M.S. THESIS

High Dimensional Markov Chain Monte Carlo with Multiple GPUs

여러 대의 GPU를 활용한 고차원 마르코프 체인 몬테카를로

BY

Minchul Kim

FEBRUARY 2018

DEPARTMENT OF STATISTICS
COLLEGE OF NATURAL SCIENCES
SEOUL NATIONAL UNIVERSITY

Abstract

Allocating computation over multiple threads to reduce running time has become a key to training big models such as deep neural networks because a Graphics Processing Unit (GPU), which is parallel in nature, can speed up intensive matrix operations. We present a new MCMC algorithm that can be distributed over multiple GPUs by combining bridge sampling with Hamiltonian Monte Carlo on partitioned sample spaces. We empirically show that this approach can expedite MCMC sampling for any unnormalized target distribution such as Bayesian Neural Network in a high dimensional setting. Furthermore, in the presence of multimodality, this algorithm is expected to be more efficient in mixing MCMC chains when proper partitions are chosen. Finally, by comparing the parameter distributions of different learning method, we suggest that further studies could be conducted on the effect of a constrained sample space on the generalization error.

Keywords: Bayesian Neural Network, Hamiltonian Monte Carlo, Image Classification, Bridge Sampling, GPU, Parallelization

Student Number: 2016-20260

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural Network models are widely applied for classification and regression. Advancements in the GPU (Graphical Processing Unit) and related programming languages have allowed deep Neural Networks to be trained in a reasonable time. In many occasions, however, questions remain about the certainty of the model parameters. Bayesian learning offers a systematic way to quantify the uncertainty of the model parameters. In the Bayesian inference framework, observations x are considered to be generated from some model with unobserved parameters θ. As opposed to getting the point estimate, a distribution of the parameter can be estimated in the form of the posterior distribution $p(\theta|x)$.

In Bayesian learning, Markov Chain Monte Carlo (MCMC) is one of the popular computational methods, along with the variational inference, for approximating the posterior distribution. Variational inference gained traction in the Bayesian community due to its fast speed from taking advantage of stochastic and distributed optimizations (Robbins and Monro, 1951) and it is considered to be the only feasible choice of posterior approximation when the given

data set is so large that MCMC sampling would be computationally intractable (Blei, Kucukelbir, and McAuliffe, 2017). MCMC, on the other hand, is more computationally intensive but enjoys the theoretical guarantee of asymptotically converging to the target posterior distribution (Robert, 2004) and better posterior approximation when the target distribution is multimodal (Blei, Kucukelbir, and McAuliffe, 2017). For this reason, we attempt to accelerate the MCMC sampling procedure by utilizing multiple GPUs to achieve both the fast computation and a close approximation of the posterior distribution shape and variance.

Multiple GPUs optimization scheme is already implemented in the conventional point estimation with stochastic optimization such as training CNN with stochastic gradient descent (Abadi et al., 2016). MCMC samples, however, are serial by design, making it difficult to speed up the sampling procedure by launching multiple threads of chains. To accelerate the sampling process with multiple GPUs, one must devise an MCMC algorithm that can be parallelized.

1.1 Related Works

Past approaches to parallelizing MCMC can be divided into six categories: 1) simple multiple chains, 2) likelihood parallelization, 3) prefetching, 4) ensemble method, 5) blocking and 6) sample space partition. Firstly, simple multiple chain method is implemented by launching multiple MCMC chains and by approximating the posterior through averaging the samples from multiple chains (Glynn and Heidelberger, 1992; Rosenthal, 2000; Bradford and Thomas, 1996). Despite its simplicity, the burn-in period cannot be shortened and does not accelerate the convergence of slow MCMC chains. Therefore, this approach is often used together with other parallelization approach to see the reliability of

the chain. Secondly, likelihood parallelization approach is implemented by parallelizing the calculation of the likelihood, which could reduce the computation time dramatically if likelihood requires heavy calculation and its computation can be efficiently parallalized (Wilkinson, 2006). However, a separate programming is needed for each problem set, which is a significant drawback to those who are less concerned with a learning method and more focused on the model design and architecture. Thirdly, prefetching is an algorithm in which parts of the computations that require heavy computation are pre-calculated in in other cores Brockwell, 2006; Byrd, Jarvis, and Bhalerao, 2008. This method suffers from the similar drawback from the likelihood parallelization method. Fourthly, ensemble methods include techniques such as parallel tempering, affine invariant ensemble sampling and parallel elliptical slice sampling (Swendsen and Wang, 1986; Foreman-Mackey et al., 2013; Goodman and Weare, 2010; Nishihara, Murray, and Adams, 2014). In this approach, multiple chains are aggregated to approximate the posterior distribution. For example, in the parallel tempering, multiple chains are launched simultaneously and some chains are used to improve the mixing of other chains. These algorithms are known to be effective in high dimension, but the number of cores to launch parallel threads is limited. Fifthly, in blocking, samples are generated in a subspace of the sample space with a fewer number of dimension. Each parallel chain updates a part of the sample space. Wilkinson, 2006. But the sample space must meet strong requirements and naive decomposition of the state space will not result in the correct limiting distribution. Lastly, in the state space partition method, multiple independent chains are launched independently on each partition of the state space (Basse, Smith, and Pillai, 2016; VanDerwerken and Schmidler, 2013). This method requires finding both a good partition and an MCMC sampling scheme that can sample from the subspace of the state space.

1.2 Contribution

We improve on the state space partitioning method for parallelizing the MCMC chains by suggesting an algorithm that is efficient in high dimensional space and can take advantage of multiple GPUs. We argue that Hamiltonian Monte Carlo and bridge sampling can be used together to launch independent MCMC chains and the whole computation can be carried out in GPUs to speed up the calculation. The key idea of the proposal is to divide the sample space into partitions and to generate samples within each partition using constrained Hamiltonian Monte Carlo. The samples from each partition are then aggregated in a careful manner using bridge sampling to recover the original distribution.

The rest of the paper is organized as follows. For the completeness of the paper, basics of HMC is included in chapter 2. In chapter 3, we propose the new method for parallelizing MCMC by using Constrained Hamiltonian Monte Carlo and bridge sampling. In chapter 4, the proposed method is illustrated in applications and its benefits are emphasized. The discussion is given in chapter 5.

Chapter 2

Hamiltonian Monte Carlo

The Hamiltonian Monte Carlo was proposed as a molecular dynamics simulation method by Alder and Wainwright (Alder and Wainwright, 1959). In the molecular dynamics, one often needs simulates the classical mechanics principle known as Hamiltonian equations. In 1996, Neal, 2012 noted that efficient MCMC transition could be made using the Hamiltonian dynamics. For the completeness of the paper, a brief introduction of the Hamiltonian Monte Carlo is included in this chapter. The notation has been adopted from Liu, 2008 and Choo, 2000.

An object's motion is charaterized by its location or state variable $\mathbf{q}$ and momentum $\mathbf{p}$ at time t. For each location, object takes the potential energy $E(\mathbf{q})$ and the kinetic energy $K(\mathbf{p})$. The total energy is equivalent to $H(\mathbf{q}, \mathbf{p}) = E(\mathbf{q}) + K(\mathbf{p})$, which is also called the Hamiltonian.

The kinetic energy is defined as

$$K(\mathbf{p}) = \sum_{i=1}^{d} \frac{p_i^2}{2m_i}$$

and the potential energy $E(\mathbf{q})$ is defined in relation to the desired distribution $P(\mathbf{q})$,

$$E(\mathbf{q}) = -\log P(\mathbf{q}) - \log(Z)$$

where Z is the normalization constant, and it allows the HMC to be used when complicated probability distribution is known up to a normalization constant. In using the Hamiltonian dynamics in HMC, $P(\mathbf{q})$ becomes the desired target distribution from which we wish to generate samples. It can also be the posterior distribution in Bayesian learning. The kinetic energy can be viewed as an auxiliary variable that will be discarded when the samples are generated.

The Hamiltonian dynamics are governed by a set of differential equations known as the Hamiltonian equations.

$$\frac{dq_i}{dt} = \frac{dH}{dp_i} = \frac{dK(\mathbf{p})}{dp_i} = \frac{\mathbf{p}}{m_i}$$
$$\frac{dp_i}{dt} = -\frac{dH}{dq_i} = \frac{dE(\mathbf{q})}{dq_i}.$$

If $\frac{dE(\mathbf{q})}{dq_i}$, $\frac{dK(\mathbf{p})}{dp_i}$, q_o and p_o at time t_o are given, it is possible to predict the location and momentum of an object at time $t = t_o + T$.

The Hamiltonian Monte Carlo is a combination of Hamiltonian dynamics proposal and Metropolis rejection. In the MCMC chain, new samples are proposed by simulating movements in the Hamiltonian dynamics, and the samples are either accepted or rejected based on the usual Metropolis rejection criterion. When those two steps are combined appropriately, the samples proposed in the Hamiltonian dynamics can travel farther than the samples from the random walk proposal without breaking ergodicity and invariance condition necessary for the chain to converge to the target distribution (Choo, 2000).

Theoretically, if the Hamiltonian dynamics could be simulated perfectly, Metropolis rejection step would be unnecessary because all proposed samples

would be in the same H and therefore be accepted. It is not possible, however, to simulate the Hamiltonian dynamics on a continuous time scale, and in practical implementations one often resorts to using leapforg updates to simulate Hamiltonian dynamics in the discrete time space. Therefore the HMC is carried out in two parts: sample proposal stage and the Metropolis accept-reject stage for correcting the error due to the discretization of time. The sample proposal stage is carried out in two fold: momentum proposal and the leapfrog simulation.

2.1 Momentum proposal

The Hamiltonian Monte Carlo is a variant of Metropolis Hastings(MH) algorithm, in that the random walk proposal in MH is replaced with a random walk that preserves the Hamiltonian defiend by the target probability distribution. Randomness in the walk comes from sampling the auxiliary variable, $\mathbf{p}$ from the standard normal distribution. Sampled momentum $\mathbf{p}$ will serve as a starting point in simulating movement in the same hamiltonian $H(\mathbf{q}, \mathbf{p})$. Simulating movement in the hamiltonian is carried out by leapfrog update.

2.2 Leapfrog update

The leapfrog update is composed of 3 steps: 1) half update of momentum $\mathbf{p}$, 2) full update of state $\mathbf{q}$, and 3) half update of momentum $\mathbf{p}$.

1. For each i in $i = 1, ..., d$

$$p_i(t + \frac{\epsilon}{2}) = p_i(t) - \frac{\epsilon}{2}\frac{dE}{dq_i}(\mathbf{q}(t)). \tag{2.1}$$

2. For each i in $i = 1, ..., d$

$$q_i(t + \epsilon) = q_i(t) - \epsilon\frac{p_i(t + \frac{\epsilon}{2})}{m_i}. \tag{2.2}$$

3. For each i in $i = 1, ..., d$

$$p_i(t + \epsilon) = p_i(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{dE}{dq_i}(\mathbf{q}(t + \epsilon)). \tag{2.3}$$

Each leapfrog update is a movement in the Hamiltonian with respect to time. Therefore repeating the leapfrog multiple times will result in simulating the object's movement for longer time. The leapfrog updates are repeated L times to simulate state and momentum movement for L steps in the Hamiltonian space.

2.3 Metropolis Accept-Reject

The last step of the Hamiltonian Monte Carlo is the Metropolis accept-reject step. In the following steps, the acceptance ratio is computed.

1. Genrate $\mathbf{p}^*, \mathbf{q}^*$ from the current values $\mathbf{p^{(s)}}, \mathbf{q^{(s)}}$ using the leapfrog proposal distribution $L_s(\mathbf{p}^*, \mathbf{q}^* | \mathbf{p}^{(s)}, \mathbf{q}^{(s)})$

2. Calculate acceptance ratio by

$$r = \frac{p(\mathbf{p}^*, \mathbf{q}^*)}{p(\mathbf{p^{(s)}}, \mathbf{q^{(s)}})} \times \frac{L_s(\mathbf{p}^{(s)}, \mathbf{q}^{(s)} | \mathbf{p}^*, \mathbf{q}^*)}{L_s(\mathbf{p}^*, \mathbf{q}^* | \mathbf{p}^{(s)}, \mathbf{q}^{(s)})}.$$

3. Sample $u \sim uniform(0, 1)$ and if $u < r$, set $(\mathbf{p}^{(s+1)}, \mathbf{q}^{(s+1)}) = (\mathbf{p}^*, \mathbf{q}^*)$; Otherwise, set $(\mathbf{p}^{(s+1)}, \mathbf{q}^{(s+1)}) = (\mathbf{p^{(s)}}, \mathbf{q^{(s)}})$

Note that the leapfrog proposal is symmetric by construction. Thus

$$\frac{L_s(\mathbf{p}^{(s)}, \mathbf{q}^{(s)} | \mathbf{p}^*, \mathbf{q}^*)}{L_s(\mathbf{p}^*, \mathbf{q}^* | \mathbf{p}^{(s)}, \mathbf{q}^{(s)})} = 1.$$

In the Hamiltonian Monte Carlo, the posterior samples are generated from the target distribution $P(\mathbf{q})$ by repeating the leapfrog step and the accept-reject stage. Algorithm 1 shows the full Hamiltonian Monte Carlo procedure. Conventionally, new momentum samples are proposed using $N(0, 1)$.

Algorithm 1 Hamiltonian Monte Carlo

procedure HMC($numSample = S, stepSizeTuningParam = \eta$)
  initialize $\mathbf{p}^0 \sim N(0, I_d)$
  initialize $\mathbf{q}^0$
  for s in $1 : S$ **do**
    $p_i^s \leftarrow N(0,1)$ for (i in $1 : d$)
    $\mathbf{q}^s \leftarrow \mathbf{q}^{s-1}$
    for l in $1 : L$ **do** ▷ Leapfrog update L times
      $p_i^s(t + \frac{\epsilon}{2}) = p_i^s(t) - \frac{\epsilon}{2}\frac{dE}{dq_i^s}(\mathbf{q^s}(t))$ for (i in $1 : d$)
      $q_i^s(t + \epsilon) = q_i^s(t) - \epsilon\frac{p_i^s(t+\frac{\epsilon}{2})}{m_i}$ for (i in $1 : d$)
      $p_i^s(t + \frac{\epsilon}{2}) = p_i^s(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{dE}{dq_i^s}(\mathbf{q}^s(t + \epsilon))$ for (i in $1 : d$)
    end for
    $ratio = min[1, \exp(-H(\mathbf{q}^s, \mathbf{p}^s)) + H(\mathbf{q}^{s-1}, \mathbf{p}^{s-1}))]$
    if $Unif[0, 1] > ratio$ **then** ▷ accept-reject
      $\mathbf{q}^s \leftarrow \mathbf{q}^{s-1}$ ▷ reject the new sample
    end if
  end for
  Return $\{\mathbf{q}^s\}_{s=0}^S$
end procedure

Chapter 3

Bridged Hamiltonian Monte Carlo

In this chapter we propose a new approach of parallelizing an MCMC chain using a constrained version of the Hamiltonian Monte Carlo and the bridge sampling. We call the proposed algorithm as Bridged Hamiltonian Monte Carlo (BHMC).

3.1 Sampling from Partitioned Sample Space

One of the parallelization schemes for MCMC is partitioning the sample space to run different MCMC chains in each component of the partition (VanDerwerken and Schmidler, 2013; Basse, Smith, and Pillai, 2016). The existing approaches focused on finding efficient partition schemes but did not fully explore different sampling scheme in a restricted sample space as most of them resorted to simple Metropolis Hastings algorithm. Instead of using Metropolis Hastings algorithms, we propose to use the constrained Hamiltonian Monte Carlo for sampling from the restricted sample space, as HMC explores the high dimen-

sional space more efficiently.

Let $\pi(\theta)$ be the probability density of interest on the parameter space Θ. The parameter space Θ is partitioned into multiple sets and $\bigcup_{j=1}^{J} \Theta_j$. Here J is the number of components in the partition. For each component j, MCMC is run on the target distribution restricted to Θ_j and its MCMC samples are denoted as $\{\theta_j^1, ...\theta_j^{n_j}\}$.

The restricted target distribution is

$$\pi_j(\theta) = \frac{\pi(\theta)\mathbf{1}_{\Theta_j}(\theta)}{\int_{\Theta_j} \pi(\theta)\nu(d\theta)} = \frac{\pi(\theta)\mathbf{1}_{\Theta_j}(\theta)}{w_j},$$

where $w_j = \int_{\Theta_j} \pi(\theta)\nu(d\theta)$, the normalizing constant of the restricted target distribution and ν is the σ-finite measure on Θ dominating π. Then for any π integrable function f, the following holds.

$$E_\pi(f) = \sum_{j=1}^{J} w_j \int_{\Theta_j} f(\theta)\pi_j(\theta)\nu(d\theta).$$

If w_j and $u_j = \int_{\Theta_j} f(\theta)\pi_j(\theta)\nu(d\theta)$, for $j \in \{1, ..., J\}$, can be estimated, $E_\pi(f)$ can be estimated.

3.1.1 Constrained HMC

A Monte Carlo estimator of u_j is

$$\hat{u_j} = \frac{1}{n_j} \sum_{i=1}^{n_j} f(\theta_j^{(i)})$$

where θ_j^i are the samples from the restricted target distribution $\pi_j(\theta)$. Since the Hamiltonian Monte Carlo is more efficient than the Metropolis-Hastings algorithm in high-dimensional space, in this paper we use the Hamiltonian

Monte Carlo augmented with the "billiards" procedure proposed by Neal et al., 2011 for restricted sample space.

Consider a d-dimensional parameter space Θ. Then the restricted sample space Θ^* can be defined as a hypercube

$$\{(\theta_1, ..., \theta_d) : l_i \leq \theta_i < u_i \text{ for } i \text{ in } \{1, ..., d\}\}$$

where $l_i \in \mathbb{R}^1$ and $u_i \in \mathbb{R}^1$ are the lower and upper bound of the sample space in i-th dimension. Let U and L be the d-dimensional vectors (or matrices) with l_i and u_i as their elements. The dimension of U and L is the same as the dimension of the state $\mathbf{q}$ in HMC. Recall that $\mathbf{q}$ is the parameter of interest and second step in the leapfrog update is

$$q_i^s(t + \epsilon) = q_i^s(t) - \epsilon \frac{p_i^s(t + \frac{\epsilon}{2})}{m_i}.$$

To restrict the sample space, Neal proposes the following procedure during the leapfrog update to achieve the effect of the ball bouncing off the wall.

For each $q_i^s(t + \epsilon)$ in $i \in \{1, ..., d\}$

(a) if $q_i^s(t + \epsilon) > u_i$, replace $q_i^s(t + \epsilon)$ with $u_i - (q_i^s(t + \epsilon) - u_i)$

(b) if $q_i^s(t + \epsilon) < l_i$, replace $q_i^s(t + \epsilon)$ with $l_i + (l_i - q_i^s(t + \epsilon))$

The momentum q_i has to be negated correspondingly. The procedure achieves reversibility, so the acceptance ratio does not need to be modified. The complete modification to the HMC is given in Alorithm 2.

3.2 Combining Samples from different Sample Space

It remains to combine the different MCMC samples from different sample spaces by estimating w_j. Let $g(\theta)$ be the unnormalized target distribution $\pi(\theta)$ and C

Algorithm 2 Constrained HMC

procedure CHMC($numSample = S, stepSize = \eta, U, L$)
    ... same as HMC so far
    for l in $1 : L$ **do** ▷ Leapfrog update L times
        $p_i^s(t + \frac{\epsilon}{2}) = p_i^s(t) - \frac{\epsilon}{2}\frac{dE}{dq_i^s}(\mathbf{q^s}(t))$ for (i in $1 : d$)
        for i in $1 : d$ **do**
            $q_i^s(t + \epsilon) = q_i^s(t) - \epsilon\frac{p_i^s(t+\frac{\epsilon}{2})}{m_i}$
            if $q_i^s(t + \epsilon) > u_i$ **then**
                $q_i^s(t + \epsilon) = u_i - (q_i^s(t + \epsilon) - u_i)$
                $p_i^s(t + \frac{\epsilon}{2}) = -p_i^s(t + \frac{\epsilon}{2})$
            else if $q_i^s(t + \epsilon) < l_i$ **then**
                $q_i^s(t + \epsilon) = l_i + (l_i - q_i^s(t + \epsilon))$
                $p_i^s(t + \frac{\epsilon}{2}) = -p_i^s(t + \frac{\epsilon}{2})$
            end if
        end for
        $p_i^s(t + \frac{\epsilon}{2}) = p_i^s(t + \frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{dE}{dq_i^s}(\mathbf{q}^s(t + \epsilon))$ for (i in $1 : d$)
    end for
    ... same as HMC afterward
    Return $\{\mathbf{q}^s\}_{s=0}^S$
end procedure

be the normalizing costant. We have

$$\frac{g(\theta)}{C} = \pi(\theta).$$

Then we can write

$$\begin{aligned} w_j &= \int_{\Theta_j} \pi(\theta)\nu(d\theta) \\ &= \int_{\Theta_j} \frac{g(\theta)}{C}\nu(d\theta) \\ &= \frac{1}{C}\int_{\Theta_j} g(\theta)\nu(d\theta) \\ &= \frac{c_j}{C} \end{aligned}$$

where $c_j = \int_{\Theta_j} g(\theta)\nu(d\theta)$ is the integral of the unnormalized target distribution over j-th parameter space. Furthermore, $C = \sum_{j=1}^{J} c_j$. Therefore,

$$w_j = \frac{c_j}{\sum_{j=1}^{J} c_j}.$$

It remains to estimate c_j for each partition. Estimating the normalization constant is a major area of research and various methods have been proposed. Basse, G. et al. (2016) found that bridge sampling works well (Gelman and Meng, 1998; Basse, Smith, and Pillai, 2016).

3.2.1 Bridge Sampling

We include a derivation of the bridge sampling defined in Gronau et al. (2017). $p(y|\theta)$, $p(\theta)$ are the usual likelihood and prior distribution function. Bridge sampling defines two new functions, a proposal distribution $g(\theta)$ and a bridge function $h(\theta)$.

Consider the following identity and multiply the marginal likelihood $p(y)$ on both sides.

$$
\begin{aligned}
p(y) &= \frac{\int p(y|\theta)p(\theta)h(\theta)g(\theta)d\theta}{\int \frac{p(y|\theta)p(\theta)}{p(y)}h(\theta)g(\theta)d\theta} \\
&= \frac{\int p(y|\theta)p(\theta)h(\theta)g(\theta)d\theta}{\int h(\theta)g(\theta)p(\theta|y)d\theta} \\
&= \frac{\mathbb{E}_{g(\theta)}p(y|\theta)p(\theta)h(\theta)}{\mathbb{E}_{p(\theta|y)}h(\theta)g(\theta)}.
\end{aligned}
$$

Therefore, the marginal likelihood can be estimated by

$$
\hat{p}(y) = \frac{\frac{1}{N_2}\sum_{i=1}^{N_2} p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)h(\tilde{\theta}_i)}{\frac{1}{N_1}\sum_{j=1}^{N_1} h(\theta_j^*)g(\theta_j^*)} \tag{3.1}
$$

where $\tilde{\theta}_i \sim g(\theta)$, the proposal distribution and $\theta_j^* \sim p(\theta|y)$, the posterior distribution and N_1 is the number of posterior samples and N_2 is the number of samples from the proposal distribution. There are several choices for the bridge function $h(\theta)$. We use the optimal bridge function, which minimizes the asymptotic variance of the estimator (Gelman and Meng, 1998).

$$
h(\theta) = C \times \frac{1}{s_1 p(y|\theta)p(\theta) + s_2 p(y)g(\theta)}
$$

where $s_1 = \frac{N_1}{N_2+N_1}$ and $s_2 = \frac{N_2}{N_1+N_2}$. Then the marginal likelihood estimator becomes

$$
\hat{p}(y) = \frac{\frac{1}{N_2}\sum_{i=1}^{N_2} \frac{p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)}{s_1 p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)+s_2 p(y)g(\tilde{\theta}_i)}}{\frac{1}{N_1}\sum_{j=1}^{N_1} \frac{g(\theta_j^*)}{s_1 p(y|\theta_j^*)p(\theta_j^*)+s_2 p(y)g(\theta_j^*)}}. \tag{3.2}
$$

But notice that this equation is an iterative equation because $\hat{p}(y)$ depends on $p(y)$. So the estimate can be calculated by running the iterative equation until convergence. Usually the estimate converges within less then ten iterations.

$$
\hat{p}(y)^{(t+1)} = \frac{\frac{1}{N_2}\sum_{i=1}^{N_2} \frac{p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)}{s_1 p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)+s_2 p(y)^{(t)}g(\tilde{\theta}_i)}}{\frac{1}{N_1}\sum_{j=1}^{N_1} \frac{g(\theta_j^*)}{s_1 p(y|\theta_j^*)p(\theta_j^*)+s_2 p(y)^{(t)}g(\theta_j^*)}} \tag{3.3}
$$

where $\tilde{\theta}_i \sim g(\theta)$, the proposal distribution and $\theta_j^* \sim p(\theta|y)$, the posterior distribution. To write down the equation more concisely, $l_{1,j}$ and $l_{2,i}$ can be defined. $l_{1,j} := \frac{p(y|\theta_j^*)p(\theta_j^*)}{g(\theta_j^*)}$ $l_{2,i} := \frac{p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)}{g(\tilde{\theta}_i)}$. Then

$$\begin{aligned}\hat{p}(y) &= \frac{\frac{1}{N_2}\sum_{i=1}^{N_2} \frac{p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)}{s_1 p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)+s_2 p(y)^{(t)} g(\tilde{\theta}_i)} \frac{1/g(\tilde{\theta}_i)}{1/g(\tilde{\theta}_i)}}{\frac{1}{N_1}\sum_{j=1}^{N_1} \frac{g(\theta_j^*)}{s_1 p(y|\theta_j^*)p(\theta_j^*)+s_2 p(y)^{(t)} g(\theta_j^*)} \frac{1/g(\theta_j^*)}{1/g(\theta_j^*)}} \\ &= \frac{\frac{1}{N_2}\sum_{i=1}^{N_2} \frac{l_{2,i}}{s_1 l_{2,i}+s_2 \hat{p}(y)^{(t)}}}{\frac{1}{N_1}\sum_{j=1}^{N_1} \frac{1}{s_1 l_{1,j}+s_2 \hat{p}(y)^{(t)}}}. \end{aligned} \tag{3.4}$$

Therefore, the samples from different partitions be combined by using the following weights.

$$w_j = \frac{\hat{p}(y)}{\sum_{j=1}^{J} \hat{p}(y)}. \tag{3.5}$$

3.3 Practical Issues in Implementing Bridged Hamiltonian Monte Carlo

3.3.1 Numerical Overflow or Underflow

When calculating the likelihood, due to the floating point limits, numerical overflow and underflow can cause the weight estimate to be completely wrong. Especially when there are many parameters as in a Bayesian Neural Network, the likelihood can quickly go to zero beyond the precision limit. Therefore, the weight estimator should be modified to work with the log likelihood. Gronau et al. (2017) introduced a similar treatment. The workaround can be done in twofold; first treatment reduces the overflow in $l_{1,j}$ and the second treatment reduces the overflow in $l_{2,i}$

We let

$$\begin{aligned} ll_{1,j} &= \ln\left(\frac{p(y|\theta_j^*)p(\theta_j^*)}{g(\theta_j^*)}\right) \\ &= \ln p(y|\theta_j^*)p(\theta_j^*) - \ln g(\theta_j^*) \\ ll_{2,i} &= \ln\left(\frac{p(y|\tilde{\theta}_i)p(\tilde{\theta}_i)}{g(\tilde{\theta}_i)}\right) \\ &= \ln p(y|\tilde{\theta}_i)p(\tilde{\theta}_i) - \ln g(\tilde{\theta}_i). \end{aligned}$$

Then

$$\begin{aligned} \hat{p}(y) &= \frac{\frac{1}{N_2}\sum_{i=1}^{N_2}\frac{\exp(ll_{2,i})}{s_1\exp(ll_{2,i})+s_2\hat{p}(y)^{(t)}}}{\frac{1}{N_1}\sum_{j=1}^{N_1}\frac{1}{s_1\exp(ll_{1,j})+s_2\hat{p}(y)^{(t)}}} \\ &= \frac{\frac{1}{N_2}\sum_{i=1}^{N_2}\frac{\exp(ll_{2,i})\exp(-l^*)}{s_1\exp(ll_{2,i})\exp(-l^*)+s_2\hat{p}(y)^{(t)}\exp(-l^*)}}{\frac{1}{N_1}\sum_{j=1}^{N_1}\frac{\exp(-l^*)}{s_1\exp(ll_{1,j})\exp(-l^*)+s_2\hat{p}(y)^{(t)}\exp(-l^*)}} \\ &= \frac{\frac{1}{N_2}\sum_{i=1}^{N_2}\frac{\exp(ll_{2,i}-l^*)}{s_1\exp(ll_{2,i}-l^*)+s_2\hat{p}(y)^{(t)}\exp(-l^*)}}{\frac{1}{N_1}\sum_{j=1}^{N_1}\frac{\exp(-l^*)}{s_1\exp(ll_{1,j}-l^*)+s_2\hat{p}(y)^{(t)}\exp(-l^*)}}. \end{aligned}$$

Practically, letting l^* equal to the mean of $ll_{1,j}$ works well because the variation in $ll_{1,j}$ is smaller than $ll_{2,i}$. And we let

$$\hat{r}(t) = \hat{p}(y)\exp(-l^*)$$

to have

$$\hat{r}(y) = \frac{\frac{1}{N_2}\sum_{i=1}^{N_2}\frac{\exp(ll_{2,i}-l^*)}{s_1\exp(ll_{2,i}-l^*)+s_2\hat{r}(y)^{(t)}}}{\frac{1}{N_1}\sum_{j=1}^{N_1}\frac{1}{s_1\exp(ll_{1,j}-l^*)+s_2\hat{r}(y)^{(t)}}}. \tag{3.6}$$

In general bridge sampling, when converting $\hat{r}(t)$ to $\hat{p}(t)$, $\exp(-l^*)$ can cause numeric overflow if $|l^*|$ is bigger than 700 as $\exp(700)$ can reach up to $1.0e{+}304$. But in this scenario where the primary interest is in the relative weights of each of the partition, $\exp(-l^*)$ can be ignored as

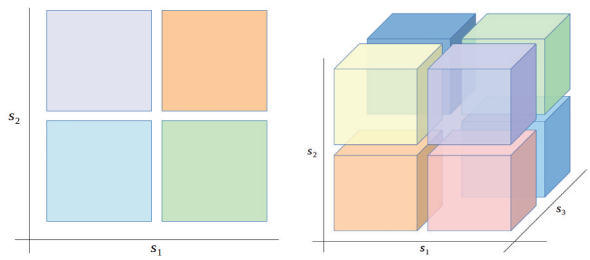


Figure 3.1: The left figure shows the $\mathbb{R}^2$ space divided at s_1 and s_2. The right figure shows the $\mathbb{R}^3$ space divided at s_1, s_2 and s_3. The number of partition after s split is 2^s. The number of partitions and where to split are tuning parameters.

$$w_j = \frac{\hat{p}(y)}{\sum_{j=1}^{J} \hat{p}(y)} = \frac{\hat{r}(y)}{\sum_{j=1}^{J} \hat{r}(y)}$$

if l^* is chosen to be same across all partition.

This transformation with l^* can help with the numeric overflow in $l_{1,j}$. But often the variation in $l_{2,i}$ is bigger than in $l_{1,i}$ because the likelihood in $l_{2,i}$ is calculated with newly generated proposal samples rather than the samples from its own distribution. Therefore, subtracting l^* sometimes does not solve the overflow issue with $l_{2,i}$. Therefore an additional measure has to be taken for $l_{2,i}$.

Notice how

$$\frac{\exp(ll_{2,i} - l^*)}{s_1 \exp(ll_{2,i} - l^*) + s_2 \hat{r}(y)^{(t)}}$$

can go to 0 or 1 when the term $\exp(ll_{2,i} - l^*)$ is either too close to 0 or to big. Therefore, in such cases, 0 or infinity was replaced with very small or large number.

3.3.2 Partitioning Scheme

The number of partitions and where to split are a tuning parameter. The number of partition is 2^s where s is the number of parameter to split.

Figure 3.1 shows examples of a partition when the sample space is $\mathbb{R}^2$ and $\mathbb{R}^3$. Usually the number of split s should be much smaller than the number of parameters. Therefore, the parameter to split the sample space could be chosen randomly or by other methods that utilize initial sampling to calculate the optimal split such as spectral decomposition used in Basse, Smith, and Pillai (2016).

The split points for Bayesian neural network in this paper were set to zero because all the inputs were mean centered and the trained parameters are usually centered around zero.

Chapter 4

Experiments

4.1 Bivariate Normal Mixture Model

We started by comparing the performance of generating samples from a mixture of bivariate normal distribution. The target distribution was

$$N = \pi_1 * N_1 + (1 - \pi_1) * N_2$$

where

$$N_1(y; \theta, \Sigma) = \frac{1}{\sqrt{2\pi}} |\Sigma|^{-0.5} \exp\left(\frac{-1}{2}(y - \theta_1)^T \Sigma (y - \theta_1)\right)$$

$$N_2(y; \theta, \Sigma) = \frac{1}{\sqrt{2\pi}} |\Sigma|^{-0.5} \exp\left(\frac{-1}{2}(y - \theta_2)^T \Sigma (y - \theta_2)\right)$$

and

$$\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

and $\theta_1 = [0, 0]$ and $\theta_2 = [5, 5]$.

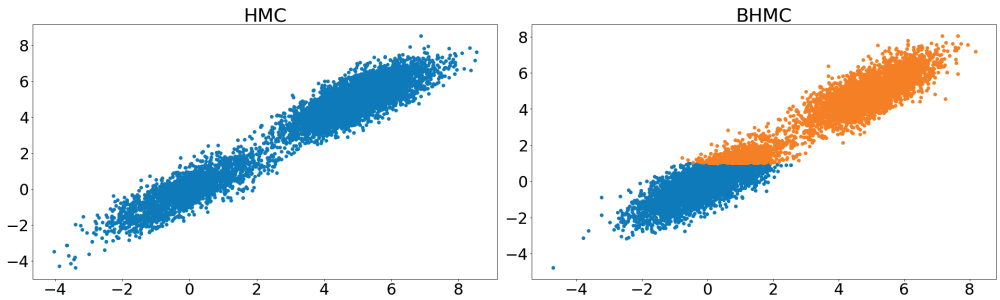
This example illustrates that partitioning sample space can achieve a better mixing when the target distribution is multimodal.

The sample space was divided into two partitions; $\Omega_1 = \mathbb{R}^2 \cdot \mathbf{1}_{(y>1)}$, $\Omega_2 = \mathbb{R}^2 \cdot \mathbf{1}_{(y\leq 1)}$. From each partitioned sample space, an independent MCMC chain was launched and 5000 posterior samples were sampled. For calculating the weights, the posterior samples were divided into two parts. The first part was used in calculating the proposal distribution. The second part was used as the posterior samples in the bridge sampling. The proposal distribution $g(\theta)$ was chosen to be a bivariate Gaussian distribution with the mean and covariance equal to the sample mean and sample covariance of the half of the samples.

The performance was compared with the single chain HMC with 10000 samples from the same target distribution.

In this example, there are two high probability regions where the HMC sampler has a difficult time crossing over due to the low probability region in between. Therefore, HMC requires many runs before MCMC samples mix well. The left figure in Figure 4.1 shows the scatter plot of the 10,000 HMC samples with the first 2,000 samples thrown away. The sample mean was calculated to be $[3.01, 2.99]$. The true mean of this mixture normal distribution is [2.5, 2.5]. And it took 23 seconds to generate 10,000 samples on a single GPU (NVidea GTX 1070) with Tensorflow.

Dividing the sample space and letting the chain run independently on each of the sample space can achieve a better mixing when the sample space is reasonably well divided. Bridged Hamiltonian Monte Carlo was run on two sample space divided at $y = 1$. In each partition, 5000 MCMC samples were gathered, and 1000 samples were thrown away for burn in period. It took about 13 seconds in each of the partitions. Since each partition is independent of the other partition, they can be run on different machines. In this experiment, only one GPU was used, but if there were more GPUs running concurrently, this method could theoretically speed up the sampling procedure drastically. In the



(a) sample mean:$[3.61, 3.61]$ (b) sample mean:$[2.47, 2.47]$

Figure 4.1: Scatter plot of the samples generated from the Hamiltonian Monte Carlo and Bridged Hamiltonian Monte Carlo. For HMC,total 10000 sampels were generated from each partition and initial 2000 samples were thrown away. For BHMC, total 10000 samples were generated from each partition and initial 2000 samples were thrown away. The true mean is design to be $[2.5, 2.5]$.

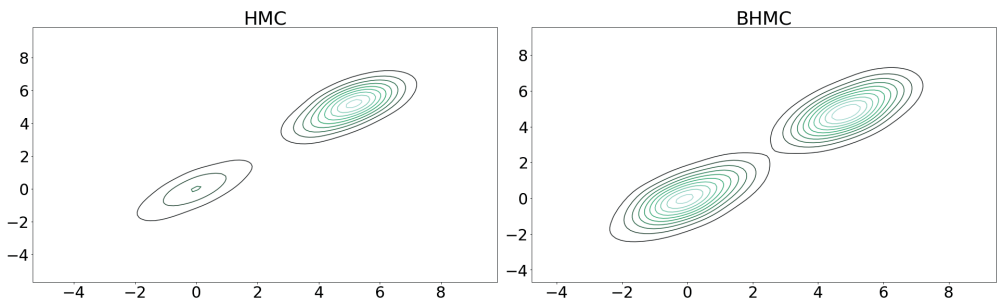


Figure 4.2: Kernel density plot of the samples generated from the Hamiltonian Monte Carlo and Bridged Hamiltonian Monte Carlo.

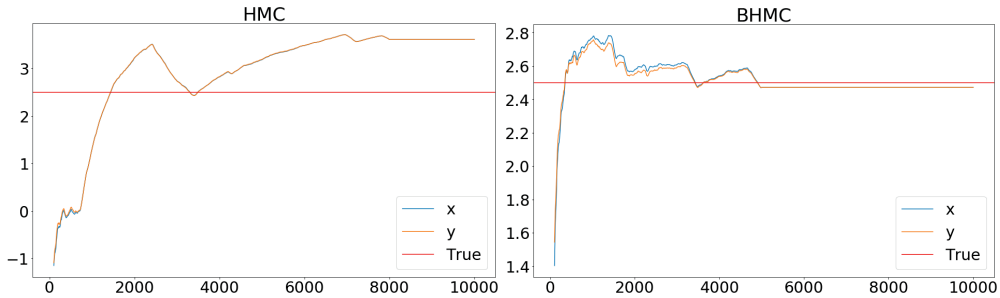


Figure 4.3: Sample mean calculated at each iteration for HMC and BHMC. The true value is 2.5 marked by the red line

Mixture Normal Sample Genration Performance Comparison					
Method	total number of samples	number of chains	number of samples in each chain	time	sample mean
HMC	$10,000$	1	$10,000$	23s	$[3.01, 2.99]$
BHMC	$10,000$	2	$5,000$	13s	$[2.47, 2.47]$

Table 4.1: The result of sampling from the mixture bivariate normal distribution with the true mean $[2.5, 2.5]$. The time signifies the time taken for each chain in BHMC without taking into account the time to calculate the partition weights. The result shows that BHMC using two chains can approximate the true mean more closely than HMC in less time.

Figure 4.2, kernel density plot shows that HMC samples are not mixed yet, whereas the BHMC samples estimate the target distribution reasonably well.

BHMC sample mean was calculated to be $[2.47, 2.47]$, which is a far better estimate of the true mean $[2.5, 2.5]$ than the simple HMC sample mean which was $[3.01, 2.99]$. It would require more samples for the HMC samples to converge to the true mean. BHMC took less time in each partition, is scalable and is more accurate in the presence of multi-modality. Figure 4.3 shows that BHMC converges to the true mean faster than HMC.

4.2 Moon Data Classification

Moon data is a well-known toy classification data in python Scikit-Learn package. This data is linearly inseparable and often used for testing the performance

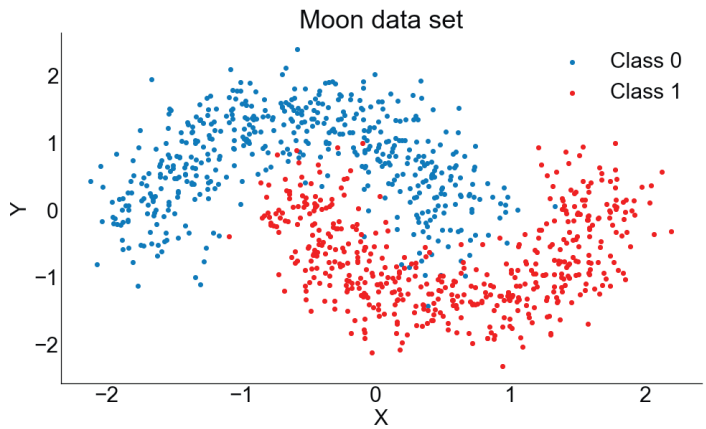


Figure 4.4: Scatter plot of the moon data set. Two classes are linearly unseparable.

of the multilayer neural network. Figure 4.4 shows the scatter plot of the moon data. Training data has the shape of 500 by 2, and the test data has the shape of 500 by 2. Each sample is either class 0 or 1. Multilayer neural network trained on HMC and BHMC will be compared.

For either HMC or BHMC, following two hidden layer architecture is used.

- L = 2 : Number of layer
- $\mathbf{h}^{(l)} = 0, ..., L+1$: lth hidden layer
- $\mathbf{h}^{(0)} = \mathbf{x}$: input data
- $\mathbf{h}^{(L+1)} = f(\mathbf{x})$: output
- Each layer $\mathbf{h}^{(l)}$ has n_l nodes. $n_0 = 2$, $n_1 = 5$, $n_2 = 5$, $n_3 = 2$

$$
\begin{aligned}
z_j^{(l)} &= \sum_{i=1}^{n_l - 1} w_{ij}^{l-1} h_i^{l-1} + b_j, \quad l = 1, ..., L+1 \quad j = 1, ..., n_l \\
h_j^{(l)} &= tanh(z_j^{(l)}), l = 1, ..., L \quad \text{hidden} \\
h_j^{(L+1)} &= Categorical(prob = norm(z^{L+1})) \quad \text{output}
\end{aligned}
$$

In Bayesian Neural network architecture, classification is often done by wrapping categorical distribution around the last layer. In this setting, weight matrices W^0, W^1, W^1 has the shape of $[2, 5]$, $[5, 5]$ and $5, 2$. The bias vectors have the length of $5, 5$ and 2. There are 57 parameters that are going to be updated at each iteration of the learning. The sample space is in a high-dimensional setting, which means that the region of probability is a shallow shell compared to the whole sample space. Therefore, usual Metropolis Hasting random walk proposal will result in too many rejection samples. Therefore, in this scenario, HMC or BHMC will be useful in generating samples with a high acceptance rate. The benefit of using a Bayesian Neural Network over regular gradient based networks is that the network parameters' certainty can be analyzed in the Bayesian Neural Network. In this case, because the input data is in two-dimension X and Y, parameters' uncertainty can be visualized. In this example, all weights have been given a normal distribution prior with mean 1 and variance 1. All biases have been given a normal prior with mean 0 and variance 1.

For simple Hamiltonian Monte Carlo sampler, total 5000 samples were generated, and the first 1000 samples were discarded for burn in period. The left figure in Figure 4.5 shows the scatter plot of the test sample classification result. In testing, the weight was taken to be the mean of the MCMC samples. The test accuracy of the classifier was 96.6%.

For Bridged Hamiltonian Monte Carlo sampler, the sample space was divided into eight partitions at the origin. In each partition, 1000 samples were generated, and the first 500 samples were discarded. The right figure in Figure 4.5 shows the scatter plot of the test sample classification result. The mean of the MCMC samples was used as the final weight for testing. The test accuracy of the classifier was 96.0%.

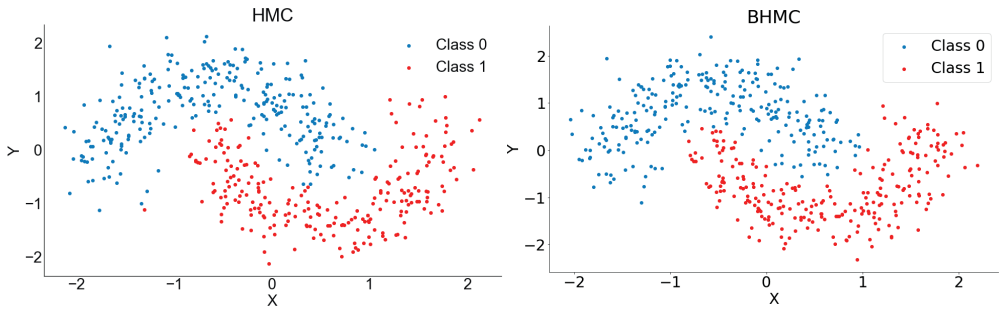


Figure 4.5: The test sample classificaiton result using Hamiltonian Monte Carlo and Bridged Hamiltonian Monte Carlo

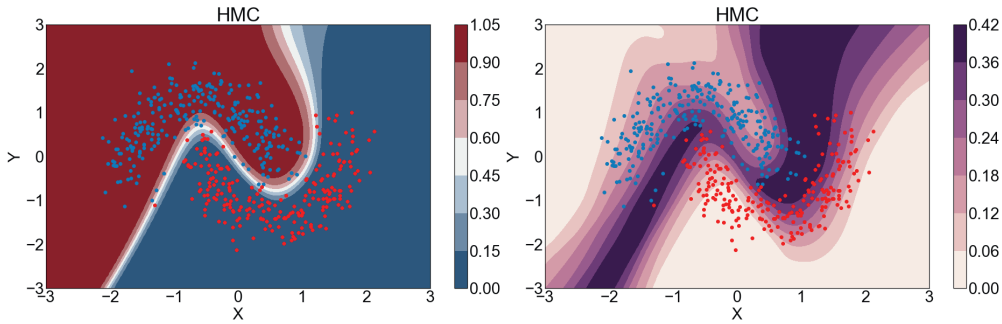


Figure 4.6: The test sample classification result using Hamiltonian Monte Carlo. The left shows the posterior mean probability of class label equal to 0. And the right plot shows the posterior predictive standard deviation of the probability of class label equal to 0.

Since the output layer is a categorical distribution, the calculated probability can be a measure of the output label's certainty. Left figures of Figure 4.6 and 4.7 show the posterior mean probability of the class label equal to 0. The figures show that the white region has a probability around $0.4 \sim 0.6$ of being a class 0.

The probability of class label equal to 0 can also be calculated by the gradient-based learning approach when the output layer is wrapped by the softmax function. However, this probability in itself is a point estimation, and the variance of the probability is not offered in the traditional gradient-based

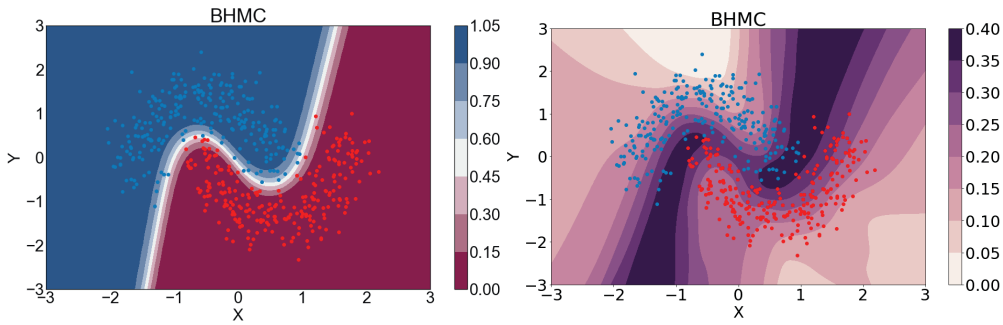


Figure 4.7: The test sample classification result using Bridged Hamiltonian Monte Carlo. The left shows the posterior mean probability of class label equal to 0. And the right plot shows the posterior predictive standard deviation of the probability of class label equal to 0.

approach. The Bayesian approach can measure how reliable is the point estimation.

Right figures in Figure 4.6 and 4.7 shows the variance of the probability. The dark region has a standard deviation of 0.4, which is a significant variation in weight estimation. Although the white region in the mean plot is relatively narrow, the dark region in the standard deviation plot is wide. 0.4 standard deviation is big enough to alter the outcome of the prediction in the region where the mean estimate is between 0.4 and 0.6.

The advantage of using BHMC over HMC is in the computation time. The deep neural network has shunned away from MCMC learning method because of its computation time and its difficulty in parallelizing the algorithm for scalability. In this example, BHMC achieved a similar result to HMC but with multiple independent parallel chains. It took 14 seconds to generate 5000 samples in HMC with GPU (NVidea GTX 1070) using Tensorflow, and it took 10 seconds to generate 1000 samples in BHMC HMC with GPU (NVidea GTX 1070) using Tensorflow (without the time in calculating the log posterior pdf in the bridge sampling). If the number of independent chains could be increased,

Moon Data Classification Performance Comparison					
Method	total number of samples	number of chains	number of samples in each chain	time	accuracy
HMC	$20,000$	1	$20,000$	57s	96.5
BHMC	$20,000$	4	$5,000$	42s	95.8
BHMC	$20,000$	8	$2,500$	23s	94.4
BHMC	$20,000$	16	$1,250$	13s	95.2

Table 4.2: The result of the moon data classification using Bayesian neural network. The time signifies the time taken for each chain in BHMC without taking into account the time to calculate the partition weights. The result shows that BHMC using more chains can decrease the time to achieve the same level of accuracy.

the necessary samples in each partition can be correspondingly decreased.

The Table 4.2 shows the result of the moon data classification under different settings. The table shows that as the number of partitions increases, the time taken to generate the same number of total samples decreases. However, the time does not decrease in a linear fashion because in BHMC, the step that check whether the sample is in the partition consumes additional computation time. Furthermore, it has been known in the experiement that the test accuracy could vary when the weight estimation using the bridge sampling is unstable. Therefore, although BHMC can utilize multiple GPUs to decrease the computation time, the parameters' variance can increase due to the variance arising from the weight calculation in each partition.

4.3 MNIST Data Classification

BHMC was also trained on the MNIST dataset. The MNIST dataset comprised of digit pictures of the size 28 by 28. It is often used as a quick benchmark for classification algorithms. Convolutional Neural Network has achieved the error rate below 0.3%. This experiment's purpose is not in attaining the lowest error rate. This experiment illustrates another usage of BHMC; it explores how the

learned parameters in each learning methods differ. Different learning methods (Adam Optimizer, HMC, BHMC)have been compared.

For all of the networks, following one hidden layer architecture is used.

- L = 1 : Number of layer
- $\mathbf{h}^{(l)} = 0, ..., L+1$: lth hidden layer
- $\mathbf{h}^{(0)} = \mathbf{x}$: input data
- $\mathbf{h}^{(L+1)} = f(\mathbf{x})$: output
- Each layer $\mathbf{h}^{(l)}$ has n_l nodes. $n_0 = 784$, $n_1 = 20$, $n_2 = 10$

$$
\begin{aligned}
z_j^{(l)} &= \sum_{i=1}^{n_l - 1} w_{ij}^{l-1} h_i^{l-1} + b_j, \quad l = 1, ..., L+1 \quad j = 1, ..., n_l \\
h_j^{(l)} &= sigmoid(z_j^{(l)}), l = 1, ..., L \quad \text{hidden} \\
h_j^{(L+1)} &= argmax(softmax(z^{L+1})) \quad \text{for adam optimizer} \\
h_j^{(L+1)} &= Categorical(prob = norm(z^{L+1})) \quad \text{for HMC and BHMC}
\end{aligned}
$$

In this example, all weights have been given a normal distribution prior with mean 1 and variance 1. All biases have been given a normal prior with mean 0 and variance 1.

4.4 Result

Adam optimizer is one of the popular optimizers in first-order gradient descent optimization method. It is based on the adaptive estimates of lower-order moments. Learning rate was chosen to be 0.01. And here cross entropy was minimized. Batch size was chosen to be 100, and 10,000 iterations were run. It

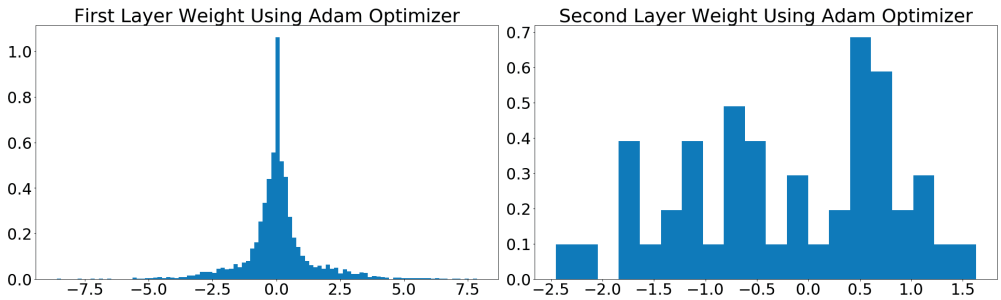


Figure 4.8: MNIST Data Adam Optimizer Weights Histogram. The left plot shows the histogram of the first layer which has the dimension 784 by 20. The second plot shows the histogram of the second layer which has the dimension 20 by 10.

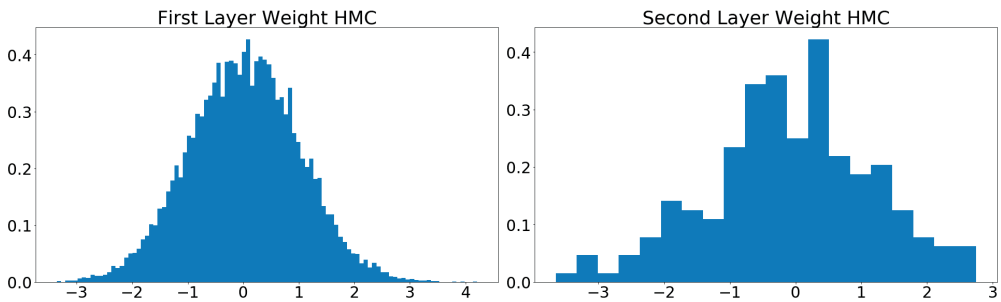


Figure 4.9: MNIST Data HMC mean weights Histogram. The left plot shows the histogram of the first layer which has the dimension 784 by 20. The second plot shows the histogram of the second layer which has the dimension 20 by 10.

took about 5 seconds, and the resulting test accuracy was 87.5%. Figure 4.8 shows the histogram of the final weights. From Figure 4.8, it is seen that the first layer weights are symmetric around 0.

For HMC no batch learning is used. Therefore, all 55,000 training samples were trained at once. Total 3,000 posterior samples were generated, and first 1,000 samples were discarded. The resulting test accuracy was 88.9%. It took 304 seconds to complete the training.

For BHMC, the sample space was constrained to be in a -1 to 1 hypercube. We tried to see if constraining the parameter to be near zero would result in

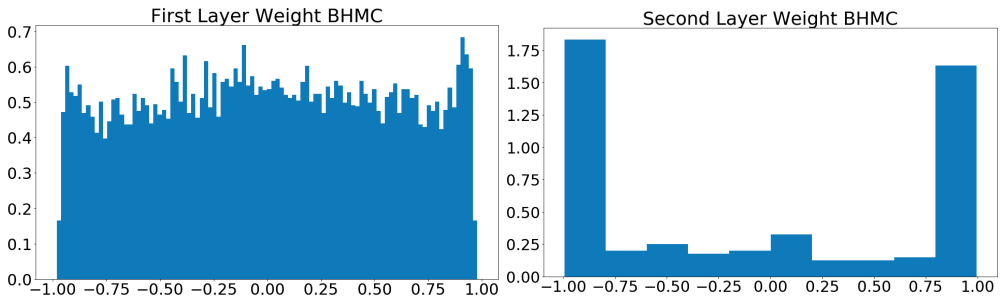


Figure 4.10: MNIST Data BHMC mean weights Histogram. The left plot shows the histogram of the first layer which has the dimension 784 by 20. The second plot shows the histogram of the second layer which has the dimension 20 by 10.

a regularization effect. 55,000 training samples were trained at once. Just as in the HMC setting, 3,000 samples were generated, and the first 1,000 samples were discarded. The final test accuracy was 89.62%. The final test accuracy was calculated by using the mean of the posterior samples. It took 329 seconds to complete the training.

The test accuracy is about the same for HMC and BHMC. But the weights look strikingly different. As shown by Figure 4.10, all BHMC parameters are within -1 and 1. A lot of second layer weights went near -1 or 1. The first layer parameters in HMC were distributed almost in a normal shape, but when the parameters were constrained in BHMC, the first layer parameter has the form of a uniform distribution.

Trace plots of 2000 samples after burn-in for HMC and BHMC posterior samples show that HMC seems to have converged to certain equilibrium from early on. The weights do not travel much. On the other hand, in BHMC, the weights seem to be still exploring the space. Although the test accuracy is same for HMC and BHMC, further run on BHMC might result in a different result.

Analysis of the test results in 4.13 show that certain misclassifications are understandable as it is difficult for human eyes to differentiate, but for most

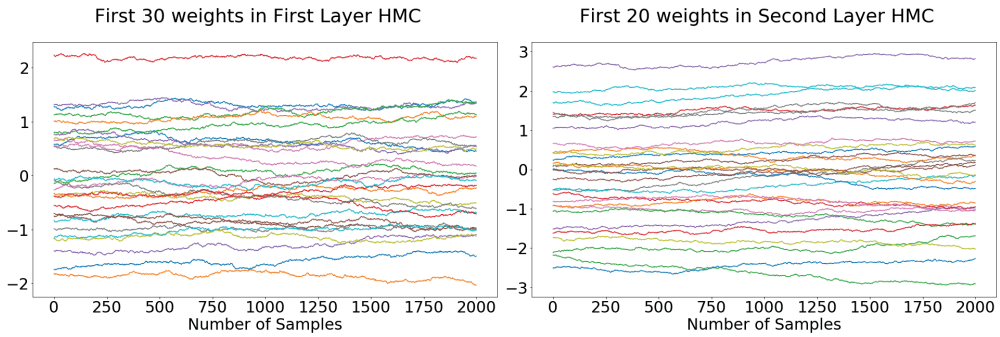


Figure 4.11: MNIST Data HMC weights Histogram. The left plot shows the trace plot of the first 30 weights of the first layer and the right plot shows the trace plot of the first 20 weights of the second layer.

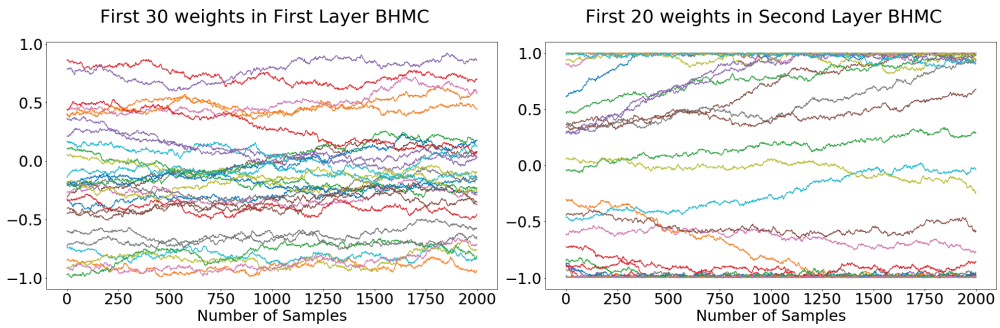


Figure 4.12: MNIST Data BHMC weights Histogram. The left plot shows the trace plot of the first 30 weights of the first layer and the right plot shows the trace plot of the first 20 weights of the second layer.

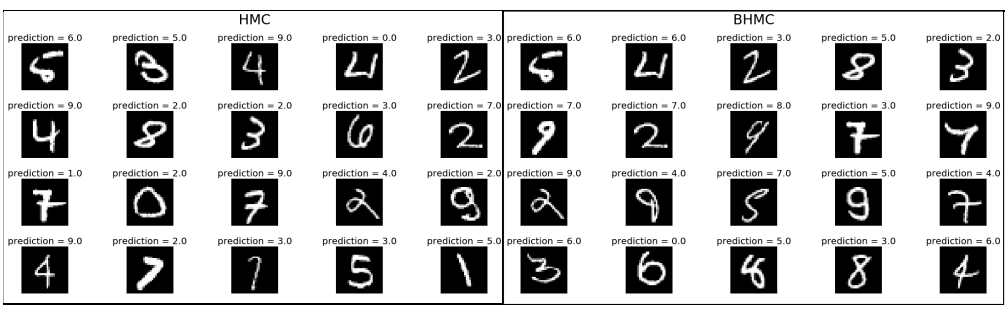


Figure 4.13: MNIST Data Misclassification Result. Left shows the misclassified samples from the HMC and the right shows the misclassified samples from the BHMC

mistakes, the problem is in the network, not the data. HMC and BHMC both seem to be making some bad mistakes.

Chapter 5

Discussion

We combined Hamiltonian Monte Carlo and bridge sampling to approximate general unnormalized target distribution using multiple independent MCMC chains. The main benefit was the speed up in generating samples that approximate the target distribution. The computation is carried out in either CPU or GPU, and unlike other single chain MCMC algorithms, Bridged Hamiltonian Monte Carlo can be carried out over multiple GPUs. The result is MCMC algorithm with desirous properties such as scalable computation time and better mixing in the presence of multimodality.

We used BHMC to sample from a mixture distribution with multimodality and fitted two different neural networks. The benefit of using BHMC over the conventional MCMC algorithm such as HMC was evident in the mixture distribution example; with a reasonable partition, multiple independent chains could allow the samples to travel in low probability regions. The question remains on how to choose the right partition boundary, and Basse et al. have worked on this topic Basse, Smith, and Pillai, 2016 using a spectral decomposition.

The Bayesian Neural Network fitted using BHMC on the toy moon data showed that significant speedup over conventional MCMC. When the sample space is high dimensional, simple random walk proposal is not efficient enough to generate samples that approximate the target distribution in a reasonable amount of time. In such scenarios, one would often resort to an auxiliary variable method such as HMC, but even HMC can be slow on deep neural networks with lots of parameters. BHMC, which split the sample space into multiple partitions to run multiple independent HMC chains, was able to speed up the sampling procedure. Theoretically, this can be scaled up by using multiple GPUs, a level of parallelization which was not possible in other MCMC algorithms.

All BHMC code is written in python's GPU friendly package, Tensorflow and the symbolic variable package, Edward. All the necessary gradients can be calculated automatically using the symbolic relations of the variables that are defined. Furthermore, by taking advantage of the Edward's Random Variable class, which is a wrapper around the tensor in Tensorflow, one can set any arbitrary unnormalized target distribution without having to calculate the likelihood equations. Therefore, fitting a Bayesian Neural Network using BHMC is as easy as writing down a similar Neural Network architecture in Tensorflow.

One step that could contribute to the instability of the algorithm is the calculation of the weight in the bridge sampling. In the bridge sampling procedure, the sum of exponentials has to be computed to calculate the weights. Chapter 3 introduced the transformation to alleviate the floating point problem. However, when the number of parameters is large, and the variance in the proposal distribution sample is large, the sum of the exponentials could still cause a numerical overflow. In the moon data example, the calculation of the weights was shown to be still unstable. When the weight calculation is unstable, the aggregated sample mean could be significantly off. Therefore, in future studies, more stable

and generalizable methods to calculate the weights have to be introduced.

Bibliography

[1] Martın Abadi et al. "TensorFlow: A system for large-scale machine learning". In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*. 2016.

[2] Berni J Alder and T E Wainwright. "Studies in molecular dynamics. I. General method". In: *The Journal of Chemical Physics* 31.2 (1959), pp. 459–466.

[3] Guillaume Basse, Aaron Smith, and Natesh Pillai. "Parallel Markov Chain Monte Carlo via Spectral Clustering". In: *Artificial Intelligence and Statistics*. 2016, pp. 1318–1327.

[4] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. "Variational inference: A review for statisticians". In: *Journal of the American Statistical Association* just-accepted (2017).

[5] Russell Bradford and Alun Thomas. "Markov chain Monte Carlo methods for family trees using a parallel processor". In: *Statistics and Computing* 6.1 (1996), pp. 67–75.

[6] Anthony E Brockwell. "Parallel Markov chain Monte Carlo simulation by pre-fetching". In: *Journal of Computational and Graphical Statistics* 15.1 (2006), pp. 246–261.

[7] Jonathan MR Byrd, Stephen A Jarvis, and Abhir H Bhalerao. "Reducing the run-time of MCMC programs by multithreading on SMP architectures". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. IEEE. 2008, pp. 1–8.

[8] Kiam Choo. "Learning hyperparameters for neural network models using Hamiltonian dynamics". PhD thesis. Citeseer, 2000.

[9] Daniel Foreman-Mackey et al. "emcee: The MCMC hammer". In: *Publications of the Astronomical Society of the Pacific* 125.925 (2013), p. 306.

[10] Andrew Gelman and Xiao-Li Meng. "Simulating normalizing constants: From importance sampling to bridge sampling to path sampling". In: *Statistical science* (1998), pp. 163–185.

[11] Peter W Glynn and Philip Heidelberger. "Analysis of initial transient deletion for parallel steady-state simulations". In: *SIAM Journal on Scientific and Statistical Computing* 13.4 (1992), pp. 904–922.

[12] Jonathan Goodman and Jonathan Weare. "Ensemble samplers with affine invariance". In: *Communications in applied mathematics and computational science* 5.1 (2010), pp. 65–80.

[13] Quentin F Gronau et al. "A tutorial on bridge sampling". In: *arXiv preprint arXiv:1703.05984* (2017).

[14] Jun S Liu. *Monte Carlo strategies in scientific computing*. Springer Science & Business Media, 2008.

[15] Radford M Neal. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media, 2012.

[16] Radford M Neal et al. "MCMC using Hamiltonian dynamics". In: *Handbook of Markov Chain Monte Carlo* 2.11 (2011).

[17] Robert Nishihara, Iain Murray, and Ryan P Adams. "Parallel MCMC with generalized elliptical slice sampling." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 2087–2112.

[18] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[19] Christian P Robert. *Monte carlo methods*. Wiley Online Library, 2004.

[20] Jeffrey S Rosenthal. "Parallel computing and Monte Carlo algorithms". In: *Far east journal of theoretical statistics* 4.2 (2000), pp. 207–236.

[21] Robert H Swendsen and Jian-Sheng Wang. "Replica Monte Carlo simulation of spin-glasses". In: *Physical Review Letters* 57.21 (1986), p. 2607.

[22] Douglas N VanDerwerken and Scott C Schmidler. "Parallel markov chain monte carlo". In: *arXiv preprint arXiv:1312.7479* (2013).

[23] Darren J Wilkinson. "Parallel bayesian computation". In: *Statistics Textbooks and Monographs* 184 (2006), p. 477.

초록

인공신경망과 같은 많은 계산을 요하는 모형이 다양한 분야에서 효과적임이 드러남에 따라 형렬 연산을 병렬처리 하기 위해 그래픽카드(GPU) 상 에서 계산하는 것이 일반화되고 있으며 이를 위해 계산을 여러 스레드로 나누는 방법을 찾는 것이 중요해지고 있다. 본 논문은 분할 된 샘플 공간에서 브리지 샘플링과 해밀토니안 몬테카를로를 결합하여 여러 GPU에 분산될 수 있는 새로운 MCMC 알고리즘을 제시한다. 이 접근법은 베이지안 뉴럴 네트워크 (Bayesian Neural Network)와 같은 타겟 분포에 대한 MCMC 샘플링을 빠르게 할 수 있다. 또한 다중 모달(Multimodality)이 존재할 때 이 알고리즘은 낮은 확률 영역에서도 샘플링을 효율적으로 잘 할수 있는것으로 나타났다. 마지막으로 이 논문은 Adam Optimizer, 해밀토니안 몬테카를로와 같은 다른 학습 방법의 변수 분포와 본 알고리즘의 변수분포를 비교함으로써, 제한된 표본 공간이 일반화 오차에 미치는 영향에 대한 추가 연구가 수행 될 수 있음을 제시한다.

주요어: 해밀토니안 몬테카를로, 브릿지 샘플링, 베이지안 신경망, GPU, 이미지 분류, 병렬 몬테카를로

학번: 2016-20260