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

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

Abstract

We reproduce the experiments contained in **sghmc** [**sghmc**] by **sghmc**.

FiXme: Give more details in abstract

List of Corrections

1 Introduction

- Overview of paper and its context.
- Which experiments replicated, and rationale for this choice.
- Target questions of paper.
- Experimental methodology.
- Implementation details.
  - Integration with Pyro.
  - Which parts are new, and which are from publicly available code?
  - Details about how key aspects were implemented.
- Link to repository.
- New aspects?

2 Background

Hamiltonian Monte Carlo (HMC) ([**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.

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

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

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. 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. 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. Unfortunately, such a dynamical system can diverge quite rapidly from the true posterior distribution [**neal-hmc**], which necessitates frequent Metropolis-Hastings steps. Such steps must be carried out using the whole dataset.

The method 'Stochastic Gradient Hamiltonian Monte Carlo' 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.

3 Implementation Details

We implemented the following algorithms from scratch: HMC, SGHMC, SGLD, SGD, SGD (with nesterov momentum) and SGNUTS - a novel extension to

SGHMC that uses ideas from the popular No U-Turn Sampler. All of our implementations subclass Pyro's `MCMCKernel` and are designed to be used directly with 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 $p(\theta \mid D)$ that we want to sample from; θ corresponds to the sampled parameters or latent parameters of the model and D corresponds to the observed parameters of the model.

Pyro already comes with the following MCMC samplers: HMC, MH and NUTS. So while the algorihtms 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 (θ, D) to U, which computes the negative log joint $-\log p(\theta, D)$. Bayes' rule tells us that $p(\theta \mid D) \propto p(\theta, D)$ which is typically all we need for MCMC samplers and even simpler ones such as Importance and Rejection samplers. Pyro also comes with the method `potential_grad`, which given (θ, 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 D is constant and so once a Pyro PP has been transformed into a potential function $U(\theta) = -\log p(\theta, D)$ 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 D by sampling minibatches $\tilde{D}$, where $\tilde{D} \subset D$. In both SGHMC and SGLD we require that the potential function has the form,

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

Unfortunately when we subsample the dataset and call `initialize_model` Pyro doesn't explicitly supply us with the likelihood term $p(\tilde{D} \mid \theta)$ - it only gives us the negative log joint $-\log p(\theta, \tilde{D})$, so we had to modify Pyro's source code to get the desired behaviour above. 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 $\tilde{U}(\theta)$ for some given minibatch $\tilde{D}$, although this is a small price to pay for much quicker gradient computations.

4 Experiments

- Describe experiments and compare with results in the paper.

4.1 Simulated examples

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,

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

Where $\eta = \epsilon^2 M^{-1}, \alpha = \epsilon M^{-1} C, \hat{\beta} = \eta M^{-1} \hat{B}$. In all our experiments in this section we set the mass matrix M to the identity, and the noise model $\hat{\beta} = \hat{B} = 0$. Other than the architecture of the BNN the only 3 hyperparameters for SGHMC are the learning rate η, the momentum decay α and the batch size $|\tilde{D}|$, in all our experiments we fixed $|\tilde{D}| = 500$ which follows what the author's of [**sghmc**] did.

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,

$$\mathbb{E}[f(\theta)] \approx \frac{\sum_{t=1}^{T} \epsilon_t f(\theta_t)}{\sum_{t=1}^{T} \epsilon_t}$$

Although since f is a classifier we can ignore the demonimator. Our BNN followed the same architecture as in [**sghmc**], that is one linear 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. The likelihood function is of course a Categorical distribution.

Our implementations of SGD and SGD with momentum are meant to be used directly with Pyro BNN, and so Guassian 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 $\eta = 2.0^{-6}, \alpha = 0.01$, and for SGD with momentum the best configuration was $\eta = 1.0^{-6}, \alpha = 0.01$.

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 configuartion for SGLD was $\eta = 4.0^{-5}$, 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 we do Bayesian averaging over all the sampled parameterisations of the BNN after warmup as is described in Section II of [**hands-on-bnn**] and report the test error. For the optimization algorithms we take most recent sample / set of parameters as a point estimate and report the test error. Figure 1 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 2 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 3 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_π as closely as possible to the true posterior it takes only hundreds of sample to characterise q_π, 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 4 presents the results of running the same BNN architecture on fashionMNIST, and Figure 5 demonstrates that our implementation of SGHMC can be used with convolutional neural networks (CNNs).

5 Conclusion

- Analysis and discussion of findings.
- Suggest what could have been done with more time.