DISCUSSION ON "EFFICIENT BAYESIAN INFERENCE WITH LATENT HAMILTONIAN NEURAL NETWORKS IN NO-U-TURN SAMPLING" BY DHULIPALA ET AL.

Shuo Hao
The Hong Kong Polytechnic University
cee-shuo.hao@connect.polyu.hk

February 28, 2025

ABSTRACT

This report presents a critical analysis of the study "Efficient Bayesian Inference with Latent Hamiltonian Neural Networks in No-U-Turn Sampling" by Dhulipala et al. [1], accompanied by the replication of their synthetic experiments utilizing the TensorFlow Probability framework. Prepared in fulfillment of the requirements for the AI & Data Science Associate Internship Program interview, the answers to the given questions are also covered in this report.

1 Overview

Hamiltonian Monte Carlo (HMC) is widely used for Bayesian inference in relatively high-dimensional parameter spaces. It leverages Hamiltonian dynamics, where the negative log-posterior serves as the potential energy function, and an auxiliary kinetic energy term is introduced to simulate parameter trajectories. Prolonged simulations guide these trajectories toward high-probability regions in the parameter space. By initializing each simulation with the terminal state of the prior, HMC constructs a Markov chain that efficiently explores the posterior distribution. In comparison with simpler Markov chain Monte Carlo algorithms, HMC enables faster convergence and improved sampling efficiency in complex probability distribution models.

The central challenge addressed in this paper arises in the numerical simulation of Hamiltonian dynamics, where symplectic integration requires iterative computation of the partial derivatives of the Hamiltonian H at each timestep. Specifically, the gradient of H with respect to the position vector $\boldsymbol{q}$, denoted as $\nabla_{\boldsymbol{q}} H$, must be evaluated repeatedly. By definition, the Hamiltonian decomposes as $H(\boldsymbol{q}, \boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p})$, where $U(\boldsymbol{q})$ is the potential energy and $K(\boldsymbol{p})$ is the kinetic energy, with $\boldsymbol{p}$ representing the momentum vector. Consequently, $\nabla_{\boldsymbol{q}} H = \nabla_{\boldsymbol{q}} U$, and the computational burden lies in evaluating $\nabla_{\boldsymbol{q}} U$. In Bayesian posterior computation, $U(\boldsymbol{q})$ corresponds to the negative logarithm of the likelihood function times the prior, which often incorporates a computationally intensive likelihood function derived from a complex hierarchical or extremely high-dimensional model, rendering the symplectic integration prohibitively expensive for many practical applications.

To circumvent the need for frequent direct calculations of $\nabla_{\boldsymbol{q}} H$, Dhulipala et al. introduced latent Hamiltonian neural networks (L-HNNs) as a surrogate model for the Hamiltonian system. An L-HNN is essentially a fully connected neural network, denoted as $H_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta}$ represents the trainable parameters. The network takes the same inputs as H, namely $\{\boldsymbol{q}, \boldsymbol{p}\}$, and outputs d latent variables λ_i, collectively denoted as $\boldsymbol{\lambda}$. The Hamiltonian is then computed via the summation of all latent variables, i.e., $H_{\boldsymbol{\theta}} = \sum_{i=1}^{d} \lambda_i$. L-HNNs are trained on data from limited simulations of Hamiltonian dynamics, where inputs $\{\boldsymbol{q}, \boldsymbol{p}\}$, corresponding time derivatives $\{\nabla_t \boldsymbol{q}, \nabla_t \boldsymbol{p}\}$, and Hamiltonian H are

recorded. The training objective is to minimize the loss function $L(\boldsymbol{\theta})$:

$$L(\boldsymbol{\theta}) = \|\nabla_{\boldsymbol{p}} H_{\boldsymbol{\theta}} - \nabla_t \boldsymbol{q}\| + \|-\nabla_{\boldsymbol{q}} H_{\boldsymbol{\theta}} - \nabla_t \boldsymbol{p}\|, \tag{1}$$

where $\|\cdot\|$ represents a specified error norm. A well-trained L-HNN, $H(\boldsymbol{\theta}^\star)$, can be evaluated by simulating Hamiltonian dynamics and comparing the resulting trajectories of $\boldsymbol{q}$ to those obtained using H under the same initial conditions. If the L-HNN is validated as a surrogate model, it can be used in HMC to approximate $\nabla_{\boldsymbol{q}} H$, which is necessary for symplectic integration, thereby improving the efficiency of the sampling process.

Another critical feature of the paper is the incorporation of efficient No-U-Turn Sampling (NUTS) to enhance the efficiency of HMC. The validated L-HNNs are seamlessly integrated into the symplectic integrator to construct the binary tree. Originally proposed by Hoffman and Gelman in 2014 [2], NUTS eliminates the need for prior specification of total simulation steps for Hamiltonian dynamics, thereby automatically avoiding unnecessary computational costs. Although NUTS is not a novel methodology, its application in this context significantly boosts the sampling efficiency in subsequent examples. Furthermore, the paper introduces an online error monitoring scheme, which defines the error metric $\epsilon \equiv H(\boldsymbol{q}, \boldsymbol{p}) + \ln u$, where $u \sim \mathcal{U}(0, \exp(-H))$. A threshold $\Delta_{\max}^{\text{HNN}}$ is manually specified, with a recommended value of $\Delta_{\max}^{\text{HNN}} = 10$. When the error exceeds this threshold, i.e., $\epsilon > \Delta_{\max}^{\text{HNN}}$, the algorithm defaults to standard leapfrog integration with numerical gradients of the target density for the subsequent N_{lf} steps, preserving both accuracy and efficiency in the NUTS algorithm. Notably, $\Delta_{\max}^{\text{HNN}}$ is intentionally set far below $\Delta_{\max}^{\text{lf}}$, a threshold suggested by Hoffman and Gelman. Exceeding $\Delta_{\max}^{\text{lf}}$ indicates unreliability in numerical gradients, triggering immediate termination of the "BuildTree" procedure to prevent divergent trajectories.

2 Identified Typographical Errors

The typographical errors identified in the attached manuscript (distinct from the published *Journal of Computational Physics* version) are listed below.

- **Page 6, Eq. 15:** The symbol for p in a set is not correct, it should be:
$$\boldsymbol{u}_p = \phi(\boldsymbol{w}_{p-1}\boldsymbol{u}_{p-1} + \boldsymbol{b}_{p-1}), p \in \{1, \cdots, P\}.$$
- **Page 7, Eq. 18:** The negative sign is missing within the exponential operator. The correct form is:
$$f(q) \propto 0.5 \exp\left(-\frac{(q-1)^2}{2 \times 0.35^2}\right) + 0.5 \exp\left(-\frac{(q+1)^2}{2 \times 0.35^2}\right).$$
- **Page 12, Algorithm 4, line 1:** The initialization of $\boldsymbol{p}^0$ is unnecessary, as it is immediately overwritten by a sample from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$ at the start of each iteration.
- **Page 12, Algorithm 4, line 16:** The condition in the "If" statement should check whether v_j is -1, not j.
- **Page 18:** There is an inconsistency between the illustration and Fig. 10. The variables are labeled as "q" in the illustration, but as "x" in the figure.
- **Page 19, Eq. 26:** The discretization of $\pi(\boldsymbol{u})$ is not correct, it should be:
$$\pi(\boldsymbol{u}) = \exp\left(-\sum_{i=0}^{d-1}\left(\frac{1}{2\Delta x}\left(u\left(i\Delta x + \Delta x\right) - u\left(i\Delta x\right)\right)^2 + \frac{\Delta x}{2}\left(V\left(u\left(i\Delta x + \Delta x\right)\right) + V\left(u\left(i\Delta x\right)\right)\right)\right)\right).$$
Note that the correct symbol is "+" instead of "−". This can be derived by approximating the integral using the trapezoidal rule, which approximates the area under a curve by summing the areas of trapezoids.

3 Critical Components for Implementations That Were Not Mentioned

3.1 Training and test datasets for L-HNNs

Training L-HNNs requires effective training and test datasets, but the paper provides limited description of the process for generating these datasets. For instance, on page 7, it is mentioned that "To generate the training data, we simulated

Hamiltonian dynamics....In total, the training of L-HNNs took 8,000 numerical gradient evaluations." However, a crucial point is that during the simulation process, the position of the trajectory at each time step, corresponding to $\boldsymbol{q}$, $\boldsymbol{p}$, $\nabla_t \boldsymbol{q}$, and $\nabla_t \boldsymbol{p}$, needs to be recorded. This allows for sufficient data to be provided for training based on the loss function constructed in Eq. 13. It is easy to confuse this with the simulation process in HMC, where only a single $\{\boldsymbol{q}^\star, \boldsymbol{p}^\star\}$ is retained at each step. In contrast, the simulation process for training L-HNNs requires saving all $\{\boldsymbol{q}, \boldsymbol{p}\}$.

3.2 Forward and inverse evaluation in leapfrog integration

The Algorithm 2 on Page 6 overlooks the incorporation of the forward and backward directions as input, which is a crucial aspect of the NUTS. Moreover, in Algorithm 5, the build tree function invokes Algorithm 2 with initial conditions $\{\boldsymbol{q}, \boldsymbol{p}\}$, allowing for the computation of $\boldsymbol{q}'$ and $\boldsymbol{p}'$. To rectify this, let $v \in \{1, -1\}$ denote the direction of integration. Then, line 7 of Algorithm 2 should be revised to:

$$q_i(t + \Delta t) = q_i(t) + v \left(\frac{\Delta t}{m_i} p_i(t) - \frac{\Delta t^2}{2 m_i} \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t)} \right),$$

and line 11 of Algorithm 2 should be:

$$p_i(t + \Delta t) = p_i(t) - \frac{v \Delta t}{2} \left(\frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t)} + \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t + \Delta t)} \right).$$

3.3 Potential numerical issues when computing Hamiltonian

When constructing the Hamiltonian using functions $f(\boldsymbol{q})$ and $f(\boldsymbol{p})$, as in Eqs. 18, 21, and 22, it is essential to first derive the analytical expressions for the logarithm of these functions to build the potential energy U and kinetic energy K. Subsequently, the Hamiltonian can be constructed as $H = U + K$, which avoids potential numerical problems.

However, a numerically unstable approach would be to directly multiply $f(\boldsymbol{q})$ and $f(\boldsymbol{p})$ to form $\exp(-U - K)$ and then take the negative logarithm to construct the Hamiltonian. The key difference between these two approaches lies in the fact that, for high-dimensional problems, the potential energy U can take on large values, e.g., 1000. In this case, computing $\exp(-U - K)$ would result in values approaching zero, exceeding the precision of floating-point numbers and rendering the construction of the Hamiltonian ineffective.

3.4 Early stopping mechanism for NUTS

Algorithm 4 in this paper and other algorithms proposed in [2] do not incorporate an early stopping mechanism. However, based on my experience in replicating the examples in the paper, I found that an effective early stopping mechanism can significantly improve the efficiency of sampling the posteriors. This is because in NUTS, the number of leapfrog steps increases exponentially after each call to BuildTree. If the "U-turn" condition is never met and the Hamiltonian is well-preserved during the iterative integration (i.e., $H(q', p') + \ln u < \Delta_{\max}^{\text{lf}}$ is always satisfied), then the computational cost of obtaining a single sample in this loop can become prohibitively expensive. Although such cases are rare, they can significantly impact the overall efficiency due to the large number of samples. Therefore, implementing an early stopping mechanism is necessary. In my code, I set a limit for the variable j, with a default value of 12, corresponding to a maximum of 4028 leapfrog steps. If j exceeds this limit, the loop is terminated. Other similar limits can be set, such as the maximum time spent to obtain each sample.

References

[1] Somayajulu L.N. Dhulipala, Yifeng Che, and Michael D. Shields. Efficient Bayesian inference with latent Hamiltonian neural networks in No-U-Turn Sampling. *Journal of Computational Physics*, 492:112425, 2023.

[2] Matthew D. Hoffman and Andrew Gelman. The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014.