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 **sghmc** [**sghmc**] by **sghmc**. We reproduce a number of the experiments from this paper, and we explain the intuition behind several of their results. On top of this, we extended their work in a number of ways. We attempted to estimate the value of the noise model $\hat{B}$ in their paper, and we test the algorithm on more than just the MNIST dataset. We also introduce a novel algorithm which we have named SGNUTS, based on the 'No U-Turn Sampler' (NUTS) introduced in [**nuts**].

1 Introduction

Hamiltonian Monte Carlo (HMC) provides us with a useful way to sample from a posterior distributions. HMC along with other MCMC sampling methods use all the available data at each step, and so we require 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 Stochastic Gradient Hamiltonian Monte Carlo (SGHMC), first introduced by the paper in question [**sghmc**]. SGHMC uses randomly sampled mini-batches of data to produce noisy estimates of the gradient $\nabla \widetilde{U}(\theta)$. 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 [**sghmc**] that we decided to reproduce:

- 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 $U(\theta) = \frac{1}{2}\theta^2$ ith perfect gradients, and noisy gradients using $\mathcal{N}(0,4)$ as a proxy for the noisy in $\widetilde{U}(\theta)$.
- Comparing the autocorrelation times of SGHMC and SGLD.
- Classifying the MNIST dataset [**mnist**] 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 [**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.

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

2 Background

2.1 HMC

Hamiltonian Monte Carlo ([**duane-hmc**; **neal-hmc**]) 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 $p(\theta \mid \mathcal{D}) \propto \exp(-U)$. 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**
  $r \sim \mathcal{N}(0, 1)$ ▷ 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 $\mathcal{D}$ may be large, and so running Algorithm 1 may be computationally expensive, as the complexity of calculating ∇U scales with $|\mathcal{D}|$. 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 $\widetilde{\mathcal{D}} \subset \mathcal{D}$ uniformly at random. We estimate the gradient $\nabla U(\theta)$ using this minibatch as follows:

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

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 [**neal-hmc**], 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 [**sghmc**] 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.

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

The algorithm is given in Algorithm 2. In the case where $\widehat{B} = B$, these dynamics accurately traverse from the posterior distribution. In practice, we must rely on an inaccurate estimate $\widehat{B}$. 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 [**sgld-fisher**]. 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

3 Implementation Details

We implemented the following algorithms from scratch: HMC, SGHMC, SGLD (stochastic gradient Langevin dynamics [**sgld**]), SGD (stochastic gradient descent), SGD with Nesterov momentum and SGNUTS — a novel extension to SGHMC that uses ideas from the popular No U-Turn Sampler [**nuts**]. 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. 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 parameters of the model.

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 parameters $(\theta, \mathscr{D})$ 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 [**bishop2006pattern**]. 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 [**sghmc**]. 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 [**sghmc**]. We reproduced these three experiments by converting the MATLAB codes provided by the authors **sghmc** [**simu_code**] 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 consisted of checking whether the samples produced by SGHMC and the other algorithms actually follow the target distribution.

Experiment 1

As in [**sghmc**] 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. To check that the theoretical convergence of Naïve SGHMC and SGHMC are true or not we introduce noise into the gradient, and so we set:

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

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 $U(\theta) = \theta^2$. The results are shown in Figures 1 and 2. It can be seen that both HMC algorithms

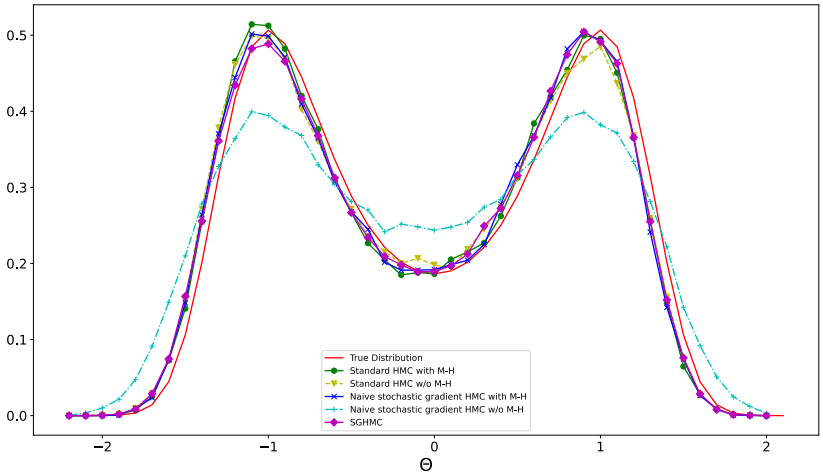


Figure 1: Distributions of different sampling algorithms for the target function $U(\theta) = -2\theta^2 + \theta^4$

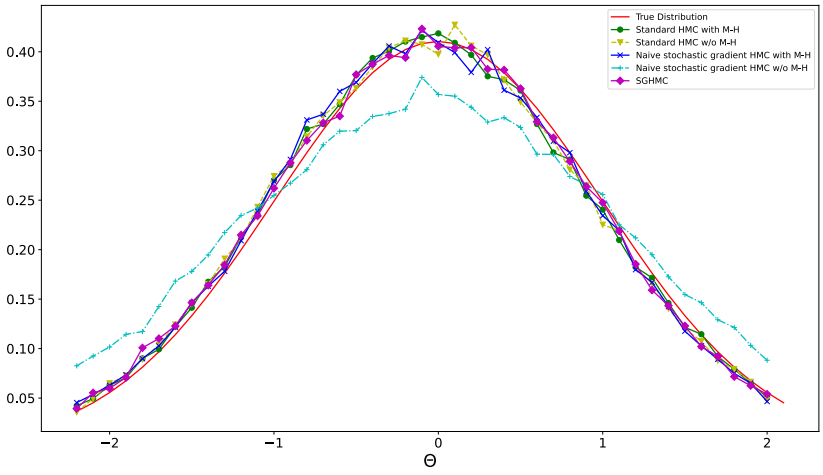


Figure 2: Distributions of different sampling algorithms for the target function $U(\theta) = \frac{1}{2}\theta^2$

perform well, as does Naïve SGHMC with MH corrections. However, as we already discussed these two 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 [**sghmc**]. 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.

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 noisy gradient as follows: $\nabla\widetilde{U}(\theta) = \nabla U(\theta) + \mathcal{N}(0, 4)$. Except in the case specified, we did not 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 [**sghmc**]. Note that in [**sghmc**] the **sghmc** 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)

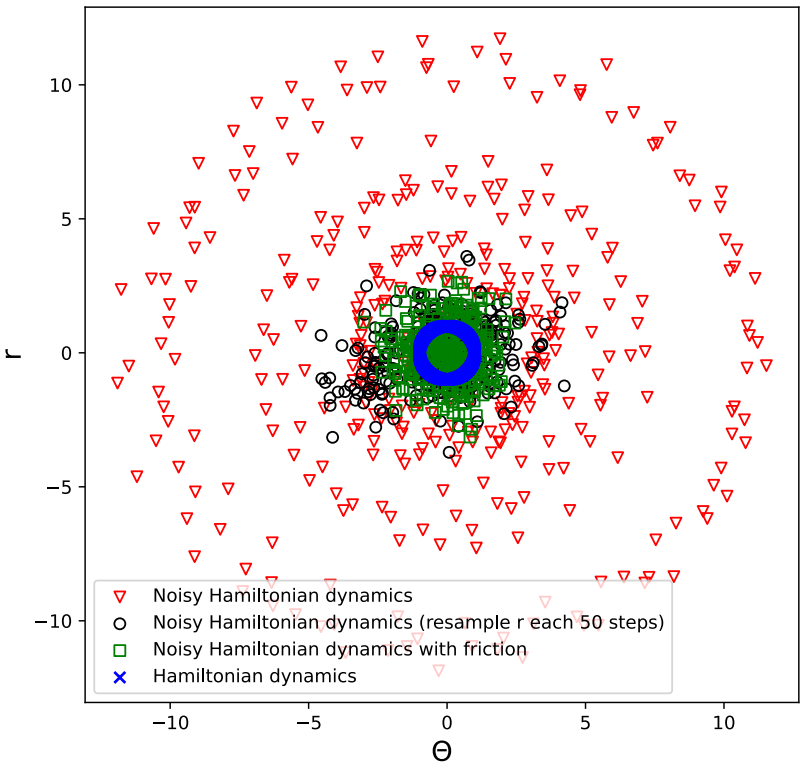


Figure 3: Plotting the trajectory of (θ, r) under various samplers for 360 steps

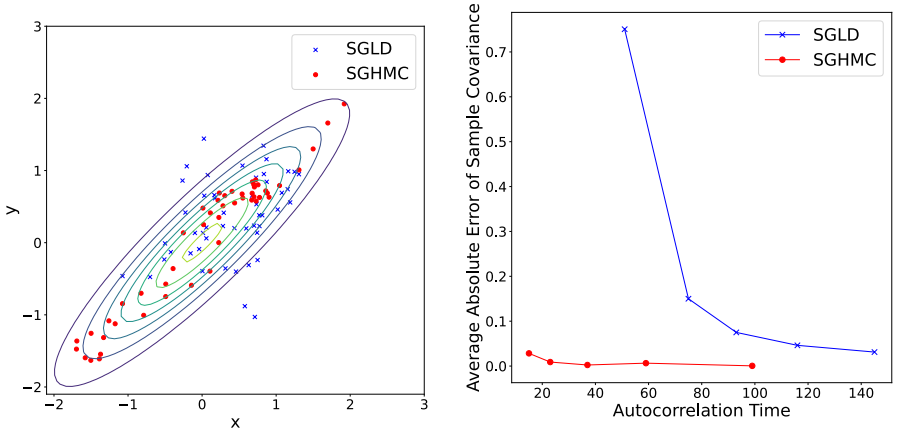
keeps Hamiltonian dynamics much closer to the true Hamiltonian dynamics (blue) in a noisy system. Resampling the momentum every 50 steps also seems to mitigate the problem which is probably why the authors **sghmc** include it in their pseudocode description of SGHMC. Figure 3 also supports the results of Theorem 3.1 in [**sghmc**], 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 [**sghmc**]. **sghmc** 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 [**sghmc**], where the authors **sghmc** 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 50 samples produced using both algorithms. The second shows the average absolute error of the sample covariance against autocorrelation time (for more details see [**sghmc**].



(a) First 50 samples of SGHMC and SGLD

(b) Autocorrelation time versus estimation error for the five setting

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

We see that SGHMC samples quickly becomec 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 a 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 [**sghmc**]. 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 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 [**sghmc**].

To build on top of the work in [**sghmc**] we implemented learning rate annealing for SGLD, and following [**sgld**] 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 [**sghmc**], 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 [**hands-on-bnn**] 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 those in [**sghmc**] 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

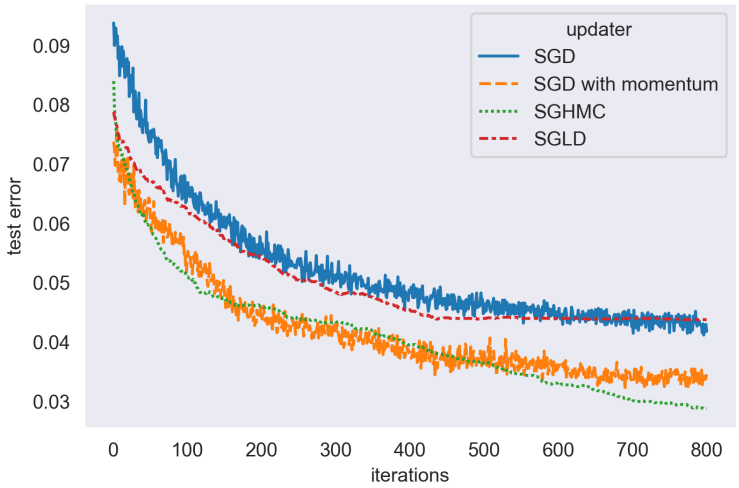
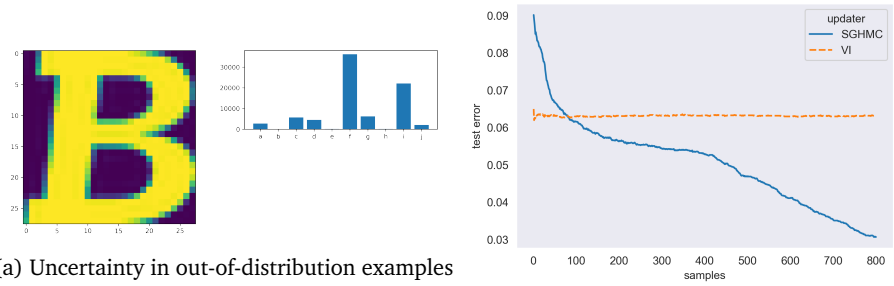


Figure 5: Reproducing the MNIST classification experiment from [**sghmc**]; SGHMC ($\eta = 2.0 \times 10^{-6}, \alpha = 0.01, \texttt{resample\_n} = 0$), SGLD ($\eta = 4.0 \times 10^{-5}$), SGD ($\eta = 1.0 \times 10^{-5}$), SGD with momentum ($\eta = 1.0 \times 10^{-6}, \alpha = 0.01$)



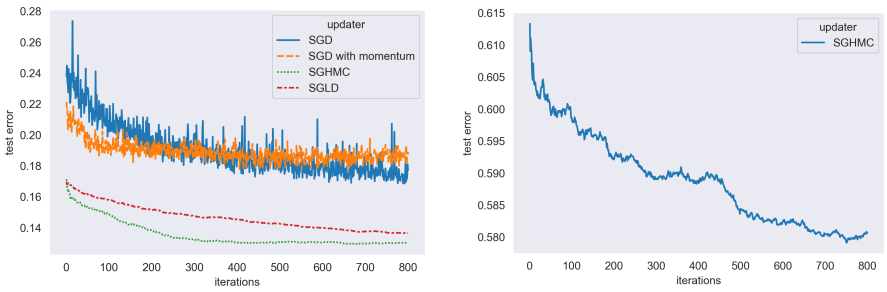
(a) Uncertainty in out-of-distribution examples

(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, $\alpha = 0.01$, $\texttt{num\_particles} = 2$) and SGHMC ($\eta = 2.0 \times 10^{-6}, \alpha = 0.01, \texttt{resample\_n} = 0$) 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 $p(\theta \mid \mathscr{D})$.

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) 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` $= 150$.