both chains are updated in this single round. In the next round, the chains are assigned to the next orthogonal groups by the block-scheduler.

to increase the between-chain parallelization, i.e. number of parallel chains. At an extreme of this approach, we can let each block become an orthogonal group by itself as in Fig. 7.1 (b) and run S independent chains in parallel. Note that in this case, we can choose not only the column splitting but any splitting scheme. Our experiment results suggest to maximize the within-chain parallelism as the dataset size increases. For smaller datasets, we may benefit more from the generalization performance of a large number of parallel chains than from a smaller number of chains using the block orthogonality.

7.4.1 Distributed SGLD Update

Since $\mathcal{X}$ (the sparse representation of $\mathbf{R}$) is partitioned into S blocks $\mathcal{X}^{(1)}, \ldots, \mathcal{X}^{(S)}$, each worker uses only one of the $\mathcal{X}^{(s)}$ for computing updates. Thus, we need to modify the bias correctors in Eqn. (7.17) so that the gradient estimator remains unbiased under this constraint. If we assume $\cup_{s=1}^{S} \mathcal{X}^{(s)} = \mathcal{X}$ and $\cap_{s=1}^{S} \mathcal{X}^{(s)} = \emptyset$, and that worker s is visited with normalized frequency $v^{(s)}$, the correction factors for users and items can be shown to be,

Algorithm 7 DSGLD for BPMF at the parameter server

```
 1: Initialize model parameters of each chain $\{\mathbf{U}_1^c, \mathbf{V}_1^c, \Lambda_1^c\}_{c=1}^C$, step sizes $\{\epsilon_t\}$
 2: for each chain $c$ parallel do
 3:     for $t$=1:max_iter do
 4:         $\mathcal{B}_c \leftarrow$ GET_ORTHO_BLOCK_GROUP$(c, t)$
 5:         for worker $s \in \mathcal{B}_c$ do
 6:             $\mathbf{U}_{t+1}^{(c,s)}, \mathbf{V}_{t+1}^{(c,s)} \leftarrow$ WKR_ROUND$(\mathbf{U}_t^{(c,s)}, \mathbf{V}_t^{(c,s)}, \Lambda_t^{(c)}, \epsilon_t)$
 7:         end for
 8:         if not burn-in then
 9:             Store $\mathbf{U}_{t+1}^{(c)}, \mathbf{V}_{t+1}^{(c)}$ as a sample of chain $c$
10:             Sample $\Lambda_{t+1}^{(c)} | \mathbf{U}_{t+1}^{(c)}, \mathbf{V}_{t+1}^{(c)}$ using Eqn. (7.20) and (7.21)
11:         end if
12:     end for
13: end for
```

Algorithm 8 DSGLD for BPMF at worker s

```
 1: Initialize $\bar{h}_{i*}, \bar{h}_{*j}$, round length $\gamma$, mini-batch size $m$
 2: function WKR_ROUND$(\mathbf{U}^{(c,s)}, \mathbf{V}^{(c,s)}, \Lambda^{(c)}, \epsilon_t)$
 3:     for $t = 1 : \gamma$ do
 4:         Sample a mini-batch $\mathcal{M}_t$ from $\mathcal{X}^{(s)}$
 5:         for each user $i$ and item $j$ in $\mathcal{M}_t$ parallel do
 6:             Update $U_i, V_j$ using Eqn. (7.25) and (7.26)
 7:         end for
 8:     end for
 9:     Send updated $\mathbf{U}^{(c,s)}$ and $\mathbf{V}^{(c,s)}$ to the parameter server
10: end function
```

respectively:

$$\bar{h}_{i*} = \sum_{s=1}^{S} v^{(s)} h_{i*}^{(s)}, \qquad \bar{h}_{*j} = \sum_{s=1}^{S} v^{(s)} h_{*j}^{(s)} \tag{7.22}$$

where:

$$h_{i*}^{(s)} = 1 - \left(1 - \frac{N_{i*}^{(s)}}{N^{(s)}}\right)^m, \;\; h_{*j}^{(s)} = 1 - \left(1 - \frac{N_{*j}^{(s)}}{N^{(s)}}\right)^m \tag{7.23}$$

here $N^{(s)} = |\mathcal{X}^{(s)}|$, the total number of ratings in s, and

$$N_{i*}^{(s)} = \sum_{n=1}^{N^{(s)}} \mathbb{I}[p_n = i | \mathcal{X}^{(s)}], \quad N_{*j}^{(s)} = \sum_{n=1}^{N^{(s)}} \mathbb{I}[q_n = j | \mathcal{X}^{(s)}]. \tag{7.24}$$

i.e. the number of ratings by user i and of item j respectively in s. Therefore, the local DSGLD update rule using block $\mathcal{X}^{(s)}$ is:

$$U_{i,t+1} \leftarrow U_{i,t} + \frac{\epsilon_t}{2}\left\{\frac{N^{(s)}}{v^{(s)}}\bar{g}(U_{i,t}; \mathcal{M}_t^{(s)}) - \frac{\Lambda_U U_{i,t}}{\bar{h}_{i*}}\right\} + \nu_t \tag{7.25}$$

$$V_{j,t+1} \leftarrow V_{j,t} + \frac{\epsilon_t}{2}\left\{\frac{N^{(s)}}{v^{(s)}}\bar{g}(V_{j,t}; \mathcal{M}_t^{(s)}) - \frac{\Lambda_V V_{j,t}}{\bar{h}_{*j}}\right\} + \nu_t. \tag{7.26}$$

The above rule updates only the sub-parameter associated with block s using only rating tuples in s.

7.5 Experiments

7.5.1 Algorithms and Models

	Optimization	MCMC
Single Machine	SGD	SGLD, Gibbs
Distributed	DSGD	DSGLD

Table 7.1: Algorithms.

We compared five algorithms: SGD, DSGD, SGLD, DSGLD, and Gibbs sampling. As shown in Table 7.1, each algorithm can be classified based on whether it is running on a single machine or a distributed architecture, and also based on whether it is an optimization or MCMC algorithm. Since Gibbs sampling was very slow, we update user/item features in parallel (as suggested in [Salakhutdinov and Mnih, 2008]) using multiple cores of a single machine. Thus, by Gibbs sampling we will mean the parallelized (but not distributed) version from now on.

For DSGLD, we tested two block-splitting schemes. Given S workers, DSGLD-S ('S' stands for square) partitions $\mathbf{R}$ into $\sqrt{S} \times \sqrt{S}$ blocks as in Fig. 7.1 (a), i.e. DSGLD-S tries to maximize the within-chain parallelism by using as many orthogonal blocks as possible. We run $\sqrt{S}$ parallel chains, where each chain updates $\sqrt{S}$ sub-parameter blocks in parallel using $\sqrt{S}$ workers. Therefore, all chains can update all parameter at every round. The second splitting scheme, called DSGLD-C ('C' stands for column blocks) divides $\mathbf{R}$ into S blocks as shown in Fig. 7.1(b). We split $\mathbf{R}$ along the rows because in our experiments we have many more users than items. The blocks in DSGLD-C are not orthogonal because all columns are shared, so we just run S independent parallel chains.

For Gibbs sampling, we use the original BPMF model[3] described in Section 7.2. For the other algorithms, we slightly extend the model described in Section 7.3 (as in Chen et al. [2014], Koren et al. [2009]). The extension includes user and item specific bias terms a_i and b_j respectively so that the predictions are modeled as:

$$R_{ij} \approx U_i^\top V_j + a_i + b_j \tag{7.27}$$

We use the following priors and hyper-priors for a_i and b_j:

$$a_i \sim \mathcal{N}(0, \lambda_a^{-1}), \quad b_j \sim \mathcal{N}(0, \lambda_b^{-1}),$$
$$\lambda_a, \lambda_b \sim \text{Gamma}(\alpha_0, \beta_0).$$

For $\mathbf{U}$ and $\mathbf{V}$, we use the same priors and hyper-priors as described in Section 7.3. Note that, in the new model, we have to sample $a_i, b_j, \lambda_a, \lambda_b$ in addition to $\mathbf{U}, \mathbf{V}, \Lambda_U, \Lambda_V$. The

[3]Using the simplified model does not reduce the computation complexity of the Gibbs sampling.

DSGLD update rules for a_i and b_j are:

$$a_{i,t+1} \leftarrow a_{i,t} + \frac{\epsilon_t}{2}\left\{\frac{N^{(s)}}{v^{(s)}}\bar{g}(a_{i,t}; \mathcal{M}_t^{(s)}) - \frac{\lambda_a a_{i,t}}{\bar{h}_{i*}}\right\} + \nu_t \tag{7.28}$$

$$b_{j,t+1} \leftarrow b_{j,t} + \frac{\epsilon_t}{2}\left\{\frac{N^{(s)}}{v^{(s)}}\bar{g}(b_{j,t}; \mathcal{M}_t^{(s)}) - \frac{\lambda_b b_{j,t}}{\bar{h}_{*j}}\right\} + \nu_t. \tag{7.29}$$

The main goal of our experiments is to answer the following questions:

- *Accuracy*: How does DSGLD compare to other methods in terms of prediction RMSE?
- *Speed*: How fast can DSGLD achieve the RMSE obtained by 1) optimization algorithms (SGD, DSGLD) 2) Gibbs sampling?
- *Factors which affect the above*: The number of workers, number of chains, block splitting schemes and the latent factor dimension.

7.5.2 Setup

Dataset	# users	# items	# ratings
Netflix	480K	18K	100M
Yahoo	1.8M	136K	700M

Table 7.2: Datasets.

We compare all 5 algorithms on two large datasets, Netflix movie ratings [Bennett and Lanning, 2007] and Yahoo music ratings (details in Table 7.2). To the best of our knowledge, the Yahoo dataset was one of the largest publicly available datasets when we performed the experiments. Note that the Yahoo dataset we use here is different from the one used in the KDD'11 Cup [Dror et al., 2012] (which has ~250M music ratings and is often referred to by the same name). For the Netflix dataset, we use 80% of the ratings for training and the remaining 20% for testing as in [Ding et al., 2014]. For the Yahoo dataset, the memory footprint was around 17GB for the train and test ratings, and around 1GB for **U** and **V** with $D = 60$ in our 64-bit float based implementation. The memory footprint of the Netflix dataset was relatively small.

We used Julia [Bezanson et al., 2014] to configure the cluster and execute the core routines of the algorithms. The core routines were implemented in `C` for high performance. For distributed computing, we used Amazon EC2 instances [ama] of type "r3" which were equipped with Intel Xeon 2.5 GHz CPUs and had memory configurable up to 244GB. Although the instances had multiple cores, we restricted all algorithms, except Gibbs sampling, to run on a single-core. For Gibbs sampling, we used a 12-core machine with the same CPU speed. All algorithms were implemented as an in-memory execution model and thus no disk I/O overheads were considered.

We annealed the step size according to the schedule $\epsilon_t = \epsilon_0(1 + t/\kappa)^{-\gamma}$, (as in [Ahn et al., 2012, Patterson and Teh, 2013]) which satisfies the convergence conditions in Eqn. (**??**). We found κ, which controls the decay rate, over the range $\kappa = [10, 50, 100, 500, 1000, 1500]$. The initial step size ϵ_0 was also selected from [9e-6,1e-6] for Netflix and [3e-6,8e-7] for Yahoo. More detailed settings are given in the Appendix. We decreased the stepsize after every *round* which we set to 50 updates. We used $\gamma = 0.51$ in all experiments.

We set the hyperparameters $\tau = 2.0$ and $\alpha_0 = 1.0$ for all experiments. We used $\beta_0 = 1.0$ for all algorithms except SGLD and DSGLD. For SGLD and DSGLD, the scale of the prior gradients sometimes became large due to multiplication by the bias correctors $1/h_{i*}$ and $1/h_{*j}$. In this case, instead of increasing the mini-batch size to reduce the scale of the correctors, we used a more appropriate scale parameter for the Gamma prior distribution ($\beta_0 = 300$), to stabilize the scale of precisions sampled from the posterior Gamma distribution.

Mini-batch sizes were set to 50K data points for Netflix and 100K for Yahoo. The initial values for the precisions Λ were all chosen to be 2.0 after testing over a range $[10, 5, 2, 1, 0.1, 0.01]$. In SGLD and DSGLD, the precision parameters were sampled every 50 rounds after burn-in.

We discarded (burned) samples until the RMSE reached 0.85 for Netflix and 1.08 for Yahoo. For DSGLD, which deploys multiple chains, we used the arithmetic mean of the RMSE of all chains to determine whether burn-in has completed. We set the thinning interval to 10 rounds, i.e. we use only every 10^{th} sample to compute the average prediction. The Gibbs sampler in our experiments was initialized near a MAP state which we found using SGD during burn-in.

Running DSGLD requires a block scheduler (line 4 in Algorithm 7) that determines which blocks (workers) are used by each chain in a round. In our experiments, the blocks and the orthogonal groups were chosen beforehand and were assigned to chains deterministically using a cyclic-shift (rotation) at every round with equal visiting frequency. This scheduling policy is illustrated in Fig. 7.2.

7.5.3 Results

Convergence and wall-clock time

We first compare the RMSE of the algorithms as a function of computational time. In this experiment, we set D=30 for both datasets and used 9 workers for Netflix and 16 workers for Yahoo. Given S workers, we used a $\sqrt{S} \times \sqrt{S}$ block-split for DSGLD-S, $S \times 1$ split for DSGLD-C and $S \times S$ split for DSGD. The total runtime was set to 50K seconds ($\approx$14 hours) for Netflix and 100K seconds ($\approx$27 hours) for Yahoo. In both Figs. 7.3 and 7.4, the x-axis is in log-scale for the figure on the left and in linear-scale for the figure on the right.

In Fig. 7.3, we show results on the Netflix dataset (which is smaller than the Yahoo dataset). We see that in the early (burn-in) stage, all algorithms except Gibbs reduce error at a similar rate. Even though DSGLD-S and DSGD uses block orthogonality to update the sub-parameters of a chain in parallel, because of communication overheads, the gain in

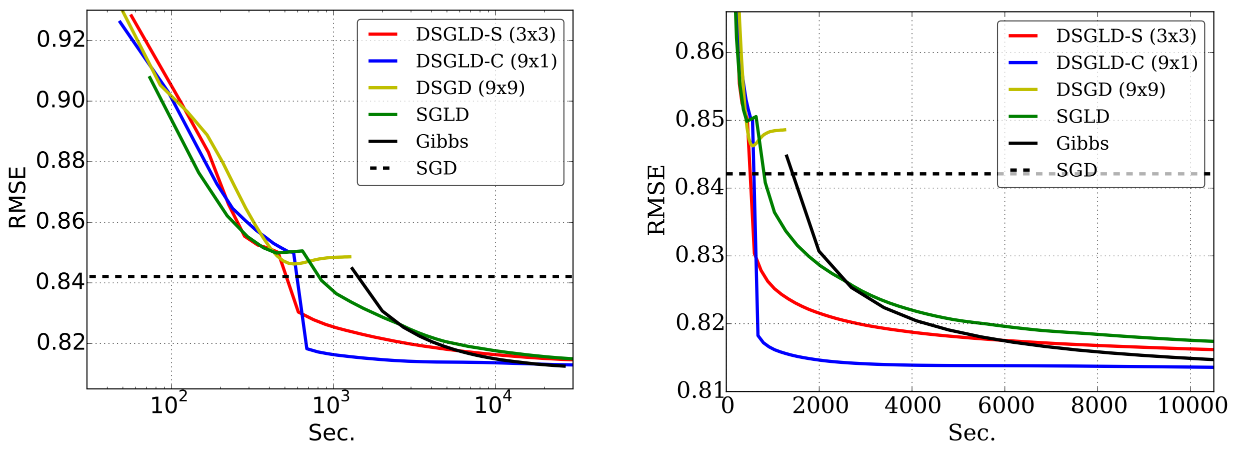


Figure 7.3: Netflix dataset ($D = 30$).

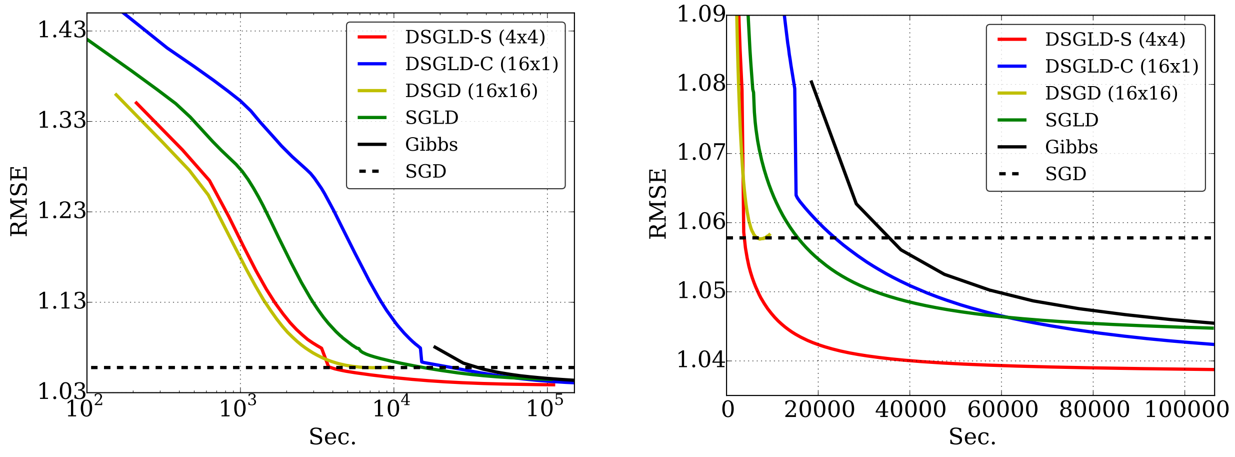


Figure 7.4: Yahoo Music Rating dataset ($D = 30$).

speed-up is not enough to outperform a non-distributed algorithm like SGLD which is able to reduce the error at a similar rate (because the dataset size is not very large) without any communication overhead. Note that because there are many chains for DSGLD, we plot the RMSE from only one chain during burn-in. The variance of RMSE across the chains was small during burn-in.

When the burn-in phase ends at around 500 - 700 seconds, MCMC algorithms (SGLD, DSGLD, and Gibbs) begin to collect samples and average their predictions over the samples, while DSGD does not and begins to overfit. Interestingly, at this point, we see a remarkably steep decrease in error for both DSGLD-S and DSGLD-C. In particular, we see the largest decrease for DSGLD-C which deploys 9 independent chains (whereas DSGLD-S uses

3 chains). Note that this is not solely a consequence of collecting a larger number of samples from multiple chains. We believe that the averaged prediction using many independent chains provides better generalization because many modes are likely to be explored (or, a large area of a single broad mode can be covered quickly if many chains reside there). After more investigation, we indeed observed that the same number of samples collected from a single chain (e.g. SGLD) cannot achieve the same level of accuracy obtained with multiple randomly initialized chains. Furthermore, we observed that given a lot more computational time, SGLD and DSGLD-S can approach the RMSE obtained by DSGLD-C as they also get a chance to explore other modes or to cover a larger area of a single mode. We will revisit the effect of multiple chains in more detail in the next section. Finally, note that Gibbs sampling achieves lower RMSE than DGSLD-C after around 20K seconds (5.5 hours) as shown in Fig. 7.3 left (but the difference to DSGLD-C is very small). Note that for this dataset, D and $L + M$ were not too large and we used 12-core single machine for parallel Gibbs sampling. Therefore the computational cost of each iteration was not extremely high.

We present our results on the Yahoo dataset in Fig. 7.4 with $S = 16$ workers. A remarkable point is that, here, unlike with the Netflix dataset, DSGLD-S outperforms DSGLD-C. This is because using orthogonal blocks increases the number of parameters updated per round, resulting in increased convergence speed even after offsetting the communication overhead. As expected, a similar effect is observed for DSGD. The progress of parameter updates in DSGLD-C is relatively slow, requiring $S = 16$ rounds to update all the parameters. Besides, DSGLD-C has a much larger communication overhead because the full matrix $\mathbf{V}$ has to be transferred between the parameter server and each of the workers, whereas only a small block of $\mathbf{V}$ is transferred in DSGLD-S. Specifically, in DSGLD-C the parameter server sends and receives packets of total size $\mathcal{O}((L + SM)D)$ per round whereas in DSGLD-S the total packet size is only $\mathcal{O}((L + M)D)$. Although DSGLD-C is rather slow during burn-in, after burn-in we still see a faster decrease in RMSE compared to SGLD because multiple chains can mix better. Gibbs sampling converges slower than it does on the Netflix dataset because for the Yahoo dataset the number of latent vectors to update, i.e. $L + M$, increases by a factor of four, and the number of ratings, N, by a factor of seven.

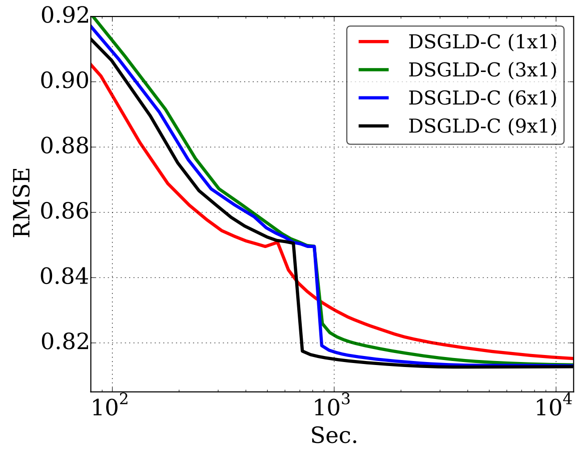
For the Netflix dataset, after 1K seconds, DSGLD-C achieved the RMSE (0.8145) that the Gibbs sampler obtains at 10K seconds. Similarly, after 11K seconds, DSGLD-S achieved the RMSE (1.0454) that the Gibbs sampler obtains at 100K seconds. Therefore, the proposed method converges an order of magnitude faster than Gibbs sampling on both datasets, which is especially important when we only have a limited computational budget.

DSGD converges to a prediction RMSE of 0.8462 on Netflix and 1.0576 on Yahoo after 1K seconds and 10K seconds respectively. Given the same amount of computational time, DSGLD achieves an error of 0.8161 on Netflix and 1.0465 on Yahoo, a relative improvement of 3.7% and 1.1%. After convergence, the RI increases to 4.1% for Netflix and 1.8% for Yahoo (See Table. 7.4 and Table. 7.3).

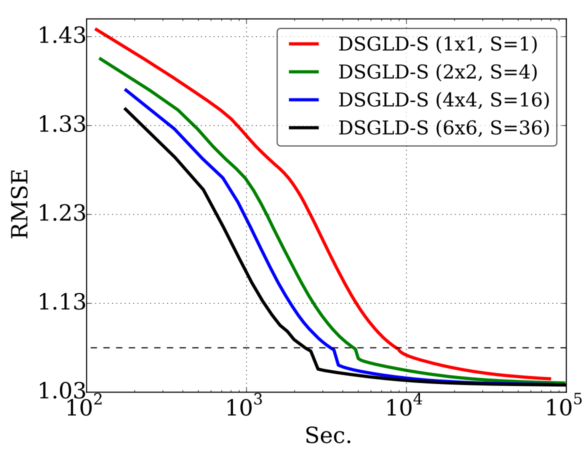
Number of chains and workers

We also investigated the effect of the number of chains and the number of workers. The results are presented in Fig. 7.5. According to the observations from the previous experiment, we used DSGLD-C for Netflix and DSGD-S for Yahoo to study this effect. The latent feature dimension was set to D=30.

In Fig. 7.5 (a), we compare DSGLD-C with $[1, 3, 6, 9]$ chains (and workers) and in each case we evenly split the rows of the rating matrix between the chains. Note that DSGLD-C (1x1) is the same as SGLD running on a single-machine. We see that during burn-in DSGLD-C (1x1) converges faster than the other splits because there is no communication overhead. After burn-in, when the chains start averaging predictions, we see a sharp decrease in error for the other splits. Although splits with more chains decrease error much faster, they all



(a) DSGLD-C on the Netflix dataset



(b) DSGLD-S on the Yahoo dataset

Figure 7.5: The effect of the number of chains, number of workers, and block split.

eventually converge to a similar value. Due to poor mixing, a single chain (i.e. SGLD) converges very slowly.

In Fig. 7.5 (b), we show results for DSGLD-S on the Yahoo dataset. We increased the number of workers to $[1, 4, 16, 36]$ to compare $[1, 2, 4, 6]$ parallel chains. Again DSGLD-S (1x1) denotes SGLD running on a single machine. We see that SGLD converges much more slowly because the dataset is larger than Netflix and SGLD has to update more parameters sequentially. Using more orthogonal blocks, DSGLD-S can update more parameters in parallel and we see more speed-up as we increase the number of workers. Although we increase the number of workers quadratically between the experiments, the packet size transferred between the parameter server and the workers stays constant at $\mathcal{O}((L+M)D)$ because the block size also reduces accordingly. Even after burn-in (horizontal dotted black line at 1.08 RMSE) we see that with more chains we can decrease the error faster. This is because (i) multiple chains help to mix better by exploring a broader space (ii) each chain can mix faster by updating orthogonal blocks in parallel.

D	SGD	DSGD	SGLD	DSGLD-C	Gibbs
30	0.8421	0.8462	0.8143	0.8126	**0.8118**
	-3.63%	-4.13%	-0.21%	-	+0.09%
60	0.8447	0.8428	0.8097	**0.8074**	0.8259
	-4.62%	-4.38%	-0.28%	-	-2.29%
100	0.8415	0.8395	0.8082	**0.8043**	0.8339
	-4.63%	-4.37%	-0.48%	-	-3.68%

Table 7.3: RMSE and relative improvement (RI). Netflix. The percentage shown below each RMSE value is the relative improvement.

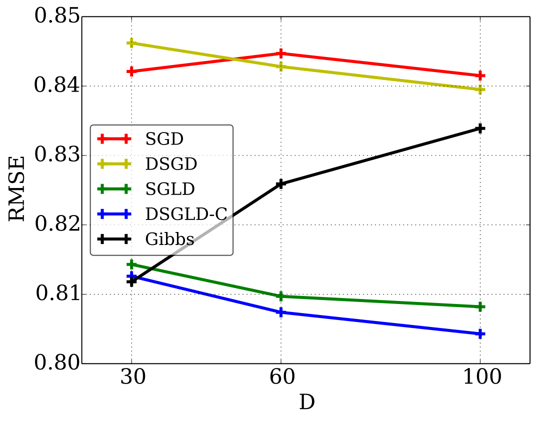
D	SGD	DSGD	SGLD	DSGLD-S	Gibbs
30	1.0578	1.0576	1.0448	**1.0387**	1.0454
	-1.83%	-1.82%	-0.58 %	-	-0.64%
60	1.0548	1.0588	1.0351	**1.0267**	1.0364
	-2.73%	-3.13%	-0.82%	-	-0.94%
100	1.0567	1.0631	1.0335	**1.0229**	1.0339
	-3.30%	-3.93%	-1.04%	-	-1.08%

Table 7.4: RMSE and relative improvement (RI). Yahoo. The percentage shown below each RMSE value is the relative improvement.

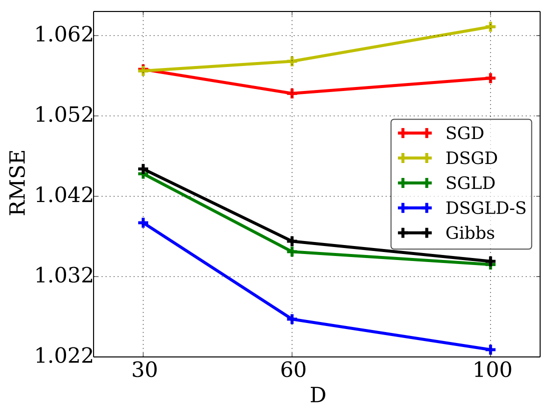
Latent feature dimension

In Fig. 7.6, we show how the latent feature dimension affects the final RMSE. The final RMSE on Netflix is measured after 50K seconds (14 hours) of computational time, because by then all algorithms had converged (except Gibbs sampling which is expected to take much longer). On the Yahoo dataset, we increased the computational time to 100K secs (1 day), 200K secs (2.3 days), and 300K secs (3.5 days) for D=[30,60,100], respectively, to give the Gibbs sampler more time to converge. In table 7.3 and 7.4, we show the RMSEs of the different algorithms and the *relative* improvement (or deterioration) compared to DSGLD. The Relative Improvement (RI) of an algorithm x is defined as $RI(x) = (r_d - r_x)/r_d$, where r_x is the RMSE achieved by algorithm x and r_d is the RMSE obtained using DSGLD.

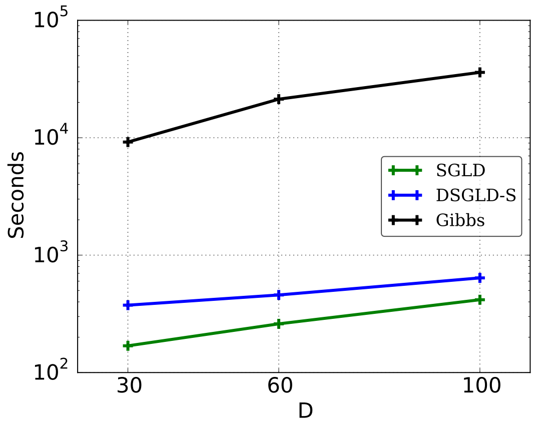
In both Fig. 7.6 (a) and (b), we see a large difference in performance between SG-MCMC (SGLD and DSGLD) and the optimization methods (SGD and DSGD). The RI is 3.6%−4.6%



(a) RMSE on Netflix



(b) RMSE on Yahoo



(c) Required time per sample

Figure 7.6: The effect of the latent feature dimension. (a) and (b) show RMSE for $D = [30, 60, 100]$ on (a) the Neflix dataset and (b) the Yahoo music ratings dataset. The maximum computational time was set to 50K seconds for Netflix and 100K (D=30), 200K (D=60), and 300K (D=100) seconds for Yahoo. (c) shows time (in seconds) required to draw a single sample on the Yahoo dataset.

on Netflix and $1.8\% - 3.9\%$ on the Yahoo dataset. As observed in [Salakhutdinov and Mnih, 2008], we see that the optimization methods do not consistently improve with increasing D. One reason is that optimization methods are highly sensitive to the hyperparameter values which become difficult to tune as the model becomes more complex. However, our method consistently improves as we increase D, because the hyper-parameters are sampled from their posterior distributions. We also see that the performance of Gibbs sampling on Netflix gets worse as D increases, because we used the same amount of computational budget for all D although the computation complexity increases as D does. On the Yahoo dataset on which we increase computational time as we increase D, we see that the RMSE for Gibbs increases as D increases, but is still lower than that of DSGLD.

In Fig. 7.6 (c), we compare the time (in seconds) required to draw a single sample for the three sampling algorithms at different values of D on the Yahoo dataset. We see that Gibbs sampling is almost two orders of magnitude slower than SGLD. For D=100, SGLD, DSGLD-S, and Gibbs generated 688, 460, and 8 samples respectively in 300K seconds of computational time. For Netflix, Gibbs generated around 100 samples in 50K seconds for D=30. Thus, even though the Gibbs sampler can produce higher quality samples (in terms of lower auto-correlation), the sampling speed is so slow that it cannot satisfactorily handle large scale datasets.

7.6 Discussion

Most applications of matrix factorization to recommender systems are based on stochastic gradient optimization algorithms because these are the only ones that can computationally handle very large datasets. However, by restricting ourselves to such simple algorithms, we miss out on all the advantages of Bayesian modelling such as quantifying uncertainty, controlling over-fitting, incorporating prior information and better prediction accuracy. In this chapter, we introduced a novel algorithm for scalable distributed Bayesian matrix factorization that achieves the best of both worlds, i.e. it inherits all the advantages of Bayesian inference at the speed of stochastic gradient optimization.

Our algorithm, based on Distributed Stochastic Gradient Langevin Dynamics, uses only a mini-batch of ratings to make each update as in Stochastic Gradient Descent optimization. By running multiple chains in parallel, and also using multiple workers within a chain to update orthogonal blocks, we can scale up Bayesian Matrix Factorization to very large datasets. Parallel chains with different random initializations also help us to average predictions from multiple modes and improve accuracy. Moreover, our algorithm can effectively handle datasets that are distributed across multiple machines unlike traditional MCMC algorithms.

We believe that our method is just one example of a much larger class of scalable distributed Bayesian matrix factorization methods. For example, we can consider using more sophisticated stochastic gradient algorithms [Ahn et al., 2012, Patterson and Teh, 2013, Chen et al., 2014, Ding et al., 2014] in place of SGLD to further improve the mixing rate.

Chapter 8

Scalable MCMC for Mixed Membership Stochastic Blockmodels

This chapter introduces a stochastic gradient Markov chain Monte Carlo (SG-MCMC) algorithm for scalable inference in the mixed-membership stochastic blockmodels (MMSB). Our algorithm is based on the stochastic gradient Riemannian Langevin dynamics (SGRLD) and achieves both faster speed and higher accuracy *at every iteration* than the current state-of-the-art algorithm based on stochastic variational inference. In addition, we develop an approximation that can handle models that entertain a very large number of communities. The experimental results show that SG-MCMC outperforms the competing algorithms and is much more efficient for a large number of communities.

This work is a joint work with Wenzhe Li at USC. I proposed the initial version of the algorithm. Then, Max Welling and Wenzhe Li helped elaborating the algorithm. Wenzhe Li performed the experiments.

8.1 Motivation

Probabilistic graphical models represent a convenient paradigm for modeling complex relationships between a potentially very large number of random variables. Bayesian graphical models [Wainwright and Jordan, 2008], where we define priors and infer posteriors over parameters also allow us to quantify model uncertainty and facilitate model selection and averaging. But an increasingly urgent question is whether these models and their inference procedures will be up to the challenge of handling very large "big data" problems.

A large subclass of Bayesian graphical models is represented by so called "topic models" such as latent Dirichlet allocation [Blei et al., 2003]. For these type of models very efficient inference algorithms have recently been developed, either based on stochastic variational Bayesian inference (SVB) [Hoffman et al., 2013, 2010a] or on stochastic gradient Markov chain Monte Carlo (SG-MCMC) [Welling and Teh, 2011, Ahn et al., 2012, 2014, Patterson and Teh, 2013]. Both methods have the important property that they only require a small subset of the data-items for every iteration. In other words, they can be applied to (infinite) streaming data.

An important class of "big data" problems are given by networks. Large networks such as social networks easily run into billions of edges and tens of millions of nodes. An interesting problem in this area is the discovery of communities: densely connected groups of nodes that are only sparsely connected to the rest of the network. Large networks may contain millions of such communities. To model overlapping communities the mixed membership stochastic blockmodel (MMSB) was introduced in Airoldi et al. [2009]. Very recently, an efficient stochastic variational inference algorithm was developed for a special case, the assortative MMSB (a-MMSB) [Gopalan et al., 2012], greatly extending the reach of Bayesian posterior inference to realistic large scale problem settings. Inspired by this work, and earlier comparisons between SVB and SG-MCMC on LDA [Patterson and Teh, 2013] we developed a scalable SG-MCMC algorithm for a-MMSB and compared it against SVB on the community detection problem.

Our conclusion is consistent with the findings of [Patterson and Teh, 2013], namely that SG-MCMC is also both faster and more accurate than SVB algorithms in this domain. While one should expect SG-MCMC to be more accurate than SVB asymptotically (SVB is asymptotically biased while SG-MCMC is not), it is interesting to observe that SG-MCMC dominates SVB across all iterations, despite the fact that SG-MCMC should have a larger variance contribution to the error.

8.2 Assortative Mixed-Membership Stochastic Blockmodels

Assortative mixed-membership stochastic blockmodel (a-MMSB) is a special case of MMSB that models the group-structure in a network of N nodes. In particular, each node a in the node set $\mathcal{V}^*$ has a K-dimension probability distribution π_a of participating in the K members of the community set $\mathcal{K}$. For every possible peer b in the network, each node a randomly draws a community z_{ab}. If a pair of nodes (a, b) in the edge set $\mathcal{E}^*$ are in the same community: $z_{ab} = z_{ba} = k$, then they have a significant probability β_k to connect, i.e., $y_{ab} = 1$. Otherwise this probability is small. Each community has its connection strength $\beta_k \in (0, 1)$ which explains how likely its members are linked each other.

The generative process of a-MMSB is then given by,

1. For each community k, draw community strength $\beta_k \sim \text{Beta}(\eta)$
2. For each node a, draw community memberships $\pi_a \sim \text{Dirichlet}(\alpha)$
3. For each pair of nodes a and b,

(a) Draw interaction indicator $z_{ab} \sim \pi_a$

(b) Draw interaction indicator $z_{ba} \sim \pi_b$

(c) Draw link $y_{ab} \sim \text{Bernoulli}(r)$, where $r = \beta_k$ if $z_{ab} = z_{ba} = k$ and $r = \epsilon$ otherwise.

Unlike the a-MMSB, the original MMSB maintains pair-wise community strength $\beta_{k,k'}$ for all pairs of the communities. Note that it is trivial to extend the results that we obtain in this chapter to the general MMSB model. The joint probability of the above process can be written as:

$$p(y, z, \pi, \beta|\alpha, \eta) = \prod_{a=1}^{N}\prod_{b>a}^{N} p(y_{ab}|z_{ab}, z_{ba}, \beta)p(z_{ab}|\pi_a)p(z_{ba}|\pi_b)\prod_{a=1}^{N} p(\pi_a|\alpha)\prod_{k=1}^{K} p(\beta_k|\eta). \quad (8.1)$$

Both variational inference [Jordan et al., 1999] and collapsed Gibbs sampling algorithms [Griffiths and Steyvers, 2004] have been used successfully for small to medium scale problems. However, the $\mathcal{O}(N^2)$ computational complexity per update prevents it from being applied to large scale networks. A stochastic variational algorithm was developed in Gopalan et al. [2012] to address this issue, where each update only depends on a small mini-batch of the nodes in the network.

8.3 Scalable MCMC for a-MMSB

Our algorithm iterates updating local parameters π and a global parameter β. Because both parameters lie on the probability simplex, we use SGRLD introduced in Section 6.2. Introducing the expand-mean parameters ϕ and θ to re-parameterize π and β respectively, the update rule is

$$\theta_k^* \leftarrow \left|\theta_k + \frac{\epsilon}{2}\left(\eta - \theta_k + \frac{N}{|\mathcal{D}_n|}\sum_{d\in\mathcal{D}_n} g_d(\theta_k)\right) + (\theta_k)^{\frac{1}{2}}\xi\right|, \quad (8.2)$$

here $\xi \sim \mathcal{N}(0, \epsilon)$, $g_d(\theta_k)$ is the gradient of the log posterior w.r.t. θ_k on a data point $d \in \mathcal{D}_n$, and η is the hyperparameter of the Dirichlet distribution being updated.

We then alternatively samples in the ϕ and θ spaces, and then obtain π and β by normalizing ϕ and θ. From the Eqn. 8.1, summing over the latent variable z, we begin with the following joint probability,

$$p(y, \pi, \beta|\alpha, \eta) = \prod_a p(\pi_a|\alpha) \prod_k p(\beta_k|\eta) \prod_a \prod_{b>a} \sum_{z_{ab}, z_{ba}} p(y_{ab}, z_{ab}, z_{ba}|\beta, \pi_a, \pi_b). \tag{8.3}$$

8.3.1 Sampling the global parameter

By the re-parameterization, we have $\beta_k = \theta_{k1}/(\theta_{k0}+\theta_{k1})$, where $\theta_{ki} \sim \text{Gamma}(\eta) \propto \theta_{ki}^{\eta-1} e^{-\theta_{ki}}$. Because $p(y, \pi, \beta|\alpha, \eta)$ decomposes into $p(y, \beta|\pi, \eta)p(\pi|\alpha)$, replacing β by θ, we compute the derivative of log of Eqn. 8.3 w.r.t. θ_{ki} for $i = \{0, 1\}$ as follows:

$$\frac{\partial \ln p(y, \theta|\pi, \eta)}{\partial \theta_{ki}} = \frac{\partial}{\partial \theta_{ki}} \ln p(\theta_{ki}|\eta) + \sum_a \sum_{b>a} g_{ab}(\theta_{ki}), \tag{8.4}$$

where $g_{ab}(\theta_{ki}) = \frac{\partial}{\partial \theta_{ki}} \ln \sum_{z_{ab}, z_{ba}} p(y_{ab}, z_{ab}, z_{ba}|\theta, \pi_a, \pi_b)$ which, similar to SGRLD for LDA, we can rewrite as

$$g_{ab}(\theta_{ki}) = \mathbb{E}\left[\mathbb{I}[z_{ab} = z_{ba} = k] \left(\frac{|1 - i - y_{ab}|}{\theta_{ki}} - \frac{1}{\theta_k}\right)\right]. \tag{8.5}$$

where $\theta_k = \sum_i \theta_{ki}$ and $\mathbb{I}[S]$ is equal to 1 if a condition S is TRUE and 0 otherwise. The expectation is w.r.t. the posterior distribution of latent variables z_{ab} and z_{ba},

$$p(z_{ab} = k, z_{ba} = l|y_{ab}, \pi_a, \pi_b, \beta) \propto f_{ab}^{(y)}(k, l) = \begin{cases} \beta_k^y (1 - \beta_k)^{(1-y)} \pi_{ak} \pi_{bk}, & \text{if } k = l \\ \epsilon^y (1 - \epsilon)^{(1-y)} \pi_{ak} \pi_{bl}, & \text{if } k \neq l. \end{cases} \tag{8.6}$$

here we used simple notation y instead of y_{ab}. Unlike the SGRLD for LDA, we compute the expectation in Eqn. (8.5) analytically by computing the normalization constant $Z_{ab}^{(y)} = \sum_{k=1}^{K}\sum_{l=1}^{K} f_{ab}^{(y)}(k,l)$ which can be reduced to $\mathcal{O}(K)$ computation as follows

$$Z_{ab}^{(y)} = \epsilon^y(1-\epsilon)^{(1-y)} + \sum_{k=1}^{K}\left(\beta_k^y(1-\beta_k)^{(1-y)} - \epsilon^y(1-\epsilon)^{(1-y)}\right)\pi_{ak}\pi_{bk}. \tag{8.7}$$

Then, the Eqn. (8.5) becomes

$$g_{ab}(\theta_{ki}) \;=\; \sum_{k=1}^{K}\frac{f_{ab}^{(y)}(k,k)}{Z_{ab}^{(y)}}\left(\frac{|1-i-y_{ab}|}{\theta_{ki}} - \frac{1}{\theta_k}\right). \tag{8.8}$$

Plugging this into Eqn. 8.2, we obtain the update rule for the global parameter,

$$\theta_{ki}^* \leftarrow \left|\theta_{ki} + \frac{\epsilon}{2}\left\{\eta - \theta_{ki} + h(\mathcal{E}_{n_t})\sum_{(a,b)\in\mathcal{E}_{n_t}} g_{ab}(\theta_{ki})\right\} + (\theta_{ki})^{\frac{1}{2}}\xi_{ki}\right|, \tag{8.9}$$

here $\mathcal{E}_{n_t}$ is a mini-batch of n_t node pairs sampled from $\mathcal{E}^*$ for which we use the following strategy.

Stratified sampling: considering that the number of links are much smaller than that of non-links, we can reduce the variance of the gradient using stratified sampling, similar to the method used in [Gopalan et al., 2012]. For this, at every iteration we first randomly select a node a and then toss a coin with probability 0.5 to decide whether to sample link edges or non-link edges for node a. If it is a link, we assign all of the link edges of node a to $\mathcal{E}_{n_t}$. Otherwise, i.e. if it is non-link, we uniformly sample a mini-batch of N/m non-link edges from the entire non-link edges and assign it to $\mathcal{E}_{n_t}$. Here, the m is a hyper-parameter. Note that the size of $|\mathcal{E}_{n_t}|$ will thus be much smaller than the entire $N(N-1)/2$ edges when m is reasonably large. Then, to ensure that the gradient is unbiased, a *scaling parameter* $h(\mathcal{E}_{n_t})$ is multiplied. Specifically, $h(\mathcal{E}_{n_t})$ is set to N when $\mathcal{E}_{n_t}$ is a set of link edges and to mN otherwise.

Because the global parameter changes not so dynamically compared to the local parameter, in practice we update only a random subset of $\mathcal{K}$ at each iteration.

8.3.2 Sampling the local parameters

Similar to the global parameter, we re-parameterize the local parameter π_a such that $\pi_{ak} = \phi_{ak}/\sum_{j=1}^{K}\phi_{aj}$, with $\phi_{ak} \sim \text{Gamma}(\alpha) \propto \phi_{ak}^{\alpha-1}e^{-\phi_{ak}}$. Then, taking the derivative of log of Eqn. 8.3 w.r.t. ϕ_{ak}, we obtain

$$\frac{\partial \ln p(y,\phi|\beta,\alpha)}{\partial \phi_{ak}} = \frac{\partial}{\partial \phi_{ak}} \ln p(\phi_{ak}|\alpha) + \sum_b g_{ab}(\phi_{ak}) \tag{8.10}$$

where $g_{ab}(\phi_{ak}) = \frac{\partial}{\partial \phi_{ak}} \ln \sum_{z_{ab},z_{ba}} p(y_{ab}, z_{ab}, z_{ba}|\beta, \phi_a, \phi_b)$ which can be written as

$$g_{ab}(\phi_{ak}) = \mathbb{E}\left[\frac{\mathbb{I}[z_{ab}=k]}{\phi_{ak}} - \frac{1}{\phi_{a\cdot}}\right]. \tag{8.11}$$

Here the expectation is w.r.t. the distribution in Eqn. (8.6). To compute the expectation analytically, we first integrate out z_{ba} from Eqn. (8.6) because the expectation depends only on z_{ab}, and obtain the following probability up to a normalization constant

$$f_{ab}^{(y)}(k) = \sum_{l=1}^{K} f_{ab}^{(y)}(k,l) = \pi_{ak}\left\{\beta_k^y(1-\beta_k)^{(1-y)}\pi_{bk} + \epsilon^y(1-\epsilon)^{(1-y)}(1-\pi_{bk})\right\}. \tag{8.12}$$

Then we obtain the normalization term by $Z_{ab}^{(y)} = \sum_{k=1}^{K} f_{ab}^{(y)}(k)$. Integrating out the expectation in the Eqn. (8.11), we obtain

$$g_{ab}(\phi_{ak}) = \frac{f_{ab}^{(y)}(k)}{Z_{ab}^{(y)}\phi_{ak}} - \frac{1}{\phi_a}. \tag{8.13}$$

Plugging this to Eqn. 8.2, we obtain the SGRLD update rule for the local parameter ϕ_{ak}

$$\phi^*_{ak} \leftarrow \left| \phi_{ak} + \frac{\epsilon}{2}\left(\alpha - \phi_{ak} + \frac{N}{|\mathcal{V}_n|}\sum_{b\in\mathcal{V}_n} g_{ab}(\phi_{ak})\right) + (\phi_{ak})^{\frac{1}{2}}\xi_{ak}\right|. \tag{8.14}$$

Here, the $\mathcal{V}_n$ is a random mini-batch of n nodes sampled from $\mathcal{V}^*$. Note that $|\mathcal{V}_n| \ll |\mathcal{V}^*| = N$.

8.3.3 Scalable local updates for a large number of communities

In some applications, the number of communities can be very large so that the local update becomes very inefficient due to its $\mathcal{O}(K|\mathcal{V}_n|)$ computation per node in $\mathcal{E}_{n_t}$ and also $\mathcal{O}(KN)$ space complexity. In this section, we extend the above algorithm further with a novel approximation in order to make the algorithm scalable in terms of both speed and memory usage even for very large number of communities which the SVI approach cannot achieve.

Community split: for each node $a \in \mathcal{V}^*$, we first split the community set $\mathcal{K}$ into three mutually exclusive subsets: the *active* set $\mathcal{A}(a)$, the *candidate* set $\mathcal{C}(a)$, and the *bulk* set $\mathcal{B}(a)$ such that $\mathcal{A}(a) \cup \mathcal{C}(a) \cup \mathcal{B}(a) = \mathcal{K}$. Then, sorting the π_a w.r.t. k in descending order, we obtain a new order of communities $k_1, \ldots, k_K$. The active set $\mathcal{A}(a)$ contains communities whose cumulative distribution $F(k_i)$ is less than a threshold $\tau \in (0, 1]$, i.e. $\mathcal{A}(a) = \{k_i \in \mathcal{K} | F(k_i) < \tau\}$. The candidate set $\mathcal{C}(a)$ includes communities which are in the active set of at least one of the neighbors of node a, i.e. $\mathcal{C}(a) = \{k \in \mathcal{K} \setminus \mathcal{A}(a) | \exists b \in \mathcal{N}(a) \text{ s.t. } k \in \mathcal{A}(b)\}$. The bulk set $\mathcal{B}(a)$ contains all the remainder, i.e. $\mathcal{B}(a) = \mathcal{K} \setminus (\mathcal{A}(a) \cup \mathcal{C}(a))$. Here, we use $\mathcal{N}(a)$ to denote the neighbors of node a.

The rationale behind this split scheme are two folds. First, due to *sparsity*, at each node only a small number of communities will have meaningful probability while a large number of communities will have very low probability π_{ak}. We want the communities of low probability to belong to the bulk set, to share a single probability $\pi_{a\boldsymbol{\kappa}}$, and thus to be updated by one-shot for all $k \in \mathcal{B}(a)$. We use $\boldsymbol{\kappa}$ to represent the representative community of a bulk set. Second, due to the *locality*, neighboring nodes are likely to have a similar distribution over communities (after all, the model only assigns high probability to links when the associated nodes have high probability of sampling the same community). That is, when a neighbor of node a has a community k in its active set, this community may be a good candidate to become active for node a as well. By maintaining a candidate set we allow communities to spread efficiently to neighboring nodes and thus through the network.

One-shot update: for communities $k \in \mathcal{B}(a)$, we apply the following approximation of the unnormalized probability in Eqn. (8.12)

$$f_{ab}^{(y)}(k \in \mathcal{B}(a)) \approx \tilde{f}_{ab}^{(y)}(\boldsymbol{\kappa}) = \pi_{a\boldsymbol{\kappa}} \left\{ \bar{\beta}_a^y (1-\bar{\beta}_a)^{(1-y)} \bar{\pi}_b + \epsilon^y (1-\epsilon)^{(1-y)} (1-\bar{\pi}_b) \right\}. \tag{8.15}$$

That is, we replace π_{bk} and β_k in Eqn. (8.12) by $\bar{\pi}_b = \frac{1}{m}\sum_{k \in \mathcal{B}_m(a)} \pi_{bk}$ and $\bar{\beta}_a = \frac{1}{m}\sum_{k \in \mathcal{B}_m(a)} \beta_k$ respectively using a random mini-batch $\mathcal{B}_m(a)$ of size m sampled from $\mathcal{B}(a)$. As a result, all $k \in \mathcal{B}(a)$ share a single value $\tilde{f}_{ab}^{(y)}(\boldsymbol{\kappa})$. Therefore, we can efficiently approximate the normalization constant by

$$Z_{ab}^{(y)} = \sum_{k \in \mathcal{K}} f_{ab}^{(y)}(k) \approx \tilde{Z}_{ab}^{(y)} = |\mathcal{B}(a)| \tilde{f}_{ab}^{(y)}(\boldsymbol{\kappa}) + \sum_{k \notin \mathcal{B}(a)} f_{ab}^{(y)}(k). \tag{8.16}$$

Note that we only sum over $|\mathcal{A}(a) \cup \mathcal{C}(a)| + 1$ terms which will be a much smaller size than $|\mathcal{B}(a)|$. Now, to compute the gradient efficiently, we apply the stratified sampling[1] for $\mathcal{V}_n$ by sampling n_1 nodes $\mathcal{V}_1$ from the neighbors $\mathcal{N}(a)$ and n_0 nodes $\mathcal{V}_0$ from non-neighbors $\mathcal{V}^* \setminus \mathcal{N}(a)$ such that $\mathcal{V}_n = \mathcal{V}_1 \cup \mathcal{V}_0$. Then, the sum of gradients for $\mathcal{V}_n$ in Eqn. (8.14) is obtained by

$$\frac{N}{|\mathcal{V}_n|} \sum_{b \in \mathcal{V}_n} g_{ab}(\phi_{ak}) \approx c_1 \sum_{b \in \mathcal{V}_1} \left(\frac{\tilde{f}_{ab}^{(1)}(k)}{\tilde{Z}_{ab}^{(1)} \phi_{ak}} - \frac{1}{\phi_a} \right) + c_0 \sum_{b' \in \mathcal{V}_0} \left(\frac{\tilde{f}_{ab'}^{(0)}(k)}{\tilde{Z}_{ab'}^{(0)} \phi_{ak}} - \frac{1}{\phi_a} \right). \tag{8.17}$$

[1]Note that, to be more efficient under the approximation, we use a sampling method which is different to the method used in the global update.

Here, we set $c_1 = |\mathcal{N}(a)|/n_1$ and $c_0 = (N - |\mathcal{N}(a)|)/n_0$ to ensure the unbiasedness of the stratified sampling. Again, it is important to note that all states in $\mathcal{B}(a)$ share a single current value $\phi_{a\boldsymbol{\kappa}}$ and also the same update equation of Eqn. (8.17). Thus for $\mathcal{B}(a)$ we compute Eqn. (8.17) only once and update all of them in one-shot. The computation cost becomes $\mathcal{O}(|\mathcal{A}(a) \cup \mathcal{C}(a)||\mathcal{V}_n|)$ per node in $\mathcal{E}_{n_t}$ which we expect to be efficient because $|\mathcal{A}(a) \cup \mathcal{C}(a)| \ll K$ due to the sparsity. For $k \notin \mathcal{B}(a)$, we simply replace $\tilde{f}_{ab}^{(y)}(k)$ in Eqn. (8.17) by $f_{ab}^{(y)}(k)$ in Eqn. (8.12), and update individually.

Promotion and demotion: after updating all $|\mathcal{A}(a) \cup \mathcal{C}(a)| + 1$ states (communities), we need to update the community split by promoting (e.g. to active or candidate set) or demoting (e.g. to candidate or bulk set) some of the states. To do this, we sort and normalize $\{\phi_{ak}\}$, and obtain the updated cdf $F(k_i)$. Then, we update $\mathcal{A}(a)$, $\mathcal{C}(a)$, and $\mathcal{B}(a)$ based on the threshold τ and based on the communities that are active in the neighboring nodes. In particular, if the cdf of the bulk state $F(\boldsymbol{\kappa})$ is less than the threshold, we promote some states in the bulk set by a random sampling. In this case, the number of states to promote is equal to $\text{int}((\tau - F(\boldsymbol{\kappa}_{-1}))/\pi_{a\boldsymbol{\kappa}})$. Here $\boldsymbol{\kappa}_{-1}$ denotes a state just left to the $\boldsymbol{\kappa}$ in the sorted community sequence. We sample a state from the pool of states that are yet not represented anywhere in the graph. The reason is that we wish to avoid creating disconnected communities of nodes, which we believe represent suboptimal local modes in the posterior distribution. Finally, we check which states in $\mathcal{B}(a)$ can be promoted to $\mathcal{C}(a)$ by checking the neighboring nodes. We provide the pseudo code of the above algorithm in the Algorithm 9.

8.4 Experiments

We evaluate the efficiency and accuracy of our algorithm on five datasets: Synthetic, US-AIR, NETSCIENCE, RELATIVITY, and HEP-PH, summarized in Table 8.1. We compare four algorithms. As exact batch-mode MCMC methods, we use collapsed Gibbs sampling

Algorithm 9 Pseudo-code for each sampling iteration t

```
1: Sample a mini-batch $\mathcal{E}$ of $n_t$ node pairs from $\mathcal{E}^*$
2: for each node $a$ in $\mathcal{E}$ do
3:     Sample a mini-batch of nodes $\mathcal{V}_n(a) = \mathcal{V}_1(a) \cup \mathcal{V}_0(a)$ from $\mathcal{V}^*$
4:     Update $\phi_{ak}$ for all $k \in \mathcal{A}(a) \cup \mathcal{C}(a)$ using Eqn. (8.17) and Eqn. (8.14)
5:     Update $\phi_{a\boldsymbol{\kappa}}$ only for the representative bulk state $\boldsymbol{\kappa}$ using Eqn. (8.17) and Eqn. (8.14)
6:     Sort and normalize to obtain $\{\pi_{ak}\}$ and the cdf $F(k_i)$ for all $|\mathcal{A}(a) \cup \mathcal{C}(a)| + 1$ states
7:     Promote or demote some states using the updated cdf, threshold $\tau$, and neighbor
       information
8: end for
9: for $k$ in a random subset of $\mathcal{K}$ do
10:    Update $\theta_{k\{0,1\}}$ by Eqn. (8.9) using $\mathcal{E}$ and obtain $\beta_k$ from $\theta_{k\{0,1\}}$
11: end for
```

	# of nodes	% of link edges
Synthetic	75	30
US-AIR	1.1K	1.2
NETSCIENCE	1.6K	0.3
HEP_PH	12K	0.16

Table 8.1: Datasets

(CGS) and Langevin Monte Carlo (LMC). We also compare to SVI as a state-of-the-art method in variational Bayes. Finally, two of our algorithms are tested, one with and the other without the approximation for large communities. We call these SGMC and SGMC-M, respectively.

We used $\alpha = 1/K$ and $\eta = 1$ for all of the models and for all experiments unless otherwise stated. For the stepsize annealing schedule we used $\varepsilon_t = (\tau_0 + t)^{-\kappa}$ with $\kappa = 0.5$ and $\tau_0 = 1024$. For the stratified sampling of the global update in SVI and SGMCs, we used m such that the size of non-link edges N/m to be $30 < N/m < 100$. And for mini-batch size of the stratified sampling of the local update in SGMCs, we used 20 samples with 10 from neighbors and 10 from non-neighboring nodes. For SGMC-M, we used the threshold $\tau = 0.9$ by default unless otherwise stated. Also, for the held-out test set, we used 1% of the total links and non-links.

As the performance metric, we use perplexity which is defined as exponential of the negative average log-likelihood of the data. Given a collection of T samples of the model parameters $\{\beta_t\}$ and $\{\pi_t\}$, the averaged perplexity on the held-out test set $\mathcal{E}_h$ is

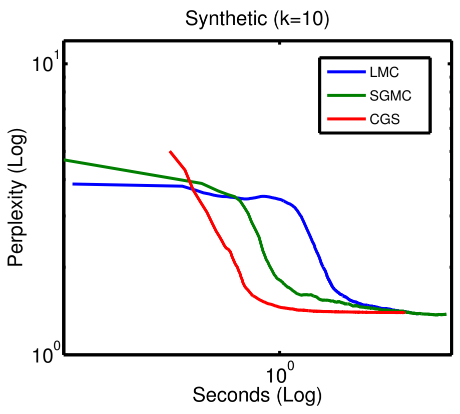
$$\mathrm{perp}_{\mathrm{avg}}(\mathcal{E}_h|\{\beta_t\},\{\pi_t\}) = \exp\left(-\frac{\sum_{(a,b)\in\mathcal{E}_h}\log\{(1/T)\sum_{t=1}^{T}p(y_{ab}|\beta_t,\pi_t)\}}{|\mathcal{E}_h|}\right). \tag{8.18}$$

8.4.1 Results

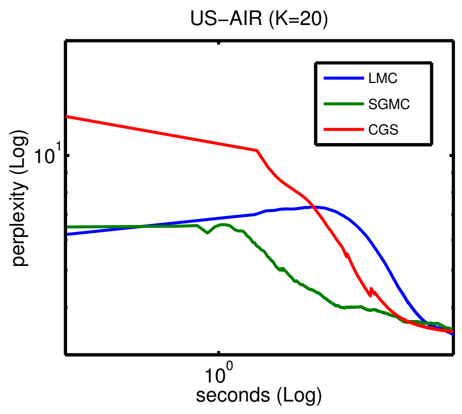
Comparison to exact batch MCMC: We first show the accuracy of our algorithm in comparison to the exact batch-mode MCMC algorithms (CGS and LMC). For this, we use two relatively small datasets, Synthetic and US-AIR, due to the slow speed of the batch algorithms. The results are shown in Fig. 8.1(a) and Fig. 8.1(b).

As expected, for the smaller dataset (Synthetic) in Fig. 8.1(a), we see that CGS converges very fast. However, it is interesting to observe that our stochastic gradient sampler (SGMC) using fixed step-size converges to the same level of accuracy in comparable time, whereas LMC converges much slower than both the collapsed Gibbs sampler and our algorithm due to its full gradient computation and the Metropolis-Hastings accept-reject step. As we move to larger network (US-AIR) in Fig. 8.1(b), we begin to see that our stochastic gradient sampler outperforms in speed the collapsed Gibbs sampler as well as the Langevin Monte Carlo. It is interesting to see that the approximation error of our algorithm due to the finite step size and the absence of accept-reject tests is negligible compared to the perplexity of the batch algorithms.

Effect of our approximation for large communities: In Fig. 8.2(a) and Fig. 8.2(b), on two large datasets, HEP-PH and NETSCIENCE, we show the speed-up effect of our approximate method (SGMC-M) compared to the SGMC without the approximation. Here we measure the time per iteration for various community size $K = [30, 50, 100, 200, 300, 500, 1000]$

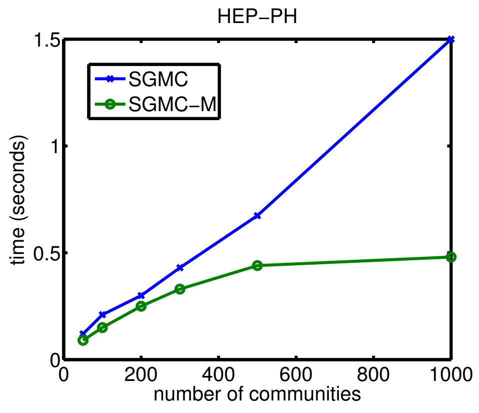


(a)

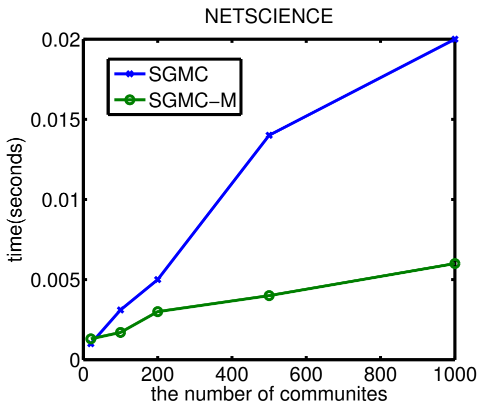


(b)

Figure 8.1: Convergence of perplexity on (a) Synthetic and (b) US-AIR datasets.



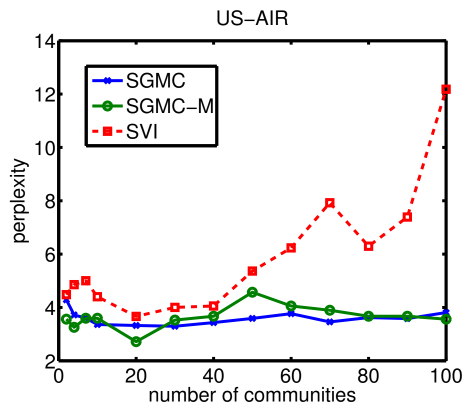
(a)



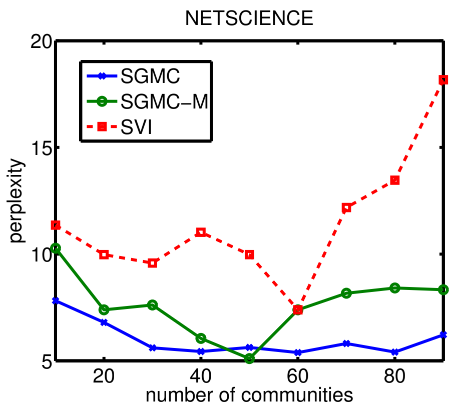
(b)

Figure 8.2: (a) Wall-clock time per iteration over increasing community sizes, on (a) HEP-PH and (b) NETSCIENCE datasets.

and set the threshold to $\tau = 0.9$. As shown, we can see that the approximate method SGMC-M only slightly increase the wall-clock time per iteration even if the community size increases. However, without the approximation (SGMC), the time per iteration increases linearly w.r.t. the community size. In fact, we can obtain more time savings as the community size increases further because the level of sparsity, i.e. the number of communities for which each node has non-negligible probability of participation, does not change much when we increase K.



(a)



(b)

Figure 8.3: Converged perplexity for various community sizes on (a) US-AIR and (b) NETSCIENCE datasets.

Furthermore, it is interesting to see in Fig. 8.3(a) and Fig. 8.3(b) that we do not lose the accuracy much despite the approximation. In particular, for Fig. 8.3(a), the SGMC-M performs as good as the SGMC. Although for Fig. 8.3(b) the SGMC-M performs worse than the SGMC, it still outperforms the SVI. Note that the results are based on converged perplexity which SGMC-M will reach much faster. The figures also reveal some interesting facts. First, the predictive accuracy is dominated by SGMC for all choices of K. Second, the curve of SVI has a V-shape indicating that the optimal value for K is in between the minimum and maximum value of K we tested. However, for SGMC the accuracy remains relatively stable as we increase K, making it less sensitive to the choice of this hyperparameter.

In Fig. 8.4(a), we also show how much memory usage the SGMC-M can reduce. In this experiment we set the threshold $\tau = 0.9$ for all datasets. As shown, the memory usage (i.e. $|\mathcal{A}(a) \cup \mathcal{C}(a)|/K$) of SGMC-M decreases as the size of community increases. This is because the sparsity usually does not change much even if the community size K increases. Note that the memory usages of SVI and SGMC are always 100%. This is a significant feature for large scale networks because without our approximation the space complexity is $\mathcal{O}(KN)$ where both K and N can be very large. It is also interesting to see that at every node the

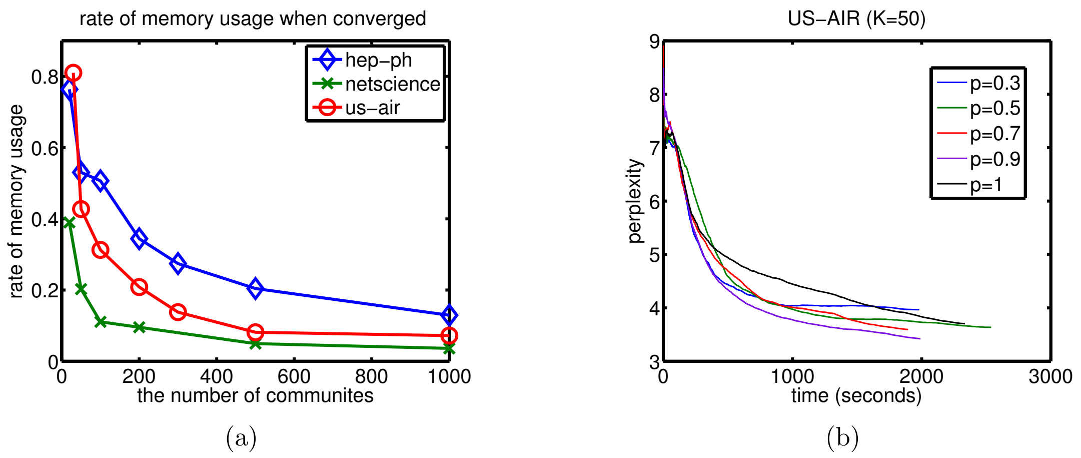


Figure 8.4: (a) Memory usage over different community sizes and datasets. The memory usage is defined as a ratio $(|\mathcal{A}| + |\mathcal{C}| + 1)/K$. (b) Convergence of perplexity for various threshold values p on US-AIR dataset.

90% of the total density is allocated to only about 10% $\sim$ 20% of the communities e.g., for $K = 500, 1000$.

Lastly, we investigate the effect of the threshold τ (in the plot we use p instead of τ) and the results are shown in Fig. 8.4(b) using US-AIR dataset. As shown, with $p = 0.9$, we obtain the best results. It is interesting because the SGMC-M only uses 50% of memory compared to that of SGMC without the approximation ($p = 1$). As expected, as we decrease the threshold, we represent smaller communities in the active and candidate set and lose some accuracy while gaining some speed.

8.5 Discussion

Bayesian inference in large network models is important for a number of reasons. Firstly, it naturally avoids overfitting by marginalizing over parameters. Secondly, it provides uncertainty estimates for predicting who is likely to be connected to whom. And thirdly, it supports active learning scenarios where, say, Facebook might not only suggest friends to a user that have significant community overlap with high probability but also friends that may have a high overlap but about which it is still uncertain. This type of serendipity is not only appreciated by users, but it is also key to explore the space of potential friends more efficiently.

In this chapter we have developed a new scalable MCMC algorithm based on stochastic gradient computations for Bayesian posterior inference in assortative mixed membership stochastic blockmodels (a-MMSB). The algorithm represents a natural extension of stochastic gradient Riemannian Langevin dynamics (SGRLD) to a-MMSBs. In line with the results reported in Patterson and Teh [2013] for LDA, SGRLD also significantly outperforms its stochastic variational Bayesian counterpart. Besides exploiting the sparsity and locality properties of real world networks, we also introduce an efficient approximation to scale up our algorithm further to very large number of communities which the standard SGRLD and the SVI cannot achieve. As was shown in Ahn et al. [2014], SGRLD algorithms are particularly suited for distributed implementation. We are currently working towards a distributed implementation of our algorithm on a HPC infrastructure allowing us to perform full Bayesian inference on very large networks of up to a billion edges and millions of communities.

Chapter 9

Conclusions and Discussions

The exponentially increasing size of modern datasets has challenged traditional machine learning algorithms. Unlike the frequentist approach for which the efficient stochastic gradient optimization algorithm is available, the traditional Bayesian inference methods such as MCMC were difficult to apply to large-scale datasets because it required processing the entire dataset at every iteration rather than using only a small subset of them. Furthermore, the traditional Bayesian inference algorithms were difficult to deploy on distributed computing systems although in many industrial problems the dataset cannot be handled in a single machine. Consequently, the powerful advantages of Bayesian inference such as quantifying uncertainty, accurate prediction performance, and preventing overfitting, have to be abandoned when it comes to large-scale problems, and the choice was limited to optimization-based frequentist methods.

In this thesis, we have introduced recent advances in scalable Bayesian inference using stochastic gradient MCMC as well as its applications. We have tackled the research challenges identified in Chapter 1 and the contributions are summarized below.

- **[RC1] Improving mixing rate**: one drawback of the SGLD is that it is limited to use relatively small step-sizes and thus the mixing rate becomes low. To resolve this problem, we have introduced the stochastic gradient Fisher scoring (SGFS) method in Chapter 3. In SGFS, by exploiting the Bayesian central limit theorem, we have made it possible to use large step sizes to improve the mixing rate. To do this, we made the sampler behave as an (approximate) Gaussian distribution sampler when the step size is large, and thus it became possible to control the discrepancy between the approximate distribution and the true posterior even if the step size is very large. As a bonus, this formulation also led us to define an efficient pre-conditioning matrix that captures the curvature information of the target distribution. Thus, unlike SGLD, it became possible to sample efficiently from highly correlated parameter spaces at all step sizes.

  One limitation of SGFS is that it requires to invert a $D \times D$ matrix at every iteration (thus requiring $\mathcal{O}(nD^3)$ computations per iteration) to obtain the preconditioning matrix. (Here, D is the parameter dimension that can be very large in many interesting problems like deep neural networks.) Although we have introduced some approximations for this, further innovations toward obtaining computationally efficient estimation of the inversion (or, its multiplication to a gradient vector) are required.

- **[RC2] Efficient distributed SG-MCMC algorithm**: in Chapter 4, we have introduced the distributed SGLD (D-SGLD) algorithm which is the first distributed sampler using stochastic gradients. In particular, we showed that with the proposed D-SGLD estimator, we can build a valid SGLD procedure when the minibatches are restricted to be sampled only from one of the partitions of the dataset at a time. Then, the extension to running parallel chains became straightforward. We showed that performing multiple (but finite) updates at a worker without jumping to other workers is allowed and helpful to reduce the communication overhead. Also, the D-SGLD method resolved some notorious problems of distributed computing systems such as the "block-by-the-slowest" problem. As a result, by extending the advantages of SG-MCMC to distributed computing, D-SGLD improved scalability of SG-MCMC further to the level of practical and large-scale problems of industrial interest.

One future direction regarding D-SGLD is to incorporate very recent SG-MCMC algorithms such as SGHMC [Chen et al., 2014] and SGNHT [Ding et al., 2014] as the local sampler of D-SGLD. This may be challenging because unlike SGLD some of the algorithms contain some quantity which requires global updates. Also, considering that multiple chains may converge to different modes from which each chain is not easy to escape, improving the mixing rate of each chain using some information of other chains (e.g., using parallel tempering [Earl and Deem, 2005]) will be an important problem.

- **[RC3] Applications to industry sized large-scale problems**: using SG-MCMC, we have shown three important applications of large-scale machine learning: (1) topic modeling using latent Dirichlet allocation, (2) recommender systems using matrix factorization, and (3) overlapping community modeling in social networks using mixed-membership stochastic blockmodels. Because each of the applications raises its own particular challenges, we could not simply apply existing algorithms but had to devise improved algorithms solving the challenges. For (1), we combined SGRLD to D-SGLD, for (2) we proposed a modified SGLD update rule which makes it possible to update only a subset of the parameters of the model. Also, we proposed an efficient parallel update method using the orthogonal blocks of the rating matrix. Finally, for (3), we proposed a fast SGRLD update rule and an approximation to handle a very large number of communities.

  One interesting application of the SG-MCMC method that we have not covered in this thesis, is the Bayesian deep neural network. Our initial experiment using SGLD has shown some success on a relatively small number of hidden units (less than 300) by outperforming the dropout method. However, with a larger number of hidden units, we could not make it outperform the dropout method. Also, the Bayesian neural network combined with the recent innovation of Bayesian model (sample) compression [Korattikara et al., 2015] will be an interesting future research direction of D-SGLD.

In conclusion, dealing with uncertainty will remain important for machine learning in the era of big data, and Bayesian inference using MCMC will also remain a powerful tool in many such problems. As an initial attempt towards scalable MCMC, we have introduced recent advances using stochastic gradient MCMC methods and have shown some successful results in terms of algorithms and applications. Although further innovations will still be desired for Bayesian inference to be applied more broadly and also to be more scalable, we hope that the results shown in this thesis will be helpful for many practitioners who seek further improvements beyond the current system based on the traditional frequentist method. Also, to many researchers, we hope this will be a guide in the journey of making innovations towards scalable Bayesian inference.

Bibliography

Amazon ec2 instances. http://aws.amazon.com/ec2/instance-types/.

R. Adams, G. Dahl, and I. Murray. Incorporating side information in probabilistic matrix factorization with gaussian processes. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 2010.

A. Ahmed, A. Aly, J. Gonzalez, S. Narayanamurthy, and A. Smola. Scalable inference in latent variable models. In *International conference on Web search and data mining*, 2012.

S. Ahn, A. Korattikara, and M. Welling. Bayesian posterior sampling via stochastic gradient fisher scoring. In *International Conference on Machine Learning*, 2012.

S. Ahn, Y. Chen, and M. Welling. Distributed and adaptive darting monte carlo through regenerations. *International Conference on Artificial Intelligence and Statistics*, 2013.

S. Ahn, B. Shahbaba, and M. Welling. Distributed stochastic gradient mcmc. In *International Conference on Machine Learning (ICML)*, 2014.

E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems*, pages 33–40, 2009.

C. Andrieu and J. Thoms. A tutorial on adaptive mcmc. *Statistics and Computing*, 18: 343–373, 2008.

R. Bardenet, A. Doucet, and C. Holmes. Towards scaling up markov chain monte carlo: an adaptive subsampling approach. In *International Conference on Machine Learning*, 2014.

J. Bennett and S. Lanning. The netflix prize. *In KDD Cup and Workshop in conjunction with KDD*, 2007.

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *CoRR*, 2014. http://dblp.uni-trier.de/rec/bib/journals/corr/BezansonEKS14.

C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

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

V. Borkar. Stochastic approximation with two time scales. *Systems and Control Letters*, 29 (5):291–294, 1997.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, volume 20, pages 161–168, 2008.

T. Chen, E. Fox, and C. Guestrin. Stochastic gradient hamiltonian monte carlo. In *International Conference on Machine Learning (ICML)*, 2014.

T. G. Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

N. Ding, Y. Fang, R. Babbush, C. Chen, R. Skeel, and H. Neven. Bayesian sampling using stochastic gradient thermostats. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The yahoo! music dataset and kdd-cup'11. In *Proceedings of KDD-Cup 2011 competition*, 2012.

D. J. Earl and M. W. Deem. Parallel tempering: Theory, applications, and new perspectives. *Physical Chemistry Chemical Physics*, 7(23):3910–3916, 2005.

R. Gemulla, E. Nijkamp, P. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011.

M. Girolami and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo. *Journal of the Royal Statistical Society B*, 73 (2):1–37, 2010.

M. Girolami and B. Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73 (2):123–214, 2011.

P. K. Gopalan, S. Gerrish, M. Freedman, D. M. Blei, and D. M. Mimno. Scalable inference of overlapping communities. In *Advances in Neural Information Processing Systems*, pages 2249–2257, 2012.

T. Griffiths and M. Steyvers. Finding scientific topics. In *Proceedings of the National Academy of Sciences (PNAS)*, pages 5228–5235, 2004.

J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29 (7):1645–1660, 2013.

K. B. Hall, S. Gilpin, and G. Mann. Mapreduce/bigtable for distributed optimization. In *NIPS LCCC Workshop*, 2010.

Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*, 2013.

M. Hoffman, F. Bach, and D. Blei. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 856–864, 2010a.

M. Hoffman, D. Blei, and F. Bach. Online learning for latent dirichlet allocation. In *Neural Information Processing Systems*, 2010b.

M. Hoffman, D. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14:1303–1347, 2013.

M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Representation Learning*, 2014.

A. Korattikara. *Approximate Markov Chain Monte Carlo Algorithms for Large Scale Bayesian Inference*. PhD thesis, University of California, Irvine, 2014.

A. Korattikara, Y. Chen, and M. Welling. Austerity in mcmc land: Cutting the metropolis-hastings budget. In *International Conference on Machine Learning (ICML)*, 2014.

A. Korattikara, V. Rathod, K. Murphy, and M. Welling. Bayesian dark knowledge. In *arXiv preprint arXiv:1506.04416*, 2015.

Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. In *IEEE Computer*, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

H. Larochelle and Y. Bengio. Classification using discriminative Restricted Boltzmann Machines. In *Proceedings of the* 25^{th} *International Conference on Machine learning*, pages 536–543. ACM, 2008.

K. B. Laskey and J. W. Myers. Population markov chain monte carlo. *Machine Learning*, 50:175–196, 2003.

L. Le Cam. *Asymptotic methods in statistical decision theory.* Springer, 1986.

G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *Neural Information Processing Systems*, 2009.

R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. In *HLT*, 2010.

A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, 2007.

K. P. Murphy. *Machine learning: a probabilistic perspective.* 2012.

R. Neal. Probabilistic inference using markov chain monte carlo methods. Technical Report CRG-TR-93-1, University of Toronto, Computer Science, 1993.

R. M. Neal and J. Zhang. High dimensional classification with bayesian neural networks and dirichlet diffusion trees. In *Feature Extraction*, pages 265–296. Springer, 2006.

D. Newman, P. Smyth, M. Welling, and A. Asuncion. Distributed inference for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 1081–1088, 2007.

F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011.

S. Patterson and Y. W. Teh. Stochastic gradient riemannian langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, 2013.

I. Porteous, A. Ascuncion, and M. Welling. Bayesian matrix factorization with side information and dirichlet process mixtures. In *AAAI Conference on Artificial Intelligence*, 2010.

B. Recht and C. Re. Parallel stochastic gradient algorithms for large-scale matrix completion. In *Mathematical Programming Computation*, 2013.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 2008.

I. Sato and H. Nakagawa. Approximation analysis of stochastic gradient langevin dynamics by using fokker-planck equation and ito process. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 982–990, 2014.

N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In M. Meila and X. Shen, editors, *Proc. 11th Intl. Conf. Artificial Intelligence and Statistics (AIstats)*, pages 436–443, San Juan, Puerto Rico, 2007.

W. Scott. Maximum likelihood estimation using the empirical fisher information matrix. *Journal of Statistical Computation and Simulation*, 72(8):599–611, 2002.

M. Seeger. Low rank updates for the cholesky decomposition. Technical report, University of California Berkeley, 2004. URL `http://lapmal.epfl.ch/papers/cholupdate.shtml`.

A. Smola and S. Narayanamurthy. An architecture for parallel topic models. *VLDB*, 2010.

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

C. Teflioudi, F. Makari, and R. Gemulla. Distributed matrix completion. In *IEEE 12th International Conference on Data Mining*, 2012.

M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1:1–305, 2008.

M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning (ICML)*, 2011.

D. J. Wilkinson. Parallel bayesian computation. *Statistics Textbooks and Monographs*, 184: 477, 2006.

Y. Zhuang, W. S. Chin, Y. C. Juan, and C. J. Lin. A fast parallel sgd for matrix factorizatio in shared memory systems. In *Proceedings of the 7th ACM conference on Recommender systems*, 2013.

M. Zinkevich, M. Weimer, and A. Smola. Parallelized stochastic gradient descent. In *Neural Information Processing Systems*, 2010.

Appendix A

Proofs

A.1 Proof of Theorem 4.3.1

We provide a proof of Theorem 4.3.1 (Chapter 4) about the correctness of the online averaging procedure for estimating the Fisher Information.

Theorem A.1.1. *Consider a sampling algorithm which generates a sample θ_t from the posterior distribution of the model parameters $p(\theta|X_N)$ in each iteration t. In each iteration, we draw a random mini-batch of size n, $X_n^t = \{x_{t_1}...x_{t_n}\}$ and compute the empirical covariance of the scores $V(\theta_t; X_n^t) = \frac{1}{n-1}\sum_{i=1}^{n}\{g(\theta_t; x_{t_i}) - \overline{g}_n(\theta_t)\}\{g(\theta_t; x_{t_i}) - \overline{g}_n(\theta_t)\}^T$. Let V_T be the average of $V(\theta_t)$ across T iterations. For large N, as $T \to \infty$, V_T converges to the Fisher information $I(\theta_0)$ plus $\mathcal{O}(\frac{1}{N})$ corrections.i.e.*

$$\lim_{T\to\infty}\left[V_T \triangleq \frac{1}{T}\sum_{t=1}^{T} V(\theta_t; X_n^t)\right] = I(\theta_0) + \mathcal{O}(\frac{1}{N}) \tag{A.1}$$

Proof. First, note that using a little algebra, we can rewrite $V(\theta_t; X_n^t)$ as:

$$\begin{aligned} V(\theta_t; X_n^t) &= \frac{1}{n}\sum_{i=1}^{n} g(\theta_t; x_{t_i}) g(\theta_t; x_{t_i})^T \\ &- \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} g(\theta_t; x_{t_i}) g(\theta_t; x_{t_j})^T \end{aligned} \tag{A.2}$$

As $T \to \infty$, the averaging of $V(\theta_t, X_n^t)$ across different iterations is equivalent to taking an expectation of $V(\theta_t, X_n^t)$ w.r.t the posterior distribution of θ_t. Also, when N is very large, averaging across the different mini-batches X_n^t drawn in different iterations is equivalent to taking an expectation w.r.t $p(x; \theta_0)$. Thus we have:

$$\begin{aligned} V_T &= E_{\theta_t, X_n^t}\left[V(\theta_t, X_n^t)\right] \\ &= E_{\theta, x}\left[g(\theta; x) g(\theta; x)^T\right] - E_{\theta, x, x'}\left[g(\theta; x) g(\theta; x')^T\right] \end{aligned} \tag{A.3}$$

According to the Bernstein-von Mises theorem, $\theta | X_N \sim \mathcal{N}(\theta_0, \frac{I^{-1}(\theta_0)}{N})$ for large N. This means that we can consider $\theta \perp\!\!\!\perp x | \theta_0$ and the expectations in Eqn. (A.3) can be taken w.r.t θ and x independently. Also, since most of the posterior density is in a small region around θ_0, we can consider a Taylor series expansion of $g(\theta; x)$ around θ_0, (treating θ as a scalar for simplicity):

$$g(\theta; x) = \sum_{k=0}^{\infty} \frac{g^{(k)}(\theta_0; x)(\theta - \theta_0)^k}{k!} \tag{A.4}$$

We will now apply this series expansion to each of the terms in Eqn. (A.3). Thus we have:

$$E_{\theta,x}\left[\{g(\theta;x)\}^2\right] = \sum_{k=0}^{\infty} A_k(\theta_0) E_\theta\left[(\theta-\theta_0)^k\right] \tag{A.5}$$

where,

$$A_k(\theta_0) = \sum_{m=0}^{k} \frac{E_x\left[g^{(m)}(\theta_0;x)g^{(k-m)}(\theta_0;x)\right]}{m!(k-m)!} \tag{A.6}$$

Since θ has a Gaussian distribution, its odd central moments are zero, and the even central moments are given by:

$$E_\theta\left[(\theta-\theta_0)^{2r}\right] = \frac{(2r)!}{2^r r!}\left[\frac{I^{-1}(\theta_0)}{N}\right]^r \tag{A.7}$$

Thus, we have:

$$E_{\theta,x}\left[\{g(\theta;x)\}^2\right] = \sum_{r=0}^{\infty} A_{2r}(\theta_0)\frac{(2r)!}{2^r r!}\left[\frac{I^{-1}(\theta_0)}{N}\right]^r \tag{A.8}$$

Similarly, we can show that:

$$E_{\theta,x,x'}\left[g(\theta;x)g(\theta;x')\right] = \sum_{r=0}^{\infty} B_{2r}(\theta_0)\frac{(2r)!}{2^r r!}\left[\frac{I^{-1}(\theta_0)}{N}\right]^r \tag{A.9}$$

where,

$$B_k(\theta_0) = \sum_{m=0}^{k} \frac{E_x\left[g^{(m)}(\theta_0;x)\right]E_x\left[g^{(k-m)}(\theta_0;x)\right]}{m!(k-m)!} \tag{A.10}$$

From Eqns. (A.8) and (A.9), we have for large N:

$$\lim_{T\to\infty} V_T = \sum_{r=0}^{\infty} C_{2r}(\theta_0)\frac{(2r)!}{2^r r!}\left[\frac{I^{-1}(\theta_0)}{N}\right]^r \tag{A.11}$$

where,

$$C_k(\theta_0) = A_k(\theta_0) - B_k(\theta_0) \tag{A.12}$$

For example,

$$C_0(\theta_0) = I(\theta_0) \tag{A.13}$$

$$\begin{aligned} C_2(\theta_0) =& E_x\left[g(\theta_0;x)g^{(2)}(\theta_0;x) + g^{(1)}(\theta_0;x)g^{(1)}(\theta_0;x)\right] \\ &- I^2(\theta_0) \end{aligned} \tag{A.14}$$

Thus for large N,

$$\begin{aligned} \lim_{T\to\infty} V_T = I(\theta_0) - \frac{I(\theta_0)}{N} + \\ & \frac{I^{-1}(\theta_0)}{N} E_x\left[g(\theta_0;x)g^{(2)}(\theta_0;x) + g^{(1)}(\theta_0;x)g^{(1)}(\theta_0;x)\right] \\ & + \sum_{r=2}^{\infty} C_{2r}(\theta_0)\frac{(2r)!}{2^r r!}\left[\frac{I^{-1}(\theta_0)}{N}\right]^r \\ & = I(\theta_0) + \mathcal{O}(\frac{1}{N}) \end{aligned} \tag{A.15}$$

□

A.2 Proof of Propositions and Corollaries in Chapter 5

Definition 1. An estimator $f(\theta, Z; X)$, where Z is a set of auxiliary random variables associated with the estimator, is said to be a *valid SGLD estimator* if $\mathbb{E}_Z[f(\theta, Z; X)] = \bar{g}(\theta; X)$, where $\mathbb{E}_Z$ denotes expectation w.r.t. the distribution $p(Z; X)$ and it has finite variance $\mathbb{V}_Z[f(\theta, Z; X)] < \infty$.

Proposition A.2.1. *For each shard $s = 1, \ldots, S$, given shard size, N_s, and the normalized shard selection frequency, q_s, such that $N_s > 0$, $\sum_{s=1}^{S} N_s = N$, $q_s \in (0, 1)$, and $\sum_{s=1}^{S} q_s = 1$,* *the following estimator is a valid SGLD estimator,*

$$\bar{g}_d(\theta; X_s^n) \overset{def}{=} \frac{N_s}{Nq_s}\bar{g}(\theta_t; X_s^n) \tag{A.16}$$

where shard s is sampled by a scheduler $h(\mathcal{Q})$ with frequencies $\mathcal{Q} = \{q_1, \ldots, q_S\}$.

Proof. We first decompose the expectation of the estimator $\mathbb{E}[\bar{g}_d(\theta; X_s^n)|X]$ w.r.t. (1) the shard s and (2) the minibatch X_s^n conditioned on the shard s, as follows

$$\mathbb{E}[\bar{g}_d(\theta; X_s^n)|X] = \mathbb{E}_s[\mathbb{E}_{X_s^n}[\bar{g}_d(\theta; X_s^n)|s]|X]. \tag{A.17}$$

Then, plugging Eqn. (A.16) in Eqn. (A.17) and rearranging, we obtain

$$\begin{aligned} &= \mathbb{E}_s\left[\mathbb{E}_{X_s^n}\left[\frac{N_s}{nNq_s}\sum_{x\in X_s^n} g(\theta; x)\middle| s\right]\middle| X\right] \\ &= \mathbb{E}_s\left[\frac{N_s}{Nq_s}\mathbb{E}_{X_s^n}\left[\frac{1}{n}\sum_{x\in X_s^n} g(\theta; x)\middle| s\right]\middle| X\right]. \end{aligned} \tag{A.18}$$

Note here that given X, the inner expectation w.r.t. the minibatches of shard s, X_s^n, is equal to the mean score over the shard X_s. That is,

$$\mathbb{E}_{X_s^n}\left[\frac{1}{n}\sum_{x\in X_s^n} g(\theta; x)\middle| s, X\right] = \frac{1}{N_s}\sum_{x\in X_s} g(\theta; x). \tag{A.19}$$

Substituting this for the inner expectation, in Eqn. (A.18), we have

$$\mathbb{E}_s\left[\frac{N_s}{Nq_s}\frac{1}{N_s}\sum_{x\in X_s} g(\theta;x)\right] \tag{A.20}$$

$$= \frac{1}{N}\mathbb{E}_s\left[\frac{1}{q_s}\sum_{x\in X_s} g(\theta;x)\right] \tag{A.21}$$

$$= \frac{1}{N}\sum_{s=1}^{S} p(s)\frac{1}{q_s}\sum_{x\in X_s} g(\theta;x). \tag{A.22}$$

Because we choose a shard s by $h(\mathcal{Q})$, $p(s)$ is equal to q_s. Thus, by plugging $p(s) = q_s$ in Eqn. (A.22) and rearranging, we obtain

$$\begin{aligned} &= \frac{1}{N}\sum_{s=1}^{S} q_s\frac{1}{q_s}\sum_{x\in X_s} g(\theta;x) \\ &= \frac{1}{N}\sum_{s=1}^{S}\sum_{x\in X_s} g(\theta;x) \\ &= \frac{1}{N}\sum_{x\in X} g(\theta;x) \\ &= \bar{g}(\theta;X). \end{aligned} \tag{A.23}$$

which completes the proof for the validity of the estimator $\bar{g}_d$,

$$\mathbb{E}[\bar{g}_d(\theta;X_s^n)|X] = \bar{g}(\theta;X). \tag{A.24}$$

□

Corollary A.2.2. *A trajectory sampler with a finite $\tau \geq 1$, obtained by redefining the worker (shard) selection process $h(\mathcal{Q})$ in Proposition A.2.1 by the process $h(\mathcal{Q},\tau)$ below, is a valid SGLD sampler. $h(\mathcal{Q},\tau)$: for chain c at iteration t, choose the next worker s_{t+1}^c by*

$$s_{t+1}^c = \begin{cases} \tilde{h}(\mathcal{Q}), & \text{if } t = k\tau \text{ for } k = 0,1,2,\ldots \\ s_t^c, & \text{otherwise,} \end{cases} \tag{A.25}$$

where $\tilde{h}(\mathcal{Q})$ is an arbitrary scheduler with selection probabilities $\mathcal{Q}$.

Proof. Because the trajectory lengths are all equal to τ for all workers $s = 1, \ldots, S$ and $\tilde{h}(\mathcal{Q})$ conforms to the frequencies $\mathcal{Q}$, the worker (shard) selection frequencies of the trajectory sampling process $h(\mathcal{Q}, \tau)$ also satisfies $\mathcal{Q}$. As a result, in the proof of Proposition A.2.1, the probability $p(s) = q_s$ is retained even if we replace $h(\mathcal{Q})$ in Proposition A.2.1 by $h(\mathcal{Q}, \tau)$. Because changing the worker selection process only affects $p(s)$ in the proof of Proposition A.2.1, the proof directly applies to the corollary. □

Corollary A.2.3. *Given τ_s, where $1 \leq \tau_s < \infty$ for $s = 1, \ldots, S$, the adaptive trajectory sampler, obtained by redefining the worker (shard) selection process $h(\mathcal{Q})$ in Proposition A.2.1 by the process $h(\mathcal{Q}, \{\tau_s\})$ below, is a valid SGLD sampler. $h(\mathcal{Q}, \{\tau_s\})$: for chain c at iteration t, choose the next worker s^c_{t+1} by*

$$s^c_{t+1} = \begin{cases} \tilde{h}(1/S), & \text{if } t = k\tau_{s^c_t} \text{ for } k = 0, 1, 2, \ldots \\ s^c_t, & \text{otherwise,} \end{cases} \tag{A.26}$$

where $\tilde{h}(1/S)$ is a scheduler with uniform selection probabilities.

Proof. Because we select the worker uniformly by $\tilde{h}(1/S)$, only the trajectory lengths $\{\tau_{s^1}, \ldots, \tau_{s^C}\}$ affect the shard selection frequency of the process $h(\mathcal{Q}, \{\tau_s\})$. Since the trajectory length τ_s is proportional to q_s ($\tau_s \overset{\text{def}}{=} \bar{\tau} S q_s$), taking τ_s consecutive updates for uniformly selected random worker s satisfies the frequency $\mathcal{Q}$. Therefore, the proof of Proposition A.2.1 also directly applies to the corollary. □