Physics-Informed Machine Learning of Dynamical Systems for Efficient Bayesian Inference

Somayajulu L. N. Dhulipala
Idaho National Laboratory
`Som.Dhulipala@inl.gov`

Yifeng Che
Idaho National Laboratory
`Yifeng.Che@inl.gov`

Michael D. Shields
Johns Hopkins University
`michael.shields@jhu.edu`

Abstract

Although the no-u-turn sampler (NUTS) is a widely adopted method for performing Bayesian inference, it requires numerous posterior gradients which can be expensive to compute in practice. Recently, there has been a significant interest in physics-based machine learning of dynamical (or Hamiltonian) systems and Hamiltonian neural networks (HNNs) is a noteworthy architecture. But these types of architectures have not been applied to solve Bayesian inference problems efficiently. We propose the use of HNNs for performing Bayesian inference efficiently without requiring numerous posterior gradients. We introduce latent variable outputs to HNNs (L-HNNs) for improved expressivity and reduced integration errors. We integrate L-HNNs in NUTS and further propose an online error monitoring scheme to prevent sampling degeneracy in regions where L-HNNs may have little training data. We demonstrate L-HNNs in NUTS with online error monitoring considering several complex high-dimensional posterior densities and compare its performance to NUTS.

1 Introduction

Hamiltonian Monte Carlo (HMC) is a popular sampling algorithm for scalable Bayesian inference considering complex and/or high-dimensional posterior distributions [8]. Given the initial position (or posterior sample) and momenta, HMC simulates the trajectory of a particle using Hamiltonian dynamics with gradients of the target posterior and its position at the end time (T) constitutes a new sample from the posterior. Therefore, the choice of this end time considerably affects the posterior inference quality, and for complex and/or high-dimensional posterior it can be difficult to manually specify the optimal end time [4]. No-u-turn sampling (NUTS) [4] has been introduced to automatically determine the optimal end time. If $\{\boldsymbol{q}, \boldsymbol{p}\}$ denotes a position-momenta (i.e., $\boldsymbol{q} - \boldsymbol{p}$) pair, NUTS builds a binary tree of $\{\boldsymbol{q}, \boldsymbol{p}\}$ states and selects a subset of these states with the aid of slice sampling [7] so as to not violate detailed balance. For each sample, NUTS terminates the tree building procedure whenever a u-turn is encountered. NUTS may also terminate the tree building procedure when the integration error while simulating the particle trajectory is large. NUTS has been shown to result in a better performance over HMC in terms of the effective sample size (ESS) [10] per gradient evaluation by several studies (e.g., [9]). However, it still requires numerous gradients estimations of the target posterior which can be expensive in practice, especially, when dealing with large datasets or computational models [2].

Recently, physics-informed machine learning of dynamical (or Hamiltonian) systems has been receiving a significant interest owing to the wide variety of applications in the physical sciences.

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

Greydanus et al. [3] introduced Hamiltonian Neural Networks (HNNs) to learn Hamiltonian systems in an unsupervised fashion by virtue of a physics-informed loss function. HNNs satisfy important properties such as time reversibility and Hamiltonian conservation and were shown to be superior than data-driven neural networks. Along these lines, several studies proposed improved architectures for learning Hamiltonian systems such as physics-informed HNNs [6], symplectic ODE-Net [11], and symplectic neural nets [5]. However, these efforts were all focused on learning Hamiltonian systems for classical mechanics problems like n-pendulum behavior and planetary motion, and not to perform Bayesian inference efficiently.

We propose latent Hamiltonian neural networks (L-HNNs) in NUTS to efficiently solve Bayesian inference problems without requiring numerous posterior gradient estimations [1]. For this, we first introduce latent variable outputs in HNNs (i.e., L-HNNs) for improved expressivity and reduced integration errors while simulating the Hamiltonian dynamics. This approach is different from the HNNs proposed by Greydanus et al. [3] which directly predict the scalar Hamiltonian instead of deriving it from the predicted latent variable values. Next, we integrate L-HNNs with NUTS and propose an online error monitoring scheme to prevent sampling degeneracy in regions of the uncertainty space where little training data may be available for L-HNNs (for e.g., tails of the posterior). In essence, this novel error monitoring scheme temporarily reverts to using posterior gradients whenever the L-HNNs integration errors are large. We demonstrate L-HNNs in NUTS with online error monitoring considering several complex posteriors and compare its performance with traditional NUTS in terms of effectively capturing the posterior features and the ESS per posterior gradient (i.e., computational efficiency).

2 Methods

L-HNNs are fully-connected feed-forward neural networks defined as:

$$\begin{aligned} &\boldsymbol{u}_p = \phi(\boldsymbol{w}_{p-1}\, \boldsymbol{u}_{p-1} + \boldsymbol{b}_{p-1}), \;\; \{p \in 1, \dots, P\} \\ &\boldsymbol{\lambda} = \boldsymbol{w}_P\, \boldsymbol{u}_P + \boldsymbol{b}_P \end{aligned} \tag{1}$$

where P is the number of hidden layers indexed by p, $\boldsymbol{u}_p$ are the outputs of the hidden layer p, $\boldsymbol{w}_p$ and $\boldsymbol{b}_p$ are the weights and biases, respectively, and $\phi(.)$ is the nonlinear activation function. $\boldsymbol{u}_0$ are the inputs defined by the position-momenta pair $\boldsymbol{z} = \{\boldsymbol{q},\, \boldsymbol{p}\}$. If d is the dimensionality of the uncertainty space, L-HNNs predict d latent variables defined by the vector $\boldsymbol{\lambda}$. The sum of these latent variables is defined as the scalar Hamiltonian predicted by L-HNNs:

$$H_{\boldsymbol{\theta}} = \sum_{i=1}^{d} \lambda_i \tag{2}$$

Whereas, HNNs directly predict the scalar value of the Hamiltonian [3], L-HNNs precit d latent variables whose sum is defined as the scalar Hamiltonian. Introduction of these latent variables improves expressivity and reduces the integration errors when simulating the Hamiltonian trajectories [1]. The training data is provided in terms of sets of $\{\boldsymbol{q},\, \boldsymbol{p}\}$ pairs. Time derivatives of these $\{\boldsymbol{q},\, \boldsymbol{p}\}$ pairs are computed. Gradients of the Hamiltonians predicted by L-HNNs [Equation (2)] are also computed. Then, the following unsupervised physics-based loss function is minimized:

$$\mathcal{L} = \Big\| \frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{p}} - \frac{\Delta \boldsymbol{q}}{\Delta t} \Big\|_2 + \Big\| - \frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{q}} - \frac{\Delta \boldsymbol{p}}{\Delta t} \Big\|_2 \tag{3}$$

The training data time derivatives $\frac{\Delta \boldsymbol{q}}{\Delta t}$ and $\frac{\Delta \boldsymbol{p}}{\Delta t}$ are, respectively, equal to the gradients of the exact Hamiltonians $\frac{\partial H}{\partial \boldsymbol{p}}$ and $-\frac{\partial H}{\partial \boldsymbol{q}}$. Thus, the loss function accounts for the physics governing Hamiltonian systems.

For Bayesian inference problems, the Hamiltonian is usually the sum of negative logarithm of the target posterior (i.e., potential energy) and negative logarithm of the distribution of momenta (i.e., kinetic energy). In most cases, the distribution of momenta is considered to be a standard multivariate Gaussian. In NUTS, it is required to simulate the particle trajectory given the initial $\boldsymbol{z}(0)$. Since the Hamilton's equations are first order ODEs, the particle trajectory is obtained by numerically simulating the following integral with the aid of symplectic integrators like the leap frog:

$$\boldsymbol{z}(T) = \boldsymbol{z}(0) + \int_0^T \begin{bmatrix} \mathbf{0}_{d\times d} & \boldsymbol{I}_{d\times d} \\ -\boldsymbol{I}_{d\times d} & \mathbf{0}_{d\times d} \end{bmatrix} \; \nabla H_{\boldsymbol{\theta}}(\boldsymbol{z})\; dt \tag{4}$$

A noteworthy component of this equation is the term $\nabla H_{\boldsymbol{\theta}}(\boldsymbol{z})$ which is computed through L-HNNs and not by using gradients of the target posterior.

For posterior sampling using NUTS, for each new sample, several trajectories of a fictitious particle need to be simulated in sequence while doubling the trajectory length each time as part of a tree building procedure. The primary criterion for terminating the tree building procedure is when any of the sub-trees makes a u-turn, which works towards reducing the serial correlations between posterior samples. The other criterion for termination is when the integration errors while simulating the trajectories are large, as defined by:

$$\varepsilon \equiv H\big(\boldsymbol{z}\big) + \ln u > \Delta_{max} \tag{5}$$

where $H\big(.\big)$ is the Hamiltonian value given a $\boldsymbol{z} = \{\boldsymbol{q}, \boldsymbol{p}\}$ pair, u is the slice value simulated from $Uniform([0,\ \exp\{-H\big(\boldsymbol{z}(0)\big)\}])$ given the initial $\boldsymbol{z}(0)$ pair, and Δ_{max} is the error threshold which is usually set to 1000. NUTS mostly employs the standard leap frog integrator using gradients of the target posterior for simulating the particle trajectories. While the replacement of the target posterior gradients with L-HNNs gradients [as discussed in Equation 4] is possible and works well for simple problems, for complex posterior spaces, this may lead to sampling degeneracy. The reason for such degeneracy is the termination criterion in Equation (5). Whenever NUTS with L-HNN gradients enters a region of the posterior space where there was little to no training data for L-HNNs, integration errors of the particle trajectories can be large, resulting in a premature termination of the tree building procedure. As a result, there will be clusters of samples at close vicinity in regions of the posterior space where L-HNNs had little to no training data. These clusters of samples constitute the sampling degeneracy problem.

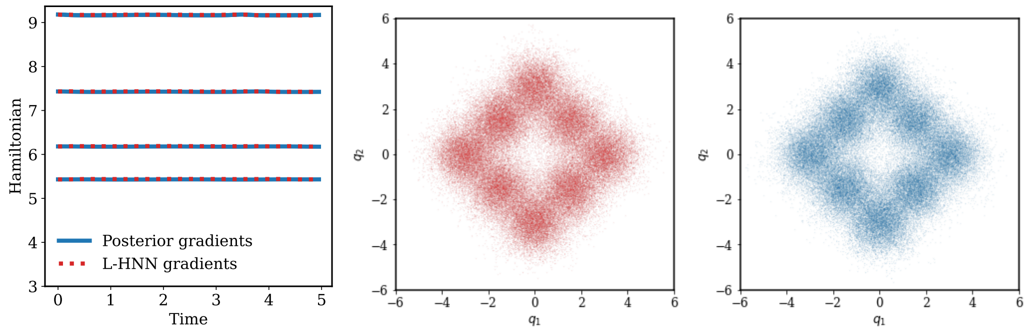
To mitigate the sampling degeneracy problem, we propose an online error monitoring scheme. We introduce two error thresholds, Δ^{hnn}_{max}, when using L-HNN gradients in the leap frog, and Δ^{lf}_{max}, when using the standard leap frog with target posterior gradients. We set $\Delta^{hnn}_{max} << \Delta^{lf}_{max}$; for example, $\Delta^{hnn}_{max} = 10$ and $\Delta^{lf}_{max} = 1000$. The particle trajectories in NUTS are, by default, simulated using the L-HNN gradients. Whenever the integration errors using L-HNNs, as denoted by ε^{hnn}, are greater than Δ^{hnn}_{max}, traditional leap frog with posterior gradients takes over for a few samples N^{lf}. The value of N^{lf} can be in between 5 and 20 and this essentially aids in NUTS returning to regions of high probability density after which the particle trajectories can again be simulated using the L-HNN gradients. Equation 6 summarizes this error monitoring scheme while using L-HNNs in NUTS:

$$\begin{aligned} \varepsilon^{hnn} \leq \Delta^{hnn}_{max} &\rightarrow \text{use leap frog with L-HNN gradients} \\ \varepsilon^{hnn} > \Delta^{hnn}_{max},\ \varepsilon^{lf} \leq \Delta^{lf}_{max} &\rightarrow \text{use leap frog with posterior gradients for } N^{lf} \text{ samples} \\ \varepsilon^{hnn} > \Delta^{hnn}_{max},\ \varepsilon^{lf} > \Delta^{lf}_{max} &\rightarrow \text{terminate tree building; move to next sample} \end{aligned} \tag{6}$$

where ε^{hnn} and ε^{lf} are the integration errors [equation (5)] when using leap frog with L-HNN and posterior gradients, respectively. The online error monitoring scheme aids in ensuring that the we sample from the right target distribution when relying on L-HNNs. In limit, as $\Delta^{hnn}_{max} \rightarrow -\infty$, NUTS with online error monitoring converges to the traditional NUTS algorithm.

3 Results

We demonstrate Bayesian inference using L-HNNs in NUTS with online error monitoring considering several case studies. The performance is compared with traditional NUTS which relies on the posterior gradients. The training dataset for L-HNNs is generated using the HMC algorithm with the number of samples, the length of the trajectory for each sample, and the step size specified. First, we consider a 2-D eight Gaussian mixture density. Each Gaussian has the same covariance which is an identity matrix but different mean vectors. Figure 1a presents the Hamiltonian evolution with time for different initial $\{\boldsymbol{q}, \boldsymbol{p}\}$ consider LHNN-NUTS and NUTS. It is noticed that LHNN-NUTS satisfactorily conserves the Hamiltonian values in a similar fashion to that of NUTS. Figures 1b and 1c present the scatter plot of $100,000$ simulated samples (first 5000 considered as burn-in) using LHNN-NUTS and NUTS, respectively. LHNN-NUTS captures all the posterior modes satisfactorily in a similar fashion to that of NUTS. Moreover, from a computational perspective, while NUTS requires more than 15 Million posterior gradients, LHNN-NUTS requires only 0.4 Million posterior gradients for the training data and during online error monitoring. The ESS per posterior gradient of LHNN-NUTS and NUTS is 0.0269 and 0.000707, respectively, which is between 1-2 orders of magnitude in improvement.



(a) (b) (c)

Figure 1: A 2-D eight Gaussian mixture is considered as the target posterior. (a) Hamiltonian conservation for random initial $\{\boldsymbol{q}, \boldsymbol{p}\}$ values considering L-HNN gradients and posterior gradients. (b) Samples from the target posterior simulated using LHNN-NUTS. (c) Samples from the target posterior simulated using traditional NUTS.

We have also tested the performance of LHNN-NUTS considering several other complex high-dimensional posteriors. Table 1 summarizes the results in terms of the number of gradients and the ESS per posterior gradient and also presents these values corresponding to NUTS for comparison. Note that the number of gradients listed below for LHNN-NUTS includes those required during training and those required during the online error monitoring scheme. Overall, it can be observed that LHNN-NUTS requires about 1-2 orders of magnitude less posterior gradients than NUTS. Moreover, LHNN-NUTS results in about an order of magnitude improvement in the ESS per posterior gradient compared to NUTS. These results contribute to the scalability and efficiency of the proposed LHNNs in NUTS with online error monitoring scheme relative to NUTS which is widely adopted for Bayesian inference.

Table 1: Summary of the performance comparison between LHNN-NUTS and NUTS (adapted from Dhulipala et al. [1]). The number of gradients listed below for LHNN-NUTS includes those required during training and those required during the online error monitoring scheme.

Posterior density	**QOI**	**LHNN-NUTS**	**NUTS**
2-D eight Gaussian mixture	# gradients	0.4 Million	15 Million
	ESS/gradient	0.0269	0.000707
10-D Rosenbrock density	# gradients	0.42 Million	7 Million
	ESS/gradient	0.0368	0.00219
24-D Bayesian logistic regression	# gradients	0.4 Million	1.2 Million
	ESS/gradient	0.243	0.0777
100-D rough well	# gradients	0.42 Million	1.28 Million
	ESS/gradient	0.0065	0.00373

4 Summary and conclusions

Physics-based neural networks that learn Hamiltonian systems (e.g., HNNs) have been receiving a great deal of interest. We proposed using HNNs to efficiently solve Bayesian inference problems by not requiring numerous posterior gradients. To increase the expressivity of HNNs and reduce the integration error while simulating Hamiltonian trajectories, we proposed HNNs with latent outputs (L-HNNs). We further proposed to use L-HNNs in NUTS with an online error monitoring scheme that reverts to using the posterior gradients for a few samples whenever the L-HNNs integration errors are high, which may be in regions of the uncertainty space where L-HNNs had little training data. Considering several complex high-dimensional posteriors, we demonstrated LHNN-NUTS with online error monitoring and compared its performance with NUTS. Overall, LHNN-NUTS with online error monitoring required 1-2 orders of magnitude lesser posterior gradients and resulted in about 1 order of magnitude better ESS per gradient compared to NUTS.

Software tools

https://github.com/IdahoLabResearch/BIhNNs

Acknowledgments

This research is supported through INL's Laboratory Directed Research and Development (LDRD) Program under U.S. Department of Energy (DOE) Idaho Operations Office contract no. DE-AC07-05ID14517.

Broader impact

Bayesian inference is a principal method for parameter calibration and uncertainty quantification in many scientific and engineering fields. Since closed-form solutions usually do not exist for practical Bayesian inference problems, Markov chain Monte Carlo (MCMC) algorithms are employed to sample from the target probability density. While random-walk MCMC algorithms are popular, they have poor scalability with the number of dimensions, and can feature large serial correlations between the samples. Therefore, algorithms under the HMC framework (including NUTS) are often employed. However, the HMC-based algorithms require numerous costly gradient estimations of the target posterior and can be expensive in practice. The proposed LHNN-NUTS with online error monitoring removes this computational hurdle and contributes to solving practical Bayesian inference problems across science and engineering while guaranteeing robustness.

References

[1] S. L. N. Dhulipala, Y. Che, and M. D. Shields. Bayesian inference with latent Hamiltonian neural networks, Aug. 2022. arXiv:2208.06120.

[2] S. L. N. Dhulipala, M. D. Shields, B. W. Spencer, C. Bolisetti, A. E. Slaughter, V. M. Laboure, and P. Chakroborty. Active learning with multifidelity modeling for efficient rare event simulation. *Journal of Computational Physics*, 468:111506, 2022.

[3] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks, Sept. 2019. arXiv:1906.01563.

[4] M. D. Hoffman and A. Gelman. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[5] P. Jin, Z. Zhang, A. Zhu, Y. Tang, and G. E. Karniadakis. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. *Neural Networks*, 132:166–179, 2020.

[6] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas. Hamiltonian neural networks for solving equations of motion. *Physical Review E*, 105:065305, 2022.

[7] R. M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003.

[8] R. M. Neal. MCMC using Hamiltonian dynamics, June 2012. arXiv:1206.1901.

[9] M. Nishio and A. Arakawa. Performance of Hamiltonian Monte Carlo and No-U-Turn Sampler for estimating genetic parameters and breeding values. *Genetics Selection Evolution*, 51:1–12, 2019.

[10] A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Burkner. Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. *Bayesian Analysis*, 16(2):667–718, 2021.

[11] Y. D. Zhong, B. Dey, and A. Chakraborty. Symplectic ode-net: Learning Hamiltonian dynamics with control, Apr. 2019. arXiv:1909.12077.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [No]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix A: Algorithms for training L-HNNs and using L-HNNs in NUTS with online error monitoring

Algorithm 1 Hamiltonian neural network training [1]

```
1: HNN parameters: $\boldsymbol{\theta}$; Training data: $\{\boldsymbol{q},\ \boldsymbol{p}\}$
2: Evaluate gradients of the training data $\frac{\Delta \boldsymbol{q}}{\Delta t}$ and $\frac{\Delta \boldsymbol{p}}{\Delta t}$
3: Initialize HNN parameters $\boldsymbol{\theta}$
4: Compute HNN Hamiltonian $H_{\boldsymbol{\theta}}$
5: Compute $H_{\boldsymbol{\theta}}$ gradients $\frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{p}}$ and $-\frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{q}}$
6: Compute loss $\mathcal{L} = ||\frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{p}} - \frac{\Delta \boldsymbol{q}}{\Delta t}|| + || - \frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{q}} - \frac{\Delta \boldsymbol{p}}{\Delta t}||$
7: Minimize $\mathcal{L}$ with respect to $\boldsymbol{\theta}$
```

Algorithm 2 Latent Hamiltonian neural networks evaluation in leapfrog integration [1]

```
1: Hamiltonian: $H$; Initial conditions: $\boldsymbol{z}(0) = \{\boldsymbol{q}(0),\ \boldsymbol{p}(0)\}$; Dimensions: $d$; Steps: $N$; End time: $T$
2: $\Delta t = \frac{T}{N}$
3: for $j = 0 : N - 1$ do
4:     $t = j\ \Delta t$
5:     Compute HNN output gradient $\frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{q}(t)}$
6:     for $i = 1 : d$ do
7:         $q_i(t + \Delta t) = q_i(t) + \frac{\Delta t}{m_i}\ p_i(t) - \frac{\Delta t^2}{2 m_i}\ \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t)}$
8:     end for
9:     Compute HNN output gradient $\frac{\partial H_{\boldsymbol{\theta}}}{\partial \boldsymbol{q}(t+\Delta t)}$
10:    for $i = 1 : d$ do
11:        $p_i(t + \Delta t) = p_i(t) - \frac{\Delta t}{2} \left( \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t)} + \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t+\Delta t)} \right)$
12:    end for
13: end for
```

Algorithm 3 L-HNNs in NUTS with online error monitoring (main loop) [1]

```
1: Hamiltonian: $H = U(\boldsymbol{q}) + K(\boldsymbol{p})$, Samples: M; Starting sample: $\{\boldsymbol{q}^0,\ \boldsymbol{p}^0\}$; Step size: $\Delta t$; Threshold for leapfrog: $\Delta^{lf}_{max}$; Threshold for L-HNNs: $\Delta^{hnn}_{max}$; Number of leapfrog samples: $N_{lf}$
2: Initialize $\mathbf{1}_{lf} = 0,\ n_{lf} = 0$
3: for $i = 1 : M$ do
4:     $\boldsymbol{p}(0) \sim \mathcal{N}(\mathbf{0},\ \boldsymbol{I}_d)$
5:     $\boldsymbol{q}(0) = \boldsymbol{q}^{i-1}$
6:     $u \sim Uniform\Big(\Big[0,\ \exp\{-H\big(\boldsymbol{q}(0),\ \boldsymbol{p}(0)\big)\}\Big]\Big)$
7:     Initialize $\boldsymbol{q}^- = \boldsymbol{q}(0), \boldsymbol{q}^+ = \boldsymbol{q}(0), \boldsymbol{p}^- = \boldsymbol{p}(0), \boldsymbol{p}^+ = \boldsymbol{p}(0), j = 0, \boldsymbol{q}^* = \boldsymbol{q}^{i-1}, n = 1, s = 1$
8:     if $\mathbf{1}_{lf} = 1$ then
9:         $n_{lf} \leftarrow n_{lf} + 1$
10:    end if
11:    if $n_{lf} = N_{lf}$ then
12:        $\mathbf{1}_{lf} = 0,\ n_{lf} = 0$
13:    end if
14:    while $s = 1$ do
15:        Choose direction $\nu_j \sim Uniform(\{-1,\ 1\})$
16:        if $j = -1$ then
17:            $\boldsymbol{q}^-, \boldsymbol{p}^-, \_, \_, \boldsymbol{q}', \boldsymbol{p}', n', s', \mathbf{1}_{lf} =$ BuildTree$(\boldsymbol{q}^-, \boldsymbol{p}^-, u, \nu_j, j, \Delta t, \mathbf{1}_{lf})$
18:        else
19:            $\_, \_, \boldsymbol{q}^+, \boldsymbol{p}^+, \boldsymbol{q}', \boldsymbol{p}', n', s', \mathbf{1}_{lf} =$ BuildTree$(\boldsymbol{q}^+, \boldsymbol{p}^+, u, \nu_j, j, \Delta t, \mathbf{1}_{lf})$
20:        end if
21:        if $s' = 1$ then
22:            With probability min$\{1,\ \frac{n'}{n}\}$, set $\{\boldsymbol{q}^i, \boldsymbol{p}^i\} \leftarrow \{\boldsymbol{q}', \boldsymbol{p}'\}$
23:        end if
24:        $n \leftarrow n + n'$
25:        $s \leftarrow s'\mathbf{1}\big[(\boldsymbol{q}^+ - \boldsymbol{q}^-)\cdot\boldsymbol{p}^- \geq 0\big]\mathbf{1}\big[(\boldsymbol{q}^+ - \boldsymbol{q}^-)\cdot\boldsymbol{p}^+ \geq 0\big]$
26:        $j \leftarrow j + 1$
27:    end while
28: end for
```

Algorithm 4 L-HNNs in NUTS with online error monitoring (build tree function) [1]

```
1: function BuildTree$(\boldsymbol{q}, \boldsymbol{p}, u, \nu, j, \Delta t, \mathbf{1}_{lf})$
2: if $j = 0$ then
3:     Base case taking one leapfrog step
4:     $\boldsymbol{q}', \boldsymbol{p}' \leftarrow$ Algorithm 2 with initial conditions: $\boldsymbol{z}(0) = \{\boldsymbol{q},\ \boldsymbol{p}\}$, Steps: 1; End Time: $\Delta t$
5:     $\mathbf{1}_{lf} \leftarrow \mathbf{1}_{lf}$ or $\mathbf{1}\big[H\big(\boldsymbol{q}',\ \boldsymbol{p}'\big) + \ln u > \Delta^{hnn}_{max}\big]$
6:     $s' \leftarrow \mathbf{1}\big[H\big(\boldsymbol{q}',\ \boldsymbol{p}'\big) + \ln u \leq \Delta^{hnn}_{max}\big]$
7:     if $\mathbf{1}_{lf} = 1$ then
8:         $\boldsymbol{q}', \boldsymbol{p}' \leftarrow$ Leapfrog integration with initial conditions: $\boldsymbol{z}(0) = \{\boldsymbol{q},\ \boldsymbol{p}\}$, Steps: 1; End Time: $\Delta t$
9:         $s' \leftarrow \mathbf{1}\big[H\big(\boldsymbol{q}',\ \boldsymbol{p}'\big) + \ln u \leq \Delta^{lf}_{max}\big]$
10:    end if
11:    $n' \leftarrow \mathbf{1}\big[u \leq \exp\{-H\big(\boldsymbol{q}',\ \boldsymbol{p}'\big)\}\big]$
12:    return $\boldsymbol{q}',\ \boldsymbol{p}', \boldsymbol{q}',\ \boldsymbol{p}', \boldsymbol{q}', \boldsymbol{p}', n', s', \mathbf{1}_{lf}$
13: else
14:    Recursion to build left and right sub-trees (follows from Algorithm 3 in [4], with $\mathbf{1}_{lf}$ additionally passed to and retrieved from every BuildTree evaluation)
15:    return $\boldsymbol{q}^-,\ \boldsymbol{p}^-, \boldsymbol{q}^+,\ \boldsymbol{p}^+, \boldsymbol{q}', \boldsymbol{p}', n', s', \mathbf{1}_{lf}$
16: end if
```