Stochastic Gradient Descent Hamiltonian Monte Carlo Applied to Bayesian Logistic Regression

Sta663 Final Project

Gilad Amitai and Beau Coker

Abstract

Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo algorithm for drawing samples from a probability distribution where proposed values are computed using Hamiltonian dynamics to find values of high acceptance probabilities. They allow us to explore sample states more efficiently than random walk proposals, but are limited by the expensive computation of the gradient of the potential energy function. Chen, Fox, and Guestrin propose the method Stochastic Gradient Hamiltonian Monte Carlo (SGHMC), a HMC algorithm that uses a subset of the data to compute the gradient. The authors find that the stochastic gradient is noisy and correct this with a friction term.

In this project, we adapt the SGHMC to be used for Bayesian Logistic regression, implement this method in Python, optimize the code for computational efficiency, validate our approach using simulated data, and apply the algorithm to real world classification problems.

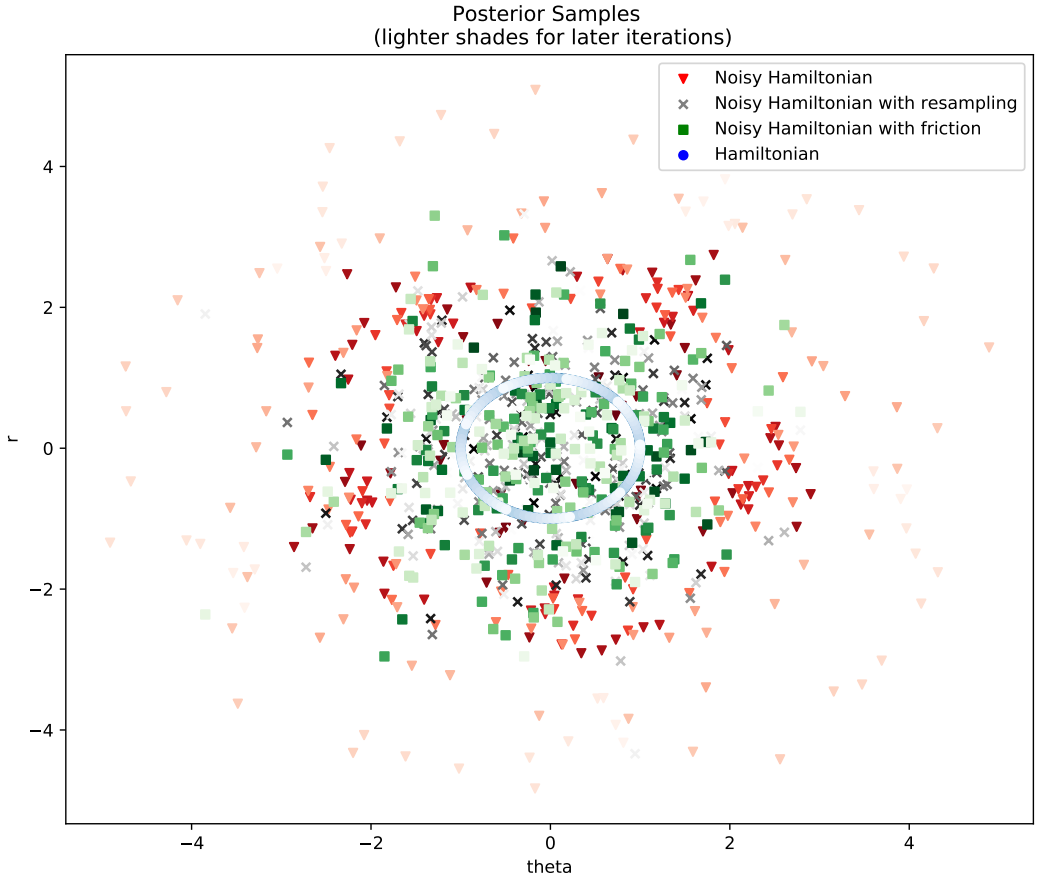
Keywords: Hamiltonian Monte Carlo, Stochastic Gradient Hamiltonian Monte Carlo, Pima Indians Diabetes Dataset, Hockey Puck, Logistic Regression, Markov chain Monte Carlo

1 Background

Because this is a project about Hamiltonian Monte Carlo, imagine a frictionless puck on an icy surface of varying heights. The state of this puck is given by it's momentum $\boldsymbol{q}$ and position $\boldsymbol{p}$. The potential energy of the puck U will be a function of only its height, while the kinetic energy will be a function of its momentum $K(q) = \frac{|\boldsymbol{q}|^2}{2m}$. If the ice is flat, the puck will move with a constant velocity. If the ice slopes upwards, the kinetic energy will decrease as the potential energy increases until it reaches zero, at which point it will slide back down. In the context of Bayesian statistics, we can think of the position of the puck as the posterior distribution we want to sample from, and the momentum variable are artificial constructs that allow us to efficiently move around our space. We use the notion of position and momentum, jointly called the Hamiltonian system, to propose samples where we set $U(p) = -\log[P(p)\mathcal{L}(p|\mathcal{D})]$

The final algorithm will be:

Input: Starting position $\theta^{(1)}$ and step size ϵ.
for *t=1, 2, ...* **do**
  Sample momentum $r^{(1)} \sim \mathcal{N}(0, M)$
  $r_0 = r_0 + \frac{\epsilon}{2}\Delta U(\theta_0)$
  for *i=1,...,m* **do**
    $\theta_i = \theta_{i-1} + \epsilon M^{-1} r_{i-1}$
    $r_i = r_{i-1} - \epsilon \Delta U(\theta_i)$
  end
  $r_m = r_m - \frac{\epsilon}{2}\Delta U(\theta_m)$
  $(\hat{\theta}, \hat{r}) = (\theta_m, r_m)$
  Sample $u \sim \text{Uniform}[0, 1]$
  $\rho = \exp\left\{H(\hat{\theta}, \hat{r}) - H(\theta^{(t)}, r^{(t)})\right\}$
  if $u < \min(1, \rho)$ **then**
    $\theta^{(t+1)} = \hat{\theta}$
  end
end



2 Description of Algorithm

In their *Stochastic Gradient Hamiltonian Monte Carlo*, Chen, Fox, and Guestrin propose using a subset $\tilde{\mathcal{D}}$ of the entire dataset $\mathcal{D}$ to compute

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

which can then be used in the Hamiltonian Monte Carlo equations in the stead of the gradient $\Delta U(\theta)$. Logistic regression assigns the probability of success to a dichotomous response variable

$$\Pr\left(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\beta}\right) = \frac{\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\}}{1+\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\}}$$

where $\boldsymbol{x}_i$ is a vector of length p covariates for data point i and $\boldsymbol{\beta}$ is a vector of regression coefficients of length p. In a Bayesian framework, we would assign the a prior distribution on our unknown parameters $P(\boldsymbol{\beta}) \sim \mathcal{N}(0, \sigma^2)$ where, for the purposes of our project, σ^2 is known. The corresponding posterior will be proportional to $P(\theta)\prod_{i=1}^n \Pr\left(y_i|\boldsymbol{x}_i, \boldsymbol{\beta}\right)$, which would give us the potential energy function

$$U(\boldsymbol{\beta}) = -\log\left[P(\boldsymbol{\beta})\right] - \sum_{i=1}^n \log\left[\Pr\left(y_i|\boldsymbol{x}_i, \boldsymbol{\beta}\right)\right] = \sum_{j=1}^p \frac{\beta_j^2}{2\sigma^2} - \sum_{i=1}^n \left[y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}) - \log(1+\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\})\right]$$

and gradient components

$$\frac{\partial U}{\partial \beta_j} = \frac{\beta_j}{\sigma^2} - \sum_{i=1}^n x_{ij}\left[y_i - \frac{\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\}}{1+\exp\left\{\boldsymbol{x}_i^T\boldsymbol{\beta}\right\}}\right].$$

In practice, the stochastic gradient is noisy since it is an approximation of the gradient. The paper suggests introducing a friction term to the momentum to dampen the movement of the chain. The algorithm will take a user specified friction term C that is element-wise bigger than the noise model B. The noise model is unknown but can be set to zero for simplicity. The final algorithm will be:

Input: Starting position $\theta^{(1)}$ and step size ϵ.
for *t=1, 2, ...* **do**
  Sample momentum $r^{(1)} \sim \mathcal{N}(0, M)$
  for *i=1,...,m* **do**
    $\theta_i = \theta_{i-1} + \epsilon M^{-1} r_{i-1}$
    $r_i = r_{i-1} + \epsilon\Delta\tilde{U}(\theta_i) - \epsilon C M^{-1} r_{i-1} + \mathcal{N}(0, 2(C-\hat{B})\epsilon)$
  end
  $(\theta^{t+1}, r^{t+1}) = (\theta_m, r_m)$
end

To better understand algorithm, we follow the example from the original paper. We consider a true potential energy function $U(\theta)$

3 Optimization

Any MCMC algorithm is inherently sequential, and so can't be parallelized (though multiple chains can be run at the same time). Each parameter's full conditional distribution depends on other parameters,

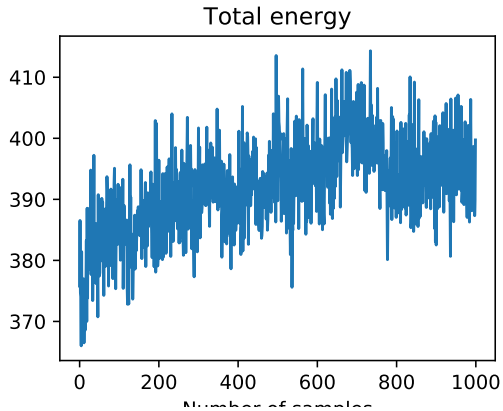
so loops are a natural implementation. Unfortunately, loops are quite slow in Python. In this section, we discuss an alternative, optimized version of our code that is written in C++ and is callable from Python via the Pybind11 package.

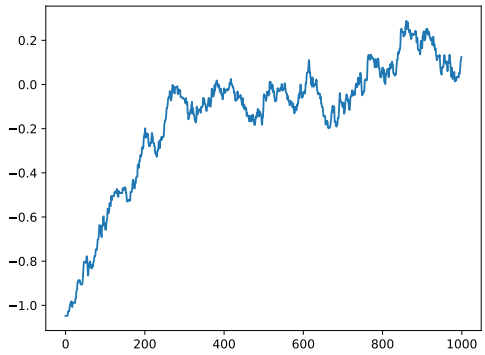
One drawback to a C++ implementation is the limited availability of easy-to-use random sampling functions. To complete our algorithm, we wrote a random multivariate normal function based on a Cholesky decomposition (necessary for the momentum updates and noise terms) and a random sampling function (necessary for the stochastic gradient descent). Unfortunately, the random sampling function is quite slow, significantly dampening the impact from the low-level speedup.

4 Application to Simulated Data

To test our implementation of the algorithm, we simulated data with 50 covariates and 500 observations, where the covariates were sampled from a normal distribution with mean zero and variances 25, 5, and 0.4 sampled form a multinomial distribution with probability vector $(.05, .05, .9)$. Data was normalized to have mean zero and a standard deviation of one. No intercept was fit.

We can see that using the Hamiltonian Monte Carlo algorithm produced good mixing with energy decreasing downwards and converging:





We can see that using the Stochastic Gradient Hamiltonian Monte Carlo algorithm produced good mixing with energy decreasing downwards and converging:

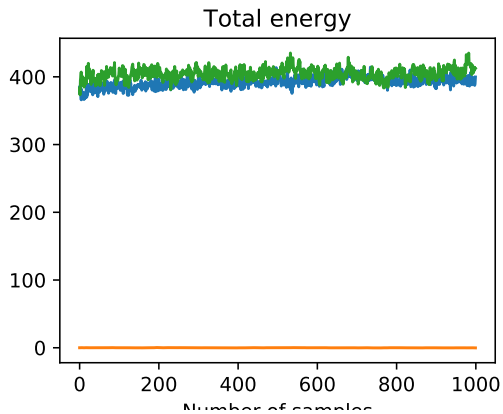


Figure 1: first figure

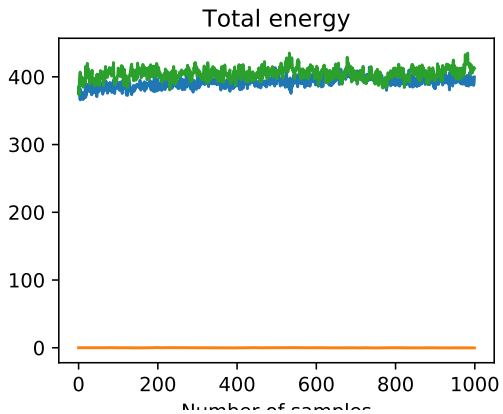


Figure 2: second figure

We compared the coefficients produced by each algorithm to the MLE estimates to check for accuracy:

5 Application to Real Data

We used the Pima Indians Diabetes Dataset from the National Institute of Diabetes and Digestive and Kidney Diseases. This dataset has a binary response variable indicating if the sample has diabetes and eight covariates on 768 samples. The data was standardized to have mean zero and standard deviation one.

We can see that using the Hamiltonian Monte Carlo algorithm produced good mixing with energy decreasing downwards and converging:

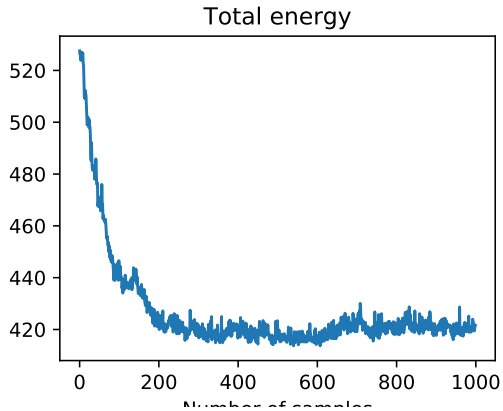


Figure 3: first figure

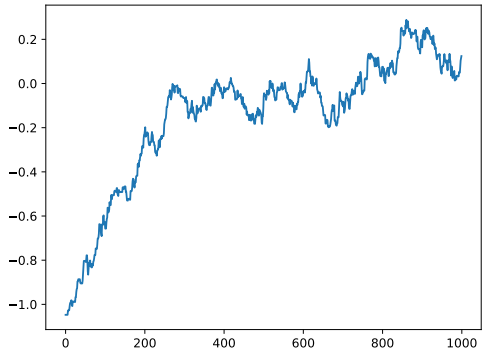


Figure 4: second figure

We can see that using the Stochastic Gradient Hamiltonian Monte Carlo algorithm produced good mixing with energy decreasing downwards and converging:

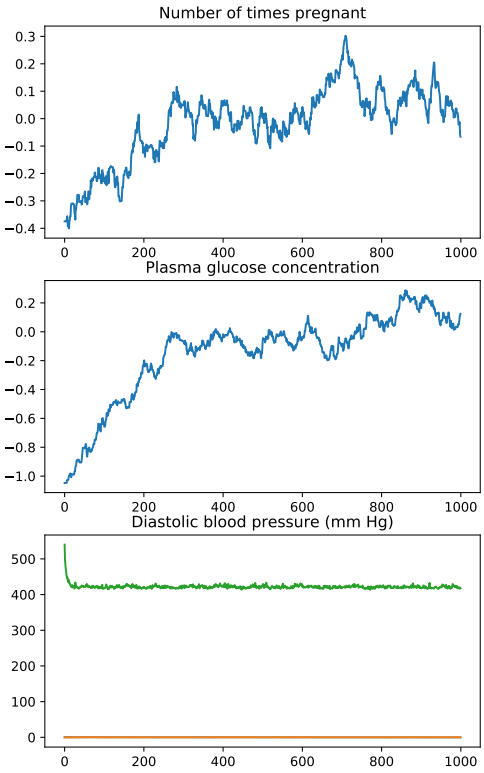


Figure 5: first figure

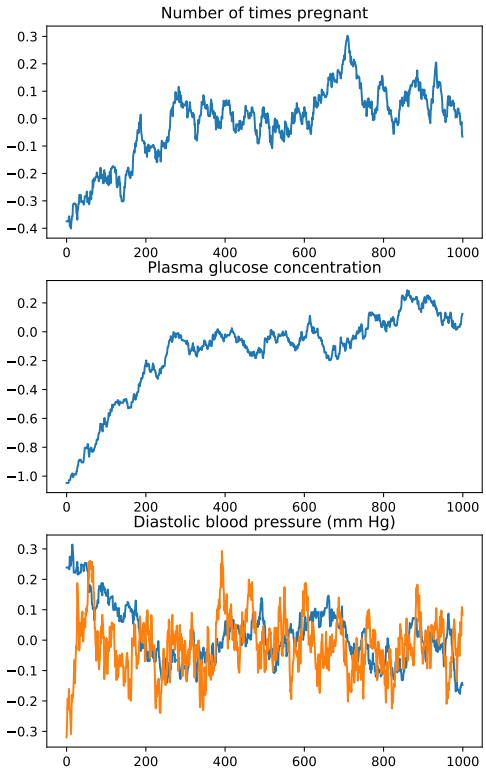
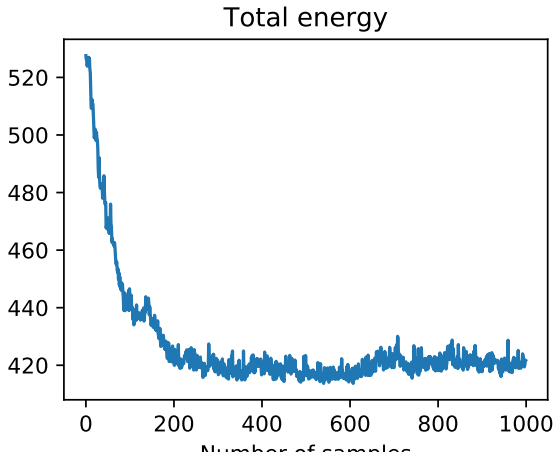


Figure 6: second figure

We compared the coefficients produced by each algorithm to the MLE estimates to check for accuracy:



6 Discussion and Conclusion

In implementing this algorithm, we found that

7 Bibliography