Semi-separable Hamiltonian Monte Carlo for inference in Bayesian neural networks

Adam D. Cobb
Dept. of Engineering Science
University of Oxford
acobb@robots.ox.ac.uk

Atılım Güneş Baydin
Dept. of Engineering Science
University of Oxford

Ivan Kiskin
Dept. of Engineering Science
University of Oxford

Andrew Markham
Dept. of Computer Science
University of Oxford

Stephen J. Roberts
Dept. of Engineering Science
University of Oxford

Abstract

We introduce a new method for performing inference in Bayesian neural networks (BNNs) using Hamiltonian Monte Carlo (HMC). We show how the previously introduced semi-separable HMC sampling scheme can be adapted to BNNs, which allows us to integrate over both the parameters and hyperparameters. We derive a suitable Riemannian metric for the BNN hyperparameters and show that it is positive definite. Our work is compared to both Monte Carlo dropout and a deterministic neural network, where our inference technique displays better calibrated uncertainties with comparable performance to current baselines. Our code is provided in a new open-source Python package, `hamiltorch`, which enables our method to scale to CNNs with over $400{,}000$ parameters and take advantage of GPUs.

1 Introduction

Current approaches to performing inference in BNNs either rely on stochastic variational inference or Markov chain Monte Carlo techniques. Variational approaches [1–4] that rely on gradient based optimisation are preferred over MCMC due to their scalability to large networks and superior performance in traditional metrics such as accuracy. However, variational inference is well-known to be mode-seeking and therefore struggles to provide reliable uncertainty estimates [5]. In this work, we introduce a new MCMC technique for performing inference in BNNs, which not only is competitive in terms of accuracy, but also significantly outperforms other approaches in estimating the uncertainty.

In particular, we introduce a new way of performing Riemannian manifold Hamiltonian Monte Carlo (RMHMC) [6] in BNNs that avoids the infeasible requirement of computing the Hessian. We build on the previous work of Zhang and Sutton [7], but adapt their work to BNNs and introduce a new inference scheme, which we call *semi-semi-separable HMC* (S3HMC), which unlike before, no longer requires all the blocks to be separable.

2 Semi-separable Hamiltonian Monte Carlo for inference in BNNs

We can start by explicitly writing the Hamiltonian from RMHMC for a BNN,

$$H(\boldsymbol{\omega}, \boldsymbol{\tau}, \mathbf{p}) = -\log\left(f(\boldsymbol{\omega}, \boldsymbol{\tau})\right) + \frac{1}{2}\log\left((2\pi)^D |\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})|\right) + \frac{1}{2}\mathbf{p}^\top \mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})^{-1}\mathbf{p}, \tag{1}$$

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

where $f(\boldsymbol{\omega}, \boldsymbol{\tau}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\omega})\, p(\boldsymbol{\omega}|\boldsymbol{\tau})p(\boldsymbol{\tau})$ is our Bayesian hierarchical model and $\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})$ is the Fisher-information matrix. The Gaussian prior over the weights, $p(\boldsymbol{\omega}|\boldsymbol{\tau})$, is parametrised by the precision hyperparameter, $\boldsymbol{\tau}$, which in itself follows a Gamma distribution, $p(\tau|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\tau^{\alpha-1}\exp(-\beta\tau)$, with the shape and rate parameters α and β as well as the gamma function $\Gamma(\cdot)$. The precision parameter controls how much the weights in the neural network are allowed to vary. We can tie each group of weights in a given layer to their own precision, such that each layer has a precision associated with the weights, $\tau_{\mathbf{w}}^{(l)}$, and a precision associated with the biases, $\tau_{\mathbf{b}}^{(l)}$. The challenge of performing RMHMC over a BNN comes from calculating the hessian in building $\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})$. For example, we are going to be performing RMHMC over a Bayesian CNN with $431{,}080$ parameters and if each element were a single-precision floating point number of 4 bytes (32 bits), then the Hessian alone would require almost a terabyte of memory.

Therefore in order to perform RMHMC over a BNN of this size, we are going to follow the formulation of semi-separable HMC (SSHMC) that was introduced in Zhang and Sutton [7], but make it more suited to BNN inference. The key insight from SSHMC is to split $\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau})$ into blocks as follows,

$$\mathbf{G}(\boldsymbol{\omega}, \boldsymbol{\tau}) = \begin{bmatrix} \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau}) & 0 \\ 0 & \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}) \end{bmatrix}, \tag{2}$$

where each block is independent from the other set of parameters. In this case $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ would be independent of $\boldsymbol{\omega}$ and $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ would be independent of $\boldsymbol{\tau}$. If we keep this block structure for Equation (1), we can split the Hamiltonian as in [7]:

$$H_{\boldsymbol{\omega}}(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}}) = -\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) - \log p(\boldsymbol{\omega}|\boldsymbol{\tau}) + A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}) + \frac{1}{2}\log|\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})| + A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}), \tag{3}$$

with $\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}$ fixed, and

$$H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}) = -\log p(\boldsymbol{\omega}|\boldsymbol{\tau}) - \log p(\boldsymbol{\tau}) + A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}) + \frac{1}{2}\log|\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})| + A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}), \tag{4}$$

with $\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}}$ fixed, where the auxiliary potential terms are defined as

$$A(\mathbf{p}_{\boldsymbol{\omega}}|\boldsymbol{\tau}) = \frac{1}{2}\mathbf{p}_{\boldsymbol{\omega}}^\top \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})\mathbf{p}_{\boldsymbol{\omega}}, \quad A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega}) = \frac{1}{2}\mathbf{p}_{\boldsymbol{\tau}}^\top \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})\mathbf{p}_{\boldsymbol{\tau}}. \tag{5}$$

The auxiliary terms appear in both Hamiltonians, however they play opposite roles in each instance. For example $A(\mathbf{p}_{\boldsymbol{\tau}}|\boldsymbol{\omega})$ is the kinetic energy component for $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ but is the auxiliary utility for $H_{\boldsymbol{\omega}}(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}})$. It is this coupling, along with the log-determinant terms in Equations (3) and (4), that enable energy to be *shared* between the two systems. Without these terms, it would be the same as performing RMHMC with Gibbs sampling. The advantage of jointly integrating over the hyperparameters and parameters is that the trajectory in the hyperparameter space is informed by the current location in the augmented parameter space $(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}})$ and vice-versa.

2.1 Choosing the metric tensor and introducing RMHMC

The remaining task is to select $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ and $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ for BNNs. Due to the possible size of the network, $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ is only feasible to store and compute if it is kept diagonal. Therefore we define $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})$ to have diagonal elements $\mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau})_{ii} = \tau_{\boldsymbol{\omega}_i}$, where $\tau_{\boldsymbol{\omega}_i}$ is the corresponding precision for NN parameter $\boldsymbol{\omega}_i$. This diagonal mass matrix is separable and cheap to compute and can therefore be used efficiently with the leapfrog integrator for separable Hamiltonians.

However, for the $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ it remains less obvious how to define an appropriate Riemannian metric. Therefore rather than building a separable Hamiltonian for $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$, we allow $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega})$ to be dependent on $\boldsymbol{\tau}$, whilst still keeping $\boldsymbol{\omega}$ fixed. We can then derive the following matrix that only contains the diagonal elements

$$\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})_{ii} = \frac{1}{\tau^2_{\boldsymbol{\omega}^{(l)}}}\left(\frac{Q_l}{2} + \alpha - 1\right), \tag{6}$$

where $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})_{ii} > 0$ if $\frac{Q_l}{2} + \alpha \geq 1$. This is always the case[1] if $\alpha > \frac{1}{2}$ as $Q_l \geq 1$. Furthermore $\tau_{\boldsymbol{\omega}^{(l)}} > 0$ by definition (See Appendix B for full derivation). Therefore if these conditions hold, the

[1] For regression, if we include the output precision, λ, as part of $\boldsymbol{\tau}$, then the condition is actually $\alpha > 1$ as $-\frac{\partial^2}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}}\log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = 0$ in this case.

matrix is positive definite, which is an important result as this matrix can be used as the metric tensor in RMHMC. We note that the independence assumption for the prior over the precisions leads to the matrix being diagonal as there are no second order cross-terms.

The advantage of employing RMHMC over the non-separable version of $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})$ does not come at the same computational cost as calculating the Hessian for $\boldsymbol{\omega}$ as there are fewer hyperparameters, e.g., two per layer. We will refer to this new method as *semi-semi-separable* HMC or S3HMC as we are using semi-separable HMC, where one of the blocks is no longer separable.

2.2 Classification example

We now move on to a classification example and show how S3HMC scales to a larger classification task. The network is the Pytorch MNIST CNN example [8] based on the LeNet architecture [9]. This network consists of two convolutional layers and two fully connected layers, which contains $431{,}080$ weights (including biases). The performance of S3HMC is compared to baselines. A deterministic NN, trained via stochastic gradient descent, and MC dropout, with $p = 0.2$. For each inference approach, the networks are trained on the first $10{,}000$ digits of the the MNIST training set. They are validated on the next $1{,}000$ and then tested on the standard $10{,}000$ test digits. S3HMC achieves an accuracy of 98.15%, stochastic gradient descent achieves 98.20% and MC dropout achieves 98.68%. Therefore all models score comparable accuracies, with MC dropout slightly outperforming in accuracy as expected.

Using a similar experimental set-up as in [4], we compare the mean predictive entropy of all models over both the MNIST test data and the notMNIST [10] data set. In Figure 1, the top plot shows the distribution of predictive entropies for all models over the test data. All models achieve good performance over the test set and this is represented in the low entropy predictions appearing on the left-hand side of the plot. However, the interesting result is shown in the bottom plot for notMNIST, where we expect our models to be uncertain due to making predictions over data that do not feature in the training data. The deterministic NN's performance is poor with the vast majority of predictions over notMNIST having a low predictive entropy (in blue), which shows overconfidence. Both Bayesian neural networks appear suitably less confident over the notMNIST data. However, S3HMC (in green) significantly outperforms the variational approach of MC dropout (in orange). This is shown by the predictive entropies of S3HMC lying much further to the right than MC dropout.

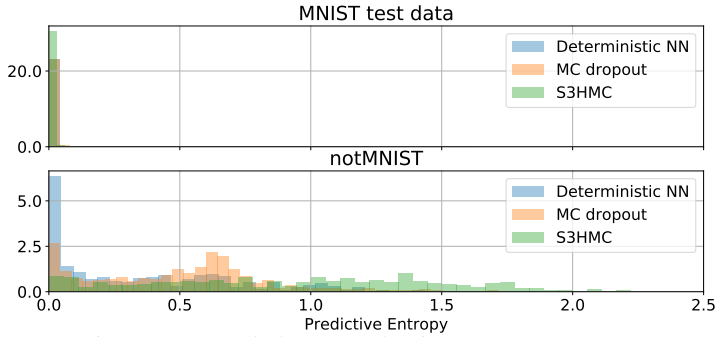


Figure 1: A histogram of the mean predictive entropies for both MNIST test data (top) and notMNIST data (bottom). All models get similar accuracy performance over the MNIST test data, as well as making low entropy (highly confident) predictions. For notMNIST, S3HMC is rightly much less confident in its predictions by giving the vast majority of its predictions a higher entropy than the baselines, with the deterministic NN displaying undesirable overconfidence.

3 Conclusion

Overall S3HMC achieves similar performance in the traditional metric of accuracy, whilst providing superior calibrated uncertainty when making predictions that lie far outside the training distribution. Furthermore, this experiment has shown that MCMC techniques can be a viable alternative to variational approaches for large networks. In Appendix C, we show further results for regression where we are able to integrate over the output noise.

References

[1] David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4 (3):448–472, 1992.

[2] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[3] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[4] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2218–2227. JMLR. org, 2017.

[5] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[6] Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[7] Yichuan Zhang and Charles Sutton. Semi-separable Hamiltonian Monte Carlo for inference in Bayesian hierarchical models. In *Advances in Neural Information Processing Systems*, pages 10–18, 2014.

[8] PyTorch MNIST example. `https://github.com/pytorch/examples/blob/master/mnist/main.py`. Accessed: 8th September 2019.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Yaroslav Bulatov. notMNIST data set. `http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html`. Accessed: 8th September 2019.

[11] Michael Betancourt. The fundamental incompatibility of scalable Hamiltonian Monte Carlo and naive data subsampling. In *International Conference on Machine Learning*, pages 533–540, 2015.

[12] Wolfgang Fruehwirt, Adam D Cobb, Martin Mairhofer, Leonard Weydemann, Heinrich Garn, Reinhold Schmidt, Thomas Benke, Peter Dal-Bianco, Gerhard Ransmayr, Markus Waser, et al. Bayesian deep neural networks for low-cost neurophysiological markers of Alzheimer's disease severity. *arXiv preprint arXiv:1812.04994*, 2018.

Semi-separable Hamiltonian Monte Carlo for inference in Bayesian neural networks
(Supplementary Material)

A Semi-separable HMC implementation

In Algorithm 1, we include the *alternating block-wise leapfrog* algorithm (ALBA), which was first introduced in Zhang and Sutton [7] for performing semi-separable HMC. When looking at ALBA, the advantage of splitting the Hamiltonian into separable blocks is made clear. We can now rely on the separable HMC sampler, which is both more scalable in the parameter space, and simple to implement. The structure of the semi-separable Hamiltonian means that the original Hamiltonian is preserved if $H_{\boldsymbol{\omega}}(\boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}})$ and $H_{\boldsymbol{\tau}}(\boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}})$ are simulated exactly. Finally, one further observation is that each leapfrog integrator within the ALBA can take multiple leapfrog steps.

Algorithm 1 Semi-separable HMC by ALBA

Inputs: $\boldsymbol{\omega}, \boldsymbol{\tau}$
Sample: $\mathbf{p}_{\boldsymbol{\omega}} \sim \mathcal{N}(\mathbf{0}, \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\tau}))$, $\mathbf{p}_{\boldsymbol{\tau}} \sim \mathcal{N}(\mathbf{0}, \mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}))$
$H = H_{\boldsymbol{\tau}} + H_{\boldsymbol{\omega}}$
 # ALBA
for l in $1, 2, \dots L$ **do**
  # Note that the integrators below can take multiple leapfrog steps.
  $\boldsymbol{\omega}^{l+\epsilon/2}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon/2} \leftarrow \text{leapfrog}(\boldsymbol{\omega}^{l}, \mathbf{p}_{\boldsymbol{\omega}}^{l}, \epsilon/2 \mid \boldsymbol{\tau}^{l}, \mathbf{p}_{\boldsymbol{\tau}}^{l})$
  $\boldsymbol{\tau}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\tau}}^{l+\epsilon} \leftarrow \text{leapfrog}(\boldsymbol{\tau}^{l}, \mathbf{p}_{\boldsymbol{\tau}}^{l}, \epsilon \mid \boldsymbol{\omega}^{l+\epsilon/2}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon/2})$
  $\boldsymbol{\omega}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon} \leftarrow \text{leapfrog}(\boldsymbol{\omega}^{l+\epsilon/2}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon/2}, \epsilon/2 \mid \boldsymbol{\tau}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\tau}}^{l+\epsilon})$
end for
 # MH correction
$H^{L\epsilon} = H_{\boldsymbol{\tau}}^{L\epsilon} + H_{\boldsymbol{\omega}}^{L\epsilon}$
$u \sim \mathcal{U}(0, 1)$
if $\log u > \min(0, H - H^{L\epsilon})$ **then**
  $\boldsymbol{\omega}^*, \mathbf{p}_{\boldsymbol{\omega}}^*, \boldsymbol{\tau}^*, \mathbf{p}_{\boldsymbol{\tau}}^* \leftarrow \boldsymbol{\omega}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\omega}}^{l+\epsilon}, \boldsymbol{\tau}^{l+\epsilon}, \mathbf{p}_{\boldsymbol{\tau}}^{l+\epsilon}$
else
  $\boldsymbol{\omega}^*, \mathbf{p}_{\boldsymbol{\omega}}^*, \boldsymbol{\tau}^*, \mathbf{p}_{\boldsymbol{\tau}}^* \leftarrow \boldsymbol{\omega}, \mathbf{p}_{\boldsymbol{\omega}}, \boldsymbol{\tau}, \mathbf{p}_{\boldsymbol{\tau}}$
end if
return $\boldsymbol{\omega}^*, \boldsymbol{\tau}^*$

B Derivation of precision metric matrix

We can define the Riemannian metric according to $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau}) = -\nabla^2_{\boldsymbol{\tau}} \log p(\boldsymbol{\omega}, \boldsymbol{\tau})$, which is the the negative Hessian of the joint log likelihood. We can write each negative log probability term in Equation (4) out explicitly as

$$1) \quad -\log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = \frac{1}{2}\left(-\sum_i \log \tau_{\boldsymbol{\omega}_i} + \sum_i \boldsymbol{\omega}_i^2 \tau_{\boldsymbol{\omega}_i} + D_{\boldsymbol{\omega}} \log 2\pi\right),$$

$$2) \quad -\log p(\boldsymbol{\tau}) = -\alpha \log \beta - (\alpha - 1)\log \boldsymbol{\tau} + \beta\boldsymbol{\tau} + \log \Gamma(\alpha),$$

This is where $\tau_{\boldsymbol{\omega}_i}$ corresponds to the precision for each network parameter $\boldsymbol{\omega}_i$. We can now form a non-separable $\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})$ by looking at $\frac{\partial^2 \log p(\boldsymbol{\omega}, \boldsymbol{\tau})}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}}$. Here, we refer to both $\tau_{\mathbf{w}}^{(l)}$ and $\tau_{\mathbf{b}}^{(l)}$ as $\tau_{\boldsymbol{\omega}^{(l)}}$ as they can be used interchangeably. Furthermore, the number of parameters tied to each $\tau_{\boldsymbol{\omega}^{(l)}}$ is defined as Q_l. Therefore

$$1) \quad -\frac{\partial^2}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}} \log p(\boldsymbol{\omega}|\boldsymbol{\tau}) = \frac{Q_l}{2\tau^2_{\boldsymbol{\omega}^{(l)}}},$$

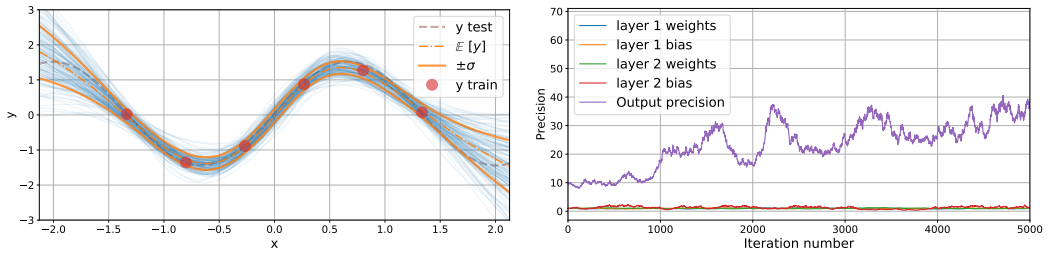
$$2) \quad -\frac{\partial^2}{\partial^2 \tau_{\boldsymbol{\omega}^{(l)}}} \log p(\boldsymbol{\tau}) = \frac{\alpha - 1}{\tau^2_{\boldsymbol{\omega}^{(l)}}},$$

Therefore we recover a diagonal matrix with elements

$$\mathbf{G}_{\boldsymbol{\tau}}(\boldsymbol{\omega}, \boldsymbol{\tau})_{ii} = \frac{1}{\tau^2_{\boldsymbol{\omega}^{(l)}}}\left(\frac{Q_l}{2} + \alpha - 1\right). \tag{7}$$

C Regression example: further results

We now look at a simple one-dimensional regression example of fitting a BNN to $y = \sin(x) + \mathcal{N}(0, 0.1^2)$. In this example, S3HMC is applied to a network with an architecture of $[1, 50, 1]$ with $\tanh$ non-linearities. We jointly integrate over all network parameters and hyperparameters, including the output precision. The key insight from this experiment in Figure 2 is that the value of α is especially significant when performing S3HMC over the output precision. Increasing the value of alpha allows the samples of the output precision to increase to a region with a higher joint log probability.



(a) S3HMC with sampling over output precision.

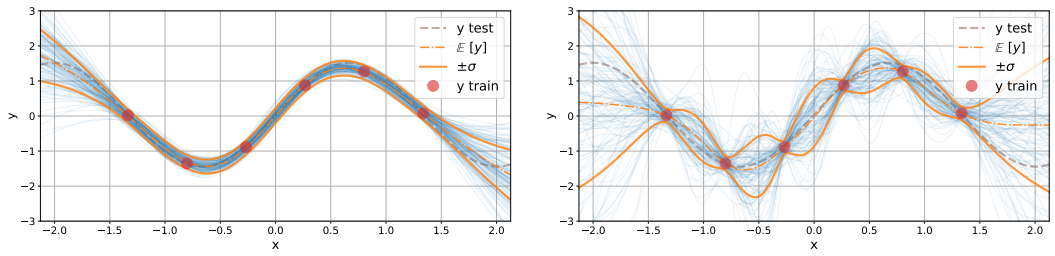
(b) Samples of $\boldsymbol{\tau}$ including output precision.

Figure 2: S3HMC regression that display performance when sampling over the parameter precisions and the output precision. The output precision rises from its initial condition of 10 and increases. These samples lead to a good performance over the training data with an expected marginal likelihood of $p(\mathbf{Y}|\mathbf{X}) = -36.25$.

In addition to integrating over the output precision, we also perform an experiment where we keep the output precision fixed at the equivalent value of the output noise ($\tau_{\text{out}}^{-1} = 0.1^2$). We compare two networks: one network has an architecture of $[1, 50, 1]$ and the other has an architecture of $[1, 50, 50, 1]$.

The smaller network demonstrates the expected behaviour of restricting the flexibility of the predictive distribution. In comparison, the predictive samples from the larger network of 2701 parameters have a much higher frequency component although still pass through the training points. Both of these models are shown in Figure 3. The estimated marginal likelihood taken over the samples for each model suggests that since the smaller model explains the data just as well as the larger model, then following along the lines of Occam's razor, we should pick the smaller network for this task ($p(\mathbf{Y}|\mathbf{X}) = -37.59$ versus $p(\mathbf{Y}|\mathbf{X}) = -640.52$).

One of the advantages of S3HMC is that it allows us to jointly integrate over the hyperparameters. Therefore we show the corresponding hyperparameter samples for the precisions in Figure 4. The majority of the precisions hover around the initial condition of setting the precisions to the value of 1.0, whereas the bias on the last layer has the most variance.



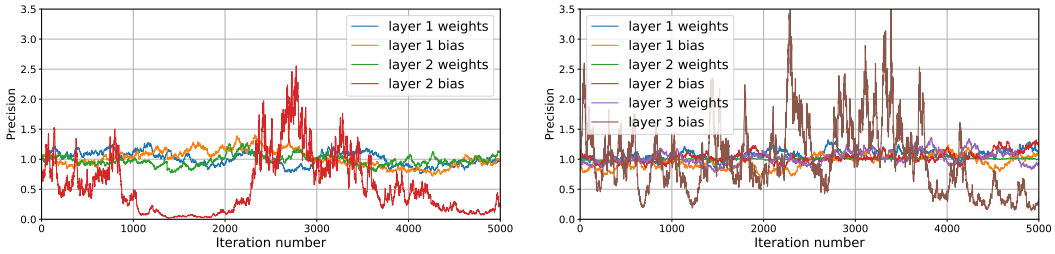
(a) $[1, 50, 1]$

(b) $[1, 50, 50, 1]$

Figure 3: Posterior draws from BNNs inferred via S3HMC. The comparison between Figures 3a and 3b shows how increasing the size of the network results in posterior draws that are more flexible.

D Limitation or advantage: scaling to large data with HMC

In the experiments above, the limiting factor on using HMC for BNNs is the size of the data set used for training. As noted by Betancourt [11], a stochastic form of HMC can be detrimental to the ability of the sampler to explore the full log probability space. However, there are many examples in the real-world where the size of the data set is in fact small and we may still want to leverage techniques such as CNNs. Examples of such instances



(a) $[1, 50, 1]$ (b) $[1, 50, 50, 1]$

Figure 4: Corresponding values of $\boldsymbol{\tau}$ drawn for Figures 3a and 3b. The bias on the last layer tends to vary the most.

can be found in medical applications, where vast amounts of data are infeasible to collect [12]. In these cases, HMC may provide better performance in terms of avoiding overfitting to small data sets and providing more reliable uncertainty estimates.