

UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Approximate Markov Chain Monte Carlo Algorithms for Large Scale Bayesian Inference

Permalink

<https://escholarship.org/uc/item/2pk858wg>

Author

Korattikara Balan, Anoop

Publication Date

2014

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Approximate Markov Chain Monte Carlo Algorithms for Large Scale Bayesian Inference

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Anoop Korattikara Balan

Dissertation Committee:
Professor Max Welling, Chair
Professor Padhraic Smyth
Professor Alexander Ihler

2014

TABLE OF CONTENTS

	Page
LIST OF FIGURES	iv
LIST OF TABLES	v
LIST OF ALGORITHMS	vi
ACKNOWLEDGMENTS	vii
CURRICULUM VITAE	viii
ABSTRACT OF THE DISSERTATION	ix
1 Introduction	1
2 Markov Chain Monte Carlo	7
2.1 A Brief Introduction to MCMC	7
2.2 MCMC Algorithms	11
2.2.1 Metropolis-Hastings	11
2.2.2 Gibbs Sampling	12
2.2.3 Langevin Monte Carlo	13
2.3 Scalability of Traditional MCMC	14
2.4 Approximate MCMC	15
2.4.1 Basic Ideas	15
2.4.2 Approximate MCMC and the Bias-Variance Trade-off	17
2.4.3 Strategies for Algorithm Design	19
3 Approximate Metropolis-Hastings	21
3.1 Metropolis-Hastings	22
3.2 Approximate Metropolis-Hastings	23
3.3 Experiments	26
3.3.1 Random Walk - Logistic Regression	26
3.3.2 Independent Component Analysis	28
3.3.3 Variable selection in Logistic Regression	29
3.4 Error Analysis and Test Design	32
3.4.1 Error Analysis and Estimation	32
3.4.2 Optimal Sequential Test Design	34

3.5	Related Work	37
4	Approximate Slice Sampling	38
4.1	Slice Sampling	38
4.2	Approximate Slice Sampling	41
4.3	Experiments	43
4.3.1	L_1 Regularized Linear Regression	43
4.3.2	Multinomial Regression	44
4.3.3	Logistic Regression	45
5	Stochastic Gradient Langevin Dynamics	47
5.1	Stochastic Gradient Langevin Dynamics	48
5.1.1	Bias-Variance Trade-off in SGLD	49
5.1.2	SGLD and Optimization	50
5.2	Error Analysis	51
5.3	MH Corrected SGLD	54
6	Stochastic Gradient Fisher Scoring	57
6.1	Preliminaries and Notation	57
6.2	SGLD with preconditioning	58
6.3	Sampling from the approximate posterior	59
6.4	The Stochastic Gradient Fisher Scoring Algorithm	60
6.4.1	Bias-Variance Trade-off in SGFS	64
6.4.2	Computational Efficiency	64
6.5	Experiments	65
6.5.1	Logistic Regression	65
6.5.2	Neural Networks	68
6.5.3	Discriminative Restricted Boltzmann Machine (DRBM)	70
6.6	Related Work	73
7	Approximate Bayesian Probabilistic Matrix Factorization	74
7.1	Model	74
7.2	Algorithms	76
7.2.1	Gibbs Sampling	76
7.2.2	SGLD and Approximate Metropolis-Hastings	77
7.3	Results	80
8	Conclusions and Future Work	83
	Bibliography	86
A	Proofs	90
A.1	Proof of Theorem 2	90
A.2	Proof of Theorem 3	92

LIST OF FIGURES

	Page
2.1 The Bias-Variance Trade-Off	18
3.1 Logistic Regression: 2D Marginals	27
3.2 Logistic Regression: Risk in mean prediction	28
3.3 ICA: Risk in mean of Amari distance	29
3.4 RJMCMC: Risk in mean prediction	31
3.5 RJMCMC: Feature inclusion probability	32
3.6 Error of a hypothesis test estimated using dynamic programming	33
3.7 Optimal Test Design	36
4.1 L_1 regularized Linear Regression: Risk in average prediction	44
4.2 Multinomial Regression: Risk in average prediction	45
4.3 Logistic Regression: Risk in average prediction	46
5.1 Pitfalls of using uncorrected SGLD	56
6.1 Logistic Regression: 2D Marginals	66
6.2 Logistic Regression: Final Error	67
6.3 Neural Networks: Test Error	68
6.4 Discriminative RBM: 2D Marginals	70
6.5 Discriminative RBM: Final Error	71
7.1 Prediction RMSE on the Yahoo! Music Dataset	81

LIST OF TABLES

	Page
6.1 Final test error rate on the KDD99 dataset.	71
7.1 RMSE of predictions on the Yahoo Music Ratings dataset.	82

LIST OF ALGORITHMS

	Page
1 Approximate Metropolis-Hastings	25
2 Slice Sampling	40
3 <code>OnSlice</code>	41
4 <code>OnSliceApprox</code>	42
5 Stochastic Gradient Fisher Scoring	63

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Max Welling, for his excellent guidance and mentorship. His wonderful insight, scientific vision and enthusiasm for research has been a constant source of inspiration to me. He always went the extra mile to help me out. I could not have asked for a better advisor and I consider myself extremely fortunate to have been his student.

I would like to thank my defense committee members, Padhraic Smyth and Alex Ihler, for their invaluable advice and help. I also thank other members of my advancement committee, Deva Ramanan and Daniel Whiteson, for their insightful feedback.

I thank my internship supervisors, Dennis DeCoste (eBay Research Labs) and Bradley Denney (Canon USA R&D Center). I learned a lot from Brad, especially how to tackle hard mathematical problems. Working with Dennis completely changed my approach to research and ultimately led me to choose a career in industry.

Thanks to Babak Shahbaba for many enlightening discussions about MCMC. I also thank fellow students and postdocs at UCI: Sungjin Ahn, Yutian Chen, Levi Boyles, Chris DuBois, Xiangxin Zhu, Shiwei Lan, Andrew Gelfand, Dilan Gorur and others. I collaborated the most with Sungjin and I learned a lot working with him.

I would like to thank UCI and the Bren School of ICS for providing a first class research environment and for supporting me through a Chairs Fellowship.

On a more personal level, I thank my parents and my brother. Without their love and support, none of this would have been possible. I would also like to thank my friends who made life in Orange County a very memorable and fun experience.

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

CURRICULUM VITAE

Anoop Korattikara Balan

EDUCATION

Doctor of Philosophy in Computer Science	2014
University of California, Irvine	<i>Irvine, CA</i>
Master of Science in Computer Science	2011
University of California, Irvine	<i>Irvine, CA</i>
Bachelor of Technology in Eletronics and Communication Engineering	2006
National Institute of Technology	<i>Calicut, India</i>

RESEARCH EXPERIENCE

Graduate Research Assistant	01/2010–12/2014
University of California, Irvine	<i>Irvine, California</i>
Intern Technical PhD	07/2012–09/2012
eBay Research Labs	<i>San Jose, California</i>
Senior Engineering Intern	06/2010–09/2010, 04/2011 - 08/2011
Canon USA R&D Centre	<i>Irvine, California</i>

REFEREED CONFERENCE PUBLICATIONS

Approximate Slice Sampling for Bayesian Posterior Inference	2014
Christopher DuBois, Anoop Korattikara Balan, Max Welling and Padhraic Smyth	
<i>Artificial Intelligence and Statistics (AISTATS)</i>	
Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget	2014
Anoop Korattikara Balan, Yutian Chen and Max Welling	
<i>International Conference on Machine Learning (ICML)</i>	
Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring	2012
Sungjin Ahn, Anoop Korattikara Balan and Max Welling	
<i>International Conference on Machine Learning (ICML)</i>	
Statistical Tests for Optimization Efficiency	2011
Levi Boyles, Anoop Korattikara Balan, Deva Ramanan and Max Welling	
<i>Neural Information Processing Systems (NIPS)</i>	
Statistical Optimization of Non-Negative Matrix Factorization	2011
Anoop Korattikara Balan, Levi Boyles, Max Welling, Jingu Kim and Haesun Park	
<i>Artificial Intelligence and Statistics (AISTATS)</i>	

ABSTRACT OF THE DISSERTATION

Approximate Markov Chain Monte Carlo Algorithms for Large Scale Bayesian Inference

By

Anoop Korattikara Balan

Doctor of Philosophy in Computer Science

University of California, Irvine, 2014

Professor Max Welling, Chair

Traditional algorithms for Bayesian posterior inference require processing the entire dataset in each iteration and are quickly getting obsoleted by the proliferation of massive datasets in various application domains. Most successful applications of learning with big data have been with simple minibatch-based algorithms such as Stochastic Gradient Descent, because they are the only ones that can computationally handle today's large datasets. However, by restricting ourselves to these algorithms, we miss out on all the advantages of Bayesian modeling, such as controlling over-fitting, estimating uncertainty and the ability to incorporate prior knowledge.

In this thesis, we attempt to scale up Bayesian posterior inference to large datasets by developing a new generation of approximate Markov Chain Monte Carlo algorithms that process only a mini-batch of data to generate each posterior sample. The approximation introduces a bias in the stationary distribution of the Markov chain, but we show that this bias is more than compensated by accelerated burn-in and lower variance due to the ability to generate a larger number of samples per unit of computational time.

Our main contributions are the following. First, we develop a fast Metropolis-Hastings (MH) algorithm by approximating each accept/reject decision using a sequential hypothesis test that processes only an adaptive mini-batch of data instead of the complete dataset.

Then, we show that the same idea can be used to speed up the slice sampling algorithm. Next, we present a theoretical analysis of Stochastic Gradient Langevin Dynamics (SGLD), a posterior sampling algorithm derived by adding Gaussian noise to Stochastic Gradient Ascent updates. We also show that the bias in SGLD can be reduced by combining it with our approximate MH test. We then propose a new algorithm called Stochastic Gradient Fisher Scoring (SGFS) which improves the mixing rate of SGLD using a preconditioning matrix that captures the curvature of the posterior distribution. Finally, we develop an efficient algorithm for Bayesian Probabilistic Matrix Factorization using a combination of SGLD and approximate Metropolis-Hastings updates.

Chapter 1

Introduction

Every day, we create 2.5 quintillion bytes of data - so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. This data is big data. - IBM¹

Recent years have seen an exponential growth of data in various application domains and machine learning algorithms that can make sense of this data are becoming more and more important. Big data is driving machine learning research in two ways. On the one hand, big data offers the opportunity to train very powerful models that can make a significant impact on important real world problems. On the other hand, big data poses numerous challenges as many traditional machine learning algorithms cannot scale up to very large datasets.

Most successes in learning with big data have been with simple optimization based approaches where learning is cast as a problem of minimizing a loss function. Given i.i.d. data, the loss function usually involves an average over loss terms, each corresponding to a single

¹<http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>

data point, and can be minimized using an efficient iterative algorithm such as Stochastic Gradient Descent (SGD) [Robbins and Monro, 1951]. SGD makes parameter updates based on a noisy gradient of the loss function computed using only a mini-batch of data, and can therefore scale well to datasets of arbitrary size. Powerful models, such as deep neural networks, trained on extremely large datasets using SGD have led to breakthrough results in image classification, speech recognition etc and is currently one of the main focuses of research in industry and academia.

In contrast, ‘batch’ algorithms, which require processing the entire dataset in each iteration, are becoming less relevant because they cannot handle the petabyte scale datasets of today. An important class of methods affected by this is Bayesian inference because traditional algorithms for Bayesian inference, such as Markov Chain Monte Carlo (MCMC) or variational inference, rely entirely on batch updates. The main goal of this thesis is to develop a toolbox of MCMC algorithms that use only a small subset of data per iteration and can therefore scale up Bayesian inference to very large datasets.

At first, developing scalable algorithms for Bayesian inference might seem like a pointless exercise. After all, with big datasets, one might expect that there is not much danger of over-fitting or that the posterior will be rather uninteresting as it concentrates around the MAP estimate of the parameters. However, there are many important reasons why we want to scale up Bayesian inference:

1. Principled Regularization: Over-fitting is a non-issue only when the model is simple relative to the size of the dataset. We argue that when we have lots of data we should be more ambitious and fit larger models, in which case avoiding over-fitting becomes very important. In fact, if one never has to deal with over-fitting it may be that one is not being ambitious enough². Some of the recent breakthroughs in various applications can be attributed to using powerful models, such as deep neural networks, and then

²Andrey Mnih, ICML 2011 talk on ‘Stochastic Gradient Langevin Dynamics’

regularizing them with various methods such as unsupervised pre-training [Hinton et al., 2006, Bengio et al., 2007], Dropout [Hinton et al., 2012] and DropConnect [Wan et al., 2013]. Bayesian posterior averaging is a more principled way of performing regularization and Bayesian neural networks have been shown to outperform DropOut in some cases [Srivastava et al., 2014]. However, Bayesian neural networks have been studied only in the context of small datasets so far, mainly because of the prohibitive computational cost of inference. Developing fast algorithms that can scale Bayesian inference to big datasets might hold one of the keys to improving the already impressive results obtained with deep neural networks.

2. Modeling Uncertainty: As machine learning applications evolve beyond the laboratory to real world products that interact with the environment, human users and other intelligent machines, it becomes more and more necessary to not just make good predictions, but also characterize the uncertainty in the model, model parameters and predictions. As an example, consider the case of an intelligent financial assistant which predicts future stock prices and offers investment advice. Even a state of the art algorithm might be expected to make a few prediction errors because it does not have a lot of data for a particular company. If the algorithm predicts highly optimistic results for a particular stock based on very little data, an agent that also estimates its uncertainty can advise a user that this is a very risky bet, whereas an agent that relies on predictions alone will advise the user to invest in the stock. Similarly, consider the case of a ‘robot doctor’ that can treat patients by predicting ailments from symptoms. If the robot is not very confident of its predictions, and is aware of this, it can try to reduce the uncertainty by running more tests as opposed to offering potentially dangerous ‘treatment’ based on insufficient data. More generally, active learning and reinforcement learning algorithms require estimates of uncertainty to decide whether to ‘exploit’ what it has learned so far or ‘explore’ to collect more data and reduce its uncertainty. Bayesian inference provides a natural and principled way to quantify

this uncertainty and is therefore necessary in a world where autonomous agents can interact with their environment.

3. Hierarchical Bayes: In certain applications, we might have access to a lot of data, but each data-point may be relevant to only a small part of the model or problem. For example, in a movie recommender system, we typically have lots of ratings for different movies by various users. However, given the task of recommending a new movie to a particular user, we may not have much directly relevant training data as that particular user might have rated only a few movies. In this case, it is helpful to use a hierarchical Bayesian model where the parameters for different users (and movies) are tied through a common prior which is also learned from the data. This allows the system to make better predictions for users with few ratings by leveraging statistical strength from the complete dataset i.e. recommendations for users with fewer ratings are regularized heavily towards movies that are popular across all users, but at the same time, more personalized recommendations can be provided for users with a large number of ratings.

Now that we have seen how big datasets can benefit from Bayesian modeling, let us discuss the Bayesian inference problem in a little more detail. In the simplest setting, we are given a dataset $X_N = \{x_1 \dots x_N\}$ consisting of N i.i.d datapoints x_i , which we model using a distribution $p(x_i|\theta)$ parameterized by θ . We choose a prior distribution $p(\theta)$ over the parameters and are interested in the posterior distribution $p(\theta|X_N) \propto \prod_{i=1}^N p(x_i|\theta)p(\theta)$. The posterior distribution can be used to answer various questions about the data. This usually involves computing expectations with respect to the posterior distribution e.g. to make predictions on a test data point x^* , we can compute statistics of the predictive distribution $p(x^*|X_N) = \int p(x^*|\theta)p(\theta|X_N)d\theta$. In general, these computations are analytically intractable and sophisticated methods are required to perform inference.

There are two popular classes of methods for performing Bayesian inference: Variational

methods [Jordan et al., 1999] and sampling algorithms, primarily Markov Chain Monte Carlo (MCMC) methods [Neal, 1993]. Variational methods attempt to approximate the posterior distribution using a tractable family of distributions $q(\theta|\lambda)$ parameterized by λ . The optimal distribution in the family can be found by minimizing $KL[q(\theta|\lambda)||p(\theta|X_N)]$ with respect to λ , usually through gradient descent or fixed point iteration. MCMC methods, on the other hand, represent the posterior with a collection of samples (generated from the posterior) which can then be used to estimate expectations with respect to the posterior. The samples are generated by constructing a Markov chain in parameter space in such a way that the probability of visiting each parameter value is proportional to its posterior density.

Variational methods usually produce biased results, as the target posterior density may not be present in the approximating family. In contrast, MCMC methods are unbiased in the limit as we collect an infinite number of samples. However, MCMC is a stochastic procedure and suffers from sampling variance inversely related to the number of samples, whereas variational inference is purely deterministic. Traditional algorithms for both variational inference and MCMC require processing the entire dataset in each iteration and therefore cannot scale up to big datasets.

However, there has been some recent progress in developing scalable Bayesian inference algorithms. Hoffman et al. [2013] developed a stochastic variational inference (SVI) algorithm where noisy gradients computed from random mini-batches of data are used to find the optimal parameters of the approximating distribution, whereas Welling and Teh [2011] developed an approximate MCMC algorithm, also based on noisy gradients computed from mini-batches. Approximate MCMC algorithms can generate more samples than traditional MCMC methods in the same amount of computational time, and can therefore produce estimates with lower variance. However, approximate MCMC methods are usually biased unlike traditional MCMC algorithms. On the other hand, SVI does not introduce any bias of its own and can find the same solution that traditional variational inference does. Unfortu-

nately, variational inference itself (and therefore SVI) is biased because we approximate the target distribution using a restricted family of approximating distributions, q , and this bias cannot be removed unless we choose a more flexible q . In contrast, the bias in approximate MCMC methods can be reduced by decreasing the amount of approximation in the algorithm over time. Because of this, approximate MCMC can outperform SVI in some applications, e.g. Latent Dirichlet Allocation [Patterson and Teh, 2013].

In this thesis, we will attempt to advance the theory and practice of approximate MCMC methods by developing a toolbox of scalable mini-batch based MCMC algorithms. The outline of the thesis is as follows. First, in chapter 2, we review basic MCMC theory and algorithms, discuss the trade-offs involved in approximate MCMC and then outline strategies for designing approximate MCMC algorithms. In chapter 3, we develop a scalable version of the Metropolis-Hastings algorithm [Metropolis et al., 1953, Hastings, 1970] using a sequential hypothesis test. Then in chapter 4, we show that these ideas can also be used to approximate the slice sampling algorithm [Neal, 2003]. In chapter 5, we present a theoretical analysis of the Stochastic Gradient Langevin Dynamics (SGLD) algorithm [Welling and Teh, 2011] and also discuss how the bias in SGLD can be reduced by combining it with our approximate Metropolis-Hastings test. Then, in chapter 6, we show how to improve the mixing speed of SGLD by equipping it with a preconditioning matrix that captures the curvature of the target posterior distribution. In chapter 7, we develop a fast algorithm for Bayesian Probabilistic Matrix Factorization using SGLD and our approximate Metropolis-Hastings test. We conclude in chapter 8 and also discuss some directions for future work.

Chapter 2

Markov Chain Monte Carlo

As discussed in chapter 1, a central task in Bayesian inference is computing the expectation of a function $f(\theta)$ with respect to a distribution $S_0(\theta)$. For example, S_0 can be a posterior distribution and f can be, say, prediction on test data. Usually these expectations are analytically intractable, and one has to resort to variational inference or sampling methods to estimate them. In this chapter, we will study sampling methods in more detail and show that traditional methods do not scale to large datasets. Then, we will discuss strategies for developing scalable approximate sampling algorithms that use only a mini-batch of data to generate each sample.

2.1 A Brief Introduction to MCMC

The general idea behind sampling methods, a.k.a Monte Carlo methods, is to estimate the expectation of a function with respect to a target distribution using a numerical average of samples generated from that distribution. i.e. to estimate $I = \langle f \rangle_{S_0} = \mathbb{E}_{S_0}[f(\theta)] =$

$\int_{\theta} f(\theta)S_0(\theta)d\theta$, we will generate T samples $\{\theta_1...\theta_T\}$ from S_0 and estimate I as:

$$I = \langle f \rangle_{S_0} \approx \hat{I} = \frac{1}{T} \sum_{t=1}^T f(\theta_t) \quad (2.1)$$

The Monte Carlo estimator $\hat{I}$ is unbiased and, if the samples are independent, its variance is equal to $\text{Var}_{S_0}[f]/T$. Simple methods for generating independent samples exist, e.g. rejection sampling and importance sampling, but these do not scale well to high dimensions. Markov Chain Monte Carlo (MCMC) is a more powerful class of sampling algorithms that can handle high dimensional distributions well. MCMC methods operate by constructing a discrete time Markov chain that moves around in parameter space such that the probability of visiting each state θ is proportional to $S_0(\theta)$. Thus, the states of the Markov chain can be considered as *correlated* samples from S_0 and can be used to compute the Monte Carlo estimator $\hat{I}$. The auto-correlation between samples does not affect the unbiasedness of $\hat{I}$, but nevertheless increases its variance as we will discuss later.

A Markov chain is a memoryless stochastic process parameterized by a transition kernel $K(\theta_{t+1}|\theta_t)$ which specifies the probability of moving from the current state θ_t to the next state θ_{t+1} . By memoryless, we mean that given θ_t , θ_{t+1} is independent of all other previous states $\{\theta_0...\theta_{t-1}\}$. If the transition kernel K does not depend on t , the Markov chain is said to be time-homogeneous. The marginal distribution of the state at time $t + 1$ can be computed as:

$$p_{t+1}(\theta_{t+1}) = \int p_t(\theta_t)K(\theta_{t+1}|\theta_t)d\theta_t \quad (2.2)$$

where $p_t(\theta)$ is the distribution of state θ_t . A distribution S_0 which satisfies the following

condition is said to be a stationary distribution of the Markov chain:

$$S_0(\theta_{t+1}) = \int S_0(\theta_t)K(\theta_{t+1}|\theta_t)d\theta_t \quad (2.3)$$

This means that if the t^{th} state is a sample from S_0 , the $t + 1^{th}$ state can also be considered a sample from S_0 , and therefore, by induction, all subsequent states can also be treated as samples from S_0 . The transition kernel K is said to leave the distribution S_0 invariant. Given a target distribution S_0 , it might seem hard to design a transition kernel K satisfying the above condition. However, it is usually possible¹ to design a K that satisfies a condition called *detailed balance*:

$$S_0(\theta_t)K(\theta_{t+1}|\theta_t) = S_0(\theta_{t+1})K(\theta_t|\theta_{t+1}) \quad (2.4)$$

It is easy to show that detailed balance is a sufficient (but not necessary) condition for ensuring that K leaves S_0 invariant. Plugging Eqn. 2.4 in Eqn. 2.3:

$$\int S_0(\theta_t)K(\theta_{t+1}|\theta_t)d\theta_t = \int S_0(\theta_{t+1})K(\theta_t|\theta_{t+1})d\theta_t = S_0(\theta_{t+1}) \int K(\theta_t|\theta_{t+1})d\theta_t = S_0(\theta_{t+1}) \quad (2.5)$$

A Markov chain that satisfies detailed balance is called a *reversible* Markov chain.

Note that satisfying the conditions in Eqn. 2.3 or Eqn. 2.4 only means that, *if* θ_t is distributed as S_0 , subsequent samples in the Markov chain will also be distributed as $S_0(\theta_0)$. Since the initial state θ_0 is drawn from some arbitrary distribution $p_0(\theta_0)$, and not S_0 , we also require an additional condition called *ergodicity* to ensure that for any choice of $p_0(\theta_0)$, the marginal distribution of the states $p_t(\theta_t) \rightarrow S_0(\theta_t)$ as $t \rightarrow \infty$. A Markov chain can be shown

¹see section 2.2.1

to be ergodic if two conditions called *irreducibility* and *aperiodicity* are satisfied [Tierney, 1994]. A chain with stationary distribution S_0 is irreducible if, starting from any state, there is a positive probability of reaching any other set of states with non-zero probability under S_0 . A chain is aperiodic if there are no portions of state space which can be visited only at certain regularly spaced times. If the Markov chain is ergodic, it will have only one stationary distribution which is called its equilibrium distribution.

Since we start our Markov chain using a sample from some arbitrary distribution $p_0(\theta)$, the initial states of the chain cannot be considered as samples from S_0 . This initial period where the marginal distribution of the states are converging to S_0 is called the burn-in period, and samples from this stage are usually discarded. After the burn-in samples have been discarded, the bias in the remaining samples can be considered negligible and $\hat{I}$ can be considered an unbiased estimator for most practical purposes. In the remainder of this thesis, θ_1 will usually refer to the first sample after the burn-in stage.

Although the auto-correlation between samples does not affect the unbiasedness of $\hat{I}$, the variance of $\hat{I}$ will be higher than if it were computed using independent samples. If we ignore burn-in bias, it can be shown that [Neal, 1995]:

$$\text{Var}[\hat{I}] = \frac{\text{Var}_{S_0}[f]\tau}{T} \quad (2.6)$$

where τ is the integrated auto-correlation time of the sequence $\{f(\theta_t)\}_{t=1}^T$. The integrated auto-correlation time is defined as $\tau = 1 + 2 \sum_{l=1}^{\infty} \rho(l)$, where $\rho(l)$ is the auto-correlation at lag l . The quantity T/τ is a measure of the number of independent samples obtained from the chain. To control the variance of $\hat{I}$, we want to keep the auto-correlation as low as possible. A chain that produces samples with low auto-correlation is said to “mix well”.

2.2 MCMC Algorithms

We will now review some popular MCMC algorithms.

2.2.1 Metropolis-Hastings

The Metropolis-Hastings (MH) algorithm [Metropolis et al., 1953, Hastings, 1970] is a general mechanism to design transition kernels that satisfy detailed balance with respect to a given distribution S_0 . The MH algorithm uses the following steps to transition from the current state θ_t to the next state θ_{t+1} :

1. Draw a candidate state θ' from a proposal distribution $q(\theta'|\theta_t)$
2. Compute the acceptance probability:

$$P_a(\theta', \theta_t) = \min \left[1, \frac{S_0(\theta')q(\theta_t|\theta')}{S_0(\theta_t)q(\theta'|\theta_t)} \right] \quad (2.7)$$

3. Draw $u \sim \text{Uniform}[0, 1]$. If $u < P_a(\theta', \theta_t)$ set $\theta_{t+1} \leftarrow \theta'$, otherwise set $\theta_{t+1} \leftarrow \theta_t$.

The test in step 3 above is called the MH accept/reject test. Setting $\theta_{t+1} \leftarrow \theta'$ is called accepting the proposal and setting $\theta_{t+1} \leftarrow \theta_t$ is called rejecting the proposal. Note in particular that rejecting a proposal does not mean we throw away state θ_{t+1} . It merely means that θ_{t+1} will be a copy of θ_t , and both of these states have to be included in computing the MCMC estimator $\hat{I}$.

It is easy to show that the MH transition rule above satisfies detailed balance with respect to S_0 . For $\theta' \neq \theta_t$, we have:

$$\begin{aligned} S_0(\theta_t)K(\theta'|\theta_t) &= S_0(\theta_t)q(\theta'|\theta_t) \min \left[1, \frac{\mathcal{S}_0(\theta')q(\theta_t|\theta')}{\mathcal{S}_0(\theta_t)q(\theta'|\theta_t)} \right] \\ &= \min [\mathcal{S}_0(\theta_t)q(\theta'|\theta_t), \mathcal{S}_0(\theta')q(\theta_t|\theta')] \\ &= \mathcal{S}_0(\theta')q(\theta_t|\theta') \min \left[1, \frac{\mathcal{S}_0(\theta_t)q(\theta'|\theta_t)}{\mathcal{S}_0(\theta')q(\theta_t|\theta')} \right] = S_0(\theta')K(\theta_t|\theta') \end{aligned}$$

A common choice for the proposal distribution is the random walk kernel, in which the next state is drawn from a symmetric distribution (such as a Gaussian) centered around the current state ². Although it is simple to implement, the random walk proposal is not very efficient. If the width of the proposal is chosen to be too small, the chain will traverse the parameter space very slowly. On the other hand, using a proposal that is too wide will result in most proposals being rejected. Both of these conditions result in poor mixing and high auto-correlation. A general design goal in MCMC, therefore, is to develop a $q(\cdot)$ that makes distant proposals which also have a high probability of acceptance. The random walk proposal cannot do this as it does not have any information about the target distribution. The next two algorithms we describe are essentially MH algorithms, but use some knowledge of the target distribution to generate more intelligent proposals that have a higher chance of getting accepted.

2.2.2 Gibbs Sampling

Gibbs sampling [Geman and Geman, 1984] is just a special case of the MH algorithm where the conditional distributions of the target distribution are used as proposals. Given a target

²Although the MH algorithm as stated above is valid for any proposal distribution q , the term “MH algorithm” is often used to refer to an MH algorithm with a random walk proposal, and MH algorithms with more sophisticated proposals usually have their own specialized names. In this thesis, “MH algorithm” will mean the general version of the algorithm with an arbitrary proposal.

distribution $S_0(\theta)$ where $\theta = \{\theta^{(1)} \dots \theta^{(D)}\}$ is a D dimensional vector, we update the variables $\theta_1 \dots \theta_D$ one at a time. To update a variable θ_i , we use the conditional distribution $S_0(\theta^{(i)} | \theta^{(j \neq i)})$ as the proposal distribution. The variable $\theta^{(i)}$ to be updated at each step can be picked in round robin fashion or at random. Instead of updating the variables one at a time, we can also update blocks of variables at a time, in which case the proposal distribution will be the conditional distribution of the variables being updated given the current value of all other variables. By plugging in the Gibbs proposal in Eqn. 2.7, we can see that the acceptance probability is always 1. Therefore, we can accept every proposal without explicitly conducting an accept/reject test.

Gibbs sampling can make more distant moves and mix faster than random walk MH. However, the proposal involves sampling from a conditional distribution, which may not always be tractable. Also, the proposals are axis aligned because we update only a few variables at a time. This makes it difficult to traverse highly correlated target distributions.

2.2.3 Langevin Monte Carlo

Langevin Monte Carlo, a.k.a. Metropolis Adjusted Langevin Algorithm (MALA) [Roberts and Tweedie, 1996], uses gradient information to make proposals to regions of higher density, and hence have a higher probability of getting accepted. The algorithm gets its name from the Langevin diffusion which is defined by the following stochastic differential equation:

$$d\theta_t = \frac{1}{2} \nabla_{\theta} \log S_0(\theta_t) + d\eta_t \tag{2.8}$$

where $\theta \in \mathbb{R}^D$ and η_t is the D -dimensional standard Brownian motion. Under certain regularity conditions (see [Roberts and Tweedie, 1996]), it can be shown that the Langevin diffusion in Eqn. 2.8 has S_0 as its equilibrium distribution. To simulate the diffusion on a

digital computer, one has to discretize the SDE, which is usually done as follows:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\alpha}{2} \nabla_{\theta} \log S_0(\theta_t) + \eta \quad \text{where} \quad \eta \sim \mathcal{N}(0, \alpha) \quad (2.9)$$

The above update equation defines an ergodic Markov chain, but its equilibrium distribution is no longer S_0 because of $O(\alpha)$ errors introduced by time discretization [Kronfeld, 1993]. Therefore, to sample correctly from S_0 , one has to treat the above as a proposal distribution q that can then be corrected with an MH accept/reject test. The Langevin proposal is:

$$q(\theta'|\theta_t) = \mathcal{N}\left(\theta'|\theta_t + \frac{\alpha}{2} \nabla_{\theta} \log S_0(\theta_t), \alpha\right) \quad (2.10)$$

As $\alpha \rightarrow 0$, acceptance probability of the Langevin proposal approaches 1, since the discretized update approaches the Langevin diffusion.

2.3 Scalability of Traditional MCMC

Most traditional MCMC algorithms are designed to satisfy detailed balance and require evaluating the target density at least once for each sample they generate. For Bayesian posterior inference, the target density is the posterior distribution $S_0 \propto p(\theta) \prod_{i=1}^N p(x_i|\theta)$, which involves the likelihood of all datapoints, and is hence very expensive to evaluate for large datasets. Therefore, traditional MCMC algorithms can generate only a limited number of samples in a reasonable amount of computational time. This negatively affects the quality of the MCMC estimator $\hat{I}$ in two ways:

1. *Burn-in is unnecessarily slow:* We want the burn-in phase to be very quick so that the remaining samples we collect are almost unbiased. In practice the burn-in phase is usually just noisy optimization to get us close to a mode, so it seems wasteful to

make precise moves that process the entire dataset in each iteration to enforce detailed balance.

2. *The variance of the estimator is too high:* Since, the variance of $\hat{I}$ is inversely proportional to the number of samples, we have to collect a large number of samples to keep the variance under control. Otherwise, the estimator may not be useful in practice even though it is unbiased.

Thus, the main goal of this thesis is to develop approximate MCMC algorithms that use only a mini-batch of data to generate each sample and can therefore scale up to arbitrarily large datasets. In the next section, we will discuss the general ideas behind approximate MCMC methods, the trade-offs involved, and strategies for designing approximate MCMC algorithms. In later chapters, we will apply these strategies to develop powerful general purpose algorithms for scalable Bayesian inference.

2.4 Approximate MCMC

2.4.1 Basic Ideas

Traditional MCMC algorithms are designed to converge to the correct stationary distribution so that the estimator $\hat{I}$ will be unbiased. However, as discussed in section 2.3, unbiasedness comes at a heavy price: traditional MCMC algorithms have to evaluate the expensive posterior density in each step. As a result, only a few samples can be generated in a reasonable amount of computational time and the variance of $\hat{I}$ will be very high. Given a fixed amount of time to finish our computation, is producing an unbiased estimator with high variance the optimal thing to do? We argue that it is not.

Imagine that, instead of evaluating the posterior distribution exactly in each step, we can

approximate some or all of the computations in the transition kernel efficiently. Let ϵ be a parameter that controls the amount of approximation. We will adopt the convention that $\epsilon = 0$ corresponds to no approximation, and higher values of ϵ correspond to cruder approximations (that are also cheaper to compute). Using an MCMC algorithm with $\epsilon > 0$ will allow us to burn-in quickly and reduce the variance of our estimator, because the cost of generating each sample is lower and we can draw more samples in the amount of computational time available.

However, because of the approximations we use, the transition kernel will, in general, not satisfy detailed balance with respect to S_0 and the stationary distribution of our Markov chain will be different from S_0 . Since samples from this biased distribution are used to estimate expectations with respect to S_0 , the estimator $\hat{I}$ will be biased as well. We will denote the biased stationary distribution as S_ϵ and the estimator computed using samples from S_ϵ as $\hat{I}_\epsilon$. We will call the bias resulting from using a wrong stationary distribution as asymptotic bias to distinguish it from burn-in bias (which becomes negligible as we collect more samples).

There are 3 sources of error in $\hat{I}_\epsilon$: burn-in bias, variance and asymptotic bias. These errors dominate at different stages of the sampling process. Initially, the burn-in bias is the dominant source of error, but it usually disappears quickly after we collect a few samples. Thereafter, the error is dominated by the variance which decreases as $\propto 1/T$. Finally, if we collect an infinite amount of samples, only the asymptotic bias will remain. The parameter ϵ controls the trade-off between these errors. The higher the value of ϵ , the faster we can sample to reduce burn-in bias and variance, but at the cost of higher asymptotic bias. If ϵ is small, the sampling process is more expensive, so burn-in and variance reduction is slow, but the asymptotic bias will be lower.

2.4.2 Approximate MCMC and the Bias-Variance Trade-off

The trade-off between these sources of errors can be studied more formally using the risk of the estimator. We define the risk of the estimator, $\hat{I}_\epsilon$, in the usual manner as its expected squared error, where the expectation is over multiple simulations of the Markov Chain. Usually, practitioners are willing to wait until the burn-in period is completed. So, for simplicity, let us assume that the burn-in bias is negligible. Then, it is easy to show that the risk is equal to the sum of variance and squared bias:

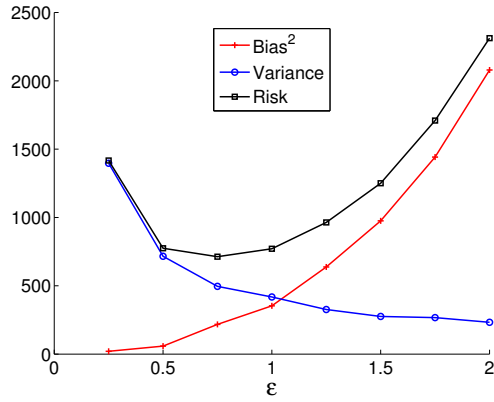
$$R_\epsilon = B_\epsilon^2 + V_\epsilon \tag{2.11}$$

where:

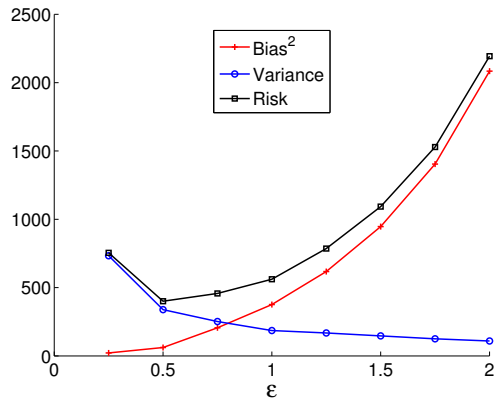
$$\begin{aligned} R_\epsilon &= \mathbb{E}[(\hat{I}_\epsilon - I)^2] \\ B_\epsilon &= I - \mathbb{E}[\hat{I}_\epsilon] = \langle f \rangle_{S_0} - \langle f \rangle_{S_\epsilon} \\ V_\epsilon &= \mathbb{E}[(\hat{I}_\epsilon - \mathbb{E}[\hat{I}_\epsilon])^2] \approx \frac{\sigma_{f, S_\epsilon}^2 \tau_\epsilon}{T_\epsilon} \end{aligned}$$

In the above, all expectations are taken with respect to multiple simulations of the Markov chain³. σ_{f, S_ϵ}^2 is the variance of f with respect to S_ϵ and τ is the integrated auto-correlation time. The bias B_ϵ is the difference in expectations of f with respect to the target distribution S_0 and the equilibrium distribution S_ϵ . Since the approximation parameter ϵ controls the trade-off between the bias and variance contributions to the risk, we can motivate the need for approximate MCMC algorithms ($\epsilon > 0$) from a risk minimization perspective.

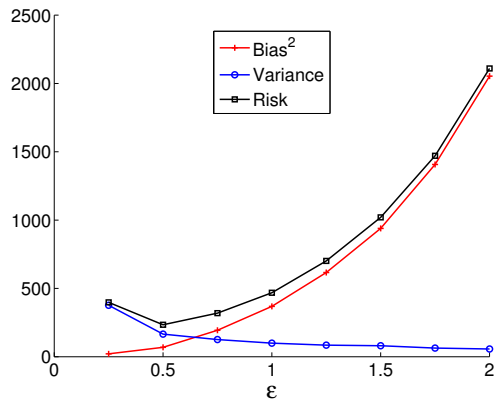
³i.e. the expectations are taken with respect to $p(\theta_1, \theta_2, \dots, \theta_T) = S_\epsilon(\theta_1)K_\epsilon(\theta_2|\theta_1)\dots K_\epsilon(\theta_T|\theta_{T-1})$. Here, $\theta_1 \sim S_\epsilon(\theta)$ since we ignore burn-in and we used K_ϵ to denote the transition kernel of our approximate Markov chain.



(a)



(b)



(c)

Figure 2.1: The risk, squared bias and variance of an approximate MCMC algorithm are plotted as a function of ϵ . Each sub-figure is for a fixed amount of computational time, which increases as we go from top to bottom.

The optimal ϵ which minimizes the risk depends on the computational time as the variance decreases with the number of samples we collect. To see this, in Figure 2.1, we have plotted the squared bias, variance and risk of an approximate MCMC algorithm⁴ as functions of ϵ . Each sub-figure is for a different amount of computational budget which increases from top to bottom (Fig. 2.1(a) to 2.1(c)). In Fig. 2.1(a), the optimal ϵ which minimizes the risk is 0.75. If we increase the computational time (Fig. 2.1(b)), we can collect more samples and the variance decreases for all values of ϵ . But the asymptotic bias stays the same because it does not depend on computational time. As a result, the optimal ϵ decreases to 0.5. In Fig. 2.1(c), we show the risk after even more computational time. Now, given an infinite amount of computational time, the variance for any value of ϵ will be 0, because we can collect an infinite number of samples. Therefore, the optimal value of ϵ in this case is 0, so that the bias is 0 as well. This is the traditional MCMC setting. However, given finite computational time, $\epsilon = 0$ is clearly not optimal. In general, the lower the amount of available computational time, the higher the value of ϵ we should use. Thus, it is necessary to develop approximate MCMC algorithms that can trade in a small amount of bias for faster variance reduction.

2.4.3 Strategies for Algorithm Design

We will now discuss some general strategies for designing approximate MCMC algorithms. The traditional MH MCMC algorithm has two steps. 1) We generate a candidate state from a proposal distribution. 2) We accept or reject the proposed state by conducting an expensive MH test. We can construct efficient approximate algorithms by modifying either of these steps as follows:

1. *Approximate the Metropolis-Hastings test with less data:* The most expensive part of the MH algorithm is the accept-reject test which requires evaluating the likelihood of

⁴The algorithm was Uncorrected Langevin Dynamics and we measure the risk in estimating θ^4 .

all datapoints in the data set. If we can approximate this test with less data, we can substantially speed up the algorithm.

2. *Develop efficient proposal distributions:* If we can develop efficient proposal distributions that generate candidates with an acceptance probability close to 1, we can accept every proposal and avoid the expensive MH test.

In the next few chapters, we apply these principles to develop various scalable MCMC algorithms.

Chapter 3

Approximate Metropolis-Hastings

The Metropolis-Hastings (MH) algorithm is a general method to design the transition kernel of a Markov chain so as to obtain a desired equilibrium distribution. Since the MH rule lies at the core of most popular MCMC methods, developing a scalable version of the MH algorithm is of fundamental importance. In this chapter, we show that the MH algorithm can be approximated using a sequential hypothesis test that processes only a fraction of the data to generate each posterior sample.

This chapter is based on results from Korattikara et al. [2014]. I came up with the idea to approximate the MH test using hypothesis tests. Max Welling and Yutian Chen helped refine the algorithm. I performed the experiments in sections 3.3.1 - 3.3.3. Yutian Chen provided the theoretical analysis and other results in section 3.4. Max Welling suggested using sampling without replacement to reduce the amount of data used per test. Alexander Ihler suggested using sequential hypothesis tests to control the error better.

3.1 Metropolis-Hastings

Given a (potentially unnormalized) target density S_0 , the Metropolis-Hastings algorithm [Metropolis et al., 1953, Hastings, 1970] uses the following steps for transitioning from the current state θ_t to the next state θ_{t+1} :

1. Draw a candidate state θ' from a proposal distribution $q(\theta'|\theta_t)$
2. Compute the acceptance probability:

$$P_a = \min \left[1, \frac{S_0(\theta')q(\theta_t|\theta')}{S_0(\theta_t)q(\theta'|\theta_t)} \right]$$

3. Draw $u \sim \text{Uniform}[0, 1]$. If $u < P_a$ set $\theta_{t+1} \leftarrow \theta'$, otherwise set $\theta_{t+1} \leftarrow \theta_t$.

It is easy to show that the above transition kernel satisfies detailed balance with respect to S_0 for any choice of the proposal distribution q (see section 2.2.1). The term MH accept/reject test is often used to refer to Steps 2 and 3 of the algorithm, since we are essentially trying to decide whether to ‘accept’ or ‘reject’ the proposed state θ' .

In Bayesian inference, we are mainly interested in generating samples from the posterior distribution $S_0 \propto p(\theta) \prod_{i=1}^N p(x_i|\theta)$. Since the acceptance probability in Step 2 involves the target density, we have to evaluate the likelihood of every datapoint for each sample we generate. This becomes prohibitively expensive when we have billions of datapoints in our dataset and we can generate only a few posterior samples in a reasonable amount of computational time. As a result, the variance of any MCMC estimators computed from these samples will be very high.

Therefore, as discussed in section 2.4, it is necessary to develop a scalable version of this algorithm, even if it is at the expense of a small amount of bias. Since many other popular

MCMC algorithms (e.g. Langevin Monte Carlo [Roberts and Tweedie, 1996], Hamiltonian Monte Carlo [Neal, 2011], Gibbs Sampling [Geman and Geman, 1984], Slice Sampling [Neal, 2003]) are derived from the MH algorithm using specialized choices for the proposal distribution, a scalable MH test is extremely useful. In the next section, we show how to approximate the MH test using a sequential hypothesis test that processes only an adaptive mini-batch of data to make each accept/reject decision.

3.2 Approximate Metropolis-Hastings

We approximate the MH test by first turning it into a decision problem about the mean difference in log-likelihoods, and then using a hypothesis test to approximate the decision using only a fraction of the data. First, note that the uniform random variable u (see MH algorithm above) can be generated before computing P_a . Since u is less than 1, we can compare u directly to $\frac{p(\theta') \prod_{i=1}^N p(x_i|\theta')q(\theta_t|\theta')}{p(\theta) \prod_{i=1}^N p(x_i|\theta)q(\theta_t|\theta)}$ instead of to P_a . Now, moving the terms in this comparison that do not depend on the data x to one side, taking a logarithm on both sides, and dividing by N , we can show that the MH test is equivalent to the following comparison:

$\mu_0 \geq \mu$, where:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{p(\theta_t)q(\theta'|\theta_t)}{p(\theta')q(\theta_t|\theta')} \right], \text{ and}$$

$$\mu = \frac{1}{N} \sum_{i=1}^N l_i \text{ where } l_i = \log p(x_i; \theta') - \log p(x_i; \theta_t)$$

Here, if $\mu > \mu_0$, we accept the proposal and set $\theta_{t+1} \leftarrow \theta'$. If $\mu \leq \mu_0$, we reject the proposal and set $\theta_{t+1} \leftarrow \theta_t$. Thus, the MH accept/reject test can be framed as a decision problem to check whether the mean of the l_i 's, the difference in log-likelihoods, is greater than or less

than a threshold that does not depend on the data. Although, this decision still involves all the datapoints, the reformulation makes it very easy to approximate the comparison using a statistical hypothesis test.

Given μ_0 and a random sample $\{l_{i_1}, \dots, l_{i_n}\}$ drawn without replacement from the population $\{l_1, \dots, l_N\}$, we would like to decide whether the population mean μ is greater than or less than the threshold μ_0 . How well we can do this depends on the precision in the random sample. If the difference between the sample mean $\bar{l}$ and μ_0 is significantly greater than the standard deviation s of $\bar{l}$, we can make the decision to accept or reject the proposal confidently. If not, we should draw more data to increase the precision of $\bar{l}$ (reduce s) until we have enough evidence to make a decision.

More formally, we test the hypotheses $H_1 : \mu > \mu_0$ vs $H_2 : \mu < \mu_0$. To do this, we proceed as follows: We compute the sample mean $\bar{l}$ and the sample standard deviation $s_l = \sqrt{(\bar{l}^2 - (\bar{l})^2) \frac{n}{n-1}}$. Then the standard deviation of $\bar{l}$ can be estimated as:

$$s = \frac{s_l}{\sqrt{n}} \sqrt{1 - \frac{n-1}{N-1}} \quad (3.1)$$

where $\sqrt{1 - \frac{n-1}{N-1}}$, the finite population correction term, is applied because we are drawing the subsample without replacement from a finite-sized population. Then, we compute:

$$t = \frac{\bar{l} - \mu_0}{s} \quad (3.2)$$

If n is large enough for the central limit theorem (CLT) to hold, t follows a standard Student-t distribution with $n-1$ degrees of freedom, when $\mu = \mu_0$. Then, we compute $\delta = 1 - \phi_{n-1}(|t|)$ where $\phi_{n-1}(\cdot)$ is the cdf of the standard Student-t distribution with $n-1$ degrees of freedom. If $\delta < \epsilon$ (a fixed threshold) we can confidently say that μ is significantly different from μ_0 . In this case, if $\bar{l} > \mu_0$, we decide $\mu > \mu_0$, otherwise we decide $\mu < \mu_0$. If $\delta \geq \epsilon$, we do not

have enough evidence to make a decision. In this case, we draw more data to reduce the uncertainty, s , in the sample mean $\bar{l}$. We keep drawing more data until we have the required confidence (i.e. until $\delta < \epsilon$). Note, that this procedure will terminate because when we have used all the available data, i.e. $n = N$, the standard deviation s is 0, the sample mean $\bar{l} = \mu$ and $\delta = 0 < \epsilon$. So, we will make the same decision as the original MH test would make. Pseudo-code for our test is shown in Algorithm 1. Here, we start with a mini-batch of size m for the first test and increase it by m datapoints when required.

Algorithm 1 Approximate MH test

Input: $\theta_t, \theta', \epsilon, \mu_0, X_N, m$

Output: *accept*

```

1: Initialize estimated means  $\bar{l} \leftarrow 0$  and  $\bar{l}^2 \leftarrow 0$ 
2: Initialize  $n \leftarrow 0$ , done  $\leftarrow$  false
3: while not done do
4:   Draw mini-batch  $\mathcal{X}$  of size  $\min(m, N - n)$  without replacement from  $X_N$  and set
      $X_N \leftarrow X_N \setminus \mathcal{X}$ 
5:   Update  $\bar{l}$  and  $\bar{l}^2$  using  $\mathcal{X}$ , and  $n \leftarrow n + |\mathcal{X}|$ 
6:   Estimate std  $s$  using Eqn. 3.1
7:   Compute  $\delta \leftarrow 1 - \phi_{n-1} \left( \left| \frac{\bar{l} - \mu_0}{s} \right| \right)$ 
8:   if  $\delta < \epsilon$  then
9:     accept  $\leftarrow$  true if  $\bar{l} > \mu_0$  and false otherwise
10:    done  $\leftarrow$  true
11:   end if
12: end while

```

The advantage of our method is that often we can make confident decisions with $n < N$ datapoints and save on computation, although we introduce a small bias in the stationary distribution. But, we can use the computational time we save to draw more samples and reduce the variance. The bias-variance trade-off can be controlled by adjusting ϵ . When ϵ is high, we make decisions without enough confidence and introduce a high bias. As $\epsilon \rightarrow 0$, we make more accurate decisions but are forced to examine more data which results in high variance.

Our algorithm will behave erratically if the assumptions of the CLT do not hold, e.g. with very

sparse datasets or datasets with extreme outliers. The CLT assumption can be easily tested empirically before running the algorithm to avoid such pathological situations. However, we expect any algorithm that does not examine every datapoint to suffer from similar problems on such datasets.

3.3 Experiments

3.3.1 Random Walk - Logistic Regression

We first test our method using a random walk proposal $q(\theta'|\theta_t) = \mathcal{N}(\theta'|\theta_t, \sigma_{RW}^2)$. Although the random walk proposal is not efficient, it is very useful for illustrating our algorithm because the proposal does not contain any information about the target distribution, unlike Langevin or Hamiltonian methods. So, the responsibility of converging to the correct distribution lies solely with the MH test. Also since q is symmetric, it does not appear in the MH test and we can use $\mu_0 = \frac{1}{N} \log [u\rho(\theta_t)/\rho(\theta')]$.

The target distribution in this experiment was the posterior for a logistic regression model trained on the MNIST dataset for classifying digits 7 vs 9. The dataset consisted of 12214 datapoints and we reduced the dimensionality from 784 to 50 using PCA. We chose a Gaussian prior over the parameters.

In figure 3.1, we show marginals of θ_1 vs θ_2 for $\epsilon = 0$ (exact MH) and $\epsilon = 0.01$ at different values of wall clock time. The red curves are marginals for 3 different runs of the random walk algorithm whereas the blue curve shows the true marginal obtained from a long run of Hybrid Monte Carlo. After 50 secs, the exact MH algorithm $\epsilon = 0$ has still not completed burn-in. Our algorithm with $\epsilon = 0.01$ is able to accelerate burn-in because it can collect more samples in a given amount of time. Theoretically, as more computational time becomes available the

exact MH algorithm will catch up with (and eventually outperform) our algorithm, because the exact MH algorithm is unbiased unlike ours. But the bias is hardly noticeable in this example and the error is still dominated by the variance even after collecting around 100K samples.

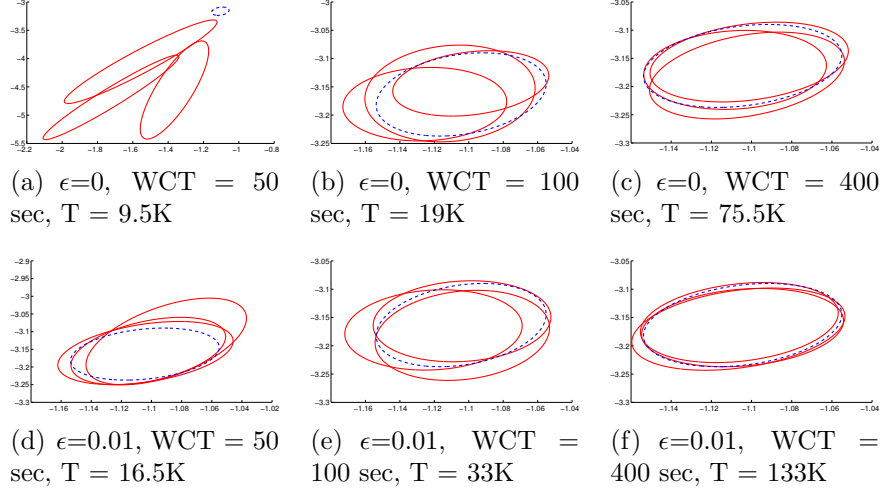


Figure 3.1: Marginals θ_1 vs θ_t for $\epsilon = 0$ and $\epsilon = 0.01$ at different values of wall clock time. The actual number of samples T is shown as well.

In Fig. 3.2, we show how the logarithm of the risk in estimating the mean prediction, decreases as a function of wall clock time. The mean prediction, given a test point x^* , is defined as $\mathbb{E}_{p(\theta|X_N)}[p(x^*|\theta)]$. To calculate the risk, we first estimate the true mean prediction using a long run of Hybrid Monte Carlo. Then, we compute multiple estimates of the mean prediction from our approximate algorithm and obtain the risk as the mean squared error in these estimates. We plot the average risk of 2037 datapoints in the test set. Since the risk $R = B^2 + V = B^2 + \frac{\sigma^2 f}{T}$, we expect it to decrease as a function of time until the bias dominates the variance. The figure shows that even after collecting a lot of samples, the risk is still dominated by the variance and the minimum risk is obtained with $\epsilon > 0$.

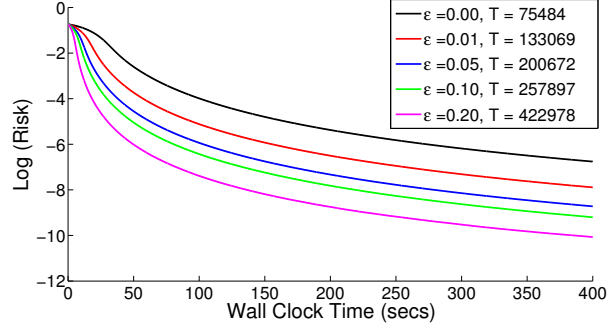


Figure 3.2: Logistic Regression: Risk in mean prediction

3.3.2 Independent Component Analysis

Next, we use our algorithm to sample from the posterior distribution of the unmixing matrix in Independent Component Analysis (ICA) [Hyvärinen and Oja, 2000]. When using prewhitened data, the unmixing matrix $W \in \mathbb{R}^{D \times D}$ is constrained to lie on the Stiefel manifold of orthonormal matrices. We choose a prior that is uniform over the manifold and zero elsewhere. We model the data as $p(x|W) = |\det(W)| \prod_{j=1}^D [4 \cosh^2(\frac{1}{2} w_j^T x)]^{-1}$ where w_j are the rows of W . Since the prior is zero outside the manifold, the same is true for the posterior. Therefore we use a random walk on the Stiefel manifold as a proposal distribution [Ouyang, 2008]. Since this is a symmetric proposal distribution, it does not appear in the MH test and we can use $\mu_0 = \frac{1}{N} \log[u]$.

To perform a large scale experiment, we created a synthetic dataset by mixing 1.95 million samples of 4 sources: (a) a Classical music recording (b) street / traffic noise (c) & (d) 2 independent Gaussian sources. To measure the correctness of the sampler, we measure the risk in estimating $I = \mathbb{E}_{p(W|X)} [d_A(W, W_0)]$ where the test function d_A is the Amari distance [Amari et al., 1996] and W_0 is the true unmixing matrix. We computed the ground truth using a long run ($T = 100K$ samples) of the exact MH algorithm. Then we ran each algorithm 10 times, each time for ≈ 6400 secs. We calculated the risk by averaging the squared error in the estimate from each Markov chain, over the 10 chains. This is shown in Fig. 3.3. Note that even after 6400 secs the variance dominates the bias, as evidenced by the still decreasing

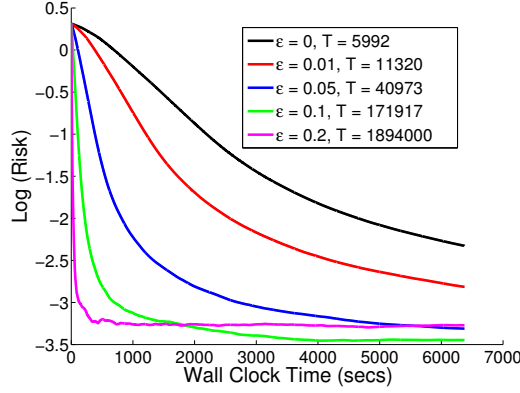


Figure 3.3: ICA: Risk in mean of Amari distance

risk, except for the most biased algorithm with $\epsilon = 0.2$. Also, the lowest risk at 6400 secs is obtained with $\epsilon = 0.1$ and not the exact MH algorithm ($\epsilon = 0$). But we expect the exact algorithm to outperform all the approximate algorithms if we were to run for an infinite time.

3.3.3 Variable selection in Logistic Regression

Now, we apply our MH test to variable selection in a logistic regression model using the reversible jump MCMC algorithm of Green [1995]. We use a model that is similar to the Bayesian LASSO model for linear regression described in Chen et al. [2011]. Specifically, given D input features, our parameter $\theta = \{\beta, \gamma\}$ where β is a vector of D regression coefficients and γ is a D dimensional binary vector that indicates whether a particular feature is included in the model or not. The prior we choose for β is $p(\beta_j | \gamma, \nu) = \frac{1}{2\nu} \exp \left\{ -\frac{|\beta_j|}{\nu} \right\}$ if $\gamma_j = 1$. If $\gamma_j = 0$, β_j does not appear in the model. Here ν is a shrinkage parameter that pushes β_j towards 0, and we choose a prior $p(\nu) \propto 1/\nu$. We also place a right truncated Poisson prior $p(\gamma | \lambda) \propto \frac{\lambda^k}{\binom{D}{k} k!}$ on γ as in Chen et al. [2011] to control the size of the model, $k = \sum_{j=1}^D \gamma_j$.

Denoting the likelihood of the data by $l_N(\beta, \gamma)$, the posterior distribution after integrating

out ν is $p(\beta, \gamma | X_N, y_N, \lambda) \propto l_N(\beta, \gamma) \|\beta\|_1^{-k} \lambda^k B(k, D-k+1)$ where $B(., .)$ is the beta function. Instead of integrating out λ , we use it as a parameter to control the size of the model. We use the same proposal distribution as in Chen et al. [2011] which is a mixture of 3 type of moves that are picked randomly in each iteration: an update move, a birth move and a death move. The update move is the usual MCMC move which involves changing the parameter vector β without changing the model γ . Specifically, we randomly pick an active component $j : \gamma_j = 1$ and set $\beta_j = \beta_j + \eta$ where $\eta \sim \mathcal{N}(0, \sigma_{update})$. The birth move involves (for $k < D$) randomly picking an inactive component $j : \gamma_j = 0$ and setting $\gamma_j = 1$. We also propose a new value for $\beta_j \sim \mathcal{N}(0, \sigma_{birth})$. The birth move is paired with a corresponding death move (for $k > 1$) which involves randomly picking an active component $j : \gamma_j = 1$ and setting $\gamma_j = 0$. The corresponding β_j is discarded. The probabilities of picking these moves $p(\gamma \rightarrow \gamma')$ are the same as in Chen et al. [2011]. The values of μ_0 used in the MH test for different moves are given below.

1. Update move:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{\|\beta\|_1^{-k}}{\|\beta'\|_1^{-k}} \right] \quad (3.3)$$

2. Birth move:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{\|\beta\|_1^{-k} p(\gamma \rightarrow \gamma') \mathcal{N}(\beta_j | 0, \sigma_{birth}) (D - k)}{\|\beta'\|_1^{-(k+1)} p(\gamma' \rightarrow \gamma) \lambda k} \right] \quad (3.4)$$

2. Death move:

$$\mu_0 = \frac{1}{N} \times \log \left[u \frac{\|\beta\|_1^{-k} p(\gamma \rightarrow \gamma')}{\|\beta'\|_1^{-(k-1)} p(\gamma' \rightarrow \gamma)} \frac{\lambda(k-1)}{\mathcal{N}(\beta_j | 0, \sigma_{birth}) (D - k + 1)} \right] \quad (3.5)$$

We used $\sigma_{update} = 0.01$ and $\sigma_{birth} = 0.1$ in this experiment.

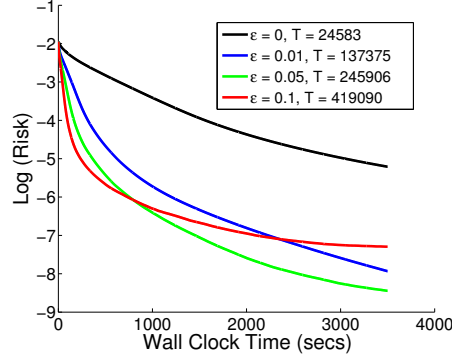


Figure 3.4: RJMCMC: Risk in mean prediction

We applied this to the MiniBooNE dataset from the UCI machine learning repository [Bache and Lichman, 2013]. Here the task is to classify electron neutrinos (signal) from muon neutrinos (background). There are 130,065 datapoints (28% in +ve class) with 50 features to which we add a constant feature of 1's. We randomly split the data into a training (80%) and testing (20%) set. To compute ground truth, we collected $T=400K$ samples using the exact reversible jump algorithm ($\epsilon = 0$). Then, we ran the approximate MH algorithm with different values of ϵ for around 3500 seconds. We plot the risk in predictive mean of test data (estimated from 10 Markov chains) in Fig. 3.4. Again we see that the lowest risk is obtained with $\epsilon > 0$.

The acceptance rates for the birth/death moves starts off at $\approx 20\%$ but dies down to $\approx 2\%$ once a good model is found. The acceptance rate for update moves is kept at $\approx 50\%$. The model also suffers from local minima. For the plot in Fig. 3.4, we started with only one variable and we ended up learning models with around 12 features, giving a classification error $\approx 15\%$. But, if we initialize the sampler with all features included and initialize β to the MAP value, we learn models with around 45 features, but with a lower classification error $\approx 10\%$. Both the exact reversible jump algorithm and our approximate version suffer from this problem. We should bear this in mind when interpreting “ground truth”. However, we have observed that when initialized with the same values, we obtain similar results with the approximate algorithm and the exact algorithm. For example, we plot the marginal posterior

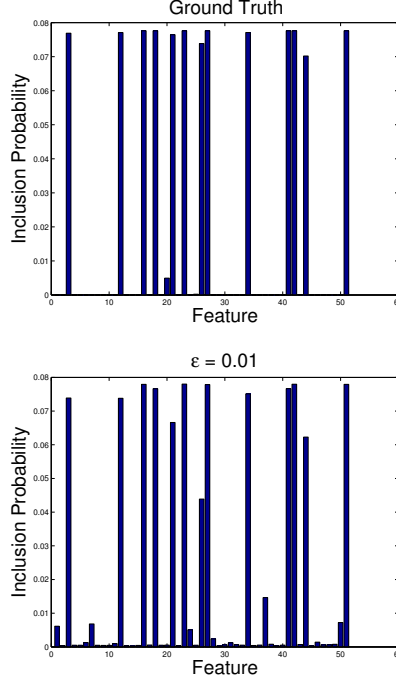


Figure 3.5: Marginal probability of features to be included in the model

probability of including a feature in the model, i.e. $p(\gamma_j = 1|X_N, y_N, \lambda)$ in figure 3.5.

3.4 Error Analysis and Test Design

We will now present an error analysis of our algorithm and discuss how to optimally set the hyper-parameters to minimize data usage given a bound on the error. We provide only a brief summary of the main results in this section. For more details, please refer to Korattikara et al. [2014].

3.4.1 Error Analysis and Estimation

The parameter ϵ is an upper-bound on the error of a single test and not the error of the complete sequential test. To compute this error, we assume a) n is large enough that the t statistic can be approximated with a z statistic, and b) the joint distribution of the $\bar{l}$'s corresponding

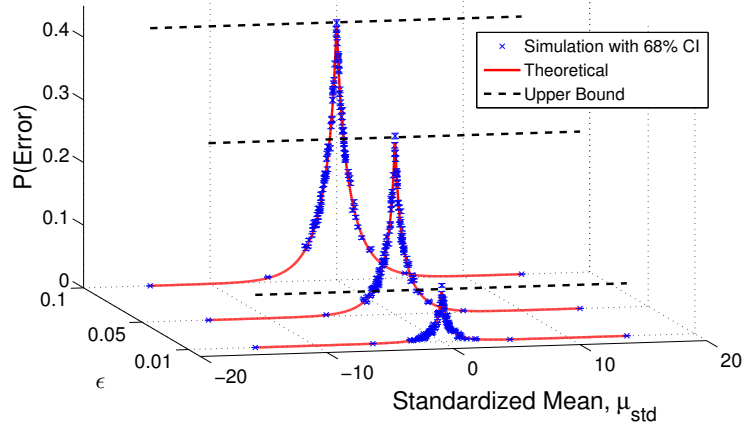


Figure 3.6: Error $\mathcal{E}$ estimated using simulation (blue cross with 1σ error bar) and dynamic programming (red line). An upper bound (black dashed line) is also shown.

to different mini-batches used in the test is multivariate normal. Under these assumptions, we can show that the test statistic at different stages of the sequential test follows a Gaussian Random Walk process. This allows us to compute the error of the sequential test $\mathcal{E}(\mu_{\text{std}}, m, \epsilon)$, and the expected proportion of the data required to reach a decision $\bar{\pi}(\mu_{\text{std}}, m, \epsilon)$, using an efficient dynamic programming algorithm. Both $\mathcal{E}$ and $\bar{\pi}$ depend on θ, θ' and u only through the ‘standardized mean’ defined as $\mu_{\text{std}}(u, \theta, \theta') \stackrel{\text{def}}{=} \frac{(\mu(\theta, \theta') - \mu_0(\theta, \theta', u)) \sqrt{N-1}}{\sigma_l(\theta, \theta')}$ where σ_l is the true standard deviation of the l_i ’s. See [Korattikara et al., 2014] for a detailed derivation and an empirical verification of the assumptions.

Fig. 3.6 shows the theoretical and actual error of 1000 sequential tests for the logistic regression model described in Section 3.3.1. The error $\mathcal{E}(\mu_{\text{std}}, m, \epsilon)$ is highest in the worst case when $\mu = \mu_0$. Therefore, $\mathcal{E}(0, m, \epsilon)$ is an upper-bound on $\mathcal{E}$. Since the error decreases sharply as μ moves away from μ_0 , we can get a more useful estimate of $\mathcal{E}$ if we have some knowledge about the distribution of μ_{std} ’s that will be encountered during the Markov chain simulation.

Now, let $P_{a,\epsilon}(\theta, \theta')$ be the actual acceptance probability of our algorithm and let $\Delta(\theta, \theta') \stackrel{\text{def}}{=} P_{a,\epsilon}(\theta, \theta') - P_a(\theta, \theta')$ be the error in $P_{a,\epsilon}$. In the supplementary section of Korattikara et al.

[2014], we show that for any (θ, θ') :

$$\Delta = \int_{P_a}^1 \mathcal{E}(\mu_{\text{std}}(u)) du - \int_0^{P_a} \mathcal{E}(\mu_{\text{std}}(u)) du \quad (3.6)$$

Thus, the errors corresponding to different u 's partly cancel each other. As a result, although $|\Delta(\theta, \theta')|$ is upper-bounded by the worst-case error $\mathcal{E}(0, m, \epsilon)$ of the sequential test, the actual error is usually much smaller. For any given (θ, θ') , Δ can be computed easily using 1-dimensional quadrature.

Finally, we show that the error in the stationary distribution is bounded linearly by $\Delta_{\max} = \sup_{\theta, \theta'} |\Delta(\theta, \theta')|$. As noted above, $\Delta_{\max} \leq \mathcal{E}(0, m, \epsilon)$ but is usually much smaller. Let $d_v(P, Q)$ denote the total variation distance¹ between two distributions, P and Q . If the transition kernel $\mathcal{K}_0$ of the exact Markov chain satisfies the contraction condition $d_v(P\mathcal{K}_0, S_0) \leq \eta d_v(P, S_0)$ for all probability distributions P with a constant $\eta \in [0, 1)$, we can prove ([Korattikara et al., 2014]) the following upper bound on the error in the stationary distribution:

Theorem 1. *The distance between the posterior distribution S_0 and the stationary distribution of our approximate Markov chain S_ϵ is upper bounded as:*

$$d_v(S_0, S_\epsilon) \leq \frac{\Delta_{\max}}{1 - \eta}$$

3.4.2 Optimal Sequential Test Design

We now briefly describe how to choose the parameters of the algorithm: ϵ , the error of a single test and m , the mini-batch size. A very simple strategy we recommend is to choose $m \approx 500$ so that the Central Limit Theorem holds and keep ϵ as small as possible while maintaining a low average data usage. This rule works well in practice and is used in Experiments 3.3.1

¹The total variation distance between two distributions P and Q , that are absolutely continuous w.r.t. measure Ω , is defined as $d_v(P, Q) \stackrel{\text{def}}{=} \frac{1}{2} \int_{\theta \in \Theta} |f_P(\theta) - f_Q(\theta)| d\Omega(\theta)$ where f_P and f_Q are their respective densities (or Radon-Nikodym derivatives to be more precise).

- 3.3.3.

The more discerning practitioner can design an optimal test that minimizes the data used while keeping the error below a given tolerance. Ideally, we want to do this based on a tolerance on the error in the stationary distribution S_ϵ . Unfortunately, this error depends on the contraction parameter η of the exact transition kernel which is difficult to compute. A more practical choice is a bound on the error Δ in the acceptance probability since the error in S_ϵ increases linearly with Δ . Since Δ is a function of (θ, θ') , we can try to control the average value of Δ over the empirical distribution of (θ, θ') that would be encountered while simulating the Markov chain. Given a tolerance Δ^* on this average error, we can find the optimal m and ϵ by solving the following optimization problem (e.g. using grid search) to minimize the average data usage :

$$\begin{aligned} \min_{m, \epsilon} \mathbb{E}_{\theta, \theta'} [\mathbb{E}_u \bar{\pi}(\mu_{\text{std}}(u, \theta, \theta'), m, \epsilon)] \\ \text{s.t. } \mathbb{E}_{\theta, \theta'} |\Delta(m, \epsilon, \theta, \theta')| \leq \Delta^* \end{aligned} \quad (3.7)$$

In the above equation, we estimate the average data usage, $\mathbb{E}_u[\bar{\pi}]$, and the error in the acceptance probability, Δ , using dynamic programming with one dimensional numerical quadrature on u . The empirical distribution for computing the expectation with respect to (θ, θ') can be obtained using a trial run of the Markov chain. Without a trial run the best we can do is to control the worst case error $\mathcal{E}(0, m, \epsilon)$ (which is also an upper-bound on Δ) in each sequential test by solving the following minimization problem:

$$\min_{m, \epsilon} \bar{\pi}(0, m, \epsilon) \quad \text{s.t. } \mathcal{E}(0, m, \epsilon) \leq \Delta^* \quad (3.8)$$

But this leads to a very conservative design as the worst case error is usually much higher than the average case error.

We now illustrate the advantages of the optimal test design by applying it to the ICA

experiment described in Section 3.3.2. We consider both design methods: the ‘average design’ (Eqn. 3.7) and the ‘worst-case design’ (Eqn. 3.8). For the average design, we collected 100 samples of the Markov chain to approximate the expectation of the error over (θ, θ') . We will call these samples the training set. The worst case design does not need the training set as it does not involve the distribution of (θ, θ') . We compute the optimal m and ϵ using grid search, for different values of the target training error, for both designs. We then collect a new set of 100 samples (θ, θ') and measure the average error and data usage on this test set (Fig. 3.7).

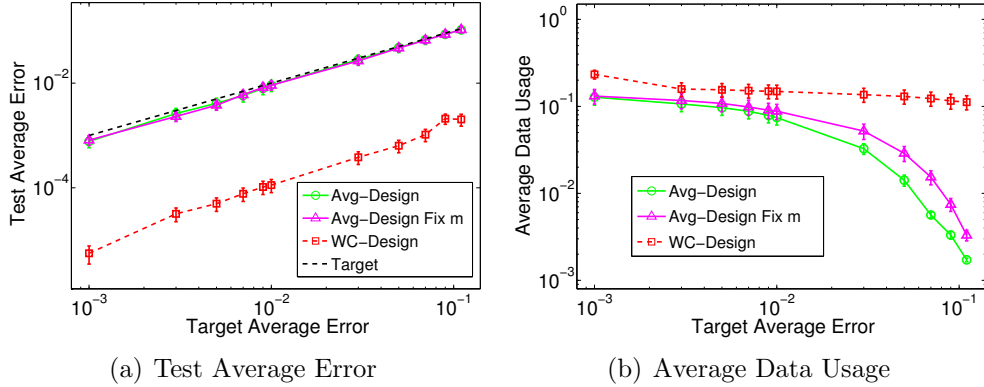


Figure 3.7: Test average error in P_a and data usage $\mathbb{E}_u[\bar{\pi}]$ for the ICA experiment using average design over both m and ϵ ($\circ$), with fixed $m = 600$ ($\triangle$), and worst-case design ($\square$).

For the same target error on the training set, the worst-case design gives a conservative parameter setting that achieves a much smaller error on the test set. In contrast, the average design achieves a test error that is almost the same as the target error (Fig. 3.7(a)). Therefore, it uses much less data than the worst-case design (Fig. 3.7(b)).

We also analyze the performance in the case where we fix $m = 600$ and only change ϵ . This is a simple heuristic we recommended at the beginning of this section. Although this usually works well, using the optimal test design ensures the best performance. In this experiment, we see that when the error is large, the optimal design uses only half the data (Fig. 3.7(b)) used by the heuristic and is therefore twice as fast.

3.5 Related Work

A similar method was used in Singh et al. [2012], where the factors of a graphical model are sub-sampled to compute fixed-width confidence intervals for the log-likelihood in the MH test. Our work is most similar to Bardenet et al. [2014] which uses concentration bounds instead of confidence intervals to measure the precision in a mini-batch. Since this algorithm does not use any central limit assumptions, it is more generally applicable than ours. But the generality comes at a price, because when the normal assumptions are indeed satisfied (as is usually the case), the algorithm based on concentration bounds uses more data than our test. However, it can be useful in cases where the normal assumptions do not hold, e.g. with very sparse datasets or datasets with extreme outliers.

There has also been some work on developing *unbiased* scalable MH algorithms. The pseudo marginal MCMC method [Andrieu and Roberts, 2009] and the method developed by Lin et al. [2000] provide a way to conduct exact accept/reject tests using unbiased estimators of the likelihood. However, in our experiments, unbiased estimators of the likelihood that can be computed from mini-batches of data, such as the Poisson estimator [Fearnhead et al., 2008] or the Kennedy-Bhanot estimator [Lin et al., 2000] have very high variance for large datasets. Because of this, once we get a very high estimate of the likelihood, almost all proposed moves are rejected and the algorithm gets stuck. A more successful approach is the Firefly algorithm [Maclaurin and Adams, 2014] which uses auxiliary binary variables to determine whether a datapoint should be included in the mini-batch or not. This algorithm can generate unbiased samples from the posterior distribution, but requires a lower bound on the per datum likelihood.

A theoretical analysis of noisy MCMC algorithms like our approximate MH test can also be found in Pillai and Smith [2014] and Alquier et al. [2014].

Chapter 4

Approximate Slice Sampling

In this chapter, we show that the sequential Metropolis-Hastings test we developed in Chapter 3 can also be used to speed up the slice sampling algorithm on Bayesian posterior distributions.

This chapter is based on results from DuBois et al. [2014]. Christopher DuBois came up with the idea of using our sequential MH test to speed up the slice sampling algorithm. I helped with the algorithm design and performed the experiments in sections 4.3.1 - 4.3.3.

4.1 Slice Sampling

Slice sampling [Neal, 2003] is an MCMC method for sampling from a (potentially unnormalized) density $S_0(\theta)$, where $\theta \in \Theta \subset \mathbb{R}^D$, by sampling uniformly from the $D + 1$ dimensional region under the density. In this section, we will first introduce slice sampling for univariate distributions and later describe how it can be easily extended to the multivariate case.

We start by defining an auxiliary variable h , and a joint distribution $p(\theta, h)$ that is uniform

over the region under the density (i.e. the set $\{(\theta, h) : \theta \in \Theta, 0 < h < S_0(\theta)\}$) and 0 elsewhere. Since marginalizing $p(\theta, h)$ over h yields $S_0(\theta)$, sampling from $p(\theta, h)$ and discarding h is equivalent to sampling from $S_0(\theta)$. However, sampling from $p(\theta, h)$ is not straightforward and is usually implemented as an MCMC method where in every iteration t , we first sample $h_t \sim p(h|\theta_{t-1})$ and then sample $\theta_t \sim p(\theta|h_t)$.

Sampling $h_t \sim p(h|\theta_{t-1})$ is trivial since $p(h|\theta_{t-1})$ is just $\text{Uniform}[0, S_0(\theta_{t-1})]$. For later convenience, we will introduce the uniform random variable $u_t \sim \text{Uniform}[0, 1]$, so that h_t can be defined deterministically as $h_t = u_t S_0(\theta_{t-1})$. Now, $p(\theta|h_t)$ is a uniform distribution over the ‘slice’ $Q[u_t, \theta_{t-1}] = \{\theta : S_0(\theta) > u_t S_0(\theta_{t-1})\}$. Sampling $\theta_t \sim p(\theta|h_t)$ is hard because of the difficulty in finding all segments of the slice in a finite amount of time.

Neal [2003] developed a method where instead of sampling from $p(\theta|h_t)$ directly, a ‘stepping-out’ procedure is used to find an interval (L, R) around θ_{t-1} that includes most or all of the slice, and θ_t is sampled uniformly from the part of (L, R) that is on $Q[u_t, \theta_{t-1}]$. Pseudo code for one iteration of slice sampling is shown in Algorithm 2.

The stepping out procedure involves placing a window (L, R) of width w around θ_{t-1} and widening the interval by stepping away from θ_{t-1} in opposite directions until L and R are no longer in the slice. The maximum number of step-outs can be limited by a constant $V_{\max}$. Note that the initial placement of the interval (L, R) around θ_{t-1} in Algorithm 2 is random, and $V_{\max}$ is randomly divided into a limit on stepping to the left, V_L , and a limit on stepping to the right, V_R . These are necessary for detailed balance to hold, as described in [Neal, 2003].

Once a suitable interval containing all or most of the slice has been found, θ_t is sampled by repeatedly drawing a candidate state θ_{prop} uniformly from (L, R) until it falls on the slice Q . At this point, we set $\theta_t \leftarrow \theta_{\text{prop}}$ and move to the next iteration. If a proposed state θ_{prop} does not fall on the slice, the interval (L, R) can be shrunk to $(L, \theta_{\text{prop}})$ or $(\theta_{\text{prop}}, R)$ such

that θ_{t-1} is still included in the new interval. A proof that Algorithm 2 defines a Markov chain with the correct stationary distribution $p(\theta, h)$ can be found in [Neal, 2003].

Algorithm 2 Slice Sampling

Input: $\theta_{t-1}, w, V_{\max}$

Output: θ_t

```

    Draw  $u_t \sim \text{Uniform}[0, 1]$ 
    // Pick an initial interval randomly:
    Draw  $\bar{w} \sim \text{Uniform}[0, 1]$ 
    Initialize  $L \leftarrow \theta_{t-1} - w\bar{w}$  and  $R \leftarrow L + w$ 
    // Determine maximum steps in each direction:
    Draw  $\bar{v} \sim \text{Uniform}[0, 1]$ 
    Set  $V_L \leftarrow \text{Floor}(V_{\max}\bar{v})$  and  $V_R \leftarrow V_{\max} - 1 - V_L$ 
    // Obtain boundaries of slice:
    while  $V_L > 0$  and  $\text{OnSlice}(L, \theta_{t-1}, u_t)$  do
         $V_L \leftarrow V_L - 1$ 
         $L \leftarrow L - w$ 
    end while
    while  $V_R > 0$  and  $\text{OnSlice}(R, \theta_{t-1}, u_t)$  do
         $V_R \leftarrow V_R - 1$ 
         $R \leftarrow R + w$ 
    end while
    // Sample until proposal is on slice:
    loop
        Draw  $\eta \sim \text{Uniform}[0, 1]$ 
         $\theta_{\text{prop}} \leftarrow L + \eta(R - L)$ 
        if  $\text{OnSlice}(\theta_{\text{prop}}, \theta_{t-1}, u_t)$  then
             $\theta_t \leftarrow \theta_{\text{prop}}$ 
            break
        end if
        if  $\theta_{\text{prop}} < \theta_{t-1}$  then
             $L \leftarrow \theta_{\text{prop}}$ 
        else
             $R \leftarrow \theta_{\text{prop}}$ 
        end if
    end loop

```

The algorithm uses Procedure `OnSlice`, both during the stepping-out phase and the rejection sampling phase, to check whether a point (L , R or θ_{prop}) lies on the slice $Q[u_t, \theta_{t-1}]$. In a practical implementation, $S_0(\theta)$ is calculated only once per iteration of the slice sampler, and does not have to be recomputed every time Procedure 3 is used.

Procedure 3 OnSlice

Input: θ', θ, u **Output:** Is θ' on $Q[u, \theta]$?1: **return** $S_0(\theta') > uS_0(\theta)$

One way to generalize the slice sampling algorithm to multiple dimensions is to pick a direction in parameter space in each iteration and update θ only in the chosen direction. The direction can be picked uniformly at random from one of the co-ordinate directions or from all possible directions. It is also valid to cycle through the co-ordinate directions instead of picking one at random.

Relative to other MCMC techniques, slice sampling can allow for larger moves, allowing one to reduce autocorrelation in the samples of the Markov chain and thereby explore the parameter space more efficiently. The technique has been extended to multivariate distributions [Thompson and Neal, 2010], truncated normal distributions [Lu, 2008], latent Gaussian models [Murray et al., 2009, Nishihara et al., 2012], and others.

4.2 Approximate Slice Sampling

A typical sampling algorithm must evaluate the posterior density exactly, atleast once for each sample that it generates. In slice sampling, this evaluation is performed by the **OnSlice** procedure which tests whether a given point (L, R or θ_{prop}) lies on the slice or not. When the dataset has billions of datapoints, computing the posterior density is very expensive as it requires $O(N)$ evaluations of the likelihood. Note that, in slice sampling, this has to be done multiple times for each sample we generate. Performing $O(N)$ computations for 1 bit of information, i.e. whether a point is on the slice or not, is not a cogent use of computational resources. Therefore, in this section, we will develop an approximate slice sampling algorithm by cutting down the computational time of the **OnSlice** procedure.

Algorithm 4 OnSliceApprox

Input: $\theta', \theta, u, \epsilon, m$ **Output:** Is θ' on $Q[u, \theta]$?

- 1: Initialize mean estimates $\bar{l} \leftarrow 0$ and $\bar{l}^2 \leftarrow 0$
 - 2: Initialize $n \leftarrow 0$
 - 3: $\mu_0 \leftarrow \frac{1}{N}[\log u + \log p(\theta) - \log p(\theta')]$
 - 4: **loop**
 - 5: Draw mini-batch $\mathcal{X}$ of size $\min(m, N - n)$ without replacement from X_N and set $X_N \leftarrow X_N \setminus \mathcal{X}$
 - 6: Update $\bar{l}$ and $\bar{l}^2$ using $\mathcal{X}$, and $n \leftarrow n + |\mathcal{X}|$
 - 7: Estimate std $s \leftarrow \sqrt{1 - \frac{n}{N}} \sqrt{\frac{\bar{l}^2 - (\bar{l})^2}{n - 1}}$
 - 8: Compute $\delta \leftarrow 1 - \phi_{n-1}\left(\left|\frac{\bar{l} - \mu_0}{s}\right|\right)$
 - 9: **if** $\delta < \epsilon$ **then**
 - 10: **break**
 - 11: **end if**
 - 12: **end loop**
 - 13: **return** $\bar{l} > \mu_0$
-

The OnSlice procedure determines whether a point θ' is on the slice $Q[u, \theta]$ by checking if $S_0(\theta') > uS_0(\theta)$. When S_0 is a Bayesian posterior distribution, we can take the logarithm of both sides of this inequality and divide by N to rephrase the test as $\mu > \mu_0$, where:

$$\begin{aligned} \mu_0 &= \frac{1}{N} \log \left[u_t \frac{p(\theta_{t-1})}{p(\theta')} \right], \quad \text{and} \\ \mu &= \frac{1}{N} \sum_{i=1}^N l_i \quad \text{where} \quad l_i = \log p(x_i; \theta') - \log p(x_i; \theta_t) \end{aligned} \tag{4.1}$$

i.e., if $\mu > \mu_0$, we can conclude that θ' lies on the slice.

Note that we are testing if the mean of the finite population $\{l_1 \dots l_N\}$ is greater than a constant that does not depend on the data. A similar scenario appears in the Metropolis-Hastings algorithm, where the mean difference in log-likelihoods is compared to a constant to determine whether to accept or reject a proposed sample. In chapter 3, we proposed a sequential hypothesis testing procedure to cut down the computational budget of this

comparison. We can easily apply this test to efficiently compare μ and μ_0 in the slice sampling algorithm. The details are described in Procedure `OnSliceApprox`. An efficient, but approximate, slice sampling algorithm can be obtained by replacing all calls to the `OnSlice` procedure in Algorithm 2 with calls to `OnSliceApprox`.

4.3 Experiments

4.3.1 L_1 Regularized Linear Regression

First, we test our method on the posterior distribution of a linear regression model with a Laplacian prior on the parameters. We trained the model using a 100K subset of the Million Song Dataset year prediction challenge[Bache and Lichman, 2013]. The goal is to predict the release year of a song from audio features. There are 90 features to which we added a constant feature of ones.

We use the risk in estimating the ‘average prediction’ as a measure to compare the relative performance of the exact and approximate slice sampling algorithms. The average prediction is defined as the expectation of the prediction under the posterior distribution, i.e. $\mathbb{E}_{p(\theta|X_N)}[\theta^T x^*]$, where x^* is a test point. To compute risk, we first compute the true average prediction using a long run (100K samples) of the exact slice sampling algorithm initialized at the mode of the distribution. Then, we run the exact and approximate slice samplers at different values of ϵ for 500 seconds. On average, we obtained $T = 12214, 48122$ and 104732 samples with $\epsilon = 0, 0.2$ and 0.3 respectively in 500 secs of computation. Each algorithm is run 10 times, and the risk is calculated as the squared error in the estimated mean prediction averaged over the 10 Markov chain simulations. We plot the risk in the average prediction, further averaged over 1000 randomly chosen test points, for different values of ϵ in Figure 4.1. $\epsilon = 0$ denotes the exact slice sampling algorithm.

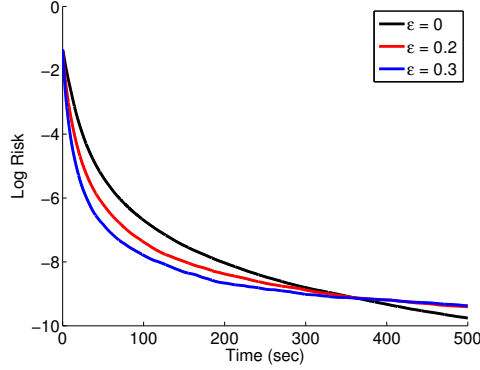


Figure 4.1: Risk in average prediction of test data for a linear regression model with a Laplacian prior

For a typical sampling algorithm, the risk is initially dominated by the burn-in bias. Burn-in usually happens very fast, after which the risk is dominated by the sampling variance and reduces as $O(1/T)$. For approximate MCMC algorithms, after collecting a large number of samples, the sampling variance will eventually be dominated by the asymptotic bias in the stationary distribution. This can be seen in Figure 4.1. Algorithms with a high ϵ are able to burn-in and reduce variance quickly. After collecting a large number of samples, their asymptotic bias will dominate the variance and prevent the risk from reducing further. The exact slice sampler is slower to burn-in and reduce variance, but it can achieve zero risk given an infinite amount of computational time.

4.3.2 Multinomial Regression

Next, we tested our method on a multinomial regression model trained on the MNIST dataset of handwritten digits. There are 60000 training points with 784 dimensions each. We reduced the dimensionality to 50 using PCA and added a constant feature of ones. Since there are 10 classes, this resulted in a 459 dimensional parameter vector. We chose a Gaussian prior over the parameters.

We plot the risk in estimating the average prediction in Figure 4.2. As in the previous

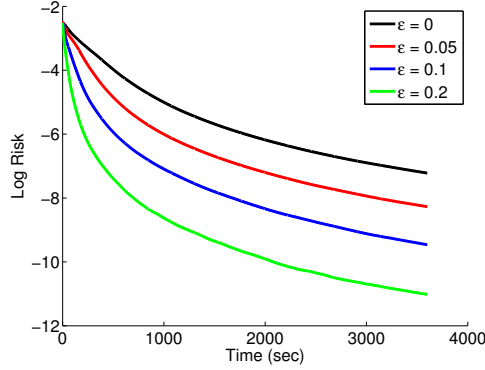


Figure 4.2: Risk in average prediction of test data for a Multinomial Regression model

experiment, ground truth was computed using 100K samples collected from the exact slice sampler and the risk was computed by averaging the squared error over 10 Markov chains for each value of ϵ . We also average the risk over 10 randomly chosen test points. We ran each algorithm for one hour and obtained $T = 13805, 27587, 56409$ and 168624 samples for $\epsilon = 0, 0.05, 0.1$ and 0.2 respectively.

The lowest risk after one hour of computation was achieved by the approximate slice sampler with the highest bias in the stationary distribution. Thus, even after one hour of computation, the risk is still dominated by sampling variance. If we were to run for a very long time, we would expect the exact slice sampler to outperform all the approximate algorithms.

4.3.3 Logistic Regression

Finally, we test the performance of our approximate slice sampler on the posterior distribution of a logistic regression model. The model was trained on the Covertype dataset[Bache and Lichman, 2013], where the goal is to predict forest cover type using cartographic variables. There are 7 cover types in the dataset, but we trained a 1-vs-rest classifier for the most populous class (Lodgepole pine). There were a total of 464809 training points with 54 features (to which we add a ‘constant’ feature of ones). We chose a Gaussian prior over the

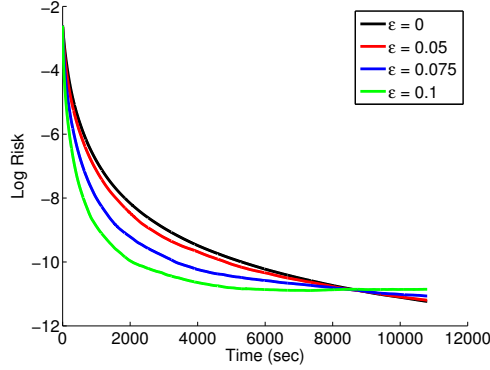


Figure 4.3: Risk in average prediction of test data for a Logistic Regression model

parameters.

To compute ground truth, we used a long run (100K samples) of the exact slice sampler initialized from the mode of the distribution. Then, we ran the exact and approximate slice sampling algorithms at different values of ϵ for 3 hours each. We obtained $T = 14808, 19182, 54156$ and 289497 samples with $\epsilon = 0, 0.05, 0.075$ and 0.1 respectively. In Figure 4.3, we plot the risk in estimating the average prediction of 10,000 randomly chosen test points.

The risk is initially dominated by the burn-in bias, and then by sampling variance. Approximate MCMC algorithms with higher values of ϵ are able to reduce both of these very quickly as shown in Figure 4.3. As more computational time becomes available, risk will eventually be dominated by the bias in the stationary distribution. In Figure 4.3, this can be seen for $\epsilon = 0.1$ which initially reduces risk very fast but is eventually outperformed by the less biased algorithms. However, this crossover does not happen until a large amount of computational time (≈ 8000 secs) has passed.

Chapter 5

Stochastic Gradient Langevin Dynamics

In chapter 2 (section 2.4.3), we discussed two strategies for designing fast approximate MCMC algorithms. The first method was to approximate the Metropolis-Hastings test directly using less data. We developed an algorithm to do this in chapter 3 and then applied the same idea to speed up the slice sampling algorithm in chapter 4.

Another strategy is to use efficient proposal distributions that have acceptance probabilities close to 1 so that we can accept every proposal and avoid the expensive MH test altogether. Welling and Teh [2011] designed a sampling algorithm called Stochastic Gradient Langevin Dynamics (SGLD) based on this idea. SGLD uses cheap noisy gradients of the target density, computed from a small mini-batch of data, to make proposals that have a high probability of being accepted.

In this chapter, we make two contributions to the theory and practice of SGLD sampling. In section 5.2, we analyze the error in the stationary distribution of the SGLD Markov Chain using tools from physics. Then in section 5.3, we show the advantages of combining SGLD

with our approximate MH test when sampling from certain target distributions. We will start with a review of the SGLD algorithm in the next section.

5.1 Stochastic Gradient Langevin Dynamics

As discussed briefly in 2, Langevin Monte Carlo [Roberts and Tweedie, 1996] is a sampling algorithm that uses gradient information to make proposals to regions of higher density, and hence have a higher probability of getting accepted. The algorithm gets its name from the Langevin diffusion which is defined by the following stochastic differential equation:

$$d\theta_t = \frac{1}{2}\psi(\theta_t) + d\eta_t \quad (5.1)$$

where $\theta \in \mathbb{R}^D$, $\psi(\theta) = \nabla_{\theta} \log S_0$ (gradient of the log density) and η_t is the D -dimensional standard Brownian motion. Under certain regularity conditions (see [Roberts and Tweedie, 1996]), it can be shown that the Langevin diffusion in Eqn. 5.1 has S_0 as its equilibrium distribution. To simulate the diffusion on a digital computer, one has to discretize the SDE, which is usually done as follows:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\alpha}{2}\psi(\theta_t) + \eta \quad \text{where } \eta \sim \mathcal{N}(0, \alpha) \quad (5.2)$$

Here, α is the discretization step size. The above update equation defines an ergodic Markov chain, but its equilibrium distribution is no longer S_0 because of $O(\alpha)$ errors introduced by the time discretization [Kronfeld, 1993]. Therefore, to sample correctly from S_0 , one has to treat the above as a proposal distribution that can then be corrected with an MH accept/reject test.

As $\alpha \rightarrow 0$, the acceptance probability approaches 1, since the discretized update approaches the Langevin diffusion. Thus, in theory, one can develop an approximate MCMC algorithm

by keeping α small and avoiding the MH test. However, the proposal requires computing the gradient $\psi(\theta) = \nabla_{\theta} S_0(\theta)$, which in the case of Bayesian posterior inference involves the gradient of the log-likelihood of all data points. Processing the entire dataset to generate each sample is prohibitively expensive when we have datasets with billions of data points.

Therefore, Welling and Teh [2011] developed the Stochastic Gradient Langevin dynamics algorithm by noting that it is possible to replace the gradient $\psi(\theta)$ in eqn. 5.2 with a noisy gradient and still keep the error in the stationary distribution under control. For Bayesian posterior inference, a noisy gradient $\hat{\psi}_n(\theta)$ can be estimated using a mini-batch of $n \ll N$ datapoints as follows:

$$\hat{\psi}_n(\theta) = \frac{N}{n} \sum_{i=1}^N \nabla_{\theta} \log p(x_i|\theta) + \nabla_{\theta} \log p(\theta) \approx \psi(\theta) = \sum_{i=1}^N \nabla_{\theta} \log p(x_i|\theta) + \nabla_{\theta} \log p(\theta) \quad (5.3)$$

Then, the update equation for SGLD is just:

$$\theta_{t+1} = \theta_t + \frac{\alpha}{2} \hat{\psi}_n(\theta) + \mathbb{N}(0, \alpha) \quad (5.4)$$

SGLD can generate each sample using only $O(n)$ computations, in contrast to traditional algorithms which require $O(N)$ computations per sample.

5.1.1 Bias-Variance Trade-off in SGLD

Recall that the variance of an MCMC estimator is proportional to τ/T , where τ is the integrated auto-correlation time and T is the number of samples. The parameters which control the bias-variance trade-off in SGLD are $\epsilon = \{\alpha, n\}$ and they affect the variance by acting on τ and T respectively.

Increasing α allows us to take bigger steps in parameter space, which reduces the auto-correlation time τ , and therefore the variance. However, since the error in the stationary

distribution is of $O(\alpha)$ (see section 5.2), increasing α increases the bias. As $\alpha \rightarrow 0$, we can obtain a theoretically unbiased algorithm, but in practice, the chain will be stuck at one point in parameter space.

On the other hand, if we increase n , the computational time required to generate each sample goes up. This limits the number of samples T that can be collected in a given amount of time and therefore increases the variance of the estimator. However, increasing n also reduces the noise in the gradient which can be shown to reduce the bias (see section 5.2). Note that it is not possible to obtain an unbiased algorithm merely by changing n . We can only obtain unbiasedness (in theory) as $\alpha \rightarrow 0$.

5.1.2 SGLD and Optimization

There is an interesting connection between SGLD and optimization. Note that the SGLD update in eqn. 5.4 is identical to that of stochastic gradient ascent except for the $O(\alpha)$ injected Gaussian noise. Intuitively, the injected noise allows SGLD to move around the MAP estimate, instead of converging to it as in optimization.

In fact, SGLD can behave either as an optimizer or as a sampler depending on the value of α . To see this, note that there are two sources of noise in SGLD: the injected Gaussian noise and the sub-sampling noise in estimating the gradient. The injected noise has variance α , whereas the sub-sampling noise has variance $O(\alpha^2)$ because the noisy gradient is multiplied by α . When α is high, the $O(\alpha^2)$ sub-sampling noise dominates the $O(\alpha)$ injected noise and the SGLD update behaves like stochastic gradient ascent optimization. On the other hand, when α is small, the $O(\alpha)$ injected noise will dominate the $O(\alpha^2)$ sub-sampling noise. In this case, we can ignore the noise in the gradient, and the algorithm behaves like a Langevin Dynamics sampler.

Welling and Teh [2011] recommend annealing α to 0 over time. Initially when α is high, SGLD behaves like stochastic gradient ascent to efficiently find the posterior mode (during burn-in). Then, as α is reduced, the algorithm seamlessly transitions into Langevin Dynamics and samples around the mode. The sampling is biased, but the bias disappears as $\alpha \rightarrow 0$.

5.2 Error Analysis

In this section, we will analyze the error¹ in the stationary distribution of the SGLD Markov chain upto first order in α . Let us denote the stationary distribution of SGLD by $S_\epsilon(\theta)$. The true target distribution is $S_0(\theta) \propto \exp\{-E(\theta)\}$ where $E(\theta) = -\sum_{i=1}^N \log p(x_i|\theta) - \log p(\theta)$ is the energy function.

The stationary distribution of any Markov chain will satisfy an infinite order differential equation called the Kramers-Moyal (KM) expansion[Coffey et al., 2004]:

$$\sum_{h=1}^{\infty} \frac{(-1)^h}{h!} \nabla_{\theta}^h [S_{\epsilon}(\theta) M_h(\theta)] = 0 \quad (5.5)$$

In the above, $M_h(\theta)$ are called the jump moments and are defined as:

$$M_h(\theta) = \int (\theta' - \theta)^h K(\theta'|\theta) d\theta' \quad (5.6)$$

where $K(\theta'|\theta)$ is the transition probability of moving from θ to θ' . We can find the stationary distribution of SGLD by solving the KM equation for S_{ϵ} . However, the KM equation is of infinite-order and we have to truncate it somehow to be able to solve it. Since we are interested in the stationary distribution as $\alpha \rightarrow 0$, the best way to truncate the KM equation is by computing the jump moments only upto some power of α . If we do this, the higher

¹For simplicity, we will assume that θ is one dimensional. The extension to multiple dimensions is straightforward, although more tedious.

order moments will vanish and drop out of the KM equation (see below).

To compute the jump moments, we rewrite the SGLD update as:

$$\theta_{t+1} - \theta_t = \phi_t = \omega(\theta_t) + \frac{\alpha N}{2n} \sum_{i=1}^n f_i(\theta_t)$$

where:

$$\begin{aligned} \omega(\theta) &\sim \mathcal{N}\left(-\frac{\alpha}{2}\nabla E(\theta), \alpha\right) \\ f_i(\theta) &= \nabla \log p(x_i|\theta) - \frac{1}{N} \sum_{j=1}^N \nabla \log p(x_j|\theta) \end{aligned} \tag{5.7}$$

We can obtain the moments of the jump $\phi_t = \theta_{t+1} - \theta_t$ via its Moment Generating Function (MGF) which can be computed from the MGFs of ω and f_i . Let M_ω denote the MGF of ω . Since ω is normally distributed, we can easily write down M_ω in terms of the moments of ω :

$$M_\omega(r) = 1 + r \left(-\frac{\alpha}{2}\nabla E\right) + \frac{r^2}{2!} \left(\frac{\alpha^2}{4}(\nabla E)^2 + \alpha\right) + \frac{r^3}{3!} \left(-\frac{3\alpha^2}{2}\nabla E\right) + \frac{r^4}{4!} (3\alpha^2) + \mathcal{O}(\alpha^3) \tag{5.8}$$

Now, we will calculate the MGF, $M_f(r)$, of the distribution² of the f_i 's. To do this, first note that $\mathbb{E}[f_i(\theta)] = 0$. Therefore, $\mathbb{E}[f_i^2(\theta)] = V(\theta)$, the empirical variance of the scores. Then we can write:

$$M_f(r) = 1 + \frac{r^2}{2!} V + \mathcal{O}(r^3) \tag{5.9}$$

²Please note that x_i is not the i^{th} data point in the dataset but instead the i^{th} data point in a randomly drawn mini-batch, i.e. x_i and f_i are random variables.

Defining $M_\phi(r)$ to be the MGF of ϕ , we have:

$$\begin{aligned}
M_\phi(r) &= M_\omega(r) \left[M_f \left(\frac{\alpha N r}{2n} \right) \right]^n \\
&= M_\omega(r) \left[1 + \frac{r^2}{2!} \frac{\alpha^2 N^2}{4n^2} V + \mathcal{O}(\alpha^3) \right]^n = M_\omega(r) \left[1 + \frac{r^2}{2!} \frac{\alpha^2 N^2}{4n} V + \mathcal{O}(\alpha^3) \right] \\
&= 1 + r \left(-\frac{\alpha}{2} \nabla E \right) + \frac{r^2}{2!} \left(\frac{\alpha^2}{4} (\nabla E)^2 + \frac{\alpha^2 N^2}{4n} V + \alpha \right) \\
&\quad + \frac{r^3}{3!} \left(-\frac{3\alpha^2}{2} \nabla E \right) + \frac{r^4}{4!} (3\alpha^2) + \mathcal{O}(\alpha^3)
\end{aligned} \tag{5.10}$$

We can easily read the jump moments off the above equation and plug them into the Kramers-Moyal expansion to obtain a differential equation for S_ϵ . If we only consider terms upto $\mathcal{O}(\alpha)$, we have:

$$\nabla \left[S_\epsilon \frac{\alpha \nabla E}{2} \right] + \frac{1}{2!} \nabla^2 [S_\epsilon \alpha] = 0 \implies \nabla [(\nabla E + \nabla) S_\epsilon] = 0 \tag{5.11}$$

The same differential equation appears in the Kramers-Moyal expansion of Langevin Dynamics. The unique solution of this equation is $S_\epsilon(\theta) \propto \exp \{-E(\theta)\}$ ([Kronfeld, 1993]). Thus, we proved the following lemma:

Lemma 1. *The stationary distribution for SGLD is $S_\epsilon(\theta) \propto \exp \{-E(\theta) + \mathcal{O}(\alpha)\}$*

To derive the higher order terms in the stationary distribution, we have to include terms upto $\mathcal{O}(\alpha^2)$ in the Kramers-Moyal equation. Including these terms and simplifying, we get:

$$\nabla [(\nabla E + \nabla) S_\epsilon] + \frac{\alpha}{4} \left[\nabla^2 \left\{ S_\epsilon \left((\nabla E)^2 + \frac{N^2 V}{n} \right) \right\} + 2 \nabla^3 \{ S_\epsilon \nabla E \} + \nabla^4 S_\epsilon \right] = 0 \tag{5.12}$$

To solve the above equation, we repeatedly use $\nabla S_\epsilon = -S_\epsilon \nabla E + \mathcal{O}(\alpha)$ (from Lemma 1) inside the square brackets to remove higher derivatives of S_ϵ . Solving the resulting equation, we get:

Theorem 2. *The stationary distribution for SGLD is:*

$$S_\epsilon(\theta) \propto \exp \left[-E(\theta) - \frac{\alpha}{4} \left\{ \nabla^2 E(\theta) - \frac{1}{2}(\nabla E(\theta))^2 + \frac{N^2}{n}(V(\theta) - \int_\theta V \nabla E) \right\} + \mathcal{O}(\alpha^2) \right]$$

A more detailed proof of Theorem 2 is provided in section A.1 of the appendix. Note that as $\alpha \rightarrow 0$, the stationary distribution S_ϵ approaches the desired target distribution $S_0 \propto \exp \{-E(\theta)\}$. Also, the error decreases as we increase the mini-batch size n . The above analysis assumed that the mini-batch was drawn with replacement from the complete dataset. If we sample the mini-batch items without replacement, variance of the f_i 's will be lower than V . Therefore, the error in the stationary distribution will be lower as well.

A similar analysis was recently presented in Sato and Nakagawa [2014] using the Fokker-Planck equation[Risken, 1984] and Itô processes. The authors showed that the stationary distribution of SGLD approaches the target distribution as $\alpha \rightarrow 0$. They also analyzed the error in discrete time SGLD versus continuous time SGLD. They showed that discrete time SGLD does not converge path-wise to continuous time SGLD (strong convergence), but the difference in expectations of a test function computed using discrete time SGLD and continuous time SGLD goes to 0 as $\alpha \rightarrow 0$ (weak convergence).

5.3 MH Corrected SGLD

In section 5.2, we showed that the error in the stationary distribution of the SGLD Markov chain is $O(\alpha)$. However, keeping α small to control the bias increases the auto-correlation time, which in turn increases sampling variance. Therefore, we would like to use a Metropolis-Hastings correction as it allows us to generate unbiased samples even at larger values of α . Since the exact MH test uses the entire dataset for each accept/reject decision, we would like to instead use the approximate version of the MH algorithm.

To apply an MH test, note that the SGLD proposal $q(\theta'|\theta)$ can be considered a mixture of proposals $q(\theta'|\theta, \mathcal{X}_n)$ corresponding to different mini-batches $\mathcal{X}_n$, i.e.

$$q(\theta'|\theta, \mathcal{X}_n) = \mathcal{N} \left(\theta'|\theta + \frac{\alpha}{2} \nabla_{\theta} \left\{ \frac{N}{n} \sum_{x \in \mathcal{X}_n} \log p(x|\theta) + \log p(\theta) \right\}, \alpha \right) \quad (5.13)$$

The mixture proposal $q(\theta'|\theta)$ will satisfy detailed balance with respect to the posterior distribution if the MH test enforces detailed balance between the posterior and each of the component proposals $q(\theta'|\theta, \mathcal{X}_n)$. We can enforce detailed balance for each proposal $q(\theta'|\theta, \mathcal{X}_n)$ using an MH test with acceptance probability:

$$P_a = \min \left[1, \frac{S_0(\theta')q(\theta|\theta', \mathcal{X}_n)}{S_0(\theta)q(\theta'|\theta, \mathcal{X}_n)} \right] \quad (5.14)$$

A case where correcting SGLD with an MH test can be very useful is described below. Consider an L1-regularized linear regression model given a dataset $\{x_i, y_i\}_{i=1}^N$ where x_i are predictors and y_i are targets. We use a Gaussian error model $p(y|x, \theta) \propto \exp \left\{ -\frac{\lambda}{2} (y - \theta^T x)^2 \right\}$ and choose a Laplacian prior for the parameters $p(\theta) \propto \exp(-\lambda_0 \|\theta\|_1)$. For pedagogical reasons, we will restrict ourselves to a toy version of the problem where θ and x are one dimensional. We use a synthetic dataset with $N = 10000$ datapoints generated as $y_i = 0.5x_i + \xi$ where $\xi \sim \mathcal{N}(0, 1/3)$. We choose $\lambda = 3$ and $\lambda_0 = 4950$, so that the prior is not washed out by the likelihood. The posterior density and the gradient of the log posterior are shown in figures 5.1(a) and 5.1(b) respectively.

An empirical histogram of samples obtained by running SGLD with $\alpha = 5 \times 10^{-6}$ is shown in Fig. 5.1(c). The effect of omitting the MH test is quite severe here. When the sampler reaches the mode of the distribution, the Langevin noise occasionally throws it into the valley to the left, where the gradient is very high. This propels the sampler far off to the right, after which it takes a long time to find its way back to the mode. However, if we had used an MH accept-reject test, most of these troublesome jumps into the valley would be rejected

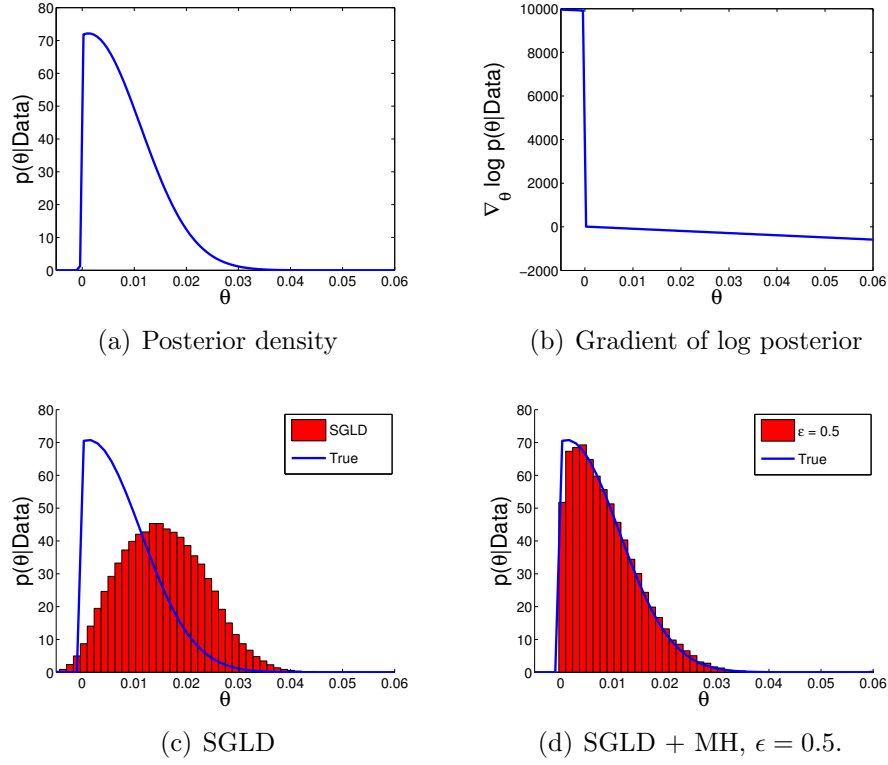


Figure 5.1: Pitfalls of using uncorrected SGLD

because the density in the valley is much lower than that at the mode.

The result of running SGLD (keeping $\alpha = 5 \times 10^{-6}$ as before) corrected using our approximate MH test, with $\epsilon = 0.5$, is shown in Fig. 5.1(d). As expected, the MH test rejects most jumps into the valley. The stationary distribution is almost indistinguishable from the true posterior. Note that when $\epsilon = 0.5$, a decision is always made in the first step (using just $m = 500$ datapoints) without querying additional data sequentially.

However, we found that the acceptance probability of SGLD is generally quite low. For models with smoother densities such as logistic regression and neural networks, the reduction in bias from conducting an MH test was not enough to compensate for the increase in auto-correlation (and therefore variance) from rejecting a large number of samples.

Chapter 6

Stochastic Gradient Fisher Scoring

A limitation of the SGLD algorithm (Chapter 5) is that we have to use a small step size to keep the bias under control. A small step size hurts the mixing rate of the algorithm which in turn affects the variance of any estimates computed from the generated samples. In this section, we propose an algorithm called Stochastic Gradient Fisher Scoring (SGFS) [Ahn et al., 2012] to improve the mixing behavior of SGLD by using a preconditioning matrix that captures the curvature of the target distribution.

Max Welling proposed the initial version of the algorithm. Sungjin Ahn performed the experiments in sections 6.5.1 and 6.5.2. I reformulated Max’s original version of the algorithm into its current form as the hybrid of two Markov chains, performed the experiments in section 6.5.3 and provided a proof for Theorem 3.

6.1 Preliminaries and Notation

We will start with some preliminaries and notation. We denote the score or the gradient of the log likelihood w.r.t. data-point x_i as $g_i(\theta) = g(\theta; x_i) = \nabla_{\theta} \log p(\theta; x_i)$. We denote the

sum of scores of a batch of n data-points $X_r = \{x_{r_1} \dots x_{r_n}\}$ by $G_n(\theta; X_r) = \sum_{i=1}^n g(\theta; x_{r_i})$ and the average by $\bar{g}_n(\theta; X_r) = \frac{1}{n} G_n(\theta; X_r)$. Sometimes we will drop the argument X_r and instead simply write $G_n(\theta)$ and $\bar{g}_n(\theta)$ 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.

We assume that the true data generating distribution is in our family of models $p(x|\theta)$ and denote the true parameter which generated the dataset as θ_0 . 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, $I_n = nI_1$. The empirical covariance of the scores computed from a batch of n data-points is called the *empirical* Fisher information, $V(\theta; X_r) = \frac{1}{n-1} \sum_{i=1}^n (g_{r_i}(\theta) - \bar{g}_n(\theta)) (g_{r_i}(\theta) - \bar{g}_n(\theta))^T$ [Scott, 2002]. 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, $p(\theta | \{x_1 \dots x_N\})$ approaches $\mathcal{N}(\theta_0, I_N^{-1})$ as N becomes very large.

6.2 SGLD with preconditioning

As discussed before, we would like to use the largest possible step size that does not incur too much bias so that we can mix fast and reduce variance quickly. For posterior distributions with different scales along different dimensions, the step size α is limited by the scale in the narrowest dimension. However, if we choose a very small step size, SGLD will take a long

time to traverse wider dimensions, which results in poor mixing and high variance. Thus we need a preconditioning matrix, e.g. like the Hessian used in optimization, to scale the step sizes along different dimensions. Welling and Teh [2011] proposed a preconditioned version of SGLD with the following update:

$$\theta_{t+1} = \theta_t + \frac{\alpha C}{2} \hat{\psi}_n(\theta_t) + \mathcal{N}(0, \alpha C) \quad (6.1)$$

where C is the preconditioning matrix. Note that C adjusts both the gradient and the injected Gaussian noise in the same way. Since, for large datasets, the posterior distribution approaches a Gaussian with covariance I_N^{-1} (the inverse Fisher information, see section 6.1), a reasonable choice for C is I_N^{-1} . This will allow SGLD to mix better by using step sizes that are properly adjusted to the width of the posterior along different dimensions. Nevertheless, if we increase the step size too much, the bias will increase as $O(\alpha)$.

However, it is easy to construct a Markov chain that samples correctly from the Gaussian approximation of the posterior even at large step sizes, as we show in the next section.

6.3 Sampling from the approximate posterior

As mentioned in section 6.1, the posterior distribution approaches $\mathcal{N}(\theta_0, I_N^{-1})$ as the size of the dataset becomes very large. We will now show that the following update equation can sample correctly from $\mathcal{N}(\theta_0, I_N^{-1})$ even when α is large:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\alpha C}{2} \{-I_N(\theta_t - \theta_0)\} + \mathcal{N}(0, \alpha C - \frac{\alpha^2}{4} C I_N C) \quad (6.2)$$

For the moment, let C in the above equation be an arbitrary matrix; we will define conditions on C shortly. The update in (6.2) is an affine transformation of θ_t plus injected independent Gaussian noise. Therefore, 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:

$$\begin{aligned}\mu_{t+1} &= (I - \frac{\alpha C}{2} I_N) \mu_t + \frac{\alpha C}{2} I_N \theta_0 \\ \Sigma_{t+1} &= (I - \frac{\alpha C}{2} I_N) \Sigma_t (I - \frac{\alpha C}{2} I_N)^T + \alpha C - \frac{\alpha^2}{4} C I_N C\end{aligned}\tag{6.3}$$

For the stationary distribution, it must hold that $\mu_{t+1} = \mu_t \triangleq \mu$ and $\Sigma_{t+1} = \Sigma_t \triangleq \Sigma$. Solving the above equations for μ and Σ , further assuming that C is 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 Eqn. (6.2) is not a Langevin equation, it can sample from the approximate posterior even at large step-sizes and does not require any MH accept/reject steps. The only requirements are that C should be symmetric and that α and/or C should be chosen so that the covariance matrix of the injected noise in Eqn. (6.2) is positive-definite.

6.4 The Stochastic Gradient Fisher Scoring Algorithm

Now, we will combine the methods in sections 6.2 and 6.3 to develop an algorithm we call Stochastic Gradient Fisher Scoring (SGFS). At high step sizes, SGFS can mix fast but will only capture a Gaussian approximation of the posterior. When the step size is reduced, it will behave like preconditioned SGLD and can capture the non-Gaussian details of the posterior. We will accomplish this using a Markov chain with the following update equation:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\alpha C}{2} \hat{\psi}_n(\theta_t) + \mathbb{N}(0, Q)\tag{6.4}$$

When the step size is small, we can choose $Q = \alpha C$ so that it behaves like the Markov chain in Eqn (6.1). We also want to choose Q in such a way that when the step size is large and the posterior is approximately Gaussian, our algorithm behaves like the Markov chain in

Eqn. (6.2). To do this, note that if n is large enough for the central limit theorem to hold, we have:

$$\hat{\psi}_n(\theta_t) \sim \mathcal{N}\left(\psi(\theta_t), \frac{N^2}{n} \text{Cov}_x[\nabla_\theta \log p(x|\theta_t)]\right) \quad (6.5)$$

Here $\text{Cov}_x[\nabla_\theta \log p(x|\theta_t)]$ is the covariance of the scores at θ_t . Using $N \text{Cov}_x[\nabla_\theta \log p(x|\theta_t)] \approx I_N$, we have:

$$\hat{\psi}_n(\theta_t) = \psi(\theta_t) + \mathbb{N}\left(0, \frac{NI_N}{n}\right) \quad (6.6)$$

Since, $\psi(\theta_t)$ is the gradient of the log posterior, if we assume that the posterior is close to its Bernstein-von Mises approximation, we have $\psi(\theta_t) \approx -I_N(\theta_t - \theta_0)$. Using this in Eqn. (6.6) and then substituting in Eqn. (6.4), we have:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\alpha C}{2} \{-I_N(\theta_t - \theta_0)\} + \mathbb{N}\left(0, \frac{\alpha^2}{4} \frac{N}{n} CI_N C\right) + \mathbb{N}(0, Q) \quad (6.7)$$

Comparing Eqn. (6.7) and Eqn. (6.2), we see that at high step sizes, we need:

$$\begin{aligned} Q + \frac{\alpha^2}{4} \frac{N}{n} CI_N C &= \alpha C - \frac{\alpha^2}{4} CI_N C \Rightarrow \\ Q &= \alpha C - \frac{\alpha^2}{4} \frac{N+n}{n} CI_N C \end{aligned} \quad (6.8)$$

Thus, we should choose Q such that:

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

where we have defined $\gamma = \frac{N+n}{n}$. Since α dominates α^2 when α is small, we can choose

$Q = \alpha C - \frac{\alpha^2}{4}\gamma C I_N C$ for both the cases above. With this, our update equation becomes:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\alpha C}{2} \hat{\psi}_n(\theta_t) + \mathbb{N}\left(0, \alpha C - \frac{\alpha^2}{4}\gamma C I_N C\right) \quad (6.9)$$

Now, we have to choose C so that the covariance matrix of the injected noise in Eqn. (6.9) is positive-definite. One way to enforce this, is by setting:

$$\alpha C - \frac{\alpha^2}{4}\gamma C I_N C = \alpha C B C \Rightarrow C = 4[\alpha\gamma I_N + 4B]^{-1} \quad (6.10)$$

where B is any symmetric positive-definite matrix. Plugging in this choice of C in Eqn. 6.9, we get:

$$\theta_{t+1} \leftarrow \theta_t + 2 \left[\gamma I_N + \frac{4B}{\alpha} \right]^{-1} \left\{ \hat{\psi}_n(\theta_t) + \eta \right\} \text{ where } \eta \sim \mathcal{N}\left(0, \frac{4B}{\alpha}\right) \quad (6.11)$$

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 :

$$\hat{I}_{1,t} = (1 - \kappa_t)\hat{I}_{1,t-1} + \kappa_t V(\theta_t; X_n^t) \quad (6.12)$$

where $\kappa_t = 1/t$. The online average converges to I_1 plus $\mathcal{O}(1/N)$ corrections if we assume that the samples are actually drawn from the posterior. This is summarized in the following theorem and proved in section A.2 of the appendix.

Theorem 3. *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 , $X_n^t = \{x_{t_1} \dots x_{t_n}\}$, and compute the empirical covariance of the scores $V(\theta_t; X_n^t) = \frac{1}{n-1} \sum_{i=1}^n \{g(\theta_t; x_{t_i}) - \bar{g}_n(\theta_t)\} \{g(\theta_t; x_{t_i}) - \bar{g}_n(\theta_t)\}^T$. Let V_T be the average of $V(\theta_t)$ across T iterations. For large N , as $T \rightarrow \infty$, V_T converges to the Fisher*

information $I(\theta_0)$ plus $\mathcal{O}(\frac{1}{N})$ corrections, i.e.

$$\lim_{T \rightarrow \infty} \left[V_T \triangleq \frac{1}{T} \sum_{t=1}^T V(\theta_t; X_n^t) \right] = I(\theta_0) + \mathcal{O}(\frac{1}{N}) \quad (6.13)$$

We would like to point out that this is not a proof of convergence of the Markov chain to the correct distribution. The result only states that if the samples are from the posterior, the online average converges to the Fisher information, i.e. we showed that sampling from the posterior and estimating the Fisher Information correctly is a fixed point of our algorithm, but we have not rigorously proved that we always reach this fixed point.

Algorithm 5 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  $X_n^t = \{x_{t_1} \dots x_{t_n}\}$ 
5:    $\bar{g}_n(\theta_t) \leftarrow \frac{1}{n} \sum_{i=1}^n g_{t_i}(\theta_t)$ 
6:    $V(\theta_t) \leftarrow \frac{1}{n-1} \sum_{i=1}^n \{g_{t_i}(\theta_t) - \bar{g}_n(\theta_t)\} \{g_{t_i}(\theta_t) - \bar{g}_n(\theta_t)\}^T$ 
7:    $\hat{I}_{1,t} \leftarrow (1 - \kappa_t) \hat{I}_{1,t-1} + \kappa_t V(\theta_t)$ 
8:   Draw  $\eta \sim \mathcal{N}[0, \frac{4B}{\alpha}]$ 
9:    $\theta_{t+1} \leftarrow \theta_t +$ 
        $2 \left( \gamma N \hat{I}_{1,t} + \frac{4B}{\alpha} \right)^{-1} \{ \nabla \log p(\theta_t) + N \bar{g}_n(\theta_t) + \eta \}$ 
10: end for
```

Also, note that estimating the Fisher Information online violates the Markov property of the chain since the update at iteration t depends on all previous samples through the online average. According to the theory of adaptive MCMC [Andrieu and Thoms, 2009] or two time scale stochastic approximations [Borkar, 1997], adapting the Markov chain using samples is valid as long as the rate of adaption is slow enough. We believe this to be true for SGFS, but a rigorous proof is lacking which we leave for future work. Pseudocode for SGFS is given in Algorithm 5.

Note that our algorithm still has a free parameter - the symmetric positive-definite matrix B . When $\alpha \rightarrow 0$, our algorithm turns into SGLD with preconditioning matrix B^{-1} . So, we recommend choosing $B \propto I_N$. With this choice, our update equation is very similar to that of Fisher scoring but with stochastic gradients. This is why we call our algorithm “Stochastic Gradient Fisher Scoring”.

6.4.1 Bias-Variance Trade-off in SGFS

During the burn-in period, SGFS behaves like noisy Fisher scoring and is an efficient optimizer that allows us to quickly reach a mode. At high step sizes/mixing rates, SGFS samples from a Gaussian approximation of the posterior whereas at low step sizes / mixing rates, it behaves like preconditioned SGLD and can sample more accurately. We recommend annealing the step size over time, so that we will first burn-in quickly to a mode and capture a rough Gaussian approximation of the posterior. Then, if more computational time becomes available, we can reduce the step size to capture the non-Gaussian details of the distribution.

6.4.2 Computational Efficiency

The main advantage of SGFS relative to traditional MCMC algorithms is that we use only a mini-batch of data instead of the entire dataset to generate each sample. However, for a model with a large number of parameters, inverting the $D \times D$ preconditioning matrix is still very expensive as it requires $O(D^3)$ computational time per iteration. If $n \ll D$, we can bring down the cost to $O(nD^2)$ using the Sherman-Morrison-Woodbury equation or by updating Cholesky factors as in Seeger [2004]. Another way of reducing the computational cost is by factoring the Fisher information into k independent blocks of d variables. In this case the computational complexity can be brought down to $O(kd^3)$. An extreme case is when the Fisher Information is approximated by a diagonal matrix (i.e. $k = D$ and $d = 1$)

in which case we perform only $O(D)$ computations. As k increases the bias also increases (especially when α is high), but we still expect the algorithm to perform better than SGLD which does not scale its step sizes at all. We use both the full covariance algorithm (SGFS-f) and the diagonal covariance algorithm (SGFS-d) in our experiments.

6.5 Experiments

We compare SGFS-f, SGFS-d, SGLD, SGD and HMC on three different models: logistic regression, neural networks and discriminative RBMs. In all experiments, stepsizes for SGD and SGLD are selected through cross-validation for at least five settings. The minibatch size n is set to ≈ 500 unless specified otherwise.

6.5.1 Logistic Regression

A logistic regression model (LR) with a zero mean Gaussian prior (with precision = 1) was trained on the MNIST dataset for binary classification of two digits 7 and 9. There were 10,000 datapoints and we reduced the dimensionality to 50 using random projections. We generated 100K samples from SGFS after discarding 3K burn-in samples. We used $B = \gamma I_N$ and tested the algorithm for a number of values of $\beta = \frac{2}{\sqrt{\alpha}}$. We compare the results to Hamilton Monte Carlo sampling [Neal, 1993] and to SGLD [Welling and Teh, 2011]. For HMC, we used a step size that gave an acceptance rate around 0.8 and for SGLD, we tested a range of different step sizes.

In figure ?? we show 2-d marginal distributions of SGFS compared to ground truth. The ground truth was computed from a long run of HMC. We used the largest possible step size for SGFS ($\beta = 0$). Even at this large step size, SGFS-f estimates the marginals almost perfectly while SGFS-d underestimates the variance (note that at lower step sizes SGFS-d

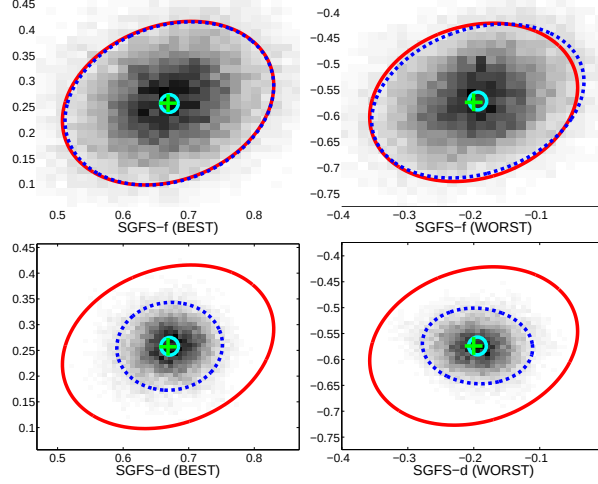


Figure 6.1: 2-d marginal posterior distributions for logistic regression. Grey colors correspond to samples from SGFS. Red solid and blue dotted ellipses represent iso-probability contours at two standard deviations away from the mean computed from HMC and SGFS, respectively. Top plots are the results 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.

becomes similar to SGLD and will sample correctly).

Next, we study the error after a fixed amount of computational time as a function of mixing rate. The error and mixing rate are averaged over 51 parameters. We define the mixing rate as the inverse of the autocorrelation time per unit of computation (ATUC)¹. We measure the relative error as follows: first we compute the mean and covariance of the parameter samples up to time t : $\bar{\theta}^t = \frac{1}{t} \sum_{t'=1}^t \theta_{t'}$ and $C^t = \frac{1}{t} \sum_{t'=1}^t (\theta_{t'} - \bar{\theta}^t)(\theta_{t'} - \bar{\theta}^t)^T$. We do the same for a long run of HMC which we denote as $\bar{\theta}^\infty$ and C^∞ . Finally we compute

$$E_{1t} = \frac{\sum_i |\bar{\theta}_i^t - \bar{\theta}_i^\infty|}{\sum_i |\bar{\theta}_i^\infty|}, \quad E_{2t} = \frac{\sum_{ij} |C_{ij}^t - C_{ij}^\infty|}{\sum_{ij} |C_{ij}^\infty|} \quad (6.14)$$

In Figure 6.2 we plot the “Error at time T” for two values of T (T=100 sec, T=3000 sec) as a function of the *inverse* ATUC. Top plots show the results for the mean and bottom plots the covariance. Each point denoted by a cross is obtained from a different setting

¹ATUC = Autocorrelation Time $\times$ Time per Sample.

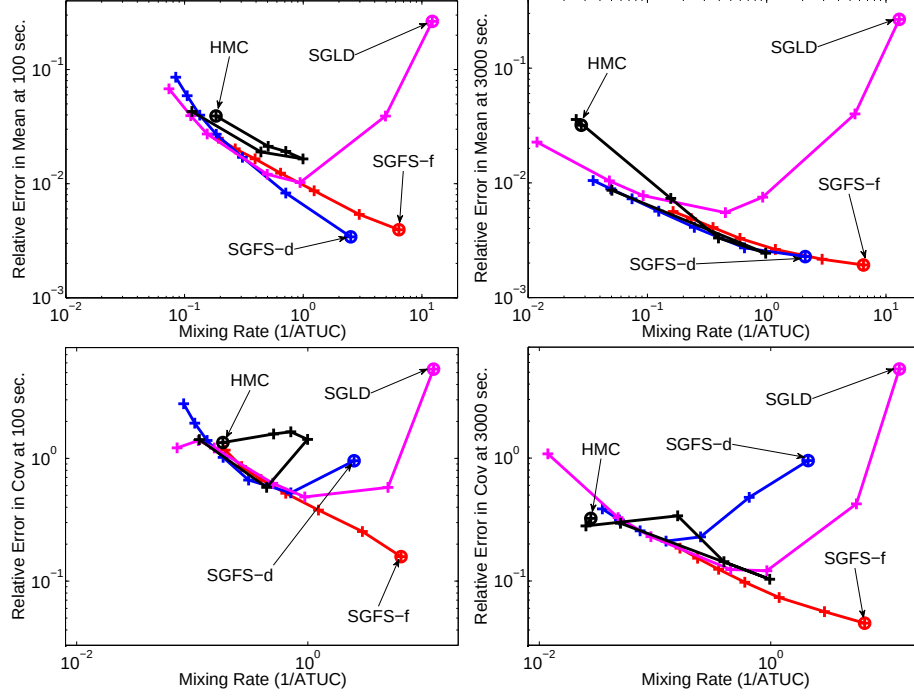


Figure 6.2: Final error of logistic regression at time T versus mixing rate for the mean (top) and covariance (bottom) estimates after 100 (left) and 3000 (right) seconds of computation.

of parameters that control the mixing rate: $\beta = [0, 1, 2, 3, 4, 5, 6]$ for SGFS, stepsizes $\alpha = [1e-3, 5e-4, 1e-4, 5e-5, 1e-5, 5e-6, 1e-6]$ for SGLD, and number of leapfrog steps $s = [50, 40, 30, 20, 10, 1]$ for HMC. The circle denotes the result for the fastest mixing chain.

For approximate algorithms like SGFS and SGLD, a negative slope indicates that the error is still dominated by the variance at time T . When the curve bends upwards, it indicates that the algorithm has reached its error floor for the higher mixing rates which means that the faster reduction in variance is dominated by the higher asymptotic bias. For the unbiased HMC algorithm, the curve bends back up on itself because when we increase the number of leapfrog steps too much, the trajectory becomes so long that the sampler ends up close to where it started after taking a U turn.

Notice that SGFS-f still has not reached the error floor even after 3000 secs of computation whereas SGFS-d reached its error floor only for the covariance estimate (but not for the mean). HMC does not perform very well because of the high cost of generating each sample.

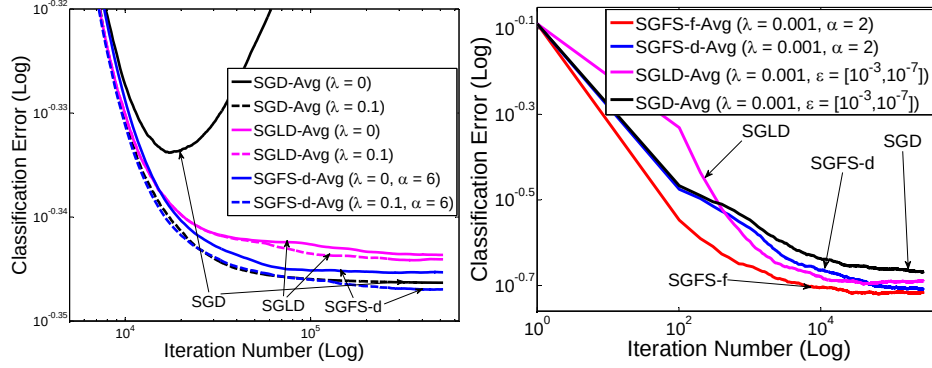


Figure 6.3: Test-set classification error of neural networks trained with SGFS-f, SGFS-d, SGLD and SGD on the HHP dataset (left) and the MNIST dataset (right)

For SGLD, we note the error becomes very high as we increase the mixing rate. This clearly shows the advantage of using SGFS over SGLD and HMC.

6.5.2 Neural Networks

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

Heritage Health Prize (HHP)

The goal of this competition is to predict the number of days (0-15) that a person will stay in hospital given his/her past three years of hospitalization records². 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 improve the results. Although we used $\beta = 6$ for SGFS-d, there was no significant difference for values in the range $3 \leq \beta \leq 6$. However, $\beta < 3$ did not work for this dataset due to the fact that many

²<http://www.heritagehealthprize.com>

features had a lot of 0 values.

For SGD, we used stepsizes with a polynomial annealing schedule $a(b + t)^{-\delta}$. Because the training error decreased slowly in the usual $\delta = [0.5, 1]$ range, 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 6.3 (left) shows the 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 finely 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 a 20 dimensional random projection and used 30 hidden units resulting in a total of 940 parameters. 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 6.3 (right). Here, we used a small regularization parameter of $\lambda = 0.001$ for all methods as overfitting was not an issue. For SGFS, $\beta = 2$ is used while for both SGD and SGLD the stepsizes were annealed from 10^{-3} to 10^{-7} using $a = 1$, $b = 1000$, and $\gamma = 1$.

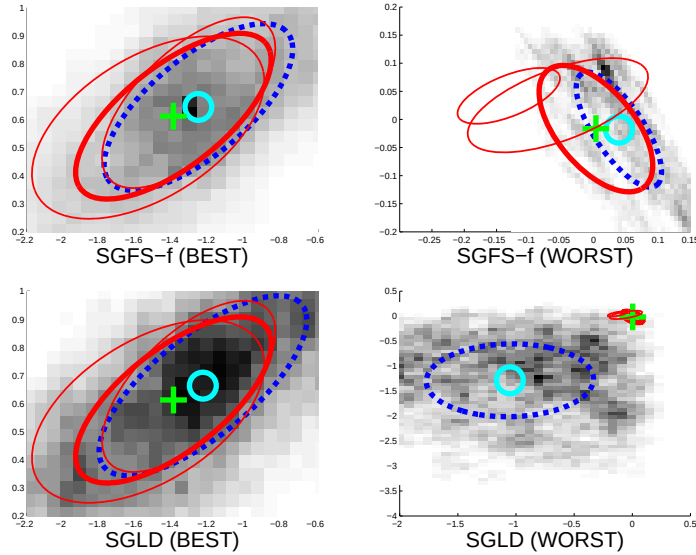


Figure 6.4: 2-d marginal posterior distributions of DRBM. Grey colors correspond to samples from SGFS/SGLD. Thick red solid lines correspond to iso-probability contours at two standard deviations away from the mean computed from HMC samples. Thin red solid lines correspond to HMC results based on subsets 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.

6.5.3 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. The task is to distinguish network intrusions or attacks from normal connections.

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% 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 values of β 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 a β 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

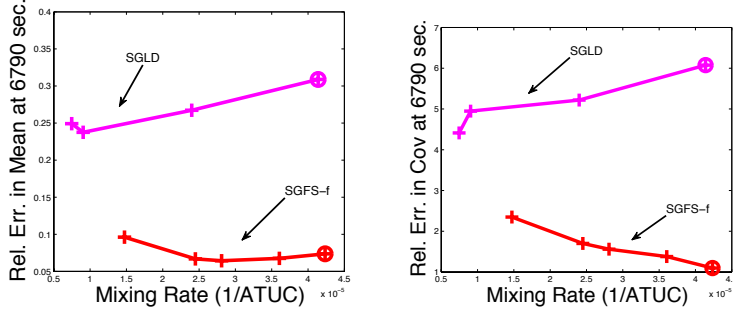


Figure 6.5: Final error for DRBM at time T versus mixing rate for the mean (left) and covariance (right) estimates after 6790 seconds of computation on a subset of KDD99.

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

Table 6.1: Final test error rate on the KDD99 dataset.

iterations. Although we experimented with different burn-in iterations, the algorithms were relatively insensitive to this choice. The final error rates are given in table 6.1 from which we conclude that the samplers based on stochastic gradients can act as effective regularizers whereas HMC 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 that of 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 $\beta = [2, 3, 4, 5, 10]$ and SGLD with fixed step sizes of $[5\text{e-}4, 1\text{e-}4, 5\text{e-}5, 1\text{e-}5, 5\text{e-}6]$. 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 6.4 (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 6.4 (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 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 6.5-left shows the results for the mean and the right for the covariance (for an explanation of the various quantities see discussion in section 6.5.1). We note again that SGLD incurs a significantly larger approximation bias at the same mixing rate as SGFS-f.

6.6 Related Work

Various approximate MCMC algorithms based on stochastic gradients have since been proposed. Patterson and Teh [2013] developed Stochastic Gradient Reimannian Langevin Dynamics (SGRLD) which uses a position specific preconditioning matrix (instead of a global one as in SGFS) to make updates in the direction of the natural gradient. Chen et al. [2014] developed the Stochastic Gradient Hamiltonian Monte Carlo algorithm which can travel large distances in parameter space avoiding random walks. Ding et al. [2014] proposed the Stochastic Gradient Nose-Hoover Thermostats (SGNHT) algorithm to stabilize momentum fluctuations caused by sub-sampling noise in the gradient. Ahn et al. [2014] developed a general framework to further speed up stochastic gradient MCMC algorithms using distributed computing.

Chapter 7

Approximate Bayesian Probabilistic Matrix Factorization

In this chapter, we show how to scale up inference in the Bayesian Probabilistic Matrix Factorization model (BPMF) [Salakhutdinov and Mnih, 2008] using SGLD and the approximate MH algorithm.

This section contains preliminary results from a project to develop a distributed algorithm for Bayesian Probabilistic Matrix factorization. This is joint work with Sungjin Ahn, Max Welling and researchers at Yahoo! labs. Max Welling, Sungjin Ahn and I designed the algorithm. I performed the experiments reported in this chapter.

7.1 Model

Consider a recommender system with P users and Q items. Each user has rated only a subset of the Q items and the goal is to predict the unknown ratings. We denote the rating for the j^{th} item by the i^{th} user as R_{ij} . We also use an indicator I_{ij} which is equal to 1 if the

i^{th} user rated the j^{th} item and 0 otherwise. Usually P and Q are very large but the number of ratings $N \ll PQ$, so we store R using a sparse representation as {user, item, rating} tuples rather than a $P \times Q$ rating matrix.

A matrix factorization model for predicting ratings assumes that the preference of a user towards an item can be modeled as a combination of a small number of user and item factors. We denote the user and item factors as $U = \{u_i\}_{i=1}^P$ and $V = \{v_j\}_{j=1}^Q$ respectively where $u_i, v_j \in \mathbb{R}^D$. Assuming Gaussian errors in the predictions, we model the ratings as $R_{ij} \sim \mathcal{N}(u_i^T v_j, \sigma^2)$. We can write the likelihood of the ratings as:

$$p(R|U, V, \sigma^2) = \prod_{i=1}^P \prod_{j=1}^Q [\mathcal{N}(R_{ij}|u_i^T v_j, \sigma^2)]^{I_{ij}} \quad (7.1)$$

We place Gaussian priors on the parameters:

$$\begin{aligned} p(U|\mu_U, \Lambda_U) &= \prod_{i=1}^P \mathcal{N}(u_i|\mu_U, \Lambda_U^{-1}) \\ p(V|\mu_V, \Lambda_V) &= \prod_{j=1}^Q \mathcal{N}(v_j|\mu_V, \Lambda_V^{-1}) \end{aligned} \quad (7.2)$$

We further place priors on the hyper-parameters $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$:

$$\begin{aligned} p(\Theta_U|\Theta_0) &= p(\mu_U|\Lambda_U)p(\Lambda_U) = \mathcal{N}(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0) \\ p(\Theta_V|\Theta_0) &= p(\mu_V|\Lambda_V)p(\Lambda_V) = \mathcal{N}(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_V|W_0, \nu_0) \end{aligned} \quad (7.3)$$

where $\mathcal{W}$ is the Wishart distribution¹ and $\Theta_0 = \{\mu_0, \beta_0, \nu_0, W_0\}$. Predictions for unknown ratings are computed by integrating over both parameters and hyper-parameters:

$$p(R_{ij}^*|R, \Theta_0) = \int \int p(R_{ij}^*|u_i, v_j)p(U, V|R, \Theta_U, \Theta_V)p(\Theta_U, \Theta_V|\Theta_0)d\{U, V\}d\{\Theta_U, \Theta_V\} \quad (7.4)$$

¹The Wishart density is $\mathcal{W}(\Lambda|W_0, \nu_0) \propto |\Lambda|^{(\nu_0-D-1)/2} \exp\{-\frac{1}{2}\text{Tr}(W_0^{-1}\Lambda)\}$

Since this integration is analytically intractable, we approximate it using MCMC methods as:

$$R_{ij}^* \approx \frac{1}{T} \sum_{t=1}^T u_{i,t}^T v_{j,t} \quad (7.5)$$

where the samples $\{u_{i,t}, v_{j,t}\}_{t=1}^T$ are generated from a Markov Chain whose equilibrium distribution is the posterior distribution over model parameters and hyper-parameters $\{U, V, \Theta_U, \Theta_V\}$.

7.2 Algorithms

7.2.1 Gibbs Sampling

Salakhutdinov and Mnih [2008] proposed a Gibbs sampling algorithm for inference in this model. Since we use conjugate priors for the parameters and hyper-parameters, the required conditional distributions can be computed in closed form. Below, we show the conditionals of the user parameters and hyper-parameters; corresponding distributions for the items are similar.

The conditional distributions of the parameters are:

$$p(u_i | R, V, \Theta_U, \sigma^2) = \mathcal{N}(u_i | \mu_i^*, [\Lambda_i^*]^{-1}) \quad (7.6)$$

where

$$\begin{aligned} \Lambda_i^* &= \Lambda_U + \frac{1}{\sigma^2} \sum_{j=1}^Q I_{ij} [V_j V_j^T] \\ \mu_i^* &= [\Lambda_i^*]^{-1} \left(\frac{1}{\sigma^2} \sum_{j=1}^Q I_{ij} [V_j R_{ij}] + \Lambda_U \mu_U \right) \end{aligned} \quad (7.7)$$

The conditional distribution of the hyper-parameters is:

$$p(\mu_U, \Lambda_U | U, \Theta_0) = \mathcal{N}(\mu_U | \mu_0^*, (\beta_0^* \Lambda_U)^{-1}) \mathcal{W}(\Lambda_U | W_0^*, \nu_0^*) \quad (7.8)$$

where

$$\begin{aligned} \mu_0^* &= \frac{\beta_0 \mu_0 + P \bar{U}}{\beta_0 + P}, & \beta_0^* &= \beta_0 + P, & \nu_0^* &= \nu_0 + P \\ [W_0^*]^{-1} &= W_0^{-1} + P \bar{S} + \frac{\beta_0 P}{\beta_0 + P} (\mu_0 - \bar{U})(\mu_0 - \bar{U})^T \\ \bar{U} &= \frac{1}{P} \sum_{i=1}^P u_i, & \bar{S} &= \frac{1}{P} \sum_{i=1}^P u_i u_i^T \end{aligned} \quad (7.9)$$

Computing the parameters in eqn. 7.9 requires $O(P)$ or $O(Q)$ time which is very expensive when we have billions of users or items. Also, computing Λ_i^* and μ_i^* for each user parameter naively as written in Eqn. 7.7 requires $O(Q)$ time and since there are P users, the total computational time is $O(PQ)$. Similarly, updating item parameters will incur $O(PQ)$ time. This is prohibitively expensive.

We can easily cut down the $O(PQ)$ computation using look-up tables that 1) map each user to a list of items that (s)he rated and the associated ratings and 2) map each item to a list of users that rated it and the associated ratings. This allows us to compute Λ_i^* and μ_i^* for all users and items in $O(N)$ amortized time, which is still very expensive. Also, storing two look-up tables requires twice the amount of memory required to store the dataset, which may be difficult considering that the original dataset is already very large.

7.2.2 SGLD and Approximate Metropolis-Hastings

To be able to handle datasets of arbitrary size, we need to make parameter updates that touch only a subset of ratings, users or items in each iteration. We will now develop an algorithm

that can do this by using SGLD to update the user/item parameters and approximate MH to update the hyper-parameters. We will adopt a slightly different notation in this section: since the dataset is usually stored using a sparse representation, we will denote it as $\mathcal{D} = \{i_n, j_n, r_n\}_{n=1}^N$ where $\{i_n, j_n, r_n\}$ is a {user, item, rating} tuple and we predict $r_n \approx u_{i_n}^T v_{j_n}$.

To update parameters u_p (or v_q) using SGLD, we need an unbiased estimate of the gradient of the log posterior that can be computed cheaply from only a few ratings. One way to write the true gradient of the log posterior with respect to u_p is:

$$g(\mathcal{D}) = \frac{1}{\sigma^2} \sum_{n=1}^N \mathbb{I}[i_n = p] (r_n - u_{i_n}^T v_{j_n}) v_{j_n} - \Lambda_U^{-1}(u_p - \mu_U) \quad (7.10)$$

We can compute an unbiased estimate of this gradient using a mini-batch $\mathcal{M}$ of $m \ll N$ ratings as:

$$g_1(\mathcal{M}) = \frac{N}{m} \frac{1}{\sigma^2} \sum_{n=1}^m \mathbb{I}[i_n = p] (r_n - u_{i_n}^T v_{j_n}) v_{j_n} - \Lambda_U^{-1}(u_p - \mu_U) \quad (7.11)$$

However, the gradient of the log prior in the above is non-zero even for users that do not have any ratings in the mini-batch. Thus, we have to update all user and item parameters in every iteration, which is very expensive and not a good use of computational time considering that the gradient of the log-likelihood will be zero for most parameters. Therefore, we can try to use the same gradient, but update only those user parameters that are in the mini-batch. In this case, the gradient is:

$$g_2(\mathcal{M}) = \frac{N}{m} \frac{1}{\sigma^2} \sum_{n=1}^m \mathbb{I}[i_n = p] (r_n - u_{i_n}^T v_{j_n}) v_{j_n} - \mathbb{I}[\exists n | i_n = p] \Lambda_U^{-1}(u_p - \mu_U) \quad (7.12)$$

where $\mathbb{I}[\exists n | i_n = p]$ is an indicator function that equals 1 if the mini-batch has at least one rating for user p . However, this is not an unbiased estimator of the true gradient (Eqn. 7.10) as the prior gradient now depends on the mini-batch and its expectation gets implicitly

scaled by γ_p , the probability of picking a mini-batch with at least one rating for user p . We can correct for this effect by scaling the prior gradient by a factor of $1/\gamma_p$. Thus, we have:

$$g_3(\mathcal{M}) = \frac{N}{m} \frac{1}{\sigma^2} \sum_{n=1}^m \mathbb{I}[i_n = p] (r_n - u_{i_n}^T v_{j_n}) v_{j_n} - \frac{1}{\gamma_p} \mathbb{I}[\exists n | i_n = p] \Lambda_U^{-1} (u_p - \mu_U) \quad (7.13)$$

It is easy to show that the expectation of g_3 over all mini-batches is equal to the true gradient, i.e. $\mathbb{E}_{\mathcal{M}}[g_3(\mathcal{M})] = g(\mathcal{D})$, since the expectation of the indicator function $\mathbb{E}_{\mathcal{M}} \{\mathbb{I}[\exists n | i_n = p]\} = \gamma_p$. If the data items in the mini-batch are drawn with replacement from the complete dataset, we can calculate γ_p as ‘1 - probability that the mini-batch does not contain any ratings for user p ’, i.e.

$$\gamma_p = 1 - \left(1 - \frac{n_p}{N}\right)^m \quad (7.14)$$

where n_p is the total number of ratings for user p . Using the gradient g_3 , we can obtain an efficient SGLD algorithm where we process only m ratings per iteration and update only (at most m) user/item parameters that have ratings in the mini-batch.

We will update hyper-parameters using the approximate MH algorithm. For μ_U and μ_V , we use a normal distribution centered around the current value of μ_U and μ_V as a proposal². For updating Λ_U and Λ_V , we propose using a Wishart distribution whose mode is equal to the current value of Λ_U and Λ_V . Since the ‘likelihood’ of the hyper-parameters depends on all the user / item parameters, it becomes very expensive to conduct the exact MH test when we have billions of users or items. Thus, we have to use our approximate MH algorithm which makes each accept/reject decision using only a subset of user / item parameters.

²Note that we can also use SGLD for updating μ_U and μ_V . However, updating Λ_U and Λ_V using SGLD is hard because of the positive semi-definiteness constraints.

7.3 Results

We test both algorithms on the Yahoo! Music Ratings dataset³ which consists of 717,872,016 ratings of 136,736 songs by 1,823,179 users. We use the official split with 699 million ratings in the training set and the remaining 18 million in the test set. The ratings are integers from 1 to 5.

For all algorithms, we set $D = 30$, $\beta_0 = 2$, $W_0 = I$, $\mu_0 = 0$, $\nu_0 = D$ and $\sigma^2 = 0.5$. It might be useful to place a prior on σ^2 and sample from the posterior over σ^2 . However, we chose not to do this because σ^2 depends on all ratings which makes it very difficult for exact algorithms such as Gibbs sampling. Also, on other datasets such as MovieLens and Netflix, sampling over σ^2 only resulted in a marginal improvement in the RMSE (root mean squared error). We compare 3 algorithms in figure 7.1:

1. SGLD for updating parameters and approximate MH with $\epsilon = 0.1$ for updating hyper-parameters (denoted as SGLDMH($\epsilon = 0.1$) in figure).
2. SGLD for updating parameters and exact MH for updating hyper-parameters (denoted as SGLDMH($\epsilon = 0$) in figure).
3. Gibbs Sampling for updating all parameters and hyper-parameters.

In the SGLDMH algorithms, we use a step size $\alpha = 1 \times 10^{-6}$ and mini-batch size $m = 100,000$ for SGLD. We thin the samples by using only every 200th sample for posterior averaging. For SGLDMH($\epsilon = 0.1$), we discard the first 50K samples as burn-in. We discard only the first 10K samples for SGLDMH($\epsilon = 0$), as the algorithm was much slower and could not generate 50K samples in a day. For Gibbs sampling, we did not discard any samples at all (for burn-in and thinning) as it was so slow that it generated only about 13 samples a day.

³R2 - Yahoo! Music User Ratings of Songs with Artist, Album, and Genre Meta Information, v. 1.0, <http://webscope.sandbox.yahoo.com/>

We initialized all algorithms by drawing U and V from a zero mean normal distribution with standard deviation 0.0001. We clipped all predictions $u_i^T v_j$ to the range $(1, 5)$ before averaging.

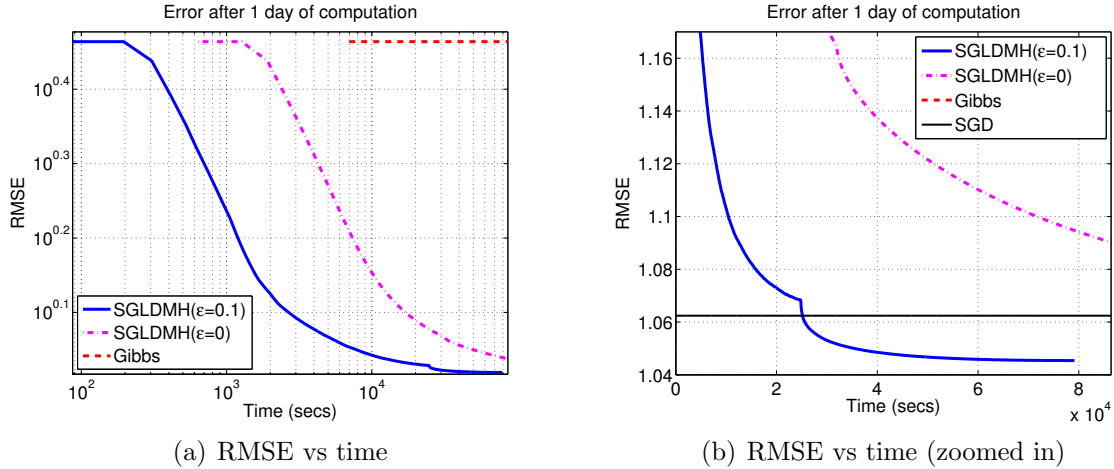


Figure 7.1: Error of different sampling algorithms on the BPMF model trained on the Yahoo Music Dataset as a function of computational time. The plot on the right is a zoomed version (and in linear scale) to show the improvement relative to SGD.

We plot the RMSE in the predictions vs computational time in seconds in Figure 7.1. In figure 7.1(a), we see that Gibbs sampling did not decrease the error at all. This is because the algorithm is extremely slow and could generate only 13 samples in a day. On the other hand SGLDMH($\epsilon = 0.1$) generated $\approx 160K$ samples and SGLDMH($\epsilon = 0$) generated $\approx 27K$ samples in a day. Thus, SGLDMH($\epsilon = 0.1$) was able to outperform Gibbs sampling and SGLDMH($\epsilon = 0$) given a computational budget of 1 day. Given a lot more computational time, we expect the other two algorithms to outperform SGLDMH($\epsilon = 0.1$) because of their lower asymptotic bias. However, note that Gibbs sampling is not very general as it cannot be used in models where the conditionals are intractable (e.g. models with non-conjugate priors).

In figure 7.1(b), we show that SGLDMH($\epsilon = 0.1$) outperforms an SGD algorithm for MAP estimation with fixed hyper-parameters. The SGD algorithm was also run with a step-size of $\alpha = 1 \times 10^{-6}$ and $m = 100,000$. SGLDMH achieved an error of 1.0454 compared to 1.0624

for SGD, an improvement of 1.6%. We believe an improvement of 1% is very significant, but there is not much published work on this dataset. On the Netflix dataset, BPMF using Gibbs sampling obtained a comparable improvement of 1.73% over SGD with $D = 30$ latent factors [Salakhutdinov and Mnih, 2008]. The only published work we found on the 700 million version of the Yahoo dataset was Schelter et al. [2012], which reported a Mean Average Error (MAE) of ≈ 0.86 using a nearest neighbour method in a map-reduce framework. The authors do not define what Mean Average Error is, but we believe they are referring to the Mean Absolute Error, a commonly used metric in the recommender systems literature. The Mean Absolute Error of SGLDMH($\epsilon = 0.1$) was 0.77. An RMSE of 1.074 and an MAE of 0.8144 was reported at <http://mymedialite.net/examples/datasets.html> using the MediaLite toolbox [Gantner et al., 2011]. Thus, SGLDMH($\epsilon = 0.1$) can obtain competitive results even though it processes only a mini-batch of data per iteration.

Algorithm	RMSE
Avg. Rating	1.5833
Avg. Item Rating	1.4974
Avg. User Rating	1.2942
SGD	1.0624
SGLDMH($\epsilon = 0.1$)	1.0454

Table 7.1: RMSE of predictions on the Yahoo Music Ratings dataset.

We also compare to other simple baselines in table 7.1. ‘Avg. Rating’ always predicts the average of all ratings in the dataset, without regard to the target user or item (i.e. no personalization). ‘Avg. Item Rating’ refers to predicting the average rating of each item irrespective of the query user and ‘Avg. User Rating’ refers to predicting the average rating of each user irrespective of the query item. We note that it is quite easy to make reasonable predictions, but extremely hard to improve upon them.

Chapter 8

Conclusions and Future Work

Bayesian modeling offers many advantages over optimization based approaches. For instance, 1) Bayesian models provide a natural way to incorporate prior knowledge, 2) posterior averaging acts as an effective regularizer to control over-fitting, 3) posterior/predictive distributions provide principled estimates of uncertainty and 4) hierarchical modeling allows us to use data-driven priors in multi-task and one-shot learning settings.

Despite these attractive qualities, Bayesian models are getting ‘left behind’ because traditional algorithms for Bayesian inference such as MCMC require processing the entire dataset in each iteration and therefore cannot scale up to modern large scale datasets. In contrast, optimization based approaches can handle arbitrarily large datasets using efficient algorithms like Stochastic Gradient Descent (SGD) which process only a mini-batch of data per iteration. However, by restricting ourselves to such simple methods we lose out on all the advantages of Bayesian modeling.

This thesis is an attempt at scaling up Bayesian inference to large datasets by developing a set of approximate MCMC algorithms that process only a mini-batch of data to generate each posterior sample. The main contributions of this thesis are:

1. We developed an approximate Metropolis-Hastings(MH) algorithm that uses a sequential hypothesis test to make accept/reject decisions using only an adaptive mini-batch of data. Since the MH test is an integral part of most other MCMC algorithms, scaling up the MH test is very important.
2. We developed an approximate slice sampling algorithm using the same sequential hypothesis testing mechanism. The slice sampling algorithm can mix faster than random walk MH by matching the width of the proposal to that of the posterior distribution.
3. We presented an error analysis of the Stochastic Gradient Langevin Dynamics(SGLD) algorithm. We also showed how SGLD can be combined with our approximate MH test to reduce the bias when sampling from certain distributions.
4. We developed an algorithm called Stochastic Gradient Fisher Scoring which improves the mixing rate of SGLD using a preconditioning matrix that captures the curvature of the target distribution.

Although these algorithms are asymptotically biased, we showed that they can produce estimates with lower risk than traditional MCMC algorithms due to the fact that approximate MCMC algorithms can generate a larger number of samples in a given amount of computational time and reduce variance faster. Below, we discuss some open questions and interesting avenues for future research.

A more rigorous error analysis and convergence guarantees for some of these algorithms are still lacking. Applications to more challenging models such as deep neural networks could be very illuminating and open up interesting research problems. Improving the mixing rate in high dimensions and further increasing the speed through distributed computing (e.g. as in Ahn et al. [2014]) can make approximate MCMC algorithms even better at handling industry scale datasets.

Balancing bias and variance to minimize the risk is an important research problem. We

saw that when we only have a small amount of computational time, it is better to use an algorithm with relatively high bias so that we can generate a large number of samples and control the variance. On the other hand, when we have lots of computational time we should use an algorithm with low bias as we have enough time to generate a large number of samples and reduce variance. The optimal strategy will involve an annealing schedule to decrease bias over time so that at any point the risk is as low as possible. However, finding this optimal annealing schedule could be very challenging. Some preliminary research in this direction can be found in Pillai and Smith [2014].

Although approximate MCMC algorithms significantly reduce the time it takes to generate samples from the posterior, another important limitation of MCMC algorithms must be addressed before they can be widely adopted in real world applications: at test time, MCMC algorithms require averaging predictions from a large number of posterior samples which is expensive both computationally as well as in terms of the space required to store these samples. This limits their use in a) devices like smartphones or medical devices where memory and power are at a premium b) applications such as self driving cars and robot control where response time is critical and c) applications where there is simply a lot of test data (e.g. Google, Facebook). This difficulty also arises with other ensemble methods such as bagging/boosting. Bucilua et al. [2006] proposed a way to address this using an approach called ‘Model Compression’ where they train a flexible non-ensemble model such as a neural network to mimic the predictions of the ensemble model. This idea can also be used to improve the test time performance of Bayesian MCMC inference. The final compressed model in the form of a neural network is much cheaper to store than a large number of posterior samples and can also make predictions much faster than posterior averaging at test time.

Bibliography

- S. Ahn, A. Korattikara, and M. Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. In *International Conference on Machine Learning*, 2012.
- S. Ahn, B. Shahbaba, and M. Welling. Distributed stochastic gradient MCMC. In *International Conference on Machine Learning*, 2014.
- P. Alquier, N. Friel, R. Everitt, and A. Boland. Noisy Monte Carlo: Convergence of Markov chains with approximate transition kernels. *arXiv preprint arXiv:1403.5496*, 2014.
- S.-i. Amari, A. Cichocki, H. H. Yang, et al. A new learning algorithm for blind signal separation. *Advances in Neural Information Processing Systems*, pages 757–763, 1996.
- C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009.
- C. Andrieu and J. Thoms. A tutorial on adaptive MCMC. *Statistics and Computing*, 18(4):343–373, 2009.
- K. Bache and M. Lichman. UCI machine learning repository, 2013. URL <http://archive.ics.uci.edu/ml>.
- R. Bardenet, A. Doucet, and C. Holmes. Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach. In *Proceedings of the 31st International Conference on Machine Learning*, pages 405–413, 2014.
- Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, pages 153–160, 2007.
- V. Borkar. Stochastic approximation with two time scales. *Systems and Control Letters*, 29(5):291–294, 1997.
- C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541. ACM, 2006.
- T. Chen, E. B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *International Conference on Machine Learning*, 2014.

- X. Chen, Z. Jane Wang, and M. J. McKeown. A Bayesian Lasso via reversible-jump MCMC. *Signal Processing*, 91(8):1920–1932, 2011.
- W. Coffey, Y. Kalmykov, and J. Waldron. *The Langevin equation: with applications to stochastic problems in physics, chemistry, and electrical engineering*, volume 14. World Scientific Publishing Company Incorporated, 2004.
- N. Ding, Y. Fang, R. Babbush, C. Chen, R. D. Skeel, and H. Neven. Bayesian sampling using stochastic gradient thermostats. *Neural Information Processing Systems*, 2014.
- C. DuBois, A. Korattikara, M. Welling, and P. Smyth. Approximate slice sampling for Bayesian posterior inference. In *Artificial Intelligence and Statistics*, 2014.
- P. Fearnhead, O. Papaspiliopoulos, and G. O. Roberts. Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(4):755–777, 2008.
- Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, pages 305–308. ACM, 2011.
- S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.
- P. J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995.
- W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.
- G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.
- G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.
- A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4):411–430, 2000.
- M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.
- A. Korattikara, Y. Chen, and M. Welling. Austerity in MCMC land: Cutting the Metropolis-Hastings budget. In *International Conference on Machine Learning*, 2014.

- A. S. Kronfeld. Dynamics of Langevin simulations. *Progress of Theoretical Physics Supplement*, 111:293–311, 1993.
- H. Larochelle and Y. Bengio. Classification using discriminative Restricted Boltzmann Machines. In *Proceedings of the 25th International Conference on Machine learning*, pages 536–543. ACM, 2008.
- L. Le Cam. *Asymptotic methods in statistical decision theory*. Springer, 1986.
- L. Lin, K. Liu, and J. Sloan. A noisy Monte Carlo algorithm. *Physical Review D*, 61(7):074505, 2000.
- J. Lu. *Multivariate slice sampling*. PhD thesis, Drexel University, 2008.
- D. Maclaurin and R. P. Adams. Firefly Monte Carlo: Exact MCMC with subsets of data. In *Uncertainty in Artificial Intelligence*, 2014.
- N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087, 1953.
- I. Murray, R. P. Adams, and D. J. MacKay. Elliptical slice sampling. *arXiv preprint arXiv:1001.0175*, 2009.
- R. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.
- R. M. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical report, University of Toronto, 1993.
- R. M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.
- R. M. Neal. Slice sampling. *Annals of Statistics*, pages 705–741, 2003.
- R. Nishihara, I. Murray, and R. P. Adams. Parallel MCMC with generalized elliptical slice sampling. *arXiv preprint arXiv:1210.7477*, 2012.
- Z. Ouyang. *Bayesian Additive Regression Kernels*. PhD thesis, Duke University, 2008.
- S. Patterson and Y. W. Teh. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, pages 3102–3110, 2013.
- N. S. Pillai and A. Smith. Ergodicity of approximate MCMC chains with applications to large data sets. *arXiv preprint arXiv:1405.0182*, 2014.
- H. Risken. *Fokker-Planck Equation*. Springer, 1984.
- H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

- G. O. Roberts and R. L. Tweedie. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, pages 341–363, 1996.
- R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using Markov Chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*, pages 880–887. ACM, 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.
- S. Schelter, C. Boden, and V. Markl. Scalable similarity-based neighborhood methods with MapReduce. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, pages 163–170. ACM, 2012.
- 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>.
- S. Singh, M. Wick, and A. McCallum. Monte Carlo MCMC: efficient inference by approximate sampling. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1104–1113. Association for Computational Linguistics, 2012.
- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- M. B. Thompson and R. M. Neal. Slice sampling with adaptive multivariate steps: The shrinking-rank method. *arXiv preprint arXiv:1011.4722*, 2010.
- L. Tierney. Markov Chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.
- L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.
- M. Welling and Y. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 681–688, 2011.

Appendix A

Proofs

A.1 Proof of Theorem 2

We provide a more detailed proof of Theorem 2 about the stationary distribution of SGLD from Chapter 5 below.

Theorem. *The stationary distribution of SGLD is:*

$$S_\epsilon(\theta) \propto \exp \left[-E(\theta) - \frac{\alpha}{4} \left\{ \nabla^2 E(\theta) - \frac{1}{2}(\nabla E(\theta))^2 + \frac{N^2}{n}(V(\theta) - \int_\theta V \nabla E) \right\} + \mathcal{O}(\alpha^2) \right]$$

Proof. Including moment terms upto $\mathcal{O}(\alpha^2)$ from the MGF (eqn. 5.10) in the Kramers-

Moyal equation, we get :

$$\begin{aligned}
& \nabla \left[S_\epsilon \frac{\alpha \nabla E}{2} \right] + \frac{1}{2!} \nabla^2 \left[S_\epsilon \left\{ \alpha + \frac{\alpha^2}{4} (\nabla E)^2 + \frac{\alpha^2 N^2}{4n} V \right\} \right] \\
& + \frac{1}{3!} \nabla^3 \left[S_\epsilon \left\{ \frac{3\alpha^2 \nabla E}{2} \right\} \right] + \frac{1}{4!} \nabla^4 [3\alpha^2 S_\epsilon] = 0 \\
& \implies \nabla [(\nabla E + \nabla) S_\epsilon] \\
& + \frac{\alpha}{4} \left[\nabla^2 \left\{ S_\epsilon \left((\nabla E)^2 + \frac{N^2 V}{n} \right) \right\} + 2\nabla^3 \{S_\epsilon \nabla E\} + \nabla^4 S_\epsilon \right] = 0
\end{aligned} \tag{A.1}$$

To solve the above equation, we repeatedly use $\nabla S_\epsilon = -S_\epsilon \nabla E + \mathcal{O}(\alpha)$ (from Lemma 1) inside the square brackets to remove higher derivatives of S_ϵ . Thus, the term in square brackets can be written as:

$$\begin{aligned}
& \nabla^2 \left\{ S_\epsilon \left((\nabla E)^2 + \frac{N^2 V}{n} \right) \right\} + 2\nabla^3 \{S_\epsilon \nabla E\} + \nabla^4 S_\epsilon \\
& = \nabla^2 \left\{ S_\epsilon \left((\nabla E)^2 + \frac{N^2 V}{n} \right) \right\} + 2\nabla^3 \{S_\epsilon \nabla E\} + \nabla^3 (-S_\epsilon \nabla E) \\
& = \nabla^2 \left\{ S_\epsilon \left((\nabla E)^2 + \frac{N^2 V}{n} \right) \right\} + \nabla^3 \{S_\epsilon \nabla E\} \\
& = \nabla^2 \left\{ S_\epsilon \left((\nabla E)^2 + \frac{N^2 V}{n} \right) \right\} + \nabla^2 \{-S_\epsilon (\nabla E)^2 + S_\epsilon (\nabla^2 E)\} \\
& = \nabla^2 \left\{ S_\epsilon \left(\nabla^2 E + \frac{N^2 V}{n} \right) \right\} \\
& = \nabla \left\{ S_\epsilon \left(\nabla^3 E + \frac{N^2 \nabla V}{n} - \nabla E \nabla^2 E - \frac{N^2}{n} V \nabla E \right) \right\}
\end{aligned} \tag{A.2}$$

Plugging the above expression back in Eqn. (A.1), we get:

$$\nabla [(\nabla \bar{E} + \nabla) S_\epsilon] = 0 \quad \text{where} \quad \nabla \bar{E} = \nabla E + \frac{\alpha}{4} \left\{ \nabla^3 E - \nabla E \nabla^2 E + \frac{N^2}{n} (\nabla V - V \nabla E) \right\} \tag{A.3}$$

By analogy to Lemma 1, the solution to the above equation is $S_\epsilon \propto \exp \{-\bar{E}(\theta) + \mathcal{O}(\alpha^2)\}$.

We can compute $\bar{E}(\theta)$ as:

$$\begin{aligned}\bar{E}(\theta) &= \int \left[\nabla E + \frac{\alpha}{4} \left\{ \nabla^3 E - \nabla E \nabla^2 E + \frac{N^2}{n} (\nabla V - V \nabla E) \right\} \right] d\theta \\ &= E(\theta) + \frac{\alpha}{4} \left[\nabla^2 E(\theta) - \frac{1}{2} \{ \nabla E(\theta) \}^2 + \frac{N^2}{n} \left\{ V(\theta) - \int V(\theta) \nabla E(\theta) d\theta \right\} \right] \quad (\text{A.4})\end{aligned}$$

□

A.2 Proof of Theorem 3

We provide a proof of Theorem 3 (Chapter 6) about the correctness of the online averaging procedure for estimating the Fisher Information.

Theorem. *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 , $X_n^t = \{x_{t_1} \dots x_{t_n}\}$ and compute the empirical covariance of the scores $V(\theta_t; X_n^t) = \frac{1}{n-1} \sum_{i=1}^n \{g(\theta_t; x_{t_i}) - \bar{g}_n(\theta_t)\} \{g(\theta_t; x_{t_i}) - \bar{g}_n(\theta_t)\}^T$. Let V_T be the average of $V(\theta_t)$ across T iterations. For large N , as $T \rightarrow \infty$, V_T converges to the Fisher information $I(\theta_0)$ plus $\mathcal{O}(\frac{1}{N})$ corrections. i.e.*

$$\lim_{T \rightarrow \infty} \left[V_T \triangleq \frac{1}{T} \sum_{t=1}^T V(\theta_t; X_n^t) \right] = I(\theta_0) + \mathcal{O}\left(\frac{1}{N}\right) \quad (\text{A.5})$$

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

$$\begin{aligned}V(\theta_t; X_n^t) &= \frac{1}{n} \sum_{i=1}^n g(\theta_t; x_{t_i}) g(\theta_t; x_{t_i})^T \\ &\quad - \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j=1, j \neq i}^n g(\theta_t; x_{t_i}) g(\theta_t; x_{t_j})^T\end{aligned} \quad (\text{A.6})$$

As $T \rightarrow \infty$, 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:

$$\begin{aligned} V_T &= E_{\theta_t, X_n^t} [V(\theta_t, X_n^t)] \\ &= E_{\theta, x} [g(\theta; x)g(\theta; x)^T] - E_{\theta, x, x'} [g(\theta; x)g(\theta; x')^T] \end{aligned} \quad (\text{A.7})$$

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.7) 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):

$$g(\theta; x) = \sum_{k=0}^{\infty} \frac{g^{(k)}(\theta_0; x)(\theta - \theta_0)^k}{k!} \quad (\text{A.8})$$

We will now apply this series expansion to each of the terms in Eqn. (A.7). Thus we have:

$$E_{\theta, x} [\{g(\theta; x)\}^2] = \sum_{k=0}^{\infty} A_k(\theta_0) E_{\theta} [(\theta - \theta_0)^k] \quad (\text{A.9})$$

where,

$$A_k(\theta_0) = \sum_{m=0}^k \frac{E_x [g^{(m)}(\theta_0; x)g^{(k-m)}(\theta_0; x)]}{m!(k-m)!} \quad (\text{A.10})$$

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

$$E_{\theta} [(\theta - \theta_0)^{2r}] = \frac{(2r)!}{2^r r!} \left[\frac{I^{-1}(\theta_0)}{N} \right]^r \quad (\text{A.11})$$

Thus, we have:

$$E_{\theta, x} [\{g(\theta; x)\}^2] = \sum_{r=0}^{\infty} A_{2r}(\theta_0) \frac{(2r)!}{2^r r!} \left[\frac{I^{-1}(\theta_0)}{N} \right]^r \quad (\text{A.12})$$

Similarly, we can show that:

$$E_{\theta, x, x'} [g(\theta; x)g(\theta; x')] = \sum_{r=0}^{\infty} B_{2r}(\theta_0) \frac{(2r)!}{2^r r!} \left[\frac{I^{-1}(\theta_0)}{N} \right]^r \quad (\text{A.13})$$

where,

$$B_k(\theta_0) = \sum_{m=0}^k \frac{E_x [g^{(m)}(\theta_0; x)] E_x [g^{(k-m)}(\theta_0; x)]}{m!(k-m)!} \quad (\text{A.14})$$

From Eqns. (A.12) and (A.13), we have for large N :

$$\lim_{T \rightarrow \infty} V_T = \sum_{r=0}^{\infty} C_{2r}(\theta_0) \frac{(2r)!}{2^r r!} \left[\frac{I^{-1}(\theta_0)}{N} \right]^r \quad (\text{A.15})$$

where,

$$C_k(\theta_0) = A_k(\theta_0) - B_k(\theta_0) \quad (\text{A.16})$$

For example,

$$C_0(\theta_0) = I(\theta_0) \quad (\text{A.17})$$

$$\begin{aligned}
C_2(\theta_0) = & E_x \left[g(\theta_0; x) g^{(2)}(\theta_0; x) + g^{(1)}(\theta_0; x) g^{(1)}(\theta_0; x) \right] \\
& - I^2(\theta_0)
\end{aligned} \tag{A.18}$$

Thus for large N ,

$$\begin{aligned}
\lim_{T \rightarrow \infty} V_T = & I(\theta_0) - \frac{I(\theta_0)}{N} + \\
& \frac{I^{-1}(\theta_0)}{N} E_x \left[g(\theta_0; x) g^{(2)}(\theta_0; x) + g^{(1)}(\theta_0; x) g^{(1)}(\theta_0; x) \right] \\
& + \sum_{r=2}^{\infty} C_{2r}(\theta_0) \frac{(2r)!}{2^r r!} \left[\frac{I^{-1}(\theta_0)}{N} \right]^r \\
= & I(\theta_0) + \mathcal{O}\left(\frac{1}{N}\right)
\end{aligned} \tag{A.19}$$

□