Implementations of the Stochastic Gradient Hamilton Monte Carlo Algorithm

Authors: Tiancheng Pan, Lingyu Zhou and Fan Zhu

Abstract

In this package, we followed and implemented the "Stochastic Gradient Hamiltonian Monte Carlo" algorithm, which was proposed by Tianqi Chen, Emily B. Fox and Carlos Guestrin (2014). To significantly reduce the computational complexity, this Hamilton Monte Carlo algorithm incorporates the stochastic gradient, which is computed on minibatchs of data with noise and counterbalances that noise by a friction term. The SGHMC algorithm was initially implemented in Python and then optimized through numba, Cython (C++) and multiprocessing. To examine whether our SGHMC algorithm works, we tested our codes on one simulated example similar to that in the original paper and one simulated example generated from a mixture normal distribution, for which we sample the posterior of the mean parameters. The behaviors of SGHMC algorithm were also compared with two competing Monte Carlo algorithms on our first simulated data. There is an up-to-date Github repository for our SGHMC codes at (`https://github.com/TianchengPAN/STA-663-Final-Project.git`). The instructions of installation and explanation of this package are available in README file.

Background

Hamiltonian Monte Carlo (HMC) algorithm was first propsed by Duane, S., Kennedy, A. D., Pendleton, B. J. and Roweth, D. in 1987. The Hamiltonian Monte Carlo sampling method basically treats the probability density as a physical system in which there exists a moving object. As the law of conservation of energy, there is a trade-off between potential energy and the momentum of the moving object. This process involves the computation of the gradient information into the proposal distribution, which, when the sample size is large, is computational costly.

To tackle with this computational issue, Chen, T., Fox, E., & Guestrin, C. proposed the Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) algorithm in 2014. The SGHMC algorithm avoids the manifold computational efforts by applying a stochastic gradient on the minibatches of data with a certain degree noise and then counteracts that noise with a “friction” term, which can maintain the desired target distribution and invariance properties.

The applications of the SGHMC are consistent with those of the Markov Chain Monte Carlo (MCMC) sampling methods. However, the SGHMC algorithm is preferred when the usual random walk methods tend to generate samples with highly correlated variables or samples with very low acceptance probabilities (Chen, T. et al, 2014). By using the minibatches of data, the SGHMC method also allows for scaling of Bayesian methods. Despite these advantages, Chen and his colleagues argued that there are two limitations of the SGHMC algorithm (2014). The first limitation is that the use of the minibatches may lead to inappropriate stochastic gradient because the subsample size may not be big enough. The other limitation is that, compared to other MCMC algorithms, the SGHMC method requires the approximations of more variables at first. Those different initializations may result in different target distributions.

Description of the Algorithm

According to the Stochastic Gradient Hamiltonian Monte Carlo by Chen, T., Fox, E., and Guestrin, C. in 2014, we can briefly described this algorithm as follows:

First, with the help of auxiliary variables in physics system, the basic Hamiltonian function is defined by:

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

Where $U(\theta)$ is the potential energy function defined by $U(\theta) = -\sum_{x \in D} logp(x|\theta) - logp(\theta)$ given a set of independent observations $x \in D$ while r and mass matrix M together define the kinetic energy term. Then, the unit change in θ and r is:

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

Chen et al introduced a stochastic gradient based on minibatch of data $\tilde{D}$ with noise into system and counterbalanced that noise by a friction term (2014). Then the stochastic gradient is defined by:

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

The stochastic gradient with noise is based on the assumptions that $\nabla \tilde{U}(\theta)$ follows a normal distribution $\nabla \tilde{U}(\theta) = \nabla U(\theta) + N(0, V(\theta))$, where $V(\theta)$ is the covariance of stochastic gradient noise. Then SGHMC introduces an altered HMC procedure to counteract the noise term, in which the unit change in θ and r is defined by:

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

For simplicity, $B(\theta)$ is abbreviated to B. $N(0, 2Bdt)$ is the noise approximated by a normal distribution where $B(\theta) = \frac{1}{2} \epsilon V(\theta)$ is the diffusion matrix contributed by gradient noise. ϵ is the step size, and it takes similar function with learning rate in gradient descent. Thus, it is a very small number. $-BM^{-1} rdt$ is the friction term. However, in practice we rarely know the exact noise model B. We use $\hat{B}$ instead to approximate noise model. As a result, a new friction term $C \succeq \hat{B}$ is introduced. The new equation of dr is:

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

Finally,the algorithm is:

initialize $(\theta_0, r_0, \epsilon, M, \hat{B}, B, C)$

when t=1,2,3....., we update θ and r with:

$$\begin{aligned} \theta_i &= \theta_{i-1} + \epsilon_t M^{-1} r_{i-1} \; r_i \\ &= r_{i-1} - \epsilon_t \nabla \tilde{U}(\theta_i) - \epsilon_t C M^{-1} r_{i-1} + N(0, 2(C - \hat{B}) \epsilon_t) \end{aligned}$$

Describe Optimization for Performance

Since vectorization is already included in our original algorithm, we will optimize our original algorithm by a better algorithm and JIT.

1. An better algorithm is achieved by the use of Cholesky decomposition on multivariate normal sampling. In this way, the covariance matrices needed to sample relevant multivariate normals are calculated outside

the loops, which saves running times.
2. Use JIT compilation on our original algorithm.
3. Re-writing our original algorithm, minibatch and gradient functions in C++ and using pybind11 to wrap them.

The following table shows the running times in seconds for different methods under simulations of figure 1 in paper and simulations of mixture of normals. For simulations of figure 1 in paper, we use 50,000 samples from different empirical distributions. For simulations of mixture of normals, we use 2,000 samples from a mixed normal distribution with 500 iterations over 4 size-50 data batches.

Running time in s	Original Method	Simplified Algorithm	Numba Version	C++ Version
Figure 1	18.18	3.22	4.65	0.21
Mixture of Normals	22.1	21.4	21.5	0.02

From results above, for simulations of figure 1, all of simplified algorithm, JIT compilation and C++ functions are able to improve the efficiency a lot. However, for simulations of mixture of normals, only C++ functions give a satisfying result. In both cases, C++ functions are most powerful optimizations compared with other methods.

Applications to Simulated Data Sets

Simulation 1

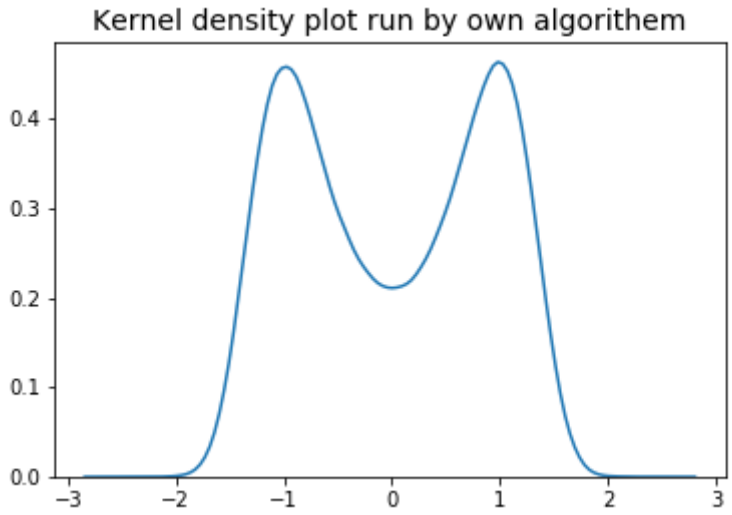

Figure 1

In the first simulation example, which is as mentioned as in the original paper, where $U(\theta) = -2\theta^2 + \theta^4$ and $\nabla\tilde{U} = \nabla U + N(0, 4)$. We set $\hat{V}$ to be a 0 matrix with shape $(1, 1)$, ϵ equals to 0.1, and batch size equal to 1. The sample size is 1000 and the number of iteration is 2000. Finally, we have a very similar density plot to that in the original paper. Thus the algorithm we have is pretty good.

Simualtion 2

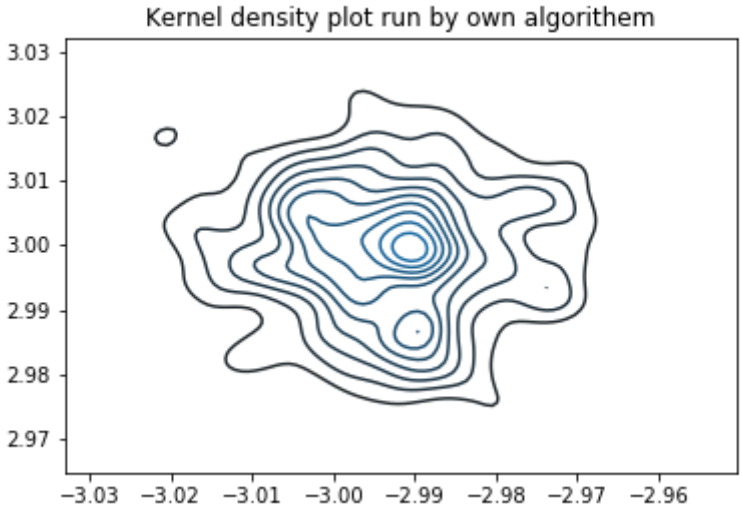

Figure 2

In the second simulation example, we choose the mixture normal distribution, where $x \sim 0.5 * N(\mu_1, 1) + 0.5 * N(\mu_2, 1)$, where $\mu_1 = -3, \mu_2 = 3$.We set $\hat{V}$ to be an identity matrix with shape $(2, 2)$, ϵ equals 0.1, and batch size equal to 1000. The sample size is 1000 and the number of iteration is 2000 again. The result is shown in figure 2, which is quite reasonable since the distribution is centered at $(3, -3)$.

Applications to Real Data Sets

Due to the complexity and the difficulty of access to data, we do not use the dataset described in the original paper. Instead, we choose the dataset about the ESG exposures for US public companies from Reprisk. The data is from Wharton Research Data Service (WRDS), which can be logged in with Duke ID or other accessible ID. Notice we do not upload the raw dataset into Github, since the raw dataset is over 25 MB. However, we did upload the code about how we clean the data with the clean data we have in the CSV files. More information on the data could be found in the "RepRisk- *Guidance* on_data_package_elements_2020_version.pdf".

The problem we want to solve is the distribution of the total number of exposed news for the companies. We assume the distribution is a Normal distribution with unknown parameters. `rep_demo.csv` contains the companies that also can be found at Financial Modeling Prep (FMP) with their ISIN. Since these companies can match the FMP, we treat them as the set of our prior. The mean is the prior mean and the variance is the prior variance. Then we run the the SGHMC with other companies in `rep.csv` with $\epsilon = 0.1$, $n = 10000$, and $niter = 50$. And here is the kernel density is shown in Figure 3, and the kernel density with the prior density is a plot in Figure 4 with the blue as posterior and yellow as prior:

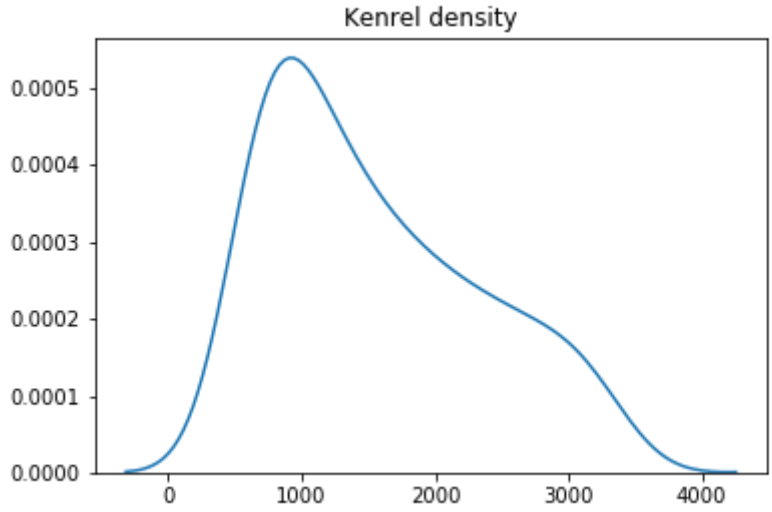

Figure 3

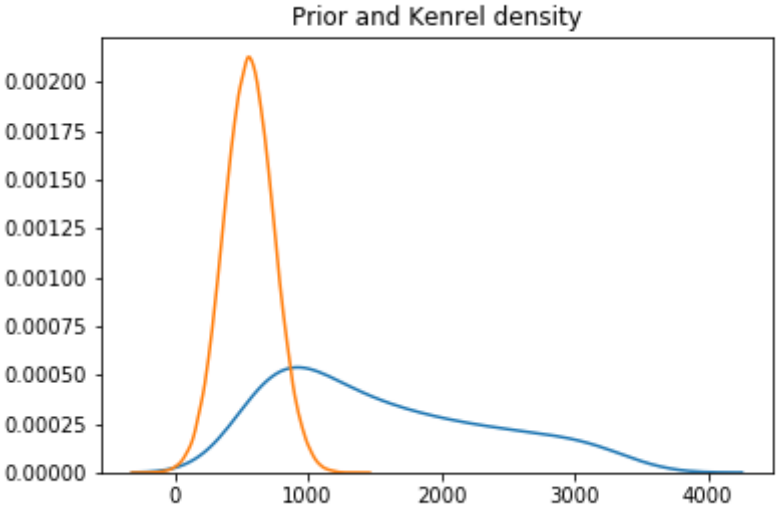

Figure 4

The result shows that Kernel density has nice Normal shape distribution before $\mu = 1000$, but when $\mu > 1200$, it shakes heavily. It may cause by some extreme large observations that affect the result for the SGHMC.

Comparative Analysis with Competing Algorithms

For the comparative analysis, we want to focus on the same problem as simulation 1, since we have the exact density plot in the original paper. I want to compare our algorithms with `hmc` in the Pyhmc package, which is a standard HMC method, and `pystan` in the Pystan package, which is a no-U-turn implementation of HMC. For `hmc`, we set the sample size to be 1000, and for `pystan`, we set the number of iteration to be 2000. All the other parameters are set to be the default.

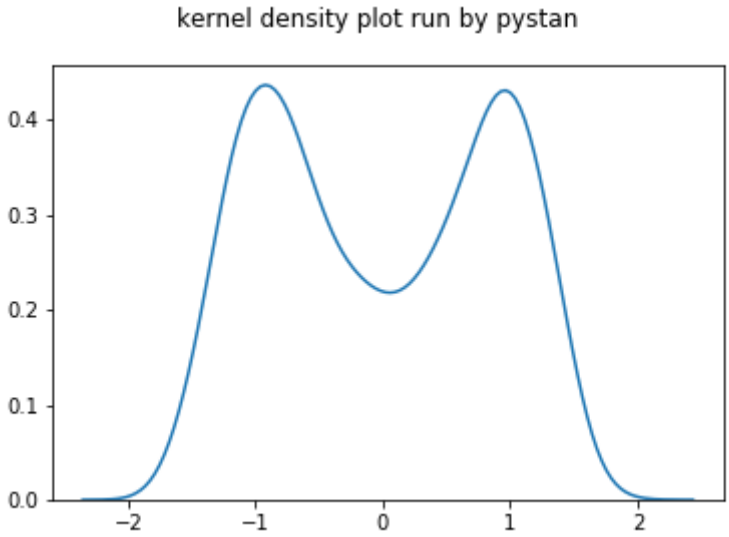

Figure 5

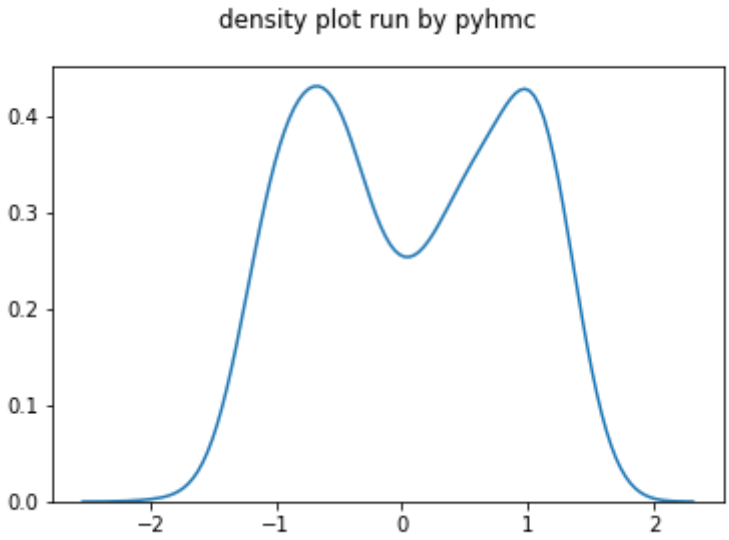

Figure 6

Figure 5 and Figure 6 are the density plot for the `pystan` and `pyhmc` with the same sample size or number of iteration as ourselves algorithm. As we can see, the density plot of `pyhmc` is significantly different from the original plot. The two peaks of density are not symmetric with the $\theta = 0$. Moreover, for the plot of `pystan` , it looks like the original plots, but it is less condense at two peaks and more condense at the nadir when $\theta = 0$. Admittedly, we do not change the default setting for both algorithm and the performance may be better if we choose a more reasonable parameter. Also, the running time for `sghmc` is longer than both of the above two. But it did show that `sghmc` did pretty well for the simulation 1.

Discussion

The simulations and application illustrate the strengths and weaknesses of the SGHMC algorithm. It does provide a precise sampler with a suitable parameter. However, the result of the SGHMC also depends a lot on that and it takes lots of time for researcher to apply different combination of parameters. Notice that in the real

world, where the truth is unknown, the basic SGHMC algorithm is insufficiently robust to be trusted. It may give thousand of results depends on different combination of parameters. Thus the algorithm may be improved for its convergence.

However, the `pystan` package, which uses the No U-Turn Sampler (NUTS) HMC algorithm, is almost as precise as `SGHMC` and without hyperparameter tuning. As noted in the original paper itself, there are a number of adaptive improvements that have been made to HMC - NUTS being one of them - which could be applied to the SGHMC algorithm. The further research can be done in such area.

References

- Chen, T., Fox, E., & Guestrin, C. (2014, January). Stochastic Gradient Hamiltonian Monte Carlo. In International Conference on Machine Learning (pp. 1683-1691).
- Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid Monte Carlo. Physics letters B, 195(2), 216-222.

Install

Our package is called `sghmczpz`. To install the package, run `pip install --index-url https://test.pypi.org/simple/ --no-deps sghmczpz` in the terminal.

The package can be import by `import sghmc`.

The main functions are in the `SGHMC_algorithm.py`, which can be import by `from sghmc import SGHMC_algorithm`. Notice package `autograd` is also required.