A Python Implementation of Stochastic Gradient Hamiltonian Monte Carlo

Jiajun Song JIAJUN.SONG@DUKE.EDU
Yiping Song YIPING.SONG@DUKE.EDU

Abstract

This project implemented the Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) algorithm (Chen et al., 2014). The sampling method is useful as it allows HMC sampling to use noisy gradient terms cacluated through *minibatches* of the data provided. Due to the fact that we are increasinglt faced with datasets have millions to billions of observations or where data comes in as a scream, the mini batching approach becomes more suitable and appropriate. The project will include both Python and multiple optimization approaches on simulated and real data. An example about using data extracted from a simulated two-component mixture of normals was also included.

1. Background

Hamiltonian Monte Carlo, referred as the HMC sampling method, provides a mechanism for defining distant proposals with high acceptance probabilities in a Metropolis-Hastings framework. The method defines a Hamiltonian function in terms of the target distribution from which we desire samples —the *potential energy*— and a *kinetic* energy term parameterized by a set of momentum auxiliary variables, making it advantageous for the sampling probability distribution, especially with high correlation. Based on the attractive properties of HMC in terms of rapid exploration of the state space, HMC methods have grown in popularity recently.

In the paper, Chen developed a variant HMC that sidesteps the computational limitation of the HMC method, which emanates from the computation of the gradient when the data is significant. Chen uses a stochastic gradient and introduces a friction term that allows the algorithm to maintain the desired target distribution and invariance properties.

Chen proposed that stochastic gradient Hamiltonian Monte Carlo algorithm is an efficiency improvement to traditional HMC orthogonal and would enable a large-scale and online Bayesian sampling algorithm with the potential to rapidly explore the posterior.

Final Project for STA 663, Duke University, 2020. Refer to https://github.com/JiajunSong629/SGHMC for source code and implementation details.

2. Descriptions of Algorithms

2.1. Hamiltonian Monte Carlo

Suppose we want to sample from the posterior distribution of θ given a set of independent observations $x \in \boldsymbol{D}$:

$$p(\theta|\boldsymbol{D}) \propto \exp(-U(\theta)),$$

where the potential energy function U is given by

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

In a Hamiltonian system, we consider particles with position θ and momentum r. The total energy of the system, $H(\theta, r) = K(r) + U(\theta)$, where K is the kinetic energy and U is the potential energy, is conserved. Such a system satisfies the following Hamiltonian equations.

$$\begin{cases} \dfrac{d\theta}{dt} = \dfrac{dK}{dr} \\ \dfrac{dr}{dt} = -\dfrac{dU}{d\theta} \end{cases}$$

We normally take $K(r) = \frac{1}{2} r^T M^{-1} r$. To propose samples, HMC simulates the Hamiltonian dynamics

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

A common approach to construct a discretized system to simulate samples is via *leapfrog* method, which is outlined in Alg.1. Because of inaccuracies introduced through the discretization, an MH step must be implemented.

We can see that HMC needs to compute the gradient of the potential energy function in order to simulate the Hamiltonian dynamical system and the MH step is also costly. However we are increasingly faced with datasets having millons to billions of observations or where data come in as a stream and we need to make inferences online. SGHMC

Algorithm 1 Hamiltonian Monte Carlo

```
Input: Start position $\theta^{(1)}$ and step size $\epsilon$
for $t = 1, 2 \dots$ do
    Resample momentum $r$
    $r^{(t)} \sim N(0, M)$
    $(\theta_0, r_0) \sim (\theta^{(t)}, r^{(t)})$
    Simulate discretization of Hamiltonian dynamics
    $r_0 \leftarrow r_0 - \frac{\epsilon}{2}\nabla U(\theta_0)$
    for $i = 1$ to $m$ do
        $\theta_i \leftarrow \theta_{i-1} + \epsilon M^{-1} r_{i-1}$
        $r_i \leftarrow r_{i-1} - \epsilon \nabla U(\theta_i)$
    end for
    $r_m \leftarrow r_m - \frac{\epsilon}{2}\nabla U(\theta_m)$
    $(\hat{\theta}, \hat{r}) = (\theta_m, r_m)$
    Metropolis-Hasting correction:
    $u \sim U[0, 1]$
    $\rho = e^(H(\hat{\theta}, \hat{r}) - H(\theta^{(t)}, r^{(t)}))$
    if $u < \min(1, \rho)$, then $\theta^{(t+1)} = \hat{\theta}$
end for
```

attempts to marry the efficiencies in state space exploration of HMC with the big-data computational efficiencies of stochastic gradients. The paper proposed that SGHMC would enable a large-scale and online Bayesian sampling algorithms with the potential to rapidly explore the posterior.

2.2. Stochastic Gradient HMC

Consider a noisy estimate based on minibatch $\tilde{D}$ sampled uniformly at random from D

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

Assume that observations x are independent and, appealing to the central limit theorem, we can approximate noisy gradient as $\nabla \tilde{U}(\theta) \approx \nabla U(\theta) + N(0, V(\theta))$. As the size of $\tilde{D}$ increases the Gaussian approximation becomes more accurate.

A naive approach would be

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

where $B(\theta) = \frac{1}{2}\epsilon V(\theta)$ is the diffusion matrix contributed by stochastic gradient noise. However, the paper showed that with the noisy stochastic gradient, the Hamiltonian function is no longer invariant, which requires a frequent costly MH correction step, or alternatively, long simulation runs with low acceptance probability. To minimize the effect of the injected nosie on dynamics, the paper considered

Algorithm 2 Stochastic Gradient HMC

```
for $t = 1, 2 \dots$ do
    optionally resample momentum $r$
    $r^{(t)} \sim N(0, M)$
    $(\theta_0, r_0) \sim (\theta^{(t)}, r^{(t)})$
    simulate with friction terms
    for $i = 1$ to $m$ do
        $\theta_i \leftarrow \theta_{i-1} + \epsilon M^{-1} r_{i-1}$
        $r_i \leftarrow r_{i-1} - \epsilon_t \nabla \tilde{U}(\theta_i) - \epsilon_t C M^{-1} r_{i-1}$
                $+ N(0, 2(C - \hat{B})dt)$
        $(\theta^{t+1}, r^{t+1}) = (\theta_m, r_m)$
    end for
end for
```

adding a *friction* term to the momentum update:

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

The paper showed that the dynamics given by the equations above have a similar invariant property to that of original Hamiltonian dynamics. The resulting method is referred as SGHMC.

In practice, the diffusion matrix $B(\theta)$ will be unknown. The paper introduce a user specified friction term $C \succeq B$ and consider the following dynamics

$$\begin{cases} d\theta = M^{-1} r dt \\ dr = -\nabla U(\theta) dt - C M^{-1} r dt \\ \qquad + N(0, 2(C - \hat{B})dt) + N(0, 2B(\theta) dt) \end{cases}$$

The resulting algorithm is outlined in Alg.2.

2.3. Connection to SGD with momentum

The paper further pointed out a relationship between SGD with momentum and SGHMC. Letting $v = \epsilon M^{-1} r$, and rewrite the update rule in Alg.2 as

$$\begin{cases} \theta_i \leftarrow \theta_{i-1} + v_{i-1} \\ v_i \leftarrow v_{i-1} - \epsilon^2 M^{-1} \nabla \tilde{U}(\theta_i) - \epsilon C M^{-1} v_{i-1} \\ \qquad + N(0, 2\epsilon^3 M^{-1}(C - \hat{B}) M^{-1}) \end{cases}$$

Define $\eta = \epsilon^2 M^{-1}, \alpha = \epsilon M^{-1} C, \hat{\beta} = \epsilon M^{-1} \hat{B}$. The update rule become

$$\begin{cases} \theta_i \leftarrow \theta_{i-1} + v_{i-1} \\ v_i \leftarrow (1 - \alpha) v_{i-1} - \eta \nabla \tilde{U}(\theta_i) + N(0, 2(\alpha - \hat{\beta})\eta) \end{cases}$$

Comparing to an SGD with momentum method, it is clear equations above that η corresponds to the learning rate and

Algorithm 3 Stochastic Gradient HMC Reparametrized

```
for $t = 1, 2 \ldots$ do
  optionally resample momentum $v$
  $v^{(t)} \sim N(0, M)$
  $(\theta_0, v_0) \sim (\theta^{(t)}, v^{(t)})$
  simulate with friction terms
  for $i = 1$ to $m$ do
    $\theta_i \leftarrow \theta_{i-1} + \nu_{i-1}$
    $v_i \leftarrow (1-\alpha)v_{i-1} - \eta\nabla\tilde{U}(\theta_i) + N(0, 2(\alpha - \hat{\beta})\eta)$
    $(\theta^{(t+1)}, v^{(t+1)}) = (\theta_m, v_m)$
  end for
end for
```

$1-\alpha$ the momentum term. We can actually use this equivalent update rule to run SGHMC and borrow experience from parameter settings of SGD with momentum to guide choice of SGHMC settings. Resulting algorithm is outline in Alg.3. More relevent details about paramter settings are in Section 2.4 below.

2.4. Parameters settings of SGHMC

As we have discussed above, in analogy to SGD with momentum, we call η the learning rate and $1-\alpha$ the momentum term. This equivalent update rule is cleaner and recommended in the paper.

The $\hat{\beta}$ term corresponds to the estimation of noise that comes from the gradient. One simple choice is ignore the gradient noise by setting $\hat{\beta} = 0$ and relying on small ϵ. This is the default value for $\hat{\beta}$ in our experiments on simulated data set and it works well. We can also set $\hat{\beta} = \eta\hat{V}/2$, where $\hat{V}$ is estimated using empirical Fisher information.

Define $\beta = \epsilon M^{-1}B = \eta V(\theta)/2$ to be the exact term induced by stochastic gradient noise. Then we have

$$\beta = O(\eta\frac{|D|}{|\tilde{D}|}I)$$

where *I* is Fisher information matrix of the gradient, $|D|$ is the size of training data, $|\tilde{D}|$ is size of minibatch, and η is the learning rate. We want to keep β small so that the resulting dynamics are governed by the user-controlled term and the sampling algorithm has a stationary distribution close to the target distribution. However there is no free lunch here: as the training size gets bigger, we can either set a small learning rate $\eta = O(\frac{1}{|D|})$ (Large η can cause divergence, since we need to multiply η by $\nabla\tilde{U}$) or use a bigger minibatch size $|\tilde{D}|$.

The paper proposed that using a minibatch size of 500 and fixing α to a small number (e.g. 0.01 or 0.1) works well in practice. The learning rate can be set as $\eta = \gamma/|D|$, where γ usually set to 0.1 or 0.01. This method of setting parameters is also commonly used for SGD with momentum.

3. Describe optimization for performance

We apply multiple methods to optimize SGHMC, specifically on the simulated data that generates Fig. 1. In Table. 1, the different results are the time taken in *ms* to run 1000 samples and 10 leapfrog steps on a noisy gradient. In *Cython 2* and *C++* the gradient function is optimized while in former ones are not. Though the derivation of the algorithm takes multiple steps to arrive at the final algorithm, its implementation is quite simple, as is outlined in Alg.3. As the algorithm is a MCMC sampler, parallelization is not possible because we have dependence within each chain. An option could be to run multiple chains and parallelize over the number of chains, but this would only give gains up to the number of chains run rather than improving the speed of the underlying algorithm itself. Hence here we focus on the bottleneck of SGHMC in terms of speed.

From profiling results of the plain numpy implementation we see that *np.dot*, *np.random.randn*, *lambda* function of the gradient are called the most times during the execution. It enables us to focus on the part of the function that needs improvement. JIT with *Numba* does not seem to have significant improvement. Most likely because the function here is not completely in explicit loop. The deteriorated performance of the JIT-compiled version here may be due to the overhead of JIT outweigh the benefits.

Also the profiling results indicate that the *lambda* function, i.e. the gradient calculation, is most time consuming. Thus we implement the gradient with *Cython* to see if we could largely boost the performance. The speed here is 70ms on average, 2 times faster than in *numpy*, *Numba* or *Cython* version without implementing the gradient. Our C++ optimization is fastest with 7ms on average, about 10 times faster than *Cython* with gradient implemented and 30 times faster than plain *numpy* implementation.

Note that in practice, we may not be able to implement the gradient or even don't have the analytical expression. The experiment here is to show that bottleneck in speed results from the gradient calculation. One promising approach would be fully embed the algorithm into Tensorflow Probability because the gradient calculation will then be based on tensor graph and run directly on optimized C++. In fact, we find an R package called sgmcmc(Baker et al., 2019) and a Python package called edward(Tran et al., 2016) that implement SGHMC on top of Tensorflow.

4. Applications to simulated data sets

The paper conducted three experiments to validate the theoretical results of HMC and SGHMC and confirm the im-

Table 1. Optimization using Numba, Cython, and C++ on simulated data in Section 4.1. The results are the time taken in *ms* to run 1000 samples and 10 leapfrog steps on a noisy gradient. In Cython 2 and C++ the gradient function is optimized while in former ones are not.

METHODS	RESULTS
PLAIN NUMPY	208± 8.47
NUMBA	235± 34.8
CYTHON 1	259± 24.6
CYTHON 2	74.6± 2.58
C++	7± 0.14

portance of adding a friction term into the dynamics.

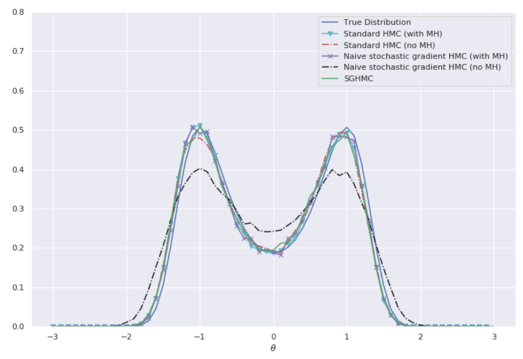

Figure 1. Various sampling algorithms results under the true target distribution with $U(\theta) = -2\theta^2 + \theta^4$. We compare the HMC method of Alg.1 with and without the MH step to: (i) a naive variant that replaces the gradient with a stochastic gradient with and without MH correction; (ii) the proposed SGHMC method, which does not use an MH correction. We use $\nabla\tilde{U}(\theta) = \nabla U(\theta) + N(0,4)$ as a noisy gradient and $\epsilon = 0.1$ in all cases. Momentum is resampled every 50 steps in all variants of HMC.

Fig. 1 shows the empirical distributions generated by the different sampling algorithms. We see that even without an MH correction, both the HMC and SGHMC algorithms provide results close to the target distribution, implying that errors from discretization are negligible. On the other hand, the results of naive stochastic gradient HMC diverge significantly from the truth unless an MH correction is added. These findings match with the theoretical results that both standard HMC and SGHMC maintain the invariance of Hamiltonian function as $\epsilon \to 0$ while naive stochastic gradient HMC does not, though this can be corrected by using a MH step. However MH is usually costly when the size of the data set is big.

We see from Fig. 2 that noisy Hamiltonian dynamics lead to diverging trajectories when friction is not introduced.

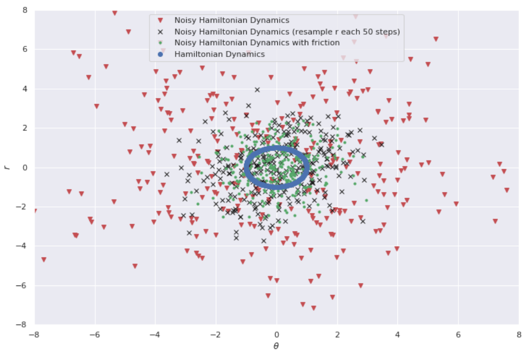

Figure 2. Points (θ, r) simulated from discretizations of various Hamiltonian dynamics using $U(\theta) = \frac{1}{2}\theta^2$ and $\epsilon = 0.1$. We use $\nabla\tilde{U}(\theta) = \theta + N(0,4)$ as the noisy gradient.

Resampling r helps control divergence, but the associated HMC stationary distribution is not correct, as illustrated in Fig. 1. These results confirm the importance of the friction term in maintaining a well-behaved Hamiltonian dynamics and leading to the correct stationary distribution.

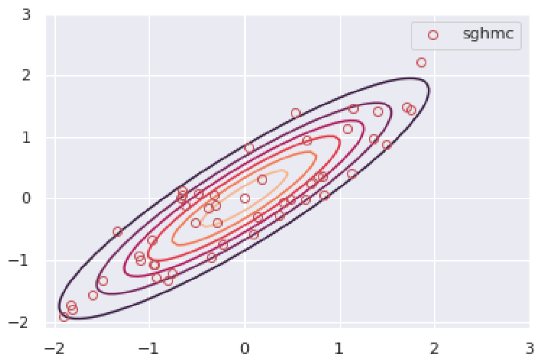

Figure 3. Constructing sampling of a bivariate Gaussian with correlation using SGHMC. $U(\theta) = \frac{1}{2}\theta^T\Sigma^{-1}\theta$,$\nabla\tilde{U}(\theta) = \Sigma^{-1}\theta + N(0, I)$ with $\Sigma_{11} = \Sigma_{22} = 1$ and correlation $\rho = \Sigma_{12} = 0.9$. The plot shows the first 50 samplings of SGHMC.

HMC is known for its advantage over many other MCMC algorithms when sampling from correlated distributions. We show in Fig. 3 that SGHMC inherits this property. We see that the momentum variable associated with SGHMC is able to drive the sampler to move along the distribution contours and hence explore the state space efficiently.

In addition to the exampled provided in the paper, we consider a simulated data that is closer to the application sce-

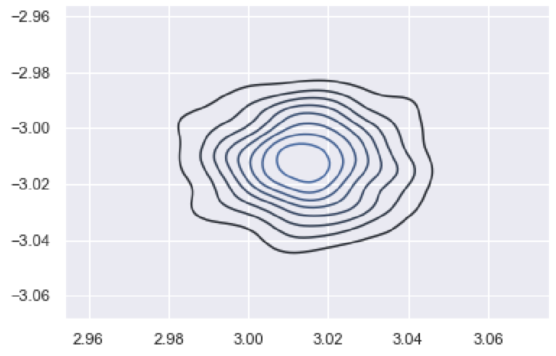

Figure 4. Constructing sampling of from the posterior distribution of mean parameters in Gaussian Mixture model. $x_1, x_2 \dots x_N$ with $X_i|\theta \sim 0.5N(\theta_1, 1) + 0.5N(\theta_2, 1)$ where $(theta_1, theta_2) = (-3, 3)$. we assume a weakly uninformative prior with $(\theta_1, \theta_2) \sim N(0, 10I_2)$.

narios. Assume we have independent and identically distributed data $x_1, x_2 \dots x_N$ with $X_i|\theta \sim 0.5N(\theta_1, 1) + 0.5N(\theta_2, 1)$ and a weakly informative prior on mean parameters with $(\theta_1, \theta_2) \sim N(0, 10I_2)$. We take $(\theta_1, \theta_2) = (-3, 3)$ and run SGHMC sampler on $N = 10^4$ simulated data starting from (0,0). Other parameters follow the default settings in Section 2.4 with $\hat{\beta} = 0, \eta = 0.01/N$, and $|\tilde{D}| = 500$. We run 5000 epochs and let the first 500 burn in. The density plot of two mean parameters are in Fig. 4. We can see that the samples from SGHMC sampler quickly converge to the region near the true parameters. However, despite that the model has two posterior modes because of its symmetric setting, the SGHMC sampler fail to travel to another mode and get struck in high probability region.

5. Applications to real data sets

The paper test SGHMC on a handwritten digits classification task using the MNIST dateset and Bayesian Neural Networks. The dataset consists of 60,000 training instances and 10,000 test instances. We adapted the source code from author's implementation and reproduced the results. In the example, we randomly split a validation set containing 10,000 instances from the training data in order to select training parameters, and use the remaining 50,000 instances for training.

For sampling-based algorithm SGHMC, the paper consider a two layer Bayesian neural network with 100 hidden variables using a sigmoid unit and an output layer using softmax. The model can be described by:

$$\Pr(y = i|x) \propto \exp(A_i^T \sigma(B^T x + b) + a_i)$$

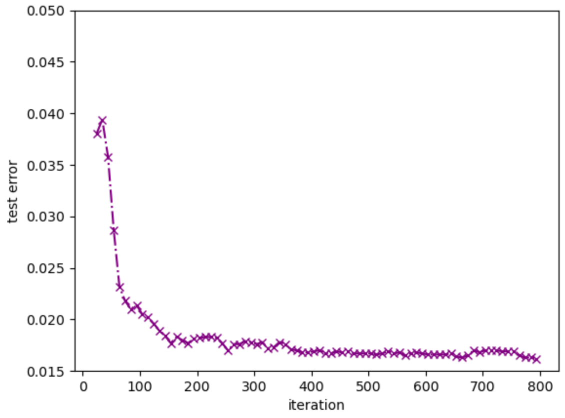

Figure 5. Convergence of test error on the MNIST dataset using SGHMC to infer model parameters of a Bayesian neural nets.

Here $y \in \{1, 2 \dots 10\}$ is the output label of a digit. $A \in R^{10\times 100}$ contains the weight for output layers and use A_i to denote ith column of A. $B \in R^{d\times 100}$ contains the weight for the first layer. $a \in R^{10}$ and $b \in R^{10}$ are bias terms in the model. In the MNIST dataset the input dimension $d = 784$. We place the following prior on the model parameters

$$P(A) \propto \exp(-\lambda_A ||A||^2), P(B) \propto \exp(-\lambda_B ||B||^2)$$
$$P(a) \propto \exp(-\lambda_a ||a||^2), P(b) \propto \exp(-\lambda_b ||b||^2)$$

And place gamma priors on hyperparameters

$$\lambda_A, \lambda_B, \lambda_a, \lambda_b \sim \Gamma(\alpha, \beta).$$

We simply set α and β to 1. The posterior distribution for all the model paramaters is

$$P(\Theta|D) \propto \prod_{y,x\in D} p(y|x, \Theta)P(\Theta),$$

where $\Theta = \{A, B, a, b, \lambda_A, \lambda_B, \lambda_a, \lambda_b\}$. The sampling procedure is carried out by alternating the following steps:

- Sampling weights from

  $$P(A, B, a, b|\lambda_A, \lambda_B, \lambda_a, \lambda_b, D)$$

  using SGHMC with minibatch of 500 instances. Sampling for 100 steps when the entire training data has been iterated. Because all weights and bias terms are considered independent here, for every w

  $$w_{i+1} = (1 - \alpha)w_i - \eta(\nabla \log p(y|x, w_i) + 2\lambda_{w_i} w_i)$$

  and we could update weights by borrowing experiences of back propagation in neural networks.

- Sampling λ from

$$P(\lambda_A, \lambda_B, \lambda_a, \lambda_b | A, B, a, b)$$

using a Gibbs step. Note that the posterior for λ is a gamma distribution by conditional conjugacy. For example,

$$\lambda_A | A, B, a, b \sim \Gamma(a, b + ||A||^2)$$

After selecting learning rate using the validation set when setting $\alpha = 0.01, \hat{\beta} = 0$, the best settings were $\eta = 0.2 \times 10^{-5}$. The final results are in Fig. 5.

6. Comparative analysis with competing algorithms

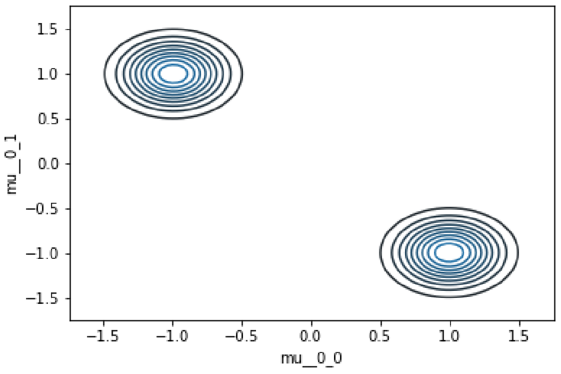

Figure 6. Using NUT sampler to construct sampling from the posterior distribution of mean parameters in Gaussian Mixture model. $x_1, x_2 \ldots x_N$ with $X_i|\theta \sim 0.5N(\theta_1, 1) + 0.5N(\theta_2, 1)$ where $(theta_1, theta_2) = (-3, 3)$. we assume a weakly uninformative prior with $(\theta_1, \theta_2) \sim N(0, 10I_2)$.

We compare the performance of SGHMC with NUT sampler through PyMC3 and PyStan package on the Gaussian Mixture model in Section 4. We did not tune these models and juts let the sampler run on its default settings. The results from two packages are similar and the density plot from PyMC3 in Fig.5. PyMC3 and PyStan both manage to detect two posterior modes for the parameters and are less concentrative than our implementation of SGHMC. SGHMC failed to catch the double modes may largely due to the lack of multiple chains. We tried another start states and SGHMC quickly converge to the other mode and stick with it. If sampling from one chain, PyMCs and PyStan will have similar problems as SGHMC. This shows us implementing the algorithm with multiple chain is very important. If there is more time, we will look into more researches on how to efficiently choose the start states for various samplers.

7. Conclusion

SGHMC builds on the fundamental framework of HMC, but using stochastic estimates of the gradient to avoid the costly full gradient computation. The friction term added to the dynamics system counteract the noise injected by stochastic estimates. Empirical results, both in simulated and real data validate these theoretical results. From testing process, we can see the SGHMC algorithm is an efficient sampler for generating high-quality, distant steps. In terms of the optimization and performance of our algorithm, one natural next step is to embed the sampler into TensorFlow probability to utilize its tensor graph. Another possible direction is to work on the multiple chains and accordingly multiprocessing implementation. As mentioned in the paper, we believe that the unification of efficient optimization and sampling techniques will enable a significant scaling of Bayesian methods.

References

Baker, Jack, Nemeth, Christopher, Fearnhead, Paul, and Fox, Emily B. sgmcmc: An R package for stochastic gradient Markov chain Monte Carlo. *Journal of Statistical Software*, 91(3):1–27, 2019. doi: 10.18637/jss.v091.i03.

Chen, Tianqi, Fox, Emily, and Guestrin, Carlos. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pp. 1683–1691, 2014.

Tran, Dustin, Kucukelbir, Alp, Dieng, Adji B., Rudolph, Maja, Liang, Dawen, and Blei, David M. Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*, 2016.