Stochastic Gradient Hamiltonian Monte Carlo

Candidate Numbers: Sam Adam-Day, 1059459, 1060482 and 1058141

University of Oxford

Stochastic Gradient Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) provides us with a useful way to sample from a posterior distributions. To do this we require a potential function $U(\theta) \propto \log p(\theta|\mathscr{D})$, as well as means to compute the gradient. At each step HMC uses all the data available to it; evaluating $U(\theta)$ and computing $\nabla U(\theta)$ is computationally expensive for large datasets, rendering HMC almost useless. As such, this motivated the development of the Stochastic Gradient Hamiltonian Monte Carlo algorithm (SGHMC), which is introduced by the paper in question [**sghmc**]. It uses randomly sampled mini-batches of data to produce noisy estimates of the gradient $\nabla\widetilde{U}(\theta)$, which gives SGHMC a significant speed-up compared to HMC. In this paper, Chen et al first introduce a Naive SGHMC algorithm, demonstrating the pitfalls of using noisy gradient estimates. They also introduce the full SGHMC algorithm that uses friction to overcome the need for a costly MH correction step. They further demonstrate the links between SGHMC and both Stochastic Gradient Descent (SGD) with momentum, and Stochastic Gradient Langevin Dynamics (SGLD). They then run experiments using SGHMC to empirically back up their theoretical claims and show that SGHMC is a candidate algorithm for scalable Bayesian inference. We implemented our own version of SGHMC along with some other algorithms and reproduced a number of Chen et al's experiments. The repository for our code can be found at https://github.com/sacktock/SGHMC.

The Core Algorithms

Below we introduce the major algorithms discussed in the paper. The first four algorithms are sampling algorithms - we sample parameters θ from the posterior distribution described by the model. We write here the transition steps that, upon iterating, gives us $\theta \sim p(\theta|\mathscr{D})$, where $\mathscr{D}$ is all the data available to us. $\widetilde{\mathscr{D}}$ is a randomly sampled batch of this data.

Hamiltonian Monte Carlo (HMC)

$$\Delta\theta \leftarrow \epsilon M^{-1} r \qquad \Delta r \leftarrow -\epsilon \nabla U(\theta)$$

where

$$\nabla U(\theta) := -\sum_{x\in\mathscr{D}} \nabla \log p(x \mid \theta) - \nabla \log p(\theta)$$

Naive Stochastic Gradient Hamiltonian Monte Carlo (Naive SGHMC)

$$\Delta\theta \leftarrow \epsilon M^{-1} r \qquad \Delta r \leftarrow -\epsilon \nabla \tilde{U}(\theta)$$

where

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

Stochastic Gradient Hamiltonian Monte Carlo (SGHMC)

$$\Delta\theta \leftarrow \epsilon M^{-1} r \qquad \Delta r \leftarrow -\epsilon \nabla \tilde{U}(\theta) - \epsilon C M^{-1} r + \mathcal{N}(0, 2(C - \hat{B})\epsilon)$$

where

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

and $\hat{B}$ is an estimation of the noise covariance B encapsulated by $\nabla\tilde{U}(\theta) = \nabla U(\theta) + \mathcal{N}(0, 2B\epsilon)$, and C is a user-defined hyperparameter

Stochastic Gradient Langevin Dynamics (SGLD)

$$\Delta\theta \leftarrow -\eta \nabla \tilde{U}(\theta) + \mathcal{N}(0, B)$$

where B is the covariance of injected noise.

The final algorithm (SGD) is an optimization algorithm, the idea is it iteratively converges to the mode of the posterior distribution - giving us a point estimate, namely, a MAP estimate.

Stochastic Gradient Descent (SGD)

$$\Delta\theta \leftarrow \alpha\Delta\theta - \eta \nabla \tilde{U}(\theta)$$

where α is a momentum hyperparameter. Standard SGD sets $\alpha = 0$

Bayesian Neural Networks for Classification

Below we present the results of running our algorithms on MNIST and FashionMNIST. We ran each of the algorithms for 800 epochs with 50 warmup epochs and 100 warm up epochs for MNIST and FashionMNIST respectively. For all the experiments we subsampled the dataset with batch sizes of 500 images. For the posterior sampling algorithms (SGHMC and SGLD) we performed Bayesian averaging over all the sampled parameterisations of the BNN after warmup and report the test accuracy as described in Section II of [**hands-on-bnn**]. For the optimization algorithms (SGD and SGD with momentum) we fixed the L2 regularization to $\lambda = 1.0$ and take the latest sample as point estimate and report the test accuracy.

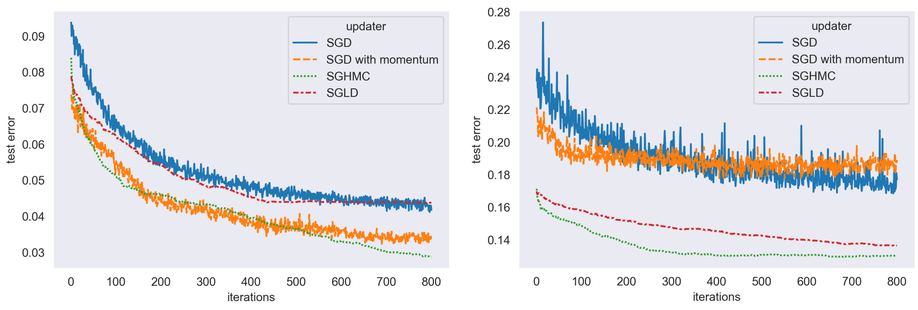

Fig. 1: **Left:** reproducing the MNIST classificiation experiment from [**sghmc**] ; SGHMC ($\eta = 2.0 \times 10^{-6}, \alpha = 0.01,$ `resample_n` $= 0$), SGLD ($\eta = 4.0 \times 10^{-5}$), SGD ($\eta = 1.0 \times 10^{-5}$), SGD with momentum ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01$) ;`warmup_epochs` $= 50$ **Right:** FashionMNIST classification experiment ; SGHMC ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01,$ `resample_n` $= 0$), SGLD ($\eta = 1.0 \times 10^{-5}$), SGD ($\eta = 1.0 \times 10^{-5}$), SGD with momentum ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01$); `warmup_epochs` $= 100$.

SGNUTS

'Stochastic Gradient No U-Turn Sampler' (SGNUTS) is our novel algorithm based on the 'No U-Turn Sampler' (NUTS) produced in [**nuts**]. NUTS removes the need for the user to pre-set the number of steps HMC performs before taking a sample. We produced SGNUTS to do the same for SGHMC. At its core, it works by repeatedly performing SGHMC steps either forward or backward in time until a 'U-turn' is seen. This is when a further step forwards in time would cause the latest sample in the trajectory to get closer to the earliest sample (or similarly for a step backwards in time). It reached accuracies of 0.94 on MNIST and 0.85 on FashionMNIST, which is similar to SGHMC (accuracies of 0.97 and 0.85 respectively). SGNUTS reaches the posterior faster than SGHMC, taking 28s to reach an accuracy of 0.87, compared to SGHMC taking 50s to reach 0.89. When at the posterior, SGNUTS takes longer than SGHMC to sample from it however.

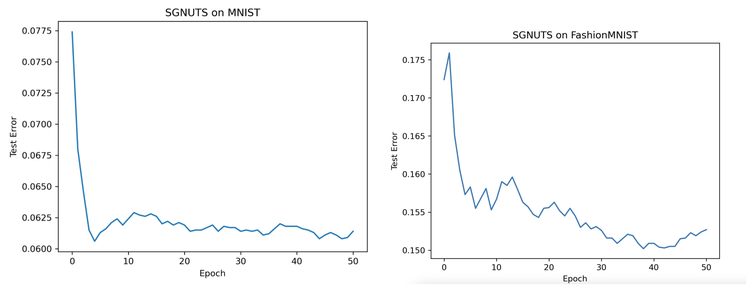

Fig. 2: SGNUTS Learning Curves on MNIST (left) and FashionMNIST (right)

References

The Reproducibility Challenge and Extensions

We reproduced the following experiments from the paper:

- Sampling θ from the potential function $U(\theta) = -2\theta^2 + \theta^4$ with noise added to the gradient $\nabla\widetilde{U}(\theta)$, using the following algorithms: HMC (with and without MH correction), Naive SGHMC (with and without MH correction) and SGHMC.
- Using HMC to sample (θ, r) from the potential function $U(\theta) = \frac{1}{2}\theta^2$ with perfect gradients, and noisy gradients using $\mathcal{N}(0, 4)$ as a proxy for the noisy in $\tilde{U}(\theta)$.
- Comparing the autocorrelation times of SGHMC and SGLD.
- Classifying the MNIST dataset using SGHMC as well as with SGD, SGD with momentum, and SGLD

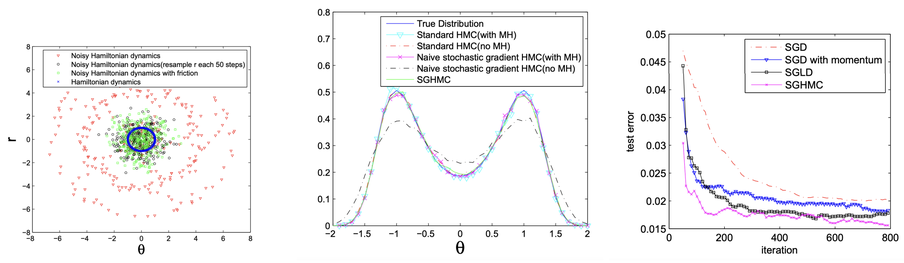

Fig. 3: Some of the figures from [**sghmc**] which we aimed to reproduce. **Left:** using HMC to sample (θ, r). **Center:** Samples from the potential function $U(\theta) = -2\theta^2 + \theta^4$. **Right:** learning curves for MNIST classification

We also extended the results of [**sghmc**] in a number of ways:

- We extended the 'No U-Turn Sampler' (NUTS) from [**nuts**] to work with SGHMC to produce our novel algorithm SGNUTS
- Ran the Bayesian neural network (BNN) for classification experiment on a new dataset, namely, FashionMNIST. [**fashion-mnist**]
- We demonstrated that our implementation of SGHMC can be used with Convolutional Neural Networks (CNNs) to classify CIFAR10 [**cifar10**].
- We implemented a scheme for estimating the gradient noise (B in the literature and in what follows) and used this to increase the algorithm's sampling accuracy.

Implementation Details

We implemented the following algorithms from scratch: HMC, SGHMC, SGLD , SGD, SGD with Nesterov momentum and SGNUTS. All of our implementations subclass Pyro's `MCMCKernel` and are designed to be used directly with Pyro [**pyro**] — a universal probabilistic programming language (PPL) written in Python. Pyro comes with useful built in functions that transform probabilistic programs (PP) into potential functions that automatically handle gradient computations. The main caveat of stochastic gradient samplers and optimizers is that we require that the potential function has the form:

$$\widetilde{U}(\theta) = -\frac{|\mathscr{D}|}{|\widetilde{\mathscr{D}}|} \log p(\widetilde{\mathscr{D}} \mid \theta) - \log p(\theta)$$

Using our implementations we illustrate below how sampling algorithms differ from optimization algorithms; while sampling algorithms visit the full posterior $p(\theta \mid \mathscr{D})$ as they draw samples, optimization algoithms hone in on the MAP estimate $\text{argmax}_\theta\{p(D \mid \theta) \cdot p(\theta)\}$.

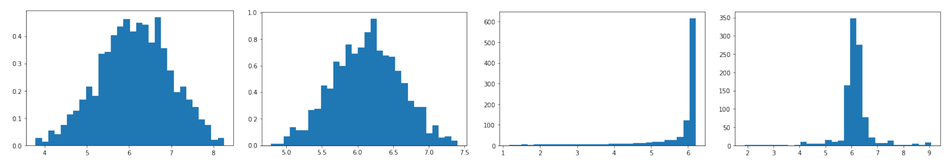

Fig. 4: **Far left**: SGHMC (`batch_size` $= 5, \eta = 0.01, \alpha = 0.1$ `num_steps`$= 10,$ `resample_n` $= 50$). **Mid left:** SGLD (`batch_size` $= 5, \eta = 0.1, \alpha = 0.1$ `num_steps`$= 5$). **Mid right:** SGD (`batch_size` $= 5, \eta = 0.001$). **Far right:** SGD with Nesterov momentum (`batch_size` $= 5, \eta = 0.001, \alpha = 0.1$)

Simulated Examples

Below we present our reproductions of the simulated scenarios in Section 4.1 of [**sghmc**]. We implemented these toy examples in Python using the Matlab codes provided by the authors of the original paper [**simu_code**]. We draw the following conclusions:

- Fig. 5(a) illustrates that the naive SGHMC fails to maintain the target distribution as its invariant distribution unless we add a costly MH correction step. Conversely, by adding friction, SGHMC maintains the target distribution as its invariant distribution, which validates the theorectical results in [**sghmc**].
- Fig. 5(b) shows that friction (green) keeps Hamiltonian dynamics much closer to the true Hamiltonian dynamics (blue) in a noisy system. Fig. 5(b) supports the results of Theorem 3.1 in [**sghmc**], that the sampled distribution tends to a uniform distribution over time, rather than the target posterior distribution.
- Shown in Fig. 5(c): we see as the step size decreases, SGLD [**sgld**] has a high autocorrelation time while SGHMC has a very low autocorrelation time with even lower estimation error. This indicates the advantage of adopting SGHMC. Fig. 5(d) shows that SGLD performs worse in exploring the tails of the distribution when compared to SGHMC.

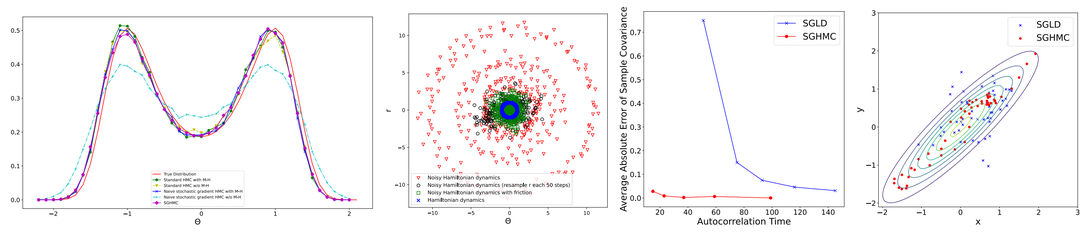

Fig. 5: **(a)**: θ samples generated from $U(\theta) = -2\theta^2 + \theta^4$. **(b):** 360 samples of (θ, r) generated from $U(\theta) = \frac{1}{2}\theta^2$. **(c):** Autocorrelation time versus mean absolute error of sample covariance for $U(\theta) = \frac{1}{2}\theta^T\Sigma^{-1}\theta$. **(d):** First 50 samples of SGHMC and SGLD generated from $U(\theta) = \frac{1}{2}\theta^T\Sigma^{-1}\theta$

Further Research

- Implementation of a hybrid SGNUTS and SGHMC algorithm - SGNUTS quickly reaches the posterior, however SGHMC is faster at sampling when at the posterior. We could investigate the power of using SGNUTS for the first few epochs or during warmup, followed by SGHMC.
- Tuning the CNN used to classify CIFAR10. Currently we have shown that our implementation of SGHMC can start classifying CIFAR10, however we could acheive better accuracies with more time to investigate other architectures and do a more thorough hyperparameter search.
- Further investigate methods of estimating the gradient noise B, as using empirical Fisher was computationally expensive for high dimensional models.
- Compare more thoroughly our implementation of SGHMC to Variational Inference.