

STOCHASTIC GRADIENT HAMILTONIAN MONTE CARLO

Sam Adam-Day, Alexander Goodall, Theo Lewy and Fanqi Xu

University of Oxford

Stochastic Gradient Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) provides a way to sample from a posterior distribution using all data available to it. To do this it must produce the potential function $U(\theta) \propto \log p(\theta|\mathcal{D})$, as well as its gradient. Using all of the data to find $U(\theta)$ is computationally expensive for large datasets, rendering HMC useless. This motivated the production of the Stochastic Gradient Hamiltonian Monte Carlo algorithm (SGHMC), which is introduced in [sghmc]. It uses batch data to produce noisy estimates of the potential function, which gives SGHMC dramatic speed-up when compared to HMC. In this paper, Chen et al introduce a Naive SGHMC algorithm, as well as SGHMC itself. They demonstrate links between SGHMC and both Stochastic Gradient Descent with momentum, and Stochastic Gradient Langevin Dynamics. They then run experiments using SGHMC. We reproduced SGHMC, and replicated a number of Chen et al's experiments. The repository for our code is found at <https://github.com/sacktock/SGHMC>.

The Core Algorithms

The first four are sampling based methods - we sample parameters θ from a model's posterior. We write here the transition step that, upon iterating, gives us $\theta \sim p(\theta|\mathcal{D})$, where \mathcal{D} is all the data available to us. \mathcal{D} is a randomly sampled batch of this data.

Hamiltonian Monte Carlo (HMC)

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

where

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

Naive Stochastic Gradient Hamiltonian Monte Carlo (Naive SGHMC)

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

where

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

Stochastic Gradient Hamiltonian Monte Carlo (SGHMC)

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

where

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

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 hyperparameter

Stochastic Gradient Langevin Dynamics (SGLD)

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

where B is the covariance of injected noise.

The next one is an optimization based method, which produces θ that are at the mode of the posterior distribution - MAP estimates.

Stochastic Gradient Descent (SGD)

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

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.

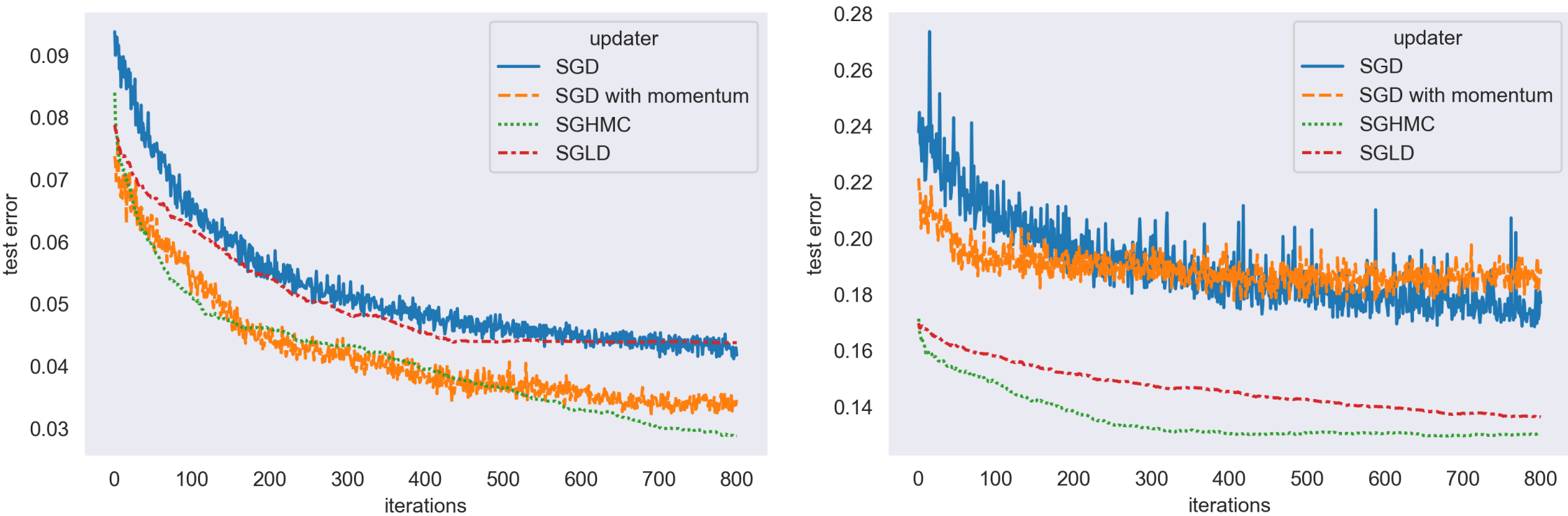


Fig. 1: **Left:** reproducing the MNIST classification experiment from [sghmc]; SGHMC ($\eta = 2.0 \times 10^{-6}, \alpha = 0.01, \text{resample_n} = 0$), SGLD ($\eta = 4.0 \times 10^{-5}$), SGD ($\eta = 1.0 \times 10^{-5}$), SGD with momentum ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01$); warmup_epochs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

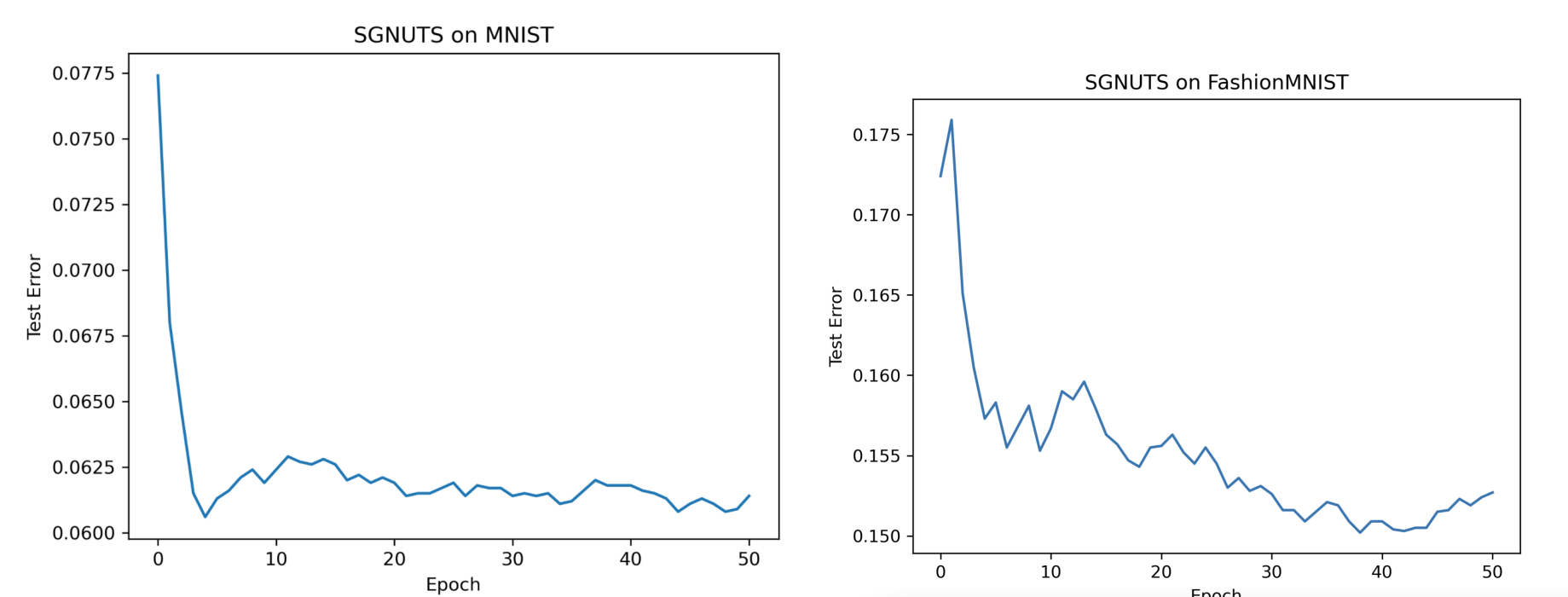


Fig. 2: SGNUTS Learning Curves on MNIST (left) and FashionMNIST (right)

Our Reproduction and Extensions

We reproduced the following experiments from the paper:

- Sampling θ from the posterior with $U(\theta) = -2\theta^2 + \theta^4$ using HMC, Naive SGHMC and SGHMC
- Sampling (θ, r) generated from $U(\theta) = \frac{1}{2}\theta^2$ with HMC, and from $U(\theta) = \frac{1}{2}\theta^2 + \mathcal{N}(0, 4)$ as a proxy for the noisy $\tilde{U}(\theta)$ in SGHMC
- Classifying the MNIST dataset using SGHMC as well as with SGD, SGD with momentum, and SGLD

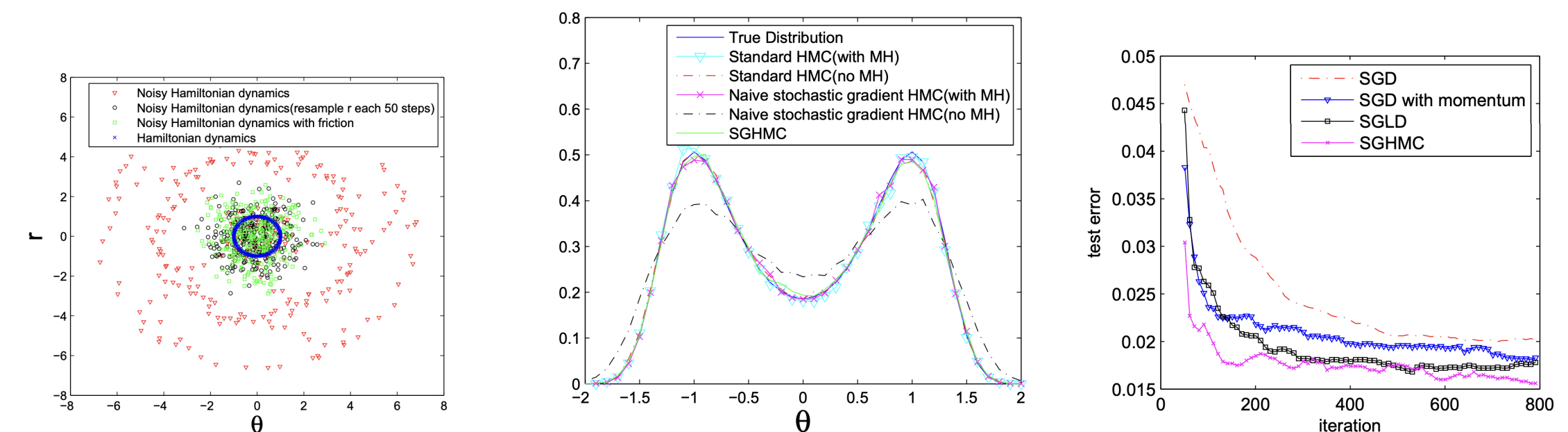


Fig. 3: Figures from [sghmc] which we aimed to reproduce. On left, (θ, r) samples generated from various methods. Center, θ samples generated from various methods. Right, learning curves for MNIST classification

We extended this paper 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
- We ran experiments on a new dataset of FashionMNIST
- We briefly introduced some Convolutional Neural Networks (CNNs) to see how accurate SGHMC was at classifying CIFAR10
- We attempted to evaluate the noisiness of the data (B in the literature and in what follows) and used this to increase the algorithm's efficiency

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:

$$\tilde{U}(\theta) = -\frac{|D|}{|\tilde{D}|} \log p(\tilde{D}|\theta) - \log p(\theta)$$

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

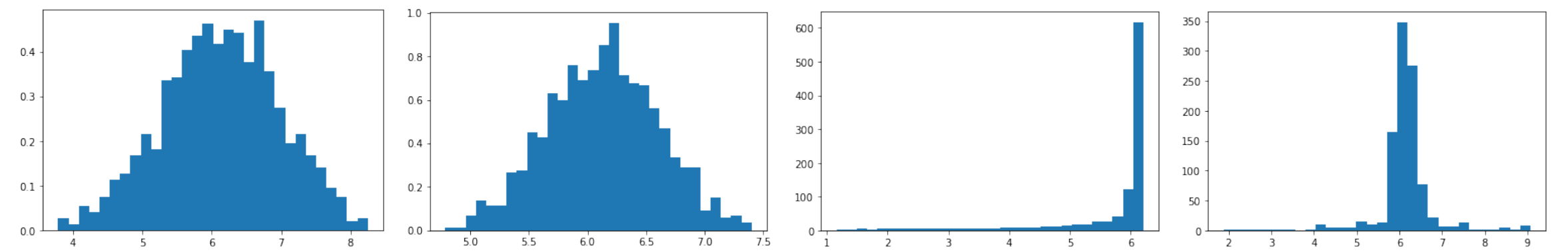


Fig. 4: **Far left:** SGHMC (batch_size = 5, $\eta = 0.01, \alpha = 0.1, \text{num_steps} = 10, \text{resample_n} = 50$). **Mid left:** SGLD (batch_size = 5, $\eta = 0.1, \alpha = 0.1, \text{num_steps} = 5$). **Mid right:** SGD (batch_size = 5, $\eta = 0.001$). **Far right:** SGD with Nesterov momentum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

Simulated Examples

Simulated scenarios are reproduced and extended from scratch based on the Matlab version of the original paper [simu_code]. Conclusions can be drawn as follows:

- As shown in Fig. 1, compared with naive stochastic gradient HMC, which requires a frequent costly MH correction step, SGHMC maintains more perfectly on both of the desired target distributions.
- As shown in Fig. 2(a), compared with naive stochastic gradient HMC without MH, which may not yield the correct target distribution, SGHMC can be used to modify the noisy dynamical system more correctly. The extension on the step size in Fig. 2(b) proves the Theorem 3.1 that $p_r(\theta, r)$ tends to be a uniform distribution over time, which can be very far from the target distribution π .
- As shown in Fig. 2(c), as step size decreases, SGLD [sgld] has a high autocorrelation time while SGHMC has very low one with even lower estimation error which indicates the efficiency of SGHMC. Fig. 2(d) shows that SGLD performs worse to explore the tails of the distribution compared to SGHMC.

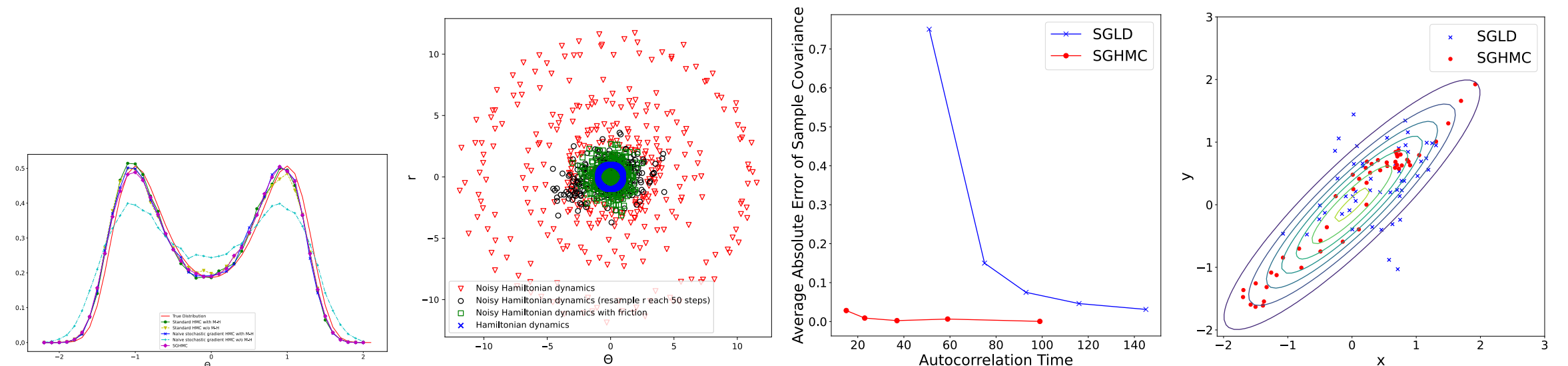


Fig. 5: (a): Points (θ, r) simulated of 15000 steps. (b): Points (θ, r) simulated of 150000 steps. (c): Autocorrelation time versus estimation error for the five settings. (d): First 50 samples of SGHMC and SGLD

Further Research

- Implementation of a joint 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, followed by SGHMC.
- Fine-tuning the CNN used to classify CIFAR10. Currently we have shown that our SGHMC implementation can start classifying CIFAR10, however our accuracies could be higher with more time to fine-tune it.
- Investigate further methods of estimating the noise model B , as using empirical Fisher was computationally expensive
- Compare thoroughly with the more popularly used Variational Inference Method.

References