Departmental Coversheet

Hilary Term 2022 Mini-project

Paper title: Reproducing the paper 'Stochastic Gradient Hamiltonian Monte Carlo' by Tianqi Chen, Emily B. Fox and Carlos Guestrin

Candidate numbers: 1302365, 1059459, 1060482 and 1058141

Degrees: DPhil Mathematics, MSc Advanced Computer Science, MSc Advanced Computer Science, DPhil Computer Science

Reproducing the paper: *Stochastic Gradient Hamiltonian Monte Carlo* by Tianqi Chen, Emily B. Fox and Carlos Guestrin

Candidate Numbers: 1302365, 1059459, 1060482 and 1058141

Abstract

In this report we focus on the Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) algorithm proposed in 'Stochastic Gradient Hamiltonian Monte Carlo' [CFG14b] by Chen, Fox and Guestrin. We reproduced a number of the experiments from this paper to confirm the theoretical results presented in 'Stochastic Gradient Hamiltonian Monte Carlo'. On top of this, we identified several directions for extending the work of Chen, Fox and Guestrin. We implemented all our algorithms so that they can be directly used with Pyro [Bin+19] and we propose a novel algorithm named SGNUTS, based on the popular 'No U-Turn Sampler' (NUTS) first introduced in [HG11]. Additionally, we implemented a scheme for estimating the value of the noise model B using the empirical Fisher information [AKW12b]. We also demonstrate that our implementation of SGHMC is flexible enough to be used to achieve good results on other datasets and more complex models such as convolutional neural networks (CNN).

1 Introduction

Hamiltonian Monte Carlo (HMC) provides us with a useful way to sample from a posterior distribution. HMC is an MCMC sampling method that uses all the available data at each step, and it requires a potential function of the form:

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

which, along with $\nabla U(\theta)$, is sufficient to sample from the posterior (as will be discussed in the background section). Computing $U(\theta)$ and $\nabla U(\theta)$ can be computationally expensive for large datasets, which has motivated the development of Stochastic Gradient Hamiltonian Monte Carlo (SGHMC), first introduced by the paper in question [CFG14b]. SGHMC uses randomly sampled mini-batches of data to produce noisy estimates of the gradient, which are denoted by $\nabla \widetilde{U}(\theta)$. We decided to investigate this paper because it convinced us that SGHMC is a strong candidate for scalable Bayesian inference and it would be interesting to investigate how it compares to more popular methods such as Variational Inference. In this paper, the authors start with a description of HMC, and then introduce Naïve SGHMC algorithm, demonstrating the pitfalls of using noisy gradient estimates. They then go on to introduce the full SGHMC algorithm which uses friction to overcome the need for a costly MH correction step. They then run a number of experiments using SGHMC to empirically back up their

theoretical claims and show that SGHMC is a candidate algorithm for scalable Bayesian inference.

We were further motivated to choose this paper as SGHMC is a relatively simple algorithm, and so we would have more time to investigate other datasets and directions. Another consideration was that running these experiments wouldn't be very computationally demanding, allowing us to run many experiments on our own machines. Below we detail exactly which experiments from [CFG14b] we decided to reproduce:

- Sampling θ using the potential function $U(\theta) = -2\theta^2 + \theta^4$, with $\mathcal{N}(0,4)$ noise added to the gradient of this to give $\nabla\widetilde{U}(\theta)$. This noise potential is a proxy for the noisy potential used by SGHMC. We used the following algorithms: HMC (with and without MH correction), Naive SGHMC (with and without MH correction) and SGHMC.
- Sampling (θ, r) using the potential function $U(\theta) = \frac{1}{2}\theta^2$, , with $\mathcal{N}(0,4)$ noise added to the gradient of this to give $\nabla\widetilde{U}(\theta)$. We used the following algorithms: HMC, Naive SGHMC (with and without momentum resampling) and SGHMC.
- Comparing the autocorrelation times, as well as the error in the covariance of the samples, of SGHMC and SGLD.
- Classifying the MNIST dataset [Den12] using SGHMC as well as with Stochastic Gradient Descent (SGD), Stochastic Gradient Descent (SGD) with momentum, and Stochastic Gradient Langevin Dynamics (SGLD).

We also considered some new ideas:

- We extended the 'No U-Turn Sampler' (NUTS) from [HG11] to work with SGHMC to produce our novel algorithm SGNUTS.
- We ran the Bayesian neural network (BNN) to classify a new dataset, namely, FashionMNIST. [XRV17].
- We demonstrated that our implementation of SGHMC can be used with Convolutional Neural Networks (CNNs) to classify CIFAR10 [Kri09].
- 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.

The repository for our code can be found at `https://github.com/sacktock/SGHMC`.

2 Background

2.1 HMC

Hamiltonian Monte Carlo ([Dua+87; Nea11]) is a Markov Chain Monte Carlo (MCMC) sampling algorithm. Given a target probability distribution — in our case the posterior distribution of a set of variables θ given independent observations $x \in \mathcal{D}$ — it produces samples by carrying out a random walk over the parameter space using Hamiltonian dynamics.

We begin with the prior distribution $p(\theta)$ and likelihood $p(x \mid \theta)$. Using these we define the *potential energy* function U:

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

Note that, using Bayes' rule, we have that the posterior $p(\theta \mid \mathscr{D}) \propto \exp(-U)$. Hamiltonian dynamics introduces an auxiliary set of momentum variables r. These dynamics have a physical interpretation in which an object moves about a landscape determined by U. We let this object have *mass matrix* M, which is typically set to the identity matrix. Then $U(\theta)$ represents the potential energy of the object, and its kinetic energy is given by $\frac{1}{2} r^\top M^{-1} r$. The total energy of the system is a quantity known as the *Hamiltonian function*:

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

The development of the system is governed by the following equations.

$$\begin{aligned} \mathrm{d}\theta &= M^{-1} r \, \mathrm{d}t \\ \mathrm{d}r &= -\nabla U(\theta) \, \mathrm{d}t \end{aligned}$$

To simulate these continuous dynamics in practice, we must use a discretised version of these equations. To correct for the inaccuracies introduced by doing so, it is necessary to make a *Metropolis-Hastings correction step*. A simple algorithm is given in Algorithm 1.

Algorithm 1 A simple HMC algorithm

for $t = 1, 2, \ldots$ **do**
  $r \sim \mathscr{N}(0, 1)$ ▷ Resample momentum
  $(\theta_0, r_0) = (\theta, r)$
  for $i = 1$ to m **do**
    $\theta \leftarrow \theta + \epsilon M^{-1} r$
    $r \leftarrow r - \epsilon \nabla U(\theta)$
  end for
  $u \sim \text{Uniform}[0, 1]$
  $\rho = \exp(H(\theta, r) - H(\theta_0, r_0))$ ▷ Acceptance probability
  if $u < \min(1, \rho)$ **then** ▷ Only accept new state with probability ρ
    $\theta = \theta_0$
  end if
end for

In practice, the dataset $\mathscr{D}$ may be large, and so running Algorithm 1 may be computationally expensive, as the complexity of calculating ∇U scales with $|\mathscr{D}|$. One idea to combat this is to simulate the Hamiltonian system using only a subset of the data at a time, in analogy with stochastic gradient descent. This method is known as Stochastic Gradient Hamiltonian Monte Carlo (SGHMC), however before explaining the SGHMC algorithm we begin with Naïve SGHMC.

2.2 Naïve SGHMC

To understand the Naïve SGHMC method, consider sampling a minibatch $\widetilde{\mathscr{D}} \subset \mathscr{D}$ uniformly at random. We estimate the gradient $\nabla U(\theta)$ using this minibatch as follows:

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

Appealing to the Central Limit Theorem, we imagine that the noisy estimate $\nabla \widetilde{U}(\theta)$ is normally distributed about $\nabla U(\theta)$, with some covariance matrix $V(\theta)$. The naïve adaptation of HMC to this stochastic scenario simply replaces $\nabla U(\theta)$

with $\nabla\widetilde{U}(\theta)$ in Algorithm 1. The corresponding discrete system is then the ϵ-discretisation of the following dynamics:

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

where $B(\theta) = \frac{1}{2}\epsilon V(\theta)$.

Unfortunately, such a dynamical system can diverge quite rapidly from the true posterior distribution [Nea11], which necessitates frequent Metropolis-Hastings steps. Such steps are costly since calculating the acceptance probability requires the use of the whole dataset. The method 'Stochastic Gradient Hamiltonian Monte Carlo' (SGHMC) proposed in [CFG14b] addresses this shortcoming. The idea is to incorporate friction into the dynamical system, which works to counteract the noise introduced by selecting a subset of the data.

2.3 SGHMC

The SGHMC method adds a 'friction term' $BM^{-1}r$ to the momentum update. Since we are unlikely to know the noise model B in practice, we instead take an estimate $\widehat{B}$ of B, together with a user-specified friction term $C \succeq \widehat{B}$ and simulate the following dynamics.

$$\begin{aligned} \mathrm{d}\theta &= M^{-1}r\,\mathrm{d}t \\ \mathrm{d}r &= -\nabla U(\theta)\mathrm{d}t - CM^{-1}r\,\mathrm{d}t + \mathcal{N}(0, 2(C-\widehat{B}(\theta))\mathrm{d}t) + \mathcal{N}(0, 2B(\theta)\mathrm{d}t) \end{aligned}$$

The algorithm is given in Algorithm 2. Ideally we would have $\widehat{B} = B$, allowing us to sample from the exact posterior distribution. In practice, we must rely on an inaccurate estimate $\widehat{B}$, which means our samples only approximately follow the posterior distribution, however typically this is sufficient for our purposes. The simplest choice is $\widehat{B} = 0$. A better but more costly estimate is $\widehat{B}(\theta) = \frac{1}{2}\epsilon\widehat{V}(\theta)$, where $\widehat{V}$ is the observed (empirical Fisher) information [AKW12a]. The friction term C can then be taken as a hyperparameter, and set so as to counteract the inaccuracies of the estimate $\widehat{B}$.

Algorithm 2 The SGHMC algorithm

for $t = 1, 2, \ldots$ **do**
  $r \sim \mathcal{N}(0, 1)$ ▷ Resample momentum
  for $i = 1$ to m **do**
    $\theta \leftarrow \theta + \epsilon M^{-1}r$
    $r \leftarrow r - \epsilon\nabla\widetilde{U}(\theta) - \epsilon CM^{-1}r + \mathcal{N}(0, 2(C-\widehat{B}(\theta))\epsilon)$
  end for
end for

Lastly, we note that the added friction term prevents the dynamical system from diverging, meaning that we no longer require the Metropolis-Hasting's steps that were required for naïve SGHMC. This means we do not need to consider all of the data at each step in the SGHMC algorithm, making it fast to run even on large datasets.

3 Implementation Details

We implemented the following algorithms from scratch: HMC, SGHMC, SGLD (stochastic gradient Langevin dynamics [WT11]), SGD (stochastic gradient

descent), SGD with Nesterov momentum and SGNUTS (stochastic gradient No U-Turn Sampler) — a novel extension to SGHMC that uses ideas from the popular No U-Turn Sampler [HG11]. All of our implementations subclass Pyro's `MCMCKernel` and are designed to be used directly with Pyro [Bin+19] — a universal probabilistic programming language (PPL) written in Python. In Pyro the user specifies a model which is a probabilistic program (PP) that describes a posterior distribution $p(\theta \mid \mathscr{D})$ that we want to sample from; θ corresponds to the sampled parameters or latent parameters of the model and $\mathscr{D}$ corresponds to the observed data.

Pyro already comes with the following MCMC samplers: HMC, MH and NUTS. So while the algorithms we implemented are not novel they are in fact innovative since Pyro doesn't come with any of them already built. The main reason for choosing to implement our algorithms on top of Pyro is because Pyro comes with a method `initialize_model` that given a Pyro PP or model P transforms it into a potential function U. Once we have U we can pass the latent and observed data $(\theta, \mathscr{D})$ to U, which computes the negative log joint $-\log p(\theta, \mathscr{D})$. Bayes' rule tells us that $p(\theta \mid \mathscr{D}) \propto p(\theta, \mathscr{D})$ which is typically all we need for MCMC samplers and even simpler ones such as Importance and Rejection samplers [BN06]. Pyro also comes with the method `potential_grad`, which given $(\theta, \mathscr{D})$ computes the gradient of U with respect to the parameters θ. The result is that we can specify any arbitrary PP and apply our algorithms to them — letting Pyro handle the transformation from PP to potential function and the gradient computations.

In traditional MCMC samplers the observed dataset $\mathscr{D}$ is constant and so once a Pyro PP has been transformed into a potential function $U(\theta) = -\log p(\theta, \mathscr{D})$ we need not change it. Unfortunately this is less straight forward for stochastic gradient samplers such as SGHMC and SGLD since we subsample the full dataset $\mathscr{D}$ by sampling minibatches $\widetilde{\mathscr{D}}$, where $\widetilde{\mathscr{D}} \subset \mathscr{D}$. In both SGHMC and SGLD we require that the potential function has the form:

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

Unfortunately when we subsample the dataset and call `initialize_model` Pyro doesn't explicitly supply us with the likelihood term $p(\widetilde{\mathscr{D}} \mid \theta)$ - it only gives us the negative log joint $-\log p(\theta, \widetilde{\mathscr{D}})$, so to get the desired behaviour above we had to get our hands dirty and modify Pyro's source code. As a result every time we generate a new sample using SGHMC or SGLD we have to call `initialize_model` so that it gives us the correct $\widetilde{U}(\theta)$ for some given minibatch $\widetilde{\mathscr{D}}$, although this is a small price to pay for much quicker gradient computations.

For the estimate of $\widehat{B}$, in our implementation we provide two options: take $\widehat{B} = 0$ or use $\widehat{B}(\theta) = \frac{1}{2}\epsilon\widehat{V}(\theta)$, where $\widehat{V}$ is the observed information, as suggested in [CFG14b]. While the latter provides a better estimate, it is much slower, and requires more memory. When $\widehat{B}$ is estimated using the observed information, we provided the option to compute it once at setup time, recalculate every sample, or recalculate every step when simulating the dynamics.

4 Experiments

4.1 Simulated examples

In this next section we present the results and intuition behind the first three experiments in [CFG14b]. We reproduced these three experiments by converting

the MATLAB codes provided by the authors Chen, Fox and Guestrin [CFG14a] into Python code. We used this, rather than our own implementation of SGHMC, as our version does not directly interface with a given potential function, but only Pyro PPs. The experiments that we replicated mostly consisted of checking whether the samples produced by SGHMC and the other algorithms actually follow the target distribution.

Experiment 1

As in [CFG14b] we considered the potential function $U(\theta) = -2\theta^2 + \theta^4$. This is sufficient for the purpose of checking the convergence to the target distribution for HMC. In [CFG14b], they show theoretically that Naïve SGHMC without MH steps should not converge, but that SGHMC should. To check these theoretical convergence results, we introduce noise into the gradient, setting:

$$\nabla \widetilde{U}(\theta) = \nabla U(\theta) + \mathcal{N}(0, 4)$$

This now corresponds to setting the noise covariance to $V = 4$ in the description of Naïve SGHMC we gave earlier. We then take samples using HMC, Naïve SGHMC and SGHMC. We also note that both HMC and Naïve SGHMC use Metropolis Hastings corrections (MH), while SGHMC has no need to. This allows us to test 5 algorithms: HMC (with MH), HMC (without MH), Naïve SGHMC (with MH), Naïve SGHMC (without MH), and SGHMC. As a small extension we also performed the same experiment with the potential function $U(\theta) = \frac{1}{2}\theta^2$. This allows us the compare the performance on bimodal and unimodal distributions. The results are shown in Figures 1 and 2. It can be

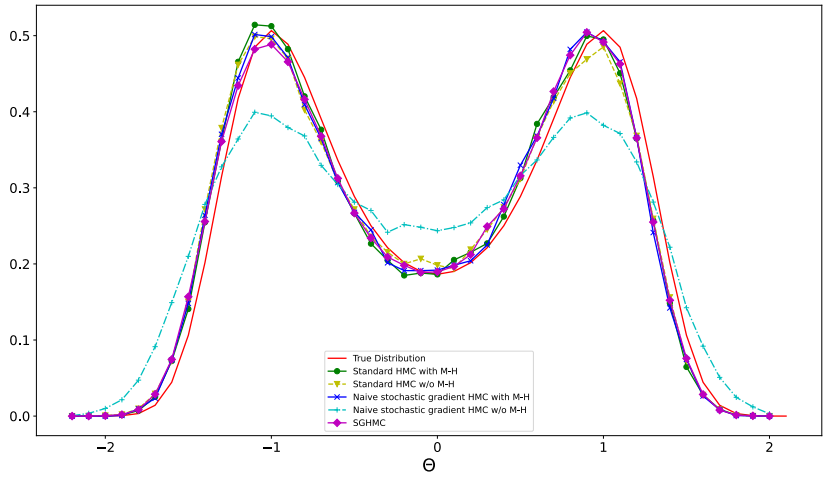


Figure 1: Distributions of different sampling algorithms for the target function $U(\theta) = -2\theta^2 + \theta^4$

seen that both HMC algorithms perform well, as does Naïve SGHMC with MH corrections. However, as we already discussed these algorithms are computationally demanding when used on large datasets. Furthermore, Naïve SGHMC does not converge to the correct distribution, empirically confirming Corollary 3.1 in [CFG14b]. On the other hand, SGHMC is fast and maintains the target distribution as its invariant distribution and so can be considered as a useful candidate for scalable Bayesian inference. We note here that our diagram for $U(\theta) = -2\theta^2 + \theta^4$ mirrors Figure 1 the paper very closely.

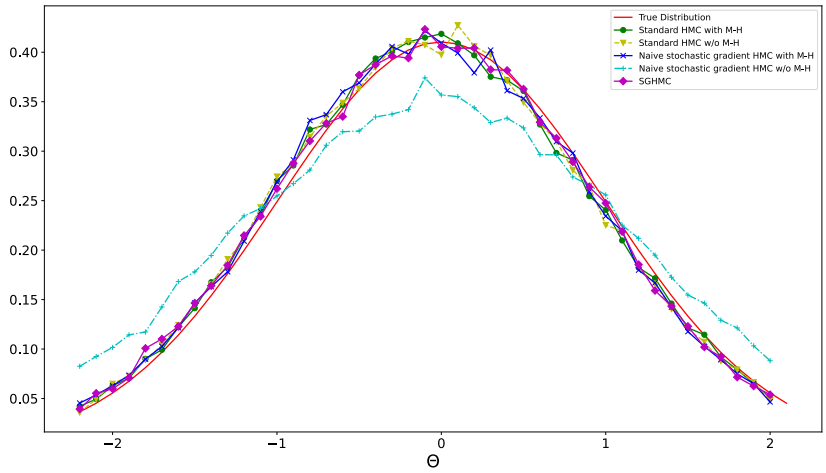


Figure 2: Distributions of different sampling algorithms for the target function $U(\theta) = \frac{1}{2}\theta^2$

Experiment 2

Next we used HMC to sample (θ, r) from the potential function $U(\theta) = \frac{1}{2}\theta^2$,and as before, we simulated the noisy gradient with $\nabla\widetilde{U}(\theta) = \nabla U(\theta) + \mathcal{N}(0,4)$. Except in the case specified, we never resample the momentum r here. We plotted the samples found using perfect gradient Hamiltonian dynamics (HMC), noisy gradient Hamiltonian dynamics (effectively Naïve SGHMC), noisy gradient Hamiltonian dynamics with momentum resampling and noisy Hamiltonian dynamics with friction (effectively SGHMC). We present the results below in Figure 3, which closely match those of [CFG14b]. Note that in [CFG14b] the Chen, Fox and Guestrin claimed that the samplers were run for 15000 steps, however their plot shows far fewer points. We simulated the aforementioned algorithms for 15000 steps and for 360, and decided that the later better illustrated the dynamics and aligned more closely with the original plot. These results show that friction (green) keeps Hamiltonian dynamics much closer to the true Hamiltonian dynamics (blue) in a noisy system. Resampling the momentum every 50 steps also seems to mitigate the problem which is probably why the authors Chen, Fox and Guestrin include it in their pseudocode description of SGHMC. Figure 3 also supports the results of Theorem 3.1 in [CFG14b], that the sampled distribution tends to the uniform distribution over time, rather than the target distribution.

Experiment 3

Finally, we consider the following correlated distribution, as is done in [CFG14b]. Chen, Fox and Guestrin note that the strength of HMC is typically in its efficiency in sampling from correlated distributions, and that SGHMC maintains this property. We reproduced the following experiment in [CFG14b], where the authors Chen, Fox and Guestrin compare the autocorrelation times of SGHMC and SGLD by considering the following potential function: $U(\theta) = \frac{1}{2}\theta^T\Sigma^{-1}\theta$, with $\nabla\widetilde{U}(\theta) = \Sigma^{-1}\theta + \mathcal{N}(0, I)$, where $\Sigma_{11} = \Sigma_{22} = 1$ and $\rho = \Sigma_{12} = 0.9$.

Figure 4 presents the results of this experiment. The first plot shows the first 50 samples produced using both algorithms. The second shows a plot of

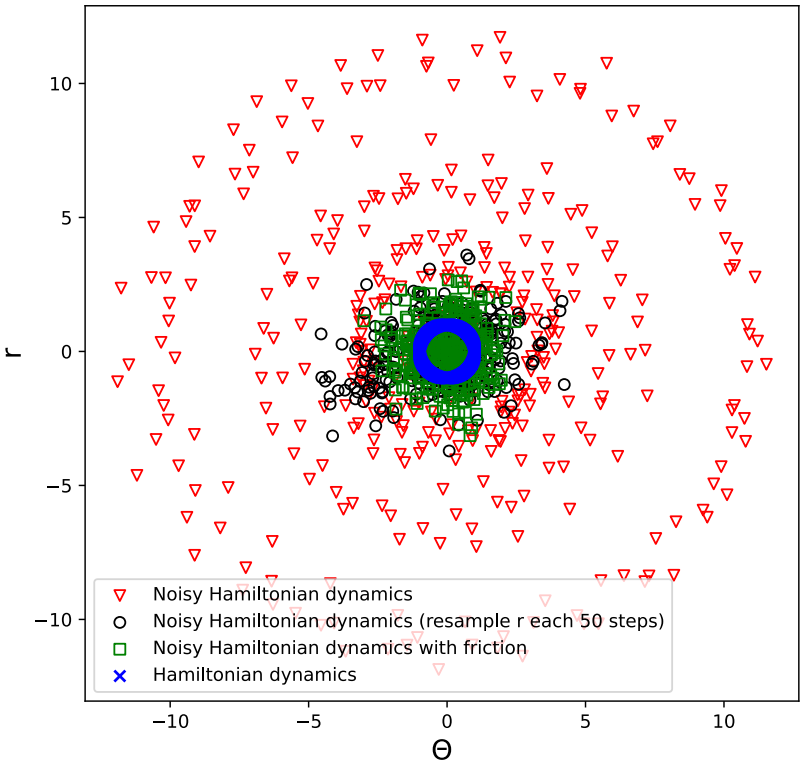


Figure 3: Plotting the trajectory of (θ, r) under various samplers for 360 steps

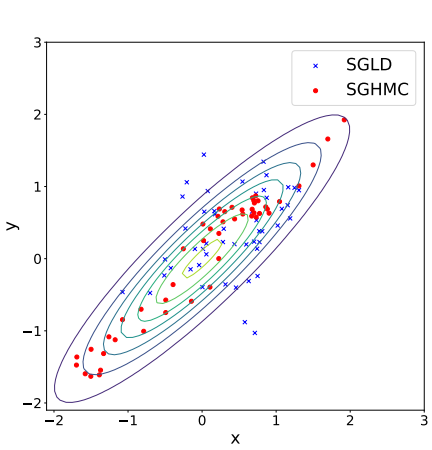
the average absolute error of the sample covariance against the autocorrelation time for 5 different values of the step-size (for more details see [CFG14b]. We see that SGHMC samples quickly become uncorrelated, which is not the case for the SGLD samples - this indicates that there is an advantage in adopting SGHMC. While both SGHMC and SGLD accurately sample from the posterior, SGHMC needs fewer samples to fully explore the posterior distribution and we see that after just a few samples SGHMC has already explored the tails of the target distribution. Therefore we empirically see that SGHMC has a much faster mixing time than SGLD which is an inherent advantage.

4.2 Bayesian Neural Networks for Classification

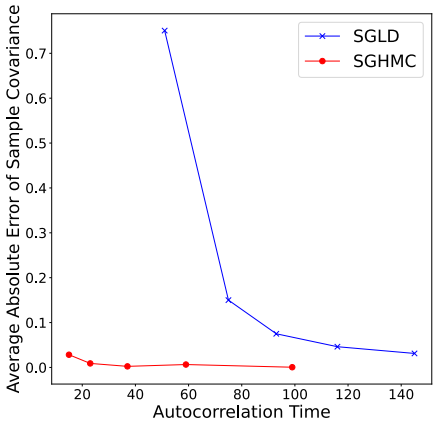
For the Bayesian neural network (BNN) MNIST classification we actually used the reparameterisation of SGHMC described in the section "Connection to SGD with Momentum" of [CFG14b]. The SGHMC algorithm is reframed in terms of learning rate and momentum decay, and simulates the following dynamics instead:

$$\begin{cases} \delta\theta = v \\ \delta v = -\eta\nabla\widetilde{U}(\theta) - \alpha v + \mathcal{N}(0, 2(\alpha - \widehat{\beta})\eta) \end{cases}$$

where $\eta = \epsilon^2 M^{-1}, \alpha = \epsilon M^{-1} C, \widehat{\beta} = \eta M^{-1}\widehat{B}$. In all our experiments in this section we set the mass matrix M to the identity, and the noise model $\widehat{\beta} = \widehat{B} = 0$. Other than the architecture of the BNN there are now only 3 hyperparameters



(a) First 50 samples of SGHMC and SGLD



(b) Autocorrelation time versus absolute error of the sample covariance, for each of five step-sizes

Figure 4: Contrasting sampling of a bivariate Gaussian with correlation using SGHMC versus SGLD.

for SGHMC, the learning rate η, the momentum decay α and the batch size $|\widetilde{\mathscr{D}}|$. In all our experiments we fixed $|\widetilde{\mathscr{D}}| = 500$ which follows from [CFG14b].

To build on top of the work in [CFG14b] we implemented learning rate annealing for SGLD, and following [WT11] we weighted the samples by the learning rate as follows:

$$\hat{\mathbf{p}} := \frac{\sum_{t=1}^{T} \eta_t f_{\theta_t}(\mathbf{x})}{\sum_{t=1}^{T} \eta_t}$$

where f_θ is our classifier parameterized by θ, $\mathbf{x}$ can be thought of as the "test set" and $\hat{\mathbf{p}}$ is an unormalized probability vector. Note that since f is a classifier we can ignore the denominator because it won't affect the argmax. Our BNN followed the same architecture as in [CFG14b], that is one linear layer with 100 hidden units followed by ReLU activation followed by another linear layer and a log softmax for multi-class classification. The weights and biases for both linear layers are sampled from univariate standard normal distributions, but the Pyro method `to_event()` declares dependence between the parameters.

Our implementations of SGD and SGD with momentum are meant to be used directly with Pyro, and so Gaussian priors on the weights and biases is equivalent to L2 regularization in the non-Bayesian paradigm. We experimented with regularization strengths of $\lambda \in \{0.1, 1.0, 10.0\}$ and found $\lambda = 1.0$ to be the most effective. Additionally we implemented weight decay for both SGD and SGD with momentum but found that this didn't improve anything in this setting.

For the momentum based algorithms, SGHMC and SGD with momentum, we tried $\eta \in \{1.0, 2.0, 4.0, 8.0\} \times 10^{-6}$, and $\alpha \in \{0.1, 0.01, 0.001\}$. For SGHMC the best configuration was $\eta = 2.0 \times 10^{-6}, \alpha = 0.01$, and for SGD with momentum the best configuration was $\eta = 1.0 \times 10^{-6}, \alpha = 0.01$.

For SGLD and SGD, we tried $\eta \in \{1.0, 2.0, 4.0, 6.0\} \times 10^{-5}$, for SGLD we also tried learning rate annealing but it proved not to make much of a difference in this setting so we ignored it in the end. The best configuration for SGLD was $\eta = 4.0 \times 10^{-5}$, and for SGD the best configuration was $\eta = 1.0 \times 10^{-5}$.

For MNIST we ran each of the algorithms for 800 epochs with 50 warmup

epochs. For the sampling algorithms the idea is that after warmup we have reached the posterior; we then perform Bayesian averaging over the entire set Θ, which consists of all of the sampled parameterizations of the BNN up to that point. The general framework of Bayesian averaging in classification tasks is described in Section II of [Jos+20] and is used to report the test error as follows:

$$\hat{\mathbf{p}} := \sum_{\theta_i \in \Theta} f_{\theta_i}(\mathbf{x}) \qquad ; \qquad \hat{\mathbf{y}} = \text{argmax}_i \{p_i \in \hat{\mathbf{p}}\}$$

However, for the optimization algorithms we take just the most recent sample / set of parameters as a point estimate and report the test error. Figure 5 presents our results. The results we get from MNIST classification align very closely with

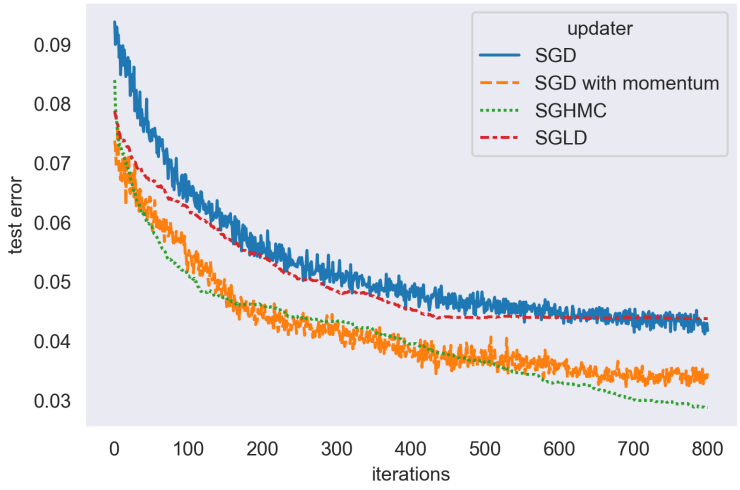
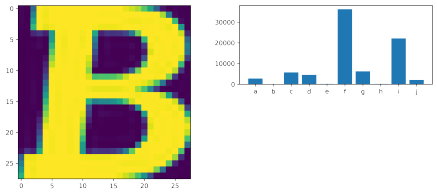


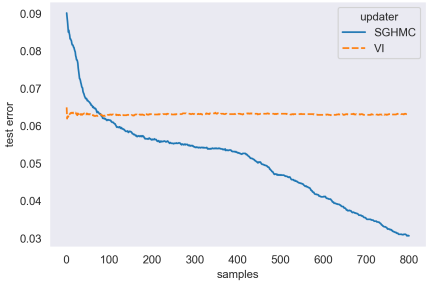
Figure 5: Reproducing the MNIST classification experiment from [CFG14b]; SGHMC ($\eta = 2.0 \times 10^{-6}, \alpha = 0.01,$ `resample_n = 0`), SGLD ($\eta = 4.0 \times 10^{-5}$), SGD ($\eta = 1.0 \times 10^{-5}$), SGD with momentum ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01$)

those in [CFG14b] and so we come to the same conclusion; the need for scalable and efficient Bayesian inference algorithms. The key benefit of BNNs is that we are not overconfident on out-of-distribution examples, Figure 6a illustrates that we still maintain this property when using SGHMC to approximately sample from the posterior distribution. We additionally conducted a brief comparison between Variational Inference (VI) and SGHMC in this setting, Figure 6b outlines our findings. The initial results suggest that SGHMC performs better than VI in this setting, although this is not the full picture. Once VI fits the variational posterior distribution q_ϕ as closely as possible to the true posterior it takes only hundreds of samples to characterise q_ϕ, whereas SGHMC requires several more samples to characterise the true posterior. In practice storing thousands of parameterisations of the same NN is very costly and so this is probably why VI is a more popular choice for Bayesian inference.

We conclude this section by demonstrating our algorithms can be applied to other datasets and more complicated models, Figure 7a presents the results of running the same BNN architecture on FashionMNIST, and Figure 7b demonstrates that our implementation of SGHMC can be used with convolutional neural networks (CNNs).

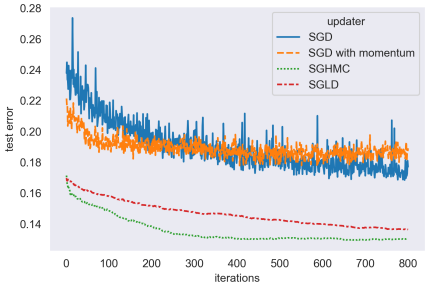


(a) Uncertainty in out-of-distribution examples

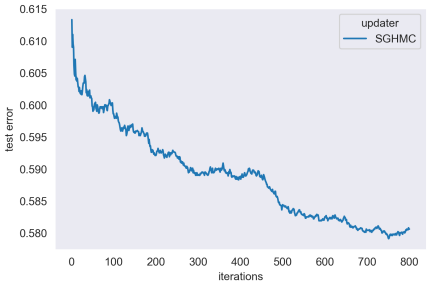


(b) VI and SGHMC on MNIST

Figure 6: **Left** (a) illustrates that we get uncertainty estimates on out-of-distribution examples. **Right** (b) compares VI (Renyi ELBO, $\alpha = 0.01$, `num_particles` $= 2$) and SGHMC ($\eta = 2.0 \times 10^{-6}, \alpha = 0.01,$ `resample_n` $= 0$) on MNIST. For VI we draw 80000 samples from the variational posterior q_ϕ and report the test error by Bayesian averaging. For SGHMC we do the same, except we are approximately sampling from the true posterior $p(\theta \,|\mathscr{D})$.



(a) Experiment on FashionMNIST



(b) Convolutional BNN on CIFAR10

Figure 7: **Left** (a) FashionMNIST classification; 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` $= 100$. **Right** (b) Convolutional BNN with 2 convolutional layers, batch norm, max pooling and tanh activations followed by two Bayesian linear layers with tanh activation; SGHMC ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01,$ `resample_n` $= 0$) ; `warmup_epochs` $= 150$.

5 NUTS

5.1 The Basic Algorithm

In our current description of the SGHMC algorithm we have the user-defined hyperparameter m, the number of steps iterated over before we take a sample. If this number is too small, our samples will be correlated, and hence successive samples would appear to follow a random walk, and we would get slow mixing times. We demonstrate this behaviour by training 3 versions of SGHMC with $m = 1, 3, 5$ (with ϵm fixed). The learning curves below in figure 8 show that increasing m increases the speed with which SGHMC reaches low error rates.

However, if m is too large we waste computational power, as we continue to step through time even though each sample is already independent of the last.

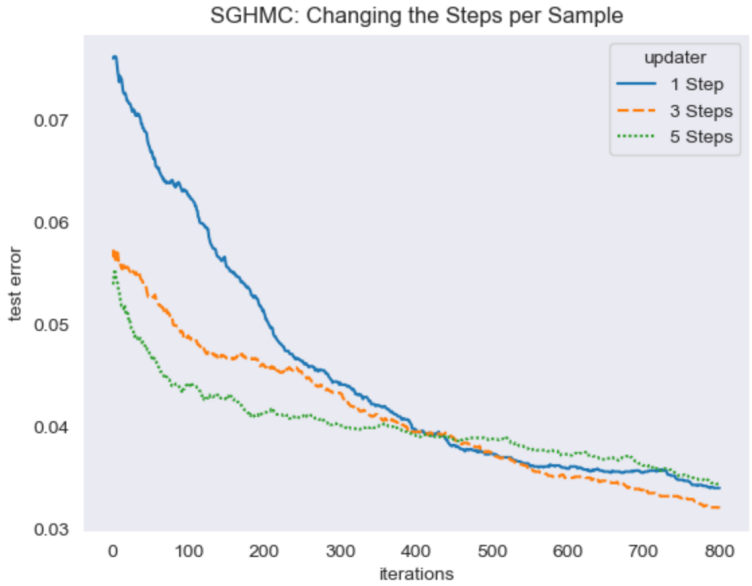


Figure 8: Changing the number of steps per sample. Each agent was run for 100 warmup epochs.

We would like to set m to its optimal value.

Hoffman and Gelman introduce the algorithm NUTS in their paper "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo" [HG11]. In its basic form this algorithm removes the need for a user to input a value for m in the standard HMC implementation. We converted this NUTS algorithm from being based on HMC to being based on SGHMC, and investigated its power. We will begin by presenting the high level overview of the original NUTS algorithm for HMC sampling.

The key idea is the concept of a 'U-Turn'. This is the point at which the samples of θ start to get closer to the initial value of θ, rather than away from it. This marks the point at which further steps will likely only waste computational power. Mathematically, for current position θ, initial position θ_0 and momentum r, this corresponds to the time at which:

$$\frac{d}{dt}|\theta-\theta_0|^2 = 2(\theta-\theta_0)\cdot\frac{d\theta}{dt} = 2(\theta-\theta_0)\cdot r < 0$$

where we use the fact that in the dynamics of HMC (and also SGHMC) we have $\frac{d\theta}{dt} = r$. This simple fact suggests an algorithm in which we draw the sample θ once we have performed enough steps so that $(\theta-\theta_0)\cdot r < 0$. However this is too simplistic an approach - as HMC is an instance of the Metropolis Hastings algorithm we require the Markov Chain of θ to be reversible, which is not the case here.

To remedy this problem, NUTS requires keeping track of a set $\mathscr{B}$, which contains all values of (θ, r) as steps are taken both forward and backwards in time. The values at the earliest and latest times considered across a single trajectory are labelled (θ^-, r^-) and (θ^+, r^+) respectively. NUTS starts from a single (θ, r) and then steps either forward or backwards one step. It then steps forward or backwards 2 steps, then 4 steps, then 8 steps etc. until a 'U-Turn' is seen. See Algorithm 3.

Algorithm 3 The NUTS algorithm

Require: (θ_0, r_0)
  $r \sim \mathcal{N}(0,1)$
  $n \leftarrow 1$
  $\mathscr{B} \leftarrow \{(\theta_0, r_0)\}$
  $(\theta^-, r^-) \leftarrow (\theta_0, r_0)$
  $(\theta^+, r^+) \leftarrow (\theta_0, r_0)$
  while there is no U-Turn at (θ^-, r^-) nor at (θ^+, r^+) (ie $(\theta^+ - \theta^-) \cdot r^- \geqslant 0$ and $(\theta^+ - \theta^-) \cdot r^+ \geqslant 0$) **do**
    With probability $\frac{1}{2}$, choose to go *forwards* or *backwards* in time
    if *forwards* **then**
      $(\theta, r) \leftarrow (\theta^+, r^+)$
      for $i = 1$ to n **do**
        $(\theta, r) \leftarrow$ step forward in time from (θ, r)
        $\mathscr{B} \leftarrow \{(\theta, r)\} \cup \mathscr{B}$
      end for
      $(\theta^+, r^+) \leftarrow (\theta, r)$
    end if
    if *backwards* **then**
      $(\theta, r) \leftarrow (\theta^-, r^-)$
      for $i = 1$ to n **do**
        $(\theta, r) \leftarrow$ step backwards in time from (θ, r)
        $\mathscr{B} \leftarrow \{(\theta, r)\} \cup \mathscr{B}$
      end for
      $(\theta^-, r^-) \leftarrow (\theta, r)$
    end if
    $n \leftarrow 2n$
  end while
  Carefully choose a subset $\mathscr{C} \subseteq \mathscr{B}$ ▷ *This step is the key to the Markov Chain being reversible; we don't go into detail here*
  Sample an element of $\mathscr{C}$

The benefit to this algorithm is that it removes the need to set the number of steps performed before we sample, as we keep stepping until a 'U-Turn' is seen. We should note here that in the original form of NUTS, the 'step' being referred to in Line 11 and Line 19 is the step of the HMC algorithm. We edited the NUTS Pyro source code to make it perform SGHMC steps, and we named this SGNUTS (Stochastic Gradient No U-Turn Sampler).

There were some doubts as to whether the NUTS algorithm would work when using SGHMC steps instead of HMC steps. This was because NUTS requires the ability to step backwards in time, while an SGHMC step includes an injection of stochastic noise. As there is no action that undoes this injection, there was a worry that the backwards step through time in NUTS would become a problem. We explain how we attempted to solve this problem at the end of the next section.

5.2 Implementation of SGNUTS

To build SGNUTS we started with the Pyro source code for NUTS [Ube19] which takes the Pyro HMC class as its parent. We altered this to instead inherit from our SGHMC class. This required removing step-size adaptation and mass matrix adaptation functionality from NUTS, as our implementation of SGHMC was not able to interface with this. It also required introducing some caching methods into the SGHMC class - to help keep things simple in our original SGHMC class we did this in a new class, named SGHMC_for_NUTS. Most importantly, we had to alter the (θ, r) step update rule which is hardcoded in NUTS. This meant that instead of being the HMC update step it was now the SGHMC update step of:

$$\theta \leftarrow \begin{cases} \theta + \epsilon M^{-1} r, & \textit{forwards step} \\ \theta - \epsilon M^{-1} r, & \textit{backwards step} \end{cases}$$

$$r \leftarrow \begin{cases} r - \epsilon \nabla \tilde{U}(\theta) - \epsilon C M^{-1} r + \mathcal{N}(0, 2(C - \hat{B})\epsilon), & \textit{forwards step} \\ r + \epsilon \nabla \tilde{U}(\theta) - \epsilon C M^{-1} r + \mathcal{N}(0, 2(C - \hat{B})\epsilon), & \textit{backwards step} \end{cases}$$

In particular, note that there is an injection of stochastic noise and momentum-reducing friction in both the forward steps and the backward steps. This is not ideal, as our backwards step in time does not undo a forward step, which likely breaks the reversibility of the Markov Chain being considered in NUTS. However we investigated this nonetheless.

5.3 Results

We tested the SGNUTS algorithm on the BNNs described earlier. We ran SGNUTS for only 20 warmup epoch and 50 epochs as the code was slower than SGHMC to run. The learning curves were as shown in Figure 9.

The final accuracies were 0.94 for MNIST and 0.85 for FashionMNIST which are similar to the accuracies obtained using SGHMC (0.97 and 0.86 respectively). This accuracy demonstrates that there is convergence to the true posterior in SGNUTS. This suggests that, like SGHMC itself, the SGNUTS Markov Chain is likely reversible in some (non traditional) sense. Denoting the posterior as $\pi(\theta, r)$, and transition kernels as $P(\theta, r|\theta', r')$, it is shown in [CFG14b] that SGHMC satisfies:

$$\pi(\theta, r) P_{SGHMC}(\theta, r|\theta', r') = \pi(\theta', -r') P_{SGHMC}(\theta', -r'|\theta, -r)$$

and it is suggested that this is the reason why the SGHMC algorithm works, despite not being reversible in the traditional sense. It would be interesting to consider if a similar property holds for P_{SGNUTS}.

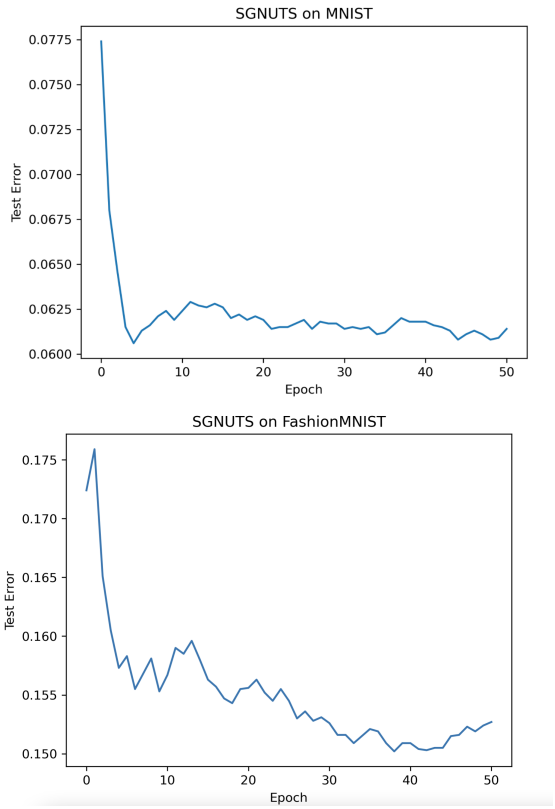


Figure 9: Classifying MNIST and FashionMNIST with SGNUTS

While accurate, SGNUTS is slower to run than SGHMC to produce a single sample. It does however reach relatively high accuracies in a short amount of time. We measured the accuracy after the first epoch of SGHMC and NUTS as we changed the number of warmup epochs, and we measured how long they took to train.

Number of Warmup-Epochs	SGHMC		SGNUTS	
	Accuracy	Time (s)	Accuracy	Time (s)
0	0.7747	3.2	0.8715	28.3
1	0.5764	4.5	0.9296	127.2
2	0.4141	6.2	0.9231	453.3
5	0.6471	11.4		
10	0.7100	20.3		
25	0.8866	50.2		

In particular, SGNUTS reached an accuracy of 0.87 in 28s, while SGHMC reached 0.89 in 50s. This suggests a new algorithm that uses both SGNUTS and SGHMC — we could run a single epoch of SGNUTS so that we quickly approach the posterior distribution, at which point we switch to running SGHMC. It should be noted the speed up suggested by the above results would only be 30s seconds, however maybe on more complex datasets this could be larger. It would be interesting to investigate this if we had more time.

6 Conclusion

For this project, we have implemented a number of algorithms for the purpose of replicating and extending the experiments in [CFG14b]. We opted for a close integration with Pyro which allowed us to make use of its probabilistic programming framework.

Replicating the analysis in [CFG14b] we compared the performance of SGHMC with related samplers. We confirmed their results that on the toy models SGHMC samples accurately from the posterior and efficiently samples from correlated distributions. Next, we tested SGHMC on a BNN for classifying MNIST digits, comparing it to SGLD, SGD and SGD with momentum. We obtained similar results to the original experiment. Moreover, we expanded the analysis by testing SGMHC on a BNN for the Fashion-MNIST dataset and a CNN for CIFAR10. In both cases SGMHC performed very well. In the former we further compared SGLD, SGD and SGD with momentum, with SGMHC producing the superior performance. Given the close integration of our codebase with Pyro, it is easy to extend to further models and datasets, which we would do if given more time.

As a further extension, we briefly compared SGHMC with the more popular variational inference method, using our BNN on the MNIST dataset. Over a large number of epochs SGHMC performs better than VI. However, the latter converges faster, and requires fewer samples to give a representative picture of the posterior. With more time, we would like to investigate this further, for example by comparing the sampling efficiency more in-depth.

Our implementation of SGHMC provided the option of estimating the noise model using the observed information, as suggested in [CFG14b]. Given more time, we would like to investigate alternative methods of estimating the noise model, ideally ones less computationally expensive.

Our last extension to [CFG14b] was the specification and implementation of our new algorithm SGNUTS, which combines SGHMC with the No-U-Turn Sampler. In spite of the lack of reversibility of the corresponding Markov process, our algorithm performs well on a BNN for the MNIST and Fashion-MNIST datasets. We found that while SGNUTS takes longer than SGHMC to produce a single sample, but reaches high accuracies in a relatively small number of steps. If we were to continue this project, we would investigate in more detail the theoretical and practical aspects of SGNUTS.

References

[AKW12a] Sungjin Ahn, Anoop Korattikara and Max Welling. *Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring*. 2012. DOI: 10.48550/ARXIV.1206.6380. URL: https://arxiv.org/abs/1206.6380.

[AKW12b] Sungjin Ahn, Anoop Korattikara and Max Welling. 'Bayesian posterior sampling via stochastic gradient Fisher scoring'. In: *arXiv preprint arXiv:1206.6380* (2012).

[Bin+19] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall and Noah D. Goodman. 'Pyro: Deep Universal Probabilistic Programming'. In: *J. Mach. Learn. Res.* 20.1 (Jan. 2019), pp. 973–978. ISSN: 1532-4435.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006. Chap. 11, pp. 523–541.

[CFG14a] Tianqi Chen, Emily Fox and Carlos Guestrin. 'Simulation Code'. In: 2014. URL: https://github.com/tqchen/ML-SGHMC/tree/master.

[CFG14b] Tianqi Chen, Emily Fox and Carlos Guestrin. 'Stochastic Gradient Hamiltonian Monte Carlo'. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, June 2014, pp. 1683–1691. URL: https://proceedings.mlr.press/v32/cheni14.html.

[Den12] Li Deng. 'The mnist database of handwritten digit images for machine learning research'. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[Dua+87] Simon Duane, A.D. Kennedy, Brian J. Pendleton and Duncan Roweth. 'Hybrid Monte Carlo'. In: *Physics Letters B* 195.2 (1987), pp. 216–222. ISSN: 0370-2693. DOI: https://doi.org/10.1016/0370-2693(87)91197-X. URL: https://www.sciencedirect.com/science/article/pii/037026938791197X.

[HG11] Matthew Hoffman and Andrew Gelman. 'The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo'. In: 2011. URL: https://arxiv.org/abs/1111.4246.

[Jos+20] Laurent Valentin Jospin, Wray Buntine, Farid Boussaid, Hamid Laga and Mohammed Bennamoun. 'Hands-on Bayesian neural networks–a tutorial for deep learning users'. In: *arXiv preprint arXiv:2007.06823* (2020).

[Kri09] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[Nea11] Radford M. Neal. 'Handbook of Markov Chain Monte Carlo'. In: ed. by BySteve Brooks, Andrew Gelman, Galin Jones and Xiao-Li Meng. Chapman and Hall/CRC, 2011. Chap. MCMC Using Hamiltonian Dynamics, pp. 113–162. ISBN: 9780429138508.

[Ube19] Inc. Uber Technologies. 'Pyro NUTS Code'. In: 2019. URL: https://docs.pyro.ai/en/stable/_modules/pyro/infer/mcmc/nuts.html.

[WT11] Max Welling and Yee W Teh. 'Bayesian learning via stochastic gradient Langevin dynamics'. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. Citeseer. 2011, pp. 681–688.

[XRV17] Han Xiao, Kashif Rasul and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 2017. DOI: 10.48550/ARXIV.1708.07747. URL: https://arxiv.org/abs/1708.07747.