Implementation of Stochastic Gradient Hamiltonian Monte Carlo

Qinzhe Wang, Yi Mi

4/25/2021

Abstract

The Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) algorithm is proposed to address the limitations of Hamiltonian Monte Carlo (HMC) sampling methods, which is not suitable for solving the problems of computing the gradient of streaming data or large sample size. SGHMC takes the advantage of noisy gradient estimates based on the subsets of data. This project implements the algorithm from the paper Stochastic Gradient Hamiltonian Monte Carlo and optimizes the performance by using Numba and C++. The algorithm was applied on both simulated data and real data and was compared to other competing algorithms.

Background

Description of algorithm

This part contains two parts: Hamiltonian Monte Carlo and Stochastic Gradient Hamiltonian Monte Carlo.

Hamiltonian Monte Carlo

HMC provides momentum variables (r) to establish a new joint distribution (θ, r) so that it can draw samples from the posterior distribution.

Let U be the potential energy function described in the article, we have

$$\pi(\theta, r) \propto \exp(-U(\theta) - \frac{1}{2}r^T M^{-1} r)$$

$$U = \sum_{x \in D} ln[p(x|D)] - ln[p(\theta)]$$

In physical field, the Hamilton function can be described as

$$H(\theta, r) = U(\theta) + \frac{1}{2}r^T M^{-1} r$$

Therefore, we have

$$d\theta = M^{-1} r dt \quad and \quad dr = -\nabla U(\theta) dt$$

Stochastic Gradient Hamiltonian Monte Carlo

Instead of computing the ∇U in $dr = -\nabla U(\theta)dt$, SGHMC uniformly chooses $\tilde{D}$ (minibatch) from the dataset and compute $\tilde{U}$. The equattion can be shown as the following:

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

From the article, we can approximate the above equation by

$$\nabla U(\theta) + N(0, V(\theta))$$

Since the equation can be regarded as a discreted system, we may want to introduce Metroplis-Hasting sampling. Let ϵ be the step term in in Metroplis-Hasting sampling, we can rewrite the dr as following:

$$dr = -\nabla U(\theta)dt = -\nabla U(\theta) + N(0, 2B(\theta)dt)$$

where $B(\theta) = \frac{1}{2}\epsilon V(\theta)$ (positive semi-definite). In addition, the article shows we need to add a friction term because $\pi(\theta, r)$ is not invariant in this case. Finally, the dynamic equations become the following:

$$d\theta = M^{-1}rdt \quad and \quad dr = -\nabla U(\theta)dt - BM^{-1}rdt + N(0, 2B(\theta)dt)$$

$$dr = -\nabla U(\theta)dt - BM^{-1}rdt + N(0, 2B(\theta)dt) \qquad (10)$$

Optimization for performance

In this part, we create a new mixture normal model for the optimization. With the true $\mu = [-3, 3]$, we have $p(x|\mu_1, \mu_2) = \frac{1}{2}N(\mu_1, 1) + \frac{1}{2}N(\mu_2, 1)$. The parameters are shown as following:

ϵ	C	V	Batch Size	Epochs	Burns
0.1	4	4	1	4000	200

Four different functions for optimization are plain python using numpy, python function with cholesky decomposition and cholesky based sampling, JIT (numba) and cpp. The running time are show in the following table:

Algorithm	Running Time
plain python (numpy)	7.91 s ± 140 ms
python with cholesky decomposition	7.51 s ± 140 ms
numba	7.38 s ± 321 ms
cpp	316 ms ± 6.25 ms

Except using cpp, the other three methods provide similar results regarding the mixture normal example. We can see that using Cholesky decomposition or using JIT compilation (numba) only improves the algorithm a little, from 7.91 s ± 140 ms per loop to 7.51 s ± 140 ms per loop and 7.38 s ± 321 ms per loop. One noteworthy point is that we use the `jacobian` function from the autograd package o calculate the autogradient. However, this function takes more time than expected. To implement SGHMC, a function to calculate the gradients of log densities are necessary. Therefore, we encourage users to write a gradient function by themselves and then use it as an argument in the sghmc function (our sghmc function allows users to use their gradient functions).

Applications to simulated data sets

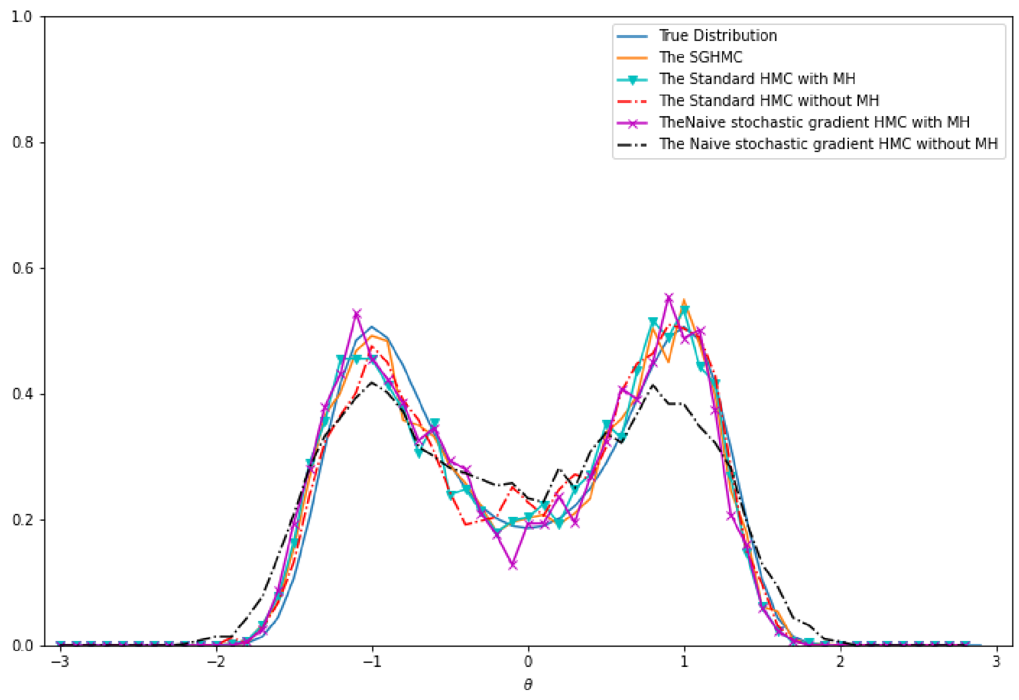
This part compares the distribution among five different algorithms (listed below) and the true distribution of the simulated data set in the original paper $U(\theta) = -2\theta^2 + \theta^4$.

- the distribution from the Stochastic Gradient Hamiltonian Monte Carlo algorithm $\nabla\tilde{U}(\theta) = \nabla U(\theta) + N(0, 4)$ with following argument:

ϵ	C	V	Batch Size	Epochs	Burns
0.1	3	4	1	5000	200

- The standard HMC with MH
- The standard HMC witthout MH
- The Naive stochastic gradient HMC with MH
- The Naive stochastic gradient HMC without MH

And a comparison figure shown as below:



All five algorithms perform similarly to the distribution of simulated data except the naive stochastic gradient HMC without MH. Theoretically, SGHMC and HMC have the invariance of Hamiltonian function for a ϵ value close to 0. However, the naive stochastic gradient HMC without MH does not unless the MH step is added. That is the reason why the naive stochastic gradient HMC without MH is barely satisfactory in the figure. Then, the MH step will be costly, so that users may need a tradeoff between the accuracy and the cost. Our focus, the distribution of SGHMC, gives a similar result as the distribution of simulated data.

Applications to real data sets

Real data sets

Competing algorithms

Discussion

References

Chen, Tianqi, Fox, Emily, and Guestrin, Carlos. Stochastic gradient hamiltonian monte carlo. ICML 2014.