Implementation of Stochastic Gradient Hamiltonian Monte Carlo

Qinzhe Wang, Yi Mi

4/25/2021

Abstract

Hamiltonian Monte Carlo (HMC) sampling method is a Markov chain Monte Carlo method that uses an approximate Hamiltonian dynamics simulation based on numerical integration, which is then corrected by performing a Metropolis acceptance step. The gradient computation for the Hamiltonian dynamical system simulation is not suitable for streaming data or large sample size. The Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) algorithm is thus proposed to address the limitations of HMC sampling methods. SGHMC takes advantage of noisy gradient estimates based on the subsets of data and utilizes a friction term that counteracts the effects of the noisy gradient, maintaining the desired target distribution as the invariant distribution. This project made an explanation and implemented the SGHMC algorithm from the paper Stochastic Gradient Hamiltonian Monte Carlo. Numba, C++, and Cholesky Decomposition were utilized to optimize the performance of the code. The algorithm was applied on the simulated dataset and tested on a handwritten digits classification task using the MNIST dataset, compared to SGLD and SGD with momentum methods. The package was created and published on TestPyPI.

Background

The paper we explored in this project is Stochastic Gradient Hamiltonian Monte Carlo, which solves the problems that exist in Hamiltonian Monte Carlo (HMC). HMC (Duane et al., 1987; Neal, 2010) is a Markov chain Monte Carlo (MCMC) sampling algorithm, which defines a Hamiltonian function based on the target function and the kinetic energy term parameterized by a set of "momentum" auxiliary variables. Based on simple updates to the momentum variables, one simulates from a Hamiltonian dynamical system that enables proposals of distant states. Under these dynamic conditions, the target distribution is constant. In practice, it is necessary to discretize the continuous-time system, which requires correction of Metropolis-Hastings (MH), although it still has a high probability of acceptance. HMC is popular for the reason that it can rapidly exploration the state space.

However, the gradient computation for the Hamiltonian dynamical system simulation is infeasible for streaming data or large sample size. To address the limitation of HMC, this paper implementing HMC using a stochastic gradient and propose variants on the Hamiltonian dynamics that are more robust to the noise introduced by the stochastic gradient estimates. It computes the noisy estimates based on minibatch sampled uniformly at random from the entire dataset, which is considered sufficient for the central limit theorem approximation to be accurate with hundreds of data points. The most significant advantage is that it significantly reduces the computational cost of the gradient.

HMC with stochastic gradients requires a frequent costly MH correction step or long simulation runs with low acceptance probabilities. Thus, a "friction" term is added to the momentum update to minimize the effect of the injected noise on the dynamics themselves to alleviate these problems.

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

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

The algorithm of Hamiltonian Monte Carlo can be shown as:

Algorithm 1: Hamiltonian Monte Carlo

Input: Starting position $\theta^{(1)}$ and step size ϵ
for $t = 1, 2 \cdots$ **do**
  Resample momentum r
  $r^{(t)} \sim \mathcal{N}(0, M)$
  $(\theta_0, r_0) = (\theta^{(t)}, r^{(t)})$
  Simulate discretization of Hamiltonian dynamics in Eq. (4):
  $r_0 \leftarrow r_0 - \frac{\epsilon}{2} \nabla U(\theta_0)$
  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
  $r_m \leftarrow r_m - \frac{\epsilon}{2} \nabla U(\theta_m)$
  $(\hat{\theta}, \hat{r}) = (\theta_m, r_m)$
  Metropolis-Hastings correction:
  $u \sim \text{Uniform}[0, 1]$
  $\rho = e^{H(\hat{\theta}, \hat{r}) - H(\theta^{(t)}, r^{(t)})}$
  if $u < \min(1, \rho)$, **then** $\theta^{(t+1)} = \hat{\theta}$
end

Figure 1: algorithm of Hamiltonian Monte Carlo

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:

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

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

The algorithm of Stochastic Gradient Hamiltonian Monte Carlo can be shown as:

Algorithm 2: Stochastic Gradient HMC

for $t = 1, 2 \cdots$ **do**
  optionally, resample momentum r *as*
  $r^{(t)} \sim \mathcal{N}(0, M)$
  $(\theta_0, r_0) = (\theta^{(t)}, r^{(t)})$
  simulate dynamics in Eq.(13)*:*
  for $i = 1$ ***to*** m **do**
    $\theta_i \leftarrow \theta_{i-1} + \epsilon_t M^{-1} r_{i-1}$
    $r_i \leftarrow r_{i-1} - \epsilon_t \nabla \tilde{U}(\theta_i) - \epsilon_t C M^{-1} r_{i-1}$
      $+\mathcal{N}(0, 2(C - \hat{B})\epsilon_t)$
  end
  $(\theta^{(t+1)}, r^{(t+1)}) = (\theta_m, r_m)$, no M-H step
end

Figure 2: algorithm of Stochastic Gradient Hamiltonian Monte Carlo

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)	7.61 s ± 140 ms
python with cholesky decomposition	7.48 s ± 316 ms
numba	7.14 s ± 88.5 ms
cpp	316 ms ± 6.25 ms

We first found that using Numba did not improve the performance a lot, so we decided to use C++ to optimize the algorithm. The main reasons that Python is slower than C++ are Python is an interpreted language and supports Garbage Collection. We use the package pybind11 to wrap C++ code for Python, which provides an automatic conversion for NumPy arrays and the C++ Eigen linear algebra library. In our experiment, we re-write the gradient function and SGHMC function in C++.

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 auto-gradient. 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).

Application 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 table shown as below:

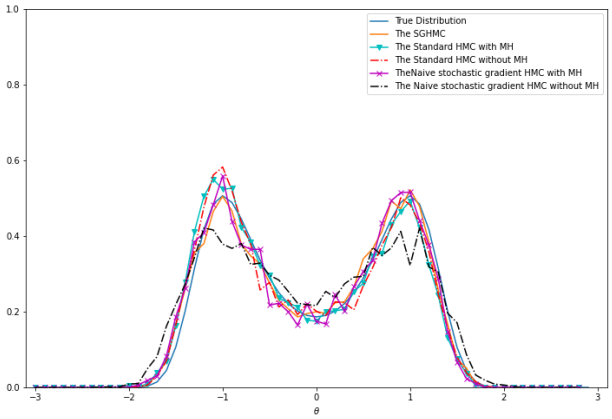

Figure 3: Distributions of different algorithms

All five algorithms perform similarly to the distribution of simulated data except the naive sThtochastic 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.

We also try to construct the sample draw from bivariate Gaussian with SGHMC correlation. Speciffically, we use $U(\theta) = 0.5 \times \theta^T \Sigma^{-1} \theta$ and $\nabla \tilde{U}(\theta) = \Sigma^{-1}\theta + N(0, I)$ where $\Sigma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & -1 \end{pmatrix}$

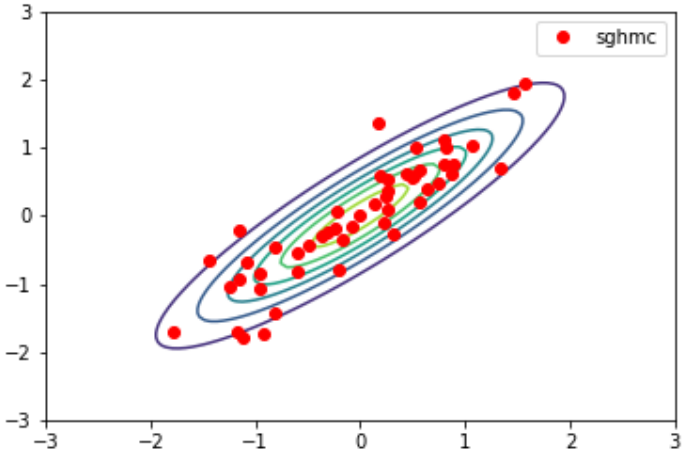

Figure 4: Figure of Bivariate Gaussian with SGHMC correlation

The above figure 4 shows the bivariate Gaussian with SGHMC correlation. The SGHMC's momentum variable moves in the same direction as the contours of the distribution. This property is from HMC when handling the distributions that are correlated.

Application to real data sets

Real data sets

In addition to the application on the simulated datasets, we also applied the SGHMC on the real datasets MNIST, which was used in the paper. MNIST is a large database of handwritten digits that is commonly used for training various image processing systems, consists of 60,000 training instances and 10,000 test instances. We utilized the author's code to train a two-layer Bayesian neural network with 100 hidden variables using a sigmoid unit and an output layer using softmax. To select training parameters, the training data is split out 10,000 instances to form a validation set and the remaining 50,000 instances are used for training. We run the samplers for 800 iterations and discard the initial 50 samples as burn-in. Three methods are tested in this experiment, SGD with momentum, SGLD and SGHMC. Fig 5 is the test result of the methods. It is shown that, based on our experiment results, both SGHMC and SGD with momentum converge to the lower test error compared to SGLD.

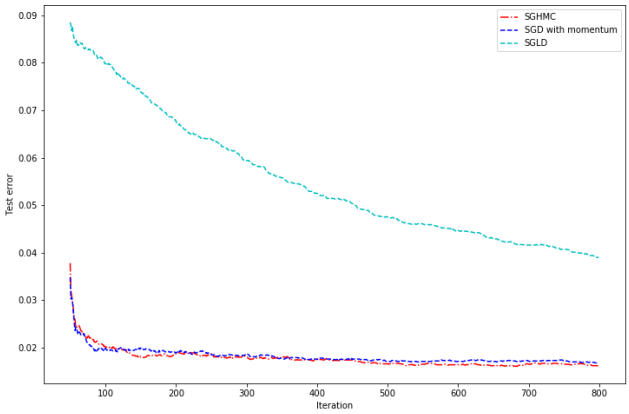

Figure 5: Convergence of test error on the MNIST dataset using SGD with momentum, SGLD, and SGHMC to infer model parameters of a Bayesian neural net.

Comparing algorithms

In this section we are going to compare SGHMC with SGD with momentum. SGD with momentum is a method adding a momentum term to SGD. Momentum is essentially a small change to the SGD parameter update so that movement through the parameter space is averaged over multiple time steps. Formally the update rule in SGD is defined like this:

$$\theta_{t+1} = \theta_t - \eta \nabla l(\theta)$$

With momentum, the SGD update rule is changed to:

$$v_{t+1} = \mu v_t - \eta \nabla l(\theta)$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

Here v is the velocity and μ is the momentum parameter which controls how fast the velocity can change and how much the local gradient influences long term movement. At every time step the velocity is updated according to the local gradient and is then applied to the parameters.

Nesterov Momentum (NAG) is used in the paper. NAG is a simple change to normal momentum. The update rule is changed to:

$$v_{t+1} = \mu v_t - \eta \nabla l(\theta + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

As discussed in the paper, there is a relationship between SGD with momentum and SGHMC. letting $v = \epsilon M^{-1} r$, we first rewrite the update rule as:

$$\begin{cases} \Delta\theta = v \\ \Delta v = -\epsilon^2 M^{-1} \nabla \tilde{U}(\theta) - \epsilon M^{-1} C v + N(0, 2\epsilon^3 M^{-1}(C - \hat{B}) M^{-1}) \end{cases}$$

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

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

Comparing SGHMC to an SGD with momentum method, we can see that η corresponds to the learning rate and $1 - \alpha$ the momentum term. When the noise is removed (via $C = \hat{B} = 0$), SGHMC naturally reduces to a stochastic gradient method with momentum.

In Fig 6 we can see that SGHMC has the better result than SGD with momentum. Additionally, according to the running time of our experiments, SGHMC is much faster than SGD with momentum.

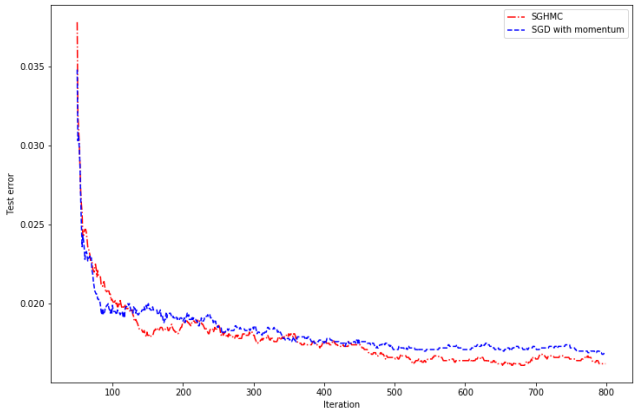

Figure 6: Comparing SGHMC and SGD with momentum

Discussion

SGHMC improves the efficiency of HMC by adding stochastic estimates of the gradient, simplifying the computation of the gradient. In addition, to offset the noise term, the friction term is added to the dynamic system. The implementation of SGHMC of both simulated data and real data shows that SGHMC is efficient. However, the algorithm needs hyperparameters, including learning rate and the noise term. Different inputted values will lead us to different distributions. In other words, the accuracy is correlated to those hyperparameters. As mentioned in the paper 'Stochastic gradient Hamiltonian Monte Carlo written by Chen, Fox, and Guestrin, it does increase the efficiency of the algorithm in the real world with a large dataset.

References

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

Appendix

URL

https://github.com/yimi97/sghmc

Installation

```
pip install -i https://test.pypi.org/simple/ sghmc-2021
```

Contributions

- Qinzhe Wang: implemented SGHMC algorithm, optimized code, applied algorithms on simulated data sets, wrote report
- Yi Mi: optimized code, applied algorithms on real data sets, wrote report, created Python package