Stochastic Gradient Hamilton Monte Carlo Implementation and Application

STA 663 Final Project at Duke

Steve Shao, Ziqian Fu

April 24, 2019

Abstract

Hamiltonian Monte Carlo (HMC) sampling is a MCMC sampler which involves intuition of hamiltonian dynamics and provides a sampling algorithm with high acceptance probabilities. Using the gradient information for the proposals would potentially lead to more efficient exploration of the target parameter space. However, in the case of a huge sample size, HMC can be quite limited or even infeasible for it requires the evaluation of the gradient of the potential energy function over the whole dataset, which is intimidatingly expensive. Recent developmennt in stochastic gradient MCMC methods provides illustrating ideas on how to solve such computational difficulties. With careful design, the SGMCMC methods can also maintain the desired target distribution with minimum computational resources. Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) resorts to second-order Langevin dynamics with a friction term that counteracts the effects of the noisy gradient and gets the desired target distribution.

In this project, we implemented SGHMC in Python, optimized its performance using C++ wrapped critical functions, carried out some simulation work to test the algorithm's performance and efficiency, did bayesian inference based on SGHMC for a real dataset and lastly compared SGHMC with several rivals of it for some insights.

Key words: Hamiltonian Monte Carlo, Stochastic Gradient Hamiltonian Monte Carlo, Stochastic Gradient MCMC, Bayesian Inference, Hamiltonian dynamics, Langevin dynamics

1. Background

We are implementing the sampling algorithm SGHMC from the paper Stochastic Gradient Hamiltonian Monte Carlo by Tianqi Chen, Emily B. Fox, Carlos Guestrin. This paper proposed a stochastic gradient MCMC sampling algorithm, which can largely reduce the computational cost of Hamiltonian Monte Carlo.

1.1 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) (Neal, 2010) is a MCMC sampling algorithm that resorts to Hamiltonian dynamics for an efficient proposal procedure. Hamiltonian dynamics preserves energy and can be used to produce distant proposals for the Metropolis algorithm, thus leading to efficient exploration of the state space. Besides the intuition, further into the math, suppose our target distribution, the distribution we want to sample from is $p(\theta|X) \propto p(\theta)p(X|\theta)$. Although our target would not necessarily be a posterior distribution, we still want to illustrate the math by doing this (mostly because HMC is often used in a Bayesian statistics context and partly because we are here at Duke).

The target distribution can be related to a potential energy function by the concept of a *canonical distribution* in statistical mechanics, which basically says that: the probability of being in a energy state $E(x)$ has the following distribution.

$$P(x) \propto \exp(-E(x)/T) \tag{1}$$

where T is the temperature of the system. Since we are less focused on physics, lets assume $T = 1$. And in Hamiltonian dynamics setting, the energy function is the summation of two independent part, *potential energy*, $U(p)$ which only depends on **position** p, and *kinetic energy*, $K(r)$ which only depends on **momentum** (or speed) r. We can let the momentum energy have the form $U(\theta) = -\log p(\theta) - \log p(X|\theta)$, then by the canonical distribution.

$$\begin{aligned} P(\theta, r) &\propto \exp(-H(\theta, r)) \\ &= \exp(-U(\theta) - K(r)) \\ &= \big(p(\theta)p(X|\theta)\big)\exp(-K(r)) \\ &\propto p(\theta|X)\exp(-\frac{r^T M^{-1} r}{2}) \end{aligned} \tag{2}$$

The invariance of $H(\theta, r)$ under Hamiltonian dynamics means that a Hamiltonian trajectory will move within a hyper-surface of constant probability density. Then we can take random initial position in the parameter space of (θ, r) and move by Hamiltonian dynamics and throw away the auxiliary variable r. We are still guaranteed (if discretized properly) to generate from the correct marginal distribution (our target), $p(\theta|X)$.

1.2 Stochastic Gradient HMC

Two problems with HMC are that 1) computation of gradient information over the whole dataset can be expensive; 2) we need a Metropolis Hastings accept-reject procedure to correct the discrepency from approximating differential equations by discretization. These problems mean that HMC can be intimidatingly costly with a huge sample size.

Therefore, the authors take some ingredients from Stochastic Gradient Markov Chain Monte Carlo (SGMCMC) theorem and add them to HMC and get SGHMC. In HMC, we have $P(\theta, r) \propto p(\theta|X)\exp(-\frac{r^T M^{-1} r}{2})$ and by Hamiltonian dynamics we have

$$\begin{cases} d\theta = & M^{-1} r dt \\ dr = & -\nabla U(\theta) dt \end{cases} \tag{3}$$

where $U = -\sum_{x \in \mathcal{D}} \log p(x|\theta) - \log p(\theta)$.

In SGHMC, to avoid costly computation of the gradient of all samples, we use a minibatch of reasonable size to approximate the true gradient by

$$\nabla \tilde{U}(\theta) = -\frac{|\mathcal{D}|}{|\tilde{\mathcal{D}}|} \sum_{x \in \mathcal{D}} \log p(x|\theta) - \log p(\theta), \tilde{\mathcal{D}} \subset \mathcal{D} \tag{4}$$

It will potentially address the problem of costly computation. But unfortunately, the author have proved in the paper that using such a stochastic gradient without further careful correction can lead to arbitrarily bad results. So they come up with the idea of adding a "friction" term in the differential equation to counter the random noise introduced by using the stochastic gradient. By adding the "friction" term, the energy of the system is conserved and we can again sample from the correct distribution. This is the idea and more details will be covered in the **Algorithm** part.

As the authors argue, if we are doing things in the right way, the SGHMC can potentially

- avoid computing gradients over the whole dataset
- give us the desired distribution withou M-H correction. Again, we don't have to iterate over the whole dataset to evaluate the potential energy function.

On the application scale, both HMC and SGHMC sampling algorithms can be implemented in a Bayesian inference setting, where very often we need to sample from the posterior and do statistical inference. More specifically, we can use them to do Bayesian regression, either linear, generalized or penalized and we can also use them in complicated Bayesian deep learning stuffs like Bayesian Neural Networks and Bayesian GANs to help use sample more efficiently.

For traditional MCMC samplers, Gibbs sampler is often limited since we can not always get a tractable full conditional of our parameters and M-H algorithm can be very inefficient with a small effective sample size. Both HMC and SGHMC can propose much better samples in the M-H framework and subsequently explore the parameter space more efficiently. And SGHMC is even better in that it is substantially more computationally friendly. We are still exploring the possibilities of SGHMC. But obviously, they are really helpful for researches in deep neural networks in a Bayesian framework.

2. Algorithm

As explained in the previous part, SGHMC can explore the parameter space of our target posterior distribution and get very efficient (high effective sample size) samples from the posterior. Similar to but not exactly the same as in HMC, it uses some dynamics (gradient information) to guide our exploration in the parameter space. In HMC, we are like having a frictionless moving back and forth (Hamiltonian dynamics). No energy loss, only transformation between potential energy and kinetic energy. SGHMC uses approximation of gradient, which is analogous to blowing a wind in our original system. Thus, energy is not conserved for such turbulence caused by noise from approximating the gradient. In SGHMC, we add some friction to the originally frictionless system and by the second order Langevin dynamics, the energy is again conserved.

Let's then go through the algorithm more mathematically. Recall that in equation (3) earlier, we have that beautifully balanced differential equation of energy. However, when we use a stochastic gradient $\nabla\tilde{U}(\theta)$ to approximate $\nabla U(\theta)$, the gradient noise is introduced and the system is no longer balanced. Instead, we can model such uncertainty by $\nabla r = -\epsilon\nabla\tilde{U}(\theta) = -\epsilon\nabla U(\theta) + \mathcal{N}(0, \epsilon^2 V)$ where V is the uncertainty from using stochastic gradient approximation. Then the differential equation in (3) becomes:

$$\begin{cases} d\theta = & M^{-1}rdt \\ dr = & -\nabla U(\theta)dt + \mathcal{N}(0, 2B(\theta)dt) \end{cases} \tag{5}$$

where $B(\theta) = \frac{1}{2}\epsilon V(\theta)$. The energy is obviously not conserved and the sampler breaks.

Then from the intuition explained earlier the authors fixed it by adding a friction term to the above equation and made it a second-order Langevin dynamics. Energy is conserved again and also we can use a more computationally friendly gradient $\nabla\tilde{U}$. By adding the friction term to counter the random noise, we get

$$\begin{cases} d\theta = & M^{-1}rdt \\ dr = & -\nabla U(\theta)dt - BM^{-1}rdt + \mathcal{N}(0, 2B(\theta)dt) \end{cases} \tag{6}$$

By using a estimated B, the dynamics is reformulated as

$$\begin{cases} d\theta = & M^{-1}rdt \\ dr = & -\nabla U(\theta)dt - CM^{-1}rdt + \mathcal{N}(0, 2(C - \hat{B})dt) + \mathcal{N}(0, 2Bdt) \end{cases} \tag{7}$$

The authors have proved in the paper and the supplementary material that sampling from such a system would give us the correct stationary distribution.

Then based on this setting, the authors set up 2 ways of practically implement SGHMC, which are proved to be equivalent.

SGHMC algorithm

The first scheme is straightforward from equation (7)

Algorithm 1: Stochastic Gradient HMC

for *t= 1,2,...* **do**
  resample momentum r as
  $r^{(t)} \sim \mathcal{N}(0, M)$
  $(\theta_0, r_0) = (\theta^{(t)}, r^{(t)})$
  simulate dynamics in equation (7): **for** *i= 1,...,m* **do**
    $\theta_i \leftarrow \theta_{i-1} + \epsilon_t M^{-1} r_{i-1}$
    $r_i \leftarrow r_{i-1} - \epsilon_t \nabla \tilde{U}(\theta_i) - \epsilon_t C M^{-1} + \mathcal{N}(0, 2(C - \hat{B})\epsilon_t)$
  end
  $(\theta^{(t+1)}, r^{(t+1)}) = (\theta_m, r_m)$, no M-H
end

where we can prespecify $C \succeq \hat{B}$ and $\hat{B} = \frac{1}{2}\epsilon\hat{V}$ and $\hat{V}$ is the empirical Fisher information. According to the Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring, we can compute V as

$$\hat{V}(\theta_t) = \frac{1}{n-1}\sum_{i=1}^{n}\{g_{t_i}(\theta_t) - \bar{g}_n(\theta_t)\}\{g_{t_i}(\theta_t) - \bar{g}_n(\theta_t)\}^T$$

where n is the size of minibatch we use, $g_{t_i}(\theta_t)$ is the gradient of nimibatch samples $\{x_{t_1}, ..., x_{t_n}\}$ and $\bar{g}_n(\theta_t)$ is the mean.

The second scheme is basically the same. We just change the parametrization a little to resemble that of a SGD with momentum and can use some experience of parameter tuning from SGD with momentum. let $v = \epsilon M^{-1} r$, define $\eta = \epsilon^2 M^{-1}, \alpha = \epsilon M^{-1} C, \hat{\beta} = \epsilon M^{-1}\hat{B}$, the update rules are as follows:

$$\begin{cases} \Delta\theta = & v \\ \Delta v = & -\eta\nabla\tilde{U}(\theta) - \alpha v + \mathcal{N}(0, 2(\alpha - \hat{\beta})\eta) \end{cases} \tag{8}$$

As the authors noted in the paper, a simple choice would be setting $\hat{B}$ or $\hat{\beta}$ to be 0 and depend on small choice of ϵ, which would still give us the desired stationary distribution.

3. Code Optimization

First we implemented the algorithm literally by the steps in **Algorithm 1**. But the performance is not so statisfactory at the beginning and it took minutes to get 1500 samples with a batch size of 500. To get the code to perform better and make the algorithm potentially more practically feasible for very large dataset, we made several improvements.

3.1 Use the simpler parametrization of algorithm

The very first change we did is change our update rules from equation (7) to equation (8). They are equivalent, but the second parametrization would avoid doing unnecessary computation of matrix multiplication for the

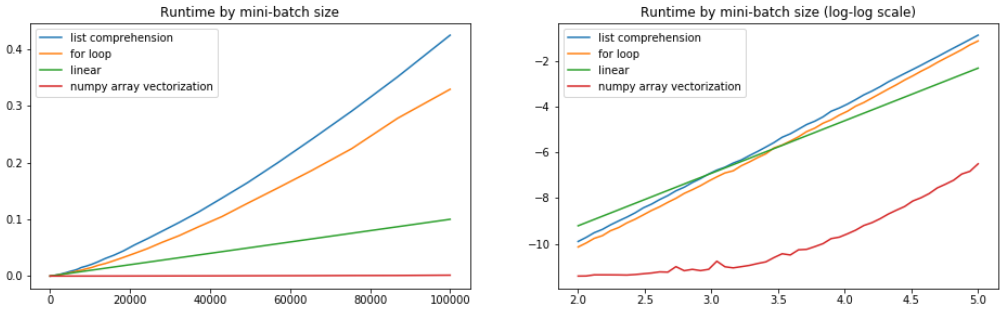


Figure 1: Runtime comparison among three ways of computation

inverse mass matrix M^{-1}. Besides, we can similarly explain the parameters as in SGD with momentum, which would give us some insights about how to appropriately tune them.

3.2 Vectorization using naive `numpy`

At the very first, we built our code in python using many forloops and list comprehensions, which surely are very inefficient. So we figured out that we could vectorize some of our functions by using `numpy` version of computation, which would be potentially be much more efficient.

We setup a easy sampling scheme of some posterior under univariate normal (see details in the Github repository), where we compared the performance of the our gradient estimate function `gradU` using all three ways, and it turn that in our example

- For *forloop* version, it takes $225 \pm 9.55\mu s$ to calculate such a gradient estimate with batch size of 500.
- For *list comprehension* version, it takes $162 \pm 2.5\mu s$
- For *numpy array* vectorized version, it takes only $9.36\mu s \pm 305ns$, which is substantially better.

Besides focusing on only one slice of batch size, we tested the performance of three ways for different batch sizes.

As the Figure 1 shows, the implication should be straightforward:

- Using numpy array built-in vectorization would largely reduce runtime of calculation quantities like gradient $\nabla \tilde{U}$ and empirical Fisher information V.
- Judging from the log-log scale plot,the runtimes of both forloop and list comprehension increase approximately polynomially. The improvement of using `numpy array` vectorization would be even greater in large batch sizes.

So we changed everything from forloop to vectorized computation if possible.

3.3 Using `Cython` and `pybind11`

Furthermore, we used `cython` to help us navigate the bottleneck of our code and found that the updating part (differential discretization) is not somewhat slow.But unfortunaly, we don't see how we could do parallelization and distributed algorithm since any MCMC algorithm is inherently sequential, so we tried in another direction. We resorted to `pybind11` to wrap some of our critical functions in `C++` and explicitly prespecified the data structure for all the arguments.

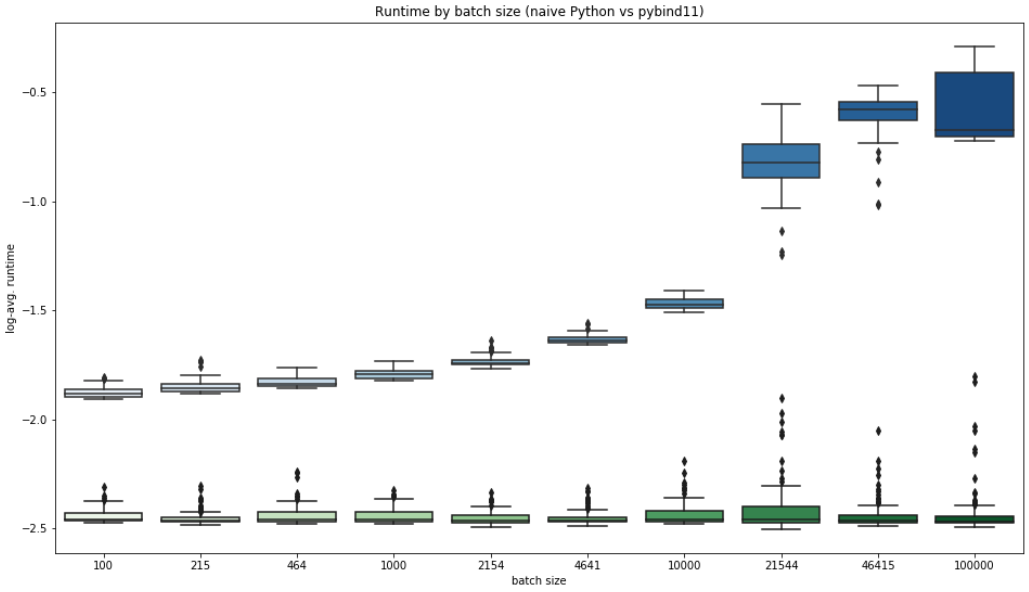


Figure 2: Runtime comparison between naive Python (blue) and pybind11 (green)

And along with the vectorization in `Eigen` library of `C++`, we get some really good `C++` code to calculate our crucial updating steps and wrapped and imported them into python by `pybind11`. Similarly, we did a comparison test between naive python version and pybind11 version.

Figure 2 shows that we are essentially doing better with the help of `pybind11`. And surprisingly, as the Python version's time complexity increases with the batch size, we found that there is no significant increase on the runtime of pybind11 version.

4. Simulation

To check the correctness of our algorithm, we set up 2 simulation scenarios of Bayesian inference. We want focus on the inference of the mean parameter of some normal distribution and setup a Bayesian statistical framework to do inference about the posterior under conjugate prior where we can easily calculate the theoretical posterior distribution of our parameter of interest.

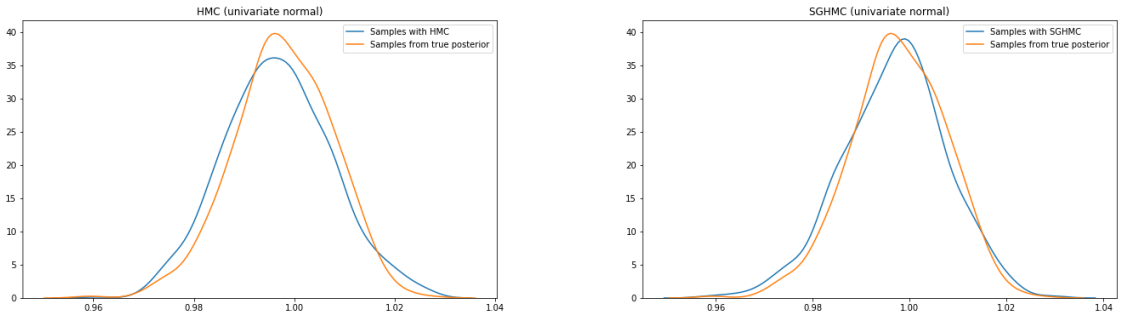
Then we compare the samples from HMC/SGHMC to those simulated from the true posterior. If they are reasonably close to each other, it might suggest that we are doing it the right way.

4.1 Univariate normal case

First we set up a univariate normal case where we specify the model as

$$\mu \sim N(0, 1)$$
$$X|\mu \sim N(\mu, 1)$$

and we simulated data $\{x_i\}_{i=1}^n$ from $N(1, 1)$. Then for conjugacy, the posterior distribution can be easily derived by



(a) HMC

(b) SGHMC

Figure 3: Simulation results for univariate normal

$$\mu|X \sim N(\mu_p, \sigma_p^2)$$

where $\sigma_p^2 = (1/1 + n/\bar{V}(x_i))^{-1}, \mu_p = \frac{0/1+\bar{x}\times n/\bar{V}(x_i)}{\sigma_p^2}$.

We get 1000 posterior samples with HMC and SGHMC and compared then with 1000 samples from the true posterior.

As Figure 3 shows, the samples with HMC and SGHMC are reasonably close to that of the true posterior's. Also we find that SGHMC is slighter better than HMC.

4.2 Bivariate normal case

Then we set up a bivariate normal case where we specify the model as

$$\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$X|\boldsymbol{\mu} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

where $\Sigma = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}$ and we simulated data $\{\mathbf{x}_i\}_{i=1}^n$ from $\mathcal{N}((1,-1)^T, \Sigma)$. Then for conjugacy, the posterior distribution can be easily derived by

$$\boldsymbol{\mu}|X \sim \mathcal{N}(\boldsymbol{\mu}_p, \Sigma_p)$$

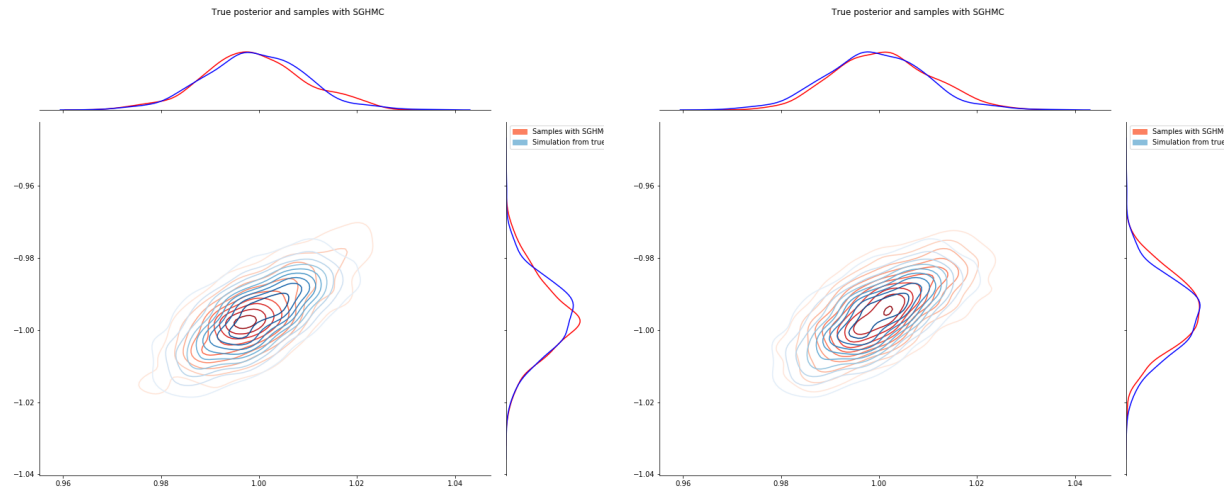
where $\Sigma_p = (\mathbf{I}^{-1} + n/\bar{\Sigma}(x_i))^{-1}, \mu_p = \Sigma_p^{-1}(0 + n\bar{\Sigma}^{-1}(\mathbf{x}_i)\bar{\mathbf{x}})$.

We get 1000 posterior samples with SGHMC from 2 sampling schemes, from equation (7) and equation (8) and compared then with 1000 samples from the true posterior.

As Figure 4 shows, the samples with bothe SGHMC update rules are close to those from the true posterior's. Thus we could see the correctness of SGHMC.

5. Real data analysis

Besides the simulation, we also applied SGHMC to a real data analysis for further test of the algorithm.



(a) SGHMC (update by equation 7)

(b) SGHMC (update by equation 8)

Figure 4: Simulation results for bivariate normal

Dataset

We chose the `Boston housing price` dataset, which is usually used for regression problems and is accessible both on Kaggle and from scikit-learn package. This dataset essentially describes the housing values in suburbs of Boston. This dataset has 14 features and 506 entries, in which `medv` median value of owner-occupied homes in $1000s is usually regarded as the target we would like to predict. We would like to do some Bayesian inference on this dataset and get some insights about the effects of all the features on the housing price in Boston Suburbs.

Models

For the models, we chose to use a Bayesian Linear Regression (with conjugate g-prior) and Bayesian Lasso (BLR with Laplace prior) to fit the dataset and use SGHMC sampler to draw posterior samples of our coefficients.

For BLR with g-prior, since it's a conjugate setting, derivng the posterior would not be hard. Consider a dataset $\{x_i, y_i\}_{i=1}^n$, the regression model is formulated as

$$y_i = x_i^T \beta + \epsilon_i$$

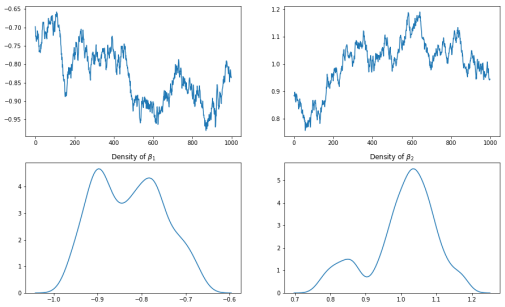
We put on a Zellner's g-prior for β and assume the prior precision ψ is known, where

$$\beta|\psi \sim \mathcal{N}(\beta_0, g\psi^{-1}(X^T X)^{-1})$$

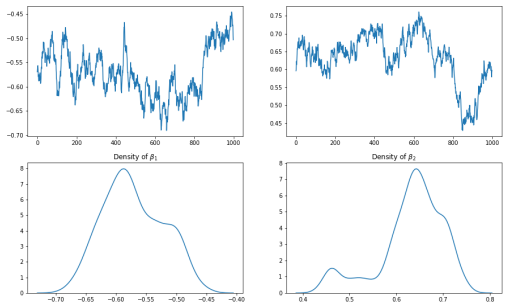
The posterior of β can be easily derived as

$$\beta|\psi, X, y \sim \mathcal{N}(\tilde{\beta}, \tilde{\Sigma})$$

where $\tilde{\beta} = q\hat{\beta}_{ols} + (1-q)\beta_0, \tilde{\Sigma} = q\psi^{-1}(X^T X)^{-1}$ and $q = g/(1+g)$.



(a) BLR SGHMC



(b) BLASSO SGHMC

Figure 5: Diagnostics for the first two coefficients

Since we have the posterior, everything in SGHMC is clean and straightforward enought. We can get some posterior samples of β to do Bayesian inference.

For Bayesian Lasso, it's essentially the same as BLR. However, since we are using a Laplace distribution as our prior, we no longer have the nice conjugacy. Then we can use SGHMC to help us explore the parameters' posterir distribution without bothering to construct a Gibbs sampler or M-H sampler. In BLASSO, when we fixed the precison ψ and penalty λ, the energy function could be written as

$$U(\beta) = \frac{1}{2}\psi(y - X\beta)^T(y - X\beta) + \lambda\sum_{j=1}^{p}|\beta_j|$$

and the derivative is tractable with the form, $\nabla U(\beta) = \psi X^T(y - X\beta) + \lambda \text{sign}(\beta)$. So we can use SGHMC to draw posterior samples and do inference.

Results

First, we did some MCMC diagnostics for our SGHMC models. Since there are 13 covariates, we just selected the first two coefficents and their traceplots and posterior densities (after deleting burn-in samples) are as in Figure 4

From the traceplot, we could see the Markov Chain is mixing fairly well, suggesting that we might reached stationarity.

Besides, the shapes of the posterior distributions of BLR and BLASSO are quite similar. But their supports are slightly different, which might be due to the different shrinkage brought by BLR and BLASSO.

As we can see, the first two covariates **crim** (per capita crime rate by town) and **zn** (proportion of residential land zoned for lots over 25,000 sq.ft.), respectively, have a significant negative impact and a significant positive impact on the housing price in both models.

Based on the dataset, we did a 5-fold cross validation on 5 models we have developed so far

- OLS
- BLR with HMC
- BLR with SGHMC
- BLASSO with HMC

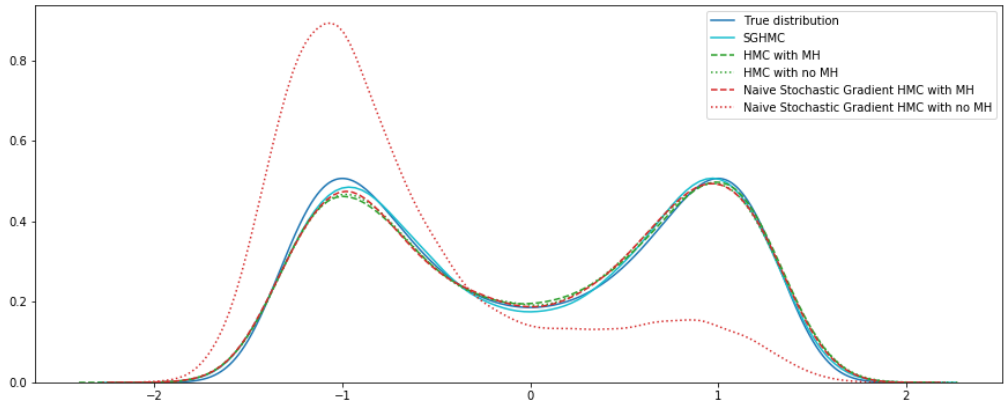


Figure 6: Empirical distributions associated with various sampling algorithms

- BLASSO with SGHMC

At the end of the day, we documented the CV-based RMSE of each model to compare each model's predictive performance.

Model	OLS	BLR_HMC	BLR_SGHMC	BLASSO_HMC	BLASSO_SGHMC
RMSE	4.8684	4.8843	4.8674	4.6825	4.9248

We can see that all 5 models are doing a decent job. Bayesian Lasso with HMC seems to be the winner here. We would probably suggest the Bayesian models since they can better capture not only the uncertainty and the distribution of our coefficients as well.

6. Comparative analysis

We replicated the one of the comparative simulation in the original paper, where we would use several sampling methods to approximate the true target distribution with potential energy $U(\theta) = -2\theta^2 + \theta^4$.

Figure 6 shows the result of this simulation.

From the plot, we can see that

- out of all sampling algorithms, SGHMC seems to have obtained the best approximation
- HMCs with or withour M-H correction are both doing a decent job.
- But for naive SGHMC (no friction correction), if we don't do M-H correction, the resulting posterior distribution would be way off what we are expecting.

7. Conclusion

SGHMC combines the benefits of using dynamics to guide our exploration in HMC and the merits of efficient computation in stochastic gradient MCMC methods. SGHMC serves as a very powerful tool in Bayesian inference, by which we can obtain samples of high-quanlity efficiently.

In this project, we implemented HMC, SGHMC and a simple version of SGLD in python. Then we did some analysis on the codes' performance in terms of runtime and optimized our codes by using numpy vectorization, C++ wrapped critical function, etc. Next, we did 2 simulation studies under the hood of simple Bayesian statistics, verifying the correctness of the algorithms. After that, a real dataset was analyzed using Bayesian linear regression and Bayesian Lasso with SGHMC sampler, which provides a bit insight on the possible application of SGHMC.

At the end of the day, through this project, we get ourselves faimilar with basic concepts of MCMC in a more convoluted context. Also, we had chance to touch on some stochastic MCMC methods like SGHMC, SGLD and many others as presented in A Complete Recipe for Stochastic Gradient MCMC. It's such a great experience of learning and sharing thoughts.

Reference

[1] T. Chen, E.B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In Proceeding of 31st International Conference on Machine Learning (ICML'14), 2014.

[2] Neal, Radford M. MCMC using Hamiltonian dynamics. Handbook of Markov Chain Monte Carlo 2, 2011.

[3] Welling, M. and Teh, Y.W. Bayesian learning via stochastic gradient Langevin dynamics. In Proceedings of the 28th International Conference on Machine Learning (ICML11), June 2011.

[4] Michael Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. arXiv preprint arXiv:1701.02434, 2017.

[5] Yi-An Ma, Tianqi Chen, and Emily B Fox. A complete recipe for stochastic gradient MCMC. In Advances in Neural Information Processing Systems, 2015.

[6] Changyou Chen, Nan Ding, and Lawrence Carin. On the convergence of stochastic gradient MCMC algorithms with high-order integrators. Advances in Neural Information Processing Systems, 2015.

[7] Xuefeng Gao, Mert G¨urb¨uzbalaban, and Lingjiong Zhu. Global convergence of stochastic gradient hamiltonian monte carlo for non-convex stochastic optimization: Non-asymptotic performance bounds and momentum-based acceleration. arXiv preprint arXiv:1809.04618, 2018

[8] Box, G. E. P., and Tiao, G. C. , Bayesian Inference in Statistical Analysis, Addison- Wesley, 1973.

[9] Trevor Park and George Casella. The Bayesian Lasso, Journal of the American Statistical Association, 2008.

[10] Ahn, S., Balan, A. K., and Welling, M. Bayesian posterior sampling via stochastic gradient Fisher scoring. In ICML, 2012.

[11] M. Mendoza, A. Allegra, and T. P. Coleman. Bayesian Lasso posterior sampling via parallelized measure transport. arXiv:1801.02106, 2018.