



Efficient Bayesian inference with latent Hamiltonian neural networks in No-U-Turn Sampling

November 2023

Changing the World's Energy Future

Som LakshmiNarasimha Dhulipala, Michael Shields, Yifeng Che



DISCLAIMER

This information was prepared as an account of work sponsored by an agency of the U.S. Government. Neither the U.S. Government nor any agency thereof, nor any of their employees, makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness, of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the U.S. Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the U.S. Government or any agency thereof.

Efficient Bayesian inference with latent Hamiltonian neural networks in No-U-Turn Sampling

Som LakshmiNarasimha Dhulipala, Michael Shields, Yifeng Che

November 2023

**Idaho National Laboratory
Idaho Falls, Idaho 83415**

<http://www.inl.gov>

**Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517**

Efficient Bayesian inference with latent Hamiltonian neural networks in No-U-Turn Sampling

Somayajulu L.N. Dhulipala ^a, Yifeng Che ^a, Michael D. Shields ^b

a) Computational Mechanics and Materials, Idaho National Laboratory, Idaho Falls, ID
83415, USA

b) Department of Civil and Systems Engineering, Johns Hopkins University, Baltimore, MD
21218, USA

Abstract

When sampling for Bayesian inference, one popular approach in the computational field is to use Hamiltonian Monte Carlo (HMC) and specifically the No-U-Turn Sampler (NUTS), which automatically decides the end time of the Hamiltonian trajectory. However, HMC and NUTS can require numerous numerical gradients of the target density, and can prove slow in practice when relying on computationally expensive forward models. We propose Latent Hamiltonian neural networks (L-HNNs) with HMC and NUTS for solving Bayesian inference problems. Once trained, L-HNNs do not require numerical gradients of the target density during sampling, and hence numerous evaluations of the forward computational model. Moreover, L-HNNs satisfy important properties such as perfect time reversibility and Hamiltonian conservation, making them well-suited for use within HMC and NUTS because stationarity can be shown. We also propose the integration of L-HNNs in an online error monitoring scheme, in which numerical gradients of the target density are used for a few samples whenever the L-HNNs prediction errors are large. This online error monitor scheme prevents sample degeneracy in regions of low probability density and ensures robust uncertainty quantification. We demonstrate L-HNNs in NUTS with online error monitoring on several analytical examples involving complex, heavy-tailed, and high-local-curvature probability densities. We then demonstrate the applicability of L-HNNs in NUTS to two computational case studies, namely the Allen-Cahn stochastic partial differential equation and an elliptic partial differential equation with 25 and 50 inference parameters, respectively. Overall, the L-HNNs in NUTS with online error monitoring satisfactorily inferred these probability densities. Compared to traditional NUTS, L-HNNs in

NUTS with online error monitoring required 1–2 orders of magnitude fewer numerical gradients of the target density and improved the effective sample size (ESS) per gradient (which is a measure of both the sampling quality and the computational expense) by an order of magnitude.

1. Introduction

Bayesian inference is a principal method for parameter calibration and uncertainty quantification for many computational applications. For example, Xia et al. [1], Warne et al. [2], and Rossat et al. [3] use Bayesian inference, respectively, for parameter inversion in multiscale systems, inferring partially observed stochastic processes, and modeling rare events. 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 are often used in the Hamiltonian Monte Carlo (HMC) [4] framework (including the No-U-Turn Sampler [NUTS] [5]). HMC methods simulate Hamiltonian dynamics to propose new samples, instead of drawing them randomly, which can drastically improve the sampling acceptance rate. However, the HMC class of algorithms can require numerous numerical gradients of the target density, which are computationally expensive, especially if the likelihood function involves calling a costly computational model. This paper proposes latent Hamiltonian neural networks (L-HNNs) with HMC and NUTS for solving Bayesian inference problems. Once trained, latent L-HNNs do not require evaluating numerical gradients of the target density while sampling, and can therefore lead to computational gains over traditional HMC and NUTS for Bayesian inference.

1.1. Literature review

Owing to the high computational cost of Bayesian inference, machine learning has garnered considerable interest to accelerate this process. For example, Che et al. [6] and Dhulipala et al. [7] utilized Gaussian processes to accelerate random-walk MCMC by replacing computationally expensive models for likelihood evaluation. Gradient-based inference methods (e.g., Langevin dynamics sampling and HMC) have better scalability than random-walk MCMC,

and machine learning techniques are being employed to accelerate these methods as well. For example, Gu et al. [8] trained neural networks that take as input the current position and the log-likelihood gradient and predict the new position in the Metropolis-adjusted Langevin algorithm (MALA). Li et al. [9] proposed neural networks that learn logarithmic likelihood and applied neural network gradients instead of the target density gradients in an HMC. Levy et al. [10] developed neural networks that take logarithmic likelihood as input and predict the transition kernel in an HMC, which then showed improved performance over traditional HMC for complex cases such as multimodal densities. One thing all these efforts to accelerate Bayesian inference have in common is the use of purely data-driven machine learning models. Furthermore, although neural network-assisted HMC has shown improved performance over traditional HMC, Hoffman and Gelman [5] demonstrated NUTS to be at least an order of magnitude more efficient than conventional HMC (in terms of effective sample size [ESS] [11] per gradient), even without relying on machine learning. The efficiency of NUTS over HMC, MALA, and random walk MCMC has also been demonstrated in many other studies (e.g., [12]).

To efficiently predict the behavior of dynamical systems, recent studies have investigated physics-based neural networks. Greydanus et al. [13] proposed HNNs to learn Hamiltonian systems in an unsupervised fashion through a loss function that accounts for Hamilton's equations. HNNs have interesting properties such as perfect time reversibility and Hamiltonian conservation, and thus proved superior in performance compared to data-driven neural networks and black-box ordinary differential equation (ODE) solvers such as neural ODEs [14]. More recently, newer architectures to learn Hamiltonian systems have been proposed. Physics-informed HNNs [15], symplectic ODE-Net [16], and symplectic neural networks [17] are examples of such architectures. However, efforts along these lines were driven from the standpoint of efficiently identifying and learning dynamical systems, rather than solving Bayesian inference problems. As such, the example applications that these studies have demonstrated correspond to classical mechanics problems such as the behavior of a n-pendulum, the gravitational attraction between bodies (e.g., planets), and the dynamics of predator-prey systems, etc.

1.2. Contributions of this paper

We propose using HNNs for Bayesian inference problems. As summarized in Fig. 1, once trained, HNNs do not require numerical gradients of the target density. As discussed earlier, HNNs satisfy important properties such as perfect time reversibility and Hamiltonian conservation, which make them amenable for HMC sampling, since stationarity, an important condition satisfied by MCMC samplers, can be shown. We also propose an HNN extension called latent HNNs (L-HNNs), which use latent variable outputs to improve the expressivity of HNNs. L-HNNs have reduced integration errors when simulating the Hamiltonian dynamics and show better performance than HNNs when sampling for Bayesian inference. Furthermore, given that the performance of NUTS has generally been shown to be superior to that of HMC, MALA, and random walk MCMC, we propose to use L-HNNs in NUTS with an online error monitoring scheme. This error monitoring scheme reverts to using numerical gradients of the target density for a few samples whenever L-HNN integration errors are large, which may be the result of inadequate training data in certain regions of the uncertainty space. Overall, the online error monitoring scheme prevents sampling degeneracy when L-HNNs are used in NUTS. In summary, the novel contributions of this paper are:

- Use of HNNs for Bayesian inference problems without requiring numerous numerical gradients of the target density.
- Development of latent variable outputs in HNNs (L-HNNs) for improved expressivity and reduced integration errors when simulating Hamiltonian dynamics.
- Use of L-HNNs in NUTS with an online error monitoring scheme to prevent sampling degeneracy.

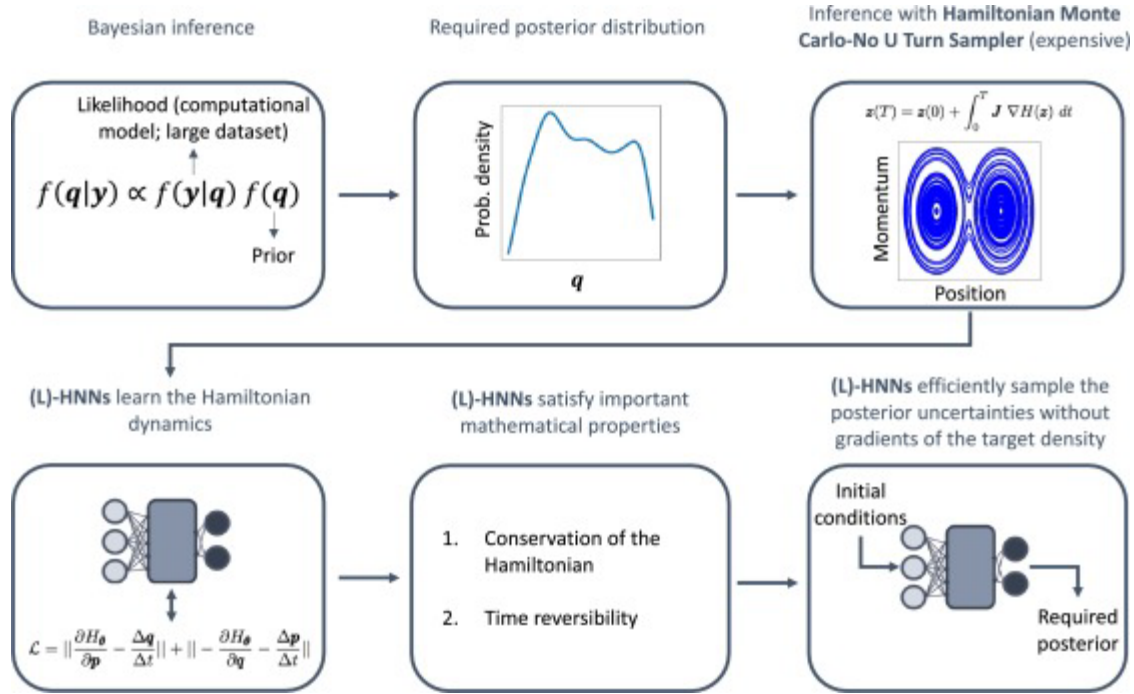


Fig. 1. Schematic of the use of (L)-HNNs for efficient Bayesian inference. Once trained, (L)-HNNs do not require numerical gradients of the posterior distribution—which are expensive to compute—for sampling.

2. Preliminaries

This section provides a brief background on Hamiltonian dynamics simulation and HMC sampling.

2.1. Simulating Hamiltonian dynamics

The basics of Hamiltonian dynamics and its numerical simulation using time integration schemes have been thoroughly discussed (e.g., [4], [18]). If \mathbf{q} and \mathbf{p} represent the position and momentum vectors, respectively, the Hamiltonian of a system is defined as:

$$H(\mathbf{z} = \{\mathbf{q}, \mathbf{p}\}) = U(\mathbf{q}) + K(\mathbf{p}) \tag{1}$$

where \mathbf{z} is a concatenated vector of the vectors \mathbf{q} and \mathbf{p} , $U(\cdot)$ is the potential energy and $K(\cdot)$ is the kinetic energy. Hamilton's equations are given by:

(2)

$$\frac{d\mathbf{z}}{dt} = \mathbf{J}\nabla H(\mathbf{z})$$

where ∇ is the gradient operator, and if d is the dimensionality of the system, \mathbf{J} is a $2d \times 2d$ matrix given by:

(3)

$$\mathbf{J} = \begin{bmatrix} \mathbf{0}_{d \times d} & \mathbf{I}_{d \times d} \\ -\mathbf{I}_{d \times d} & \mathbf{0}_{d \times d} \end{bmatrix}$$

where \mathbf{I} is an identity matrix and $\mathbf{0}$ is a zero matrix. The equations in Equation (2) can be expressed explicitly in each of the dimensions d as:

(4)

$$\begin{aligned} \frac{dq_i}{dt} &= \frac{\partial H}{\partial p_i} \\ \frac{dp_i}{dt} &= -\frac{\partial H}{\partial q_i} \end{aligned} \quad \forall i \in [1, \dots, d].$$

Equation (2) is a coupled first-order ODE with general solution given by:

(5)

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T \mathbf{J}\nabla H(\mathbf{z}) dt$$

Hamiltonian dynamics is numerically simulated using a symplectic time integrator [19], [20]. A symplectic time integrator is better suited than other integrators (e.g., Runge-Kutta) because it satisfies two key properties: (1) time reversibility and (2) volume preservation in the $\{\mathbf{q}, \mathbf{p}\}$ phase space. Both properties are critical for accurately simulating Hamiltonian dynamics over long trajectories. This paper uses the basic symplectic integrator known as the synchronized leapfrog integration scheme. It is first assumed that the mass matrix associated with the kinetic energy $K(\mathbf{p})$ is a diagonal matrix, and thus $K(\mathbf{p})$ can be expressed as:

(6)

$$K(\mathbf{p}) \propto \sum_{i=1}^d \frac{p_i^2}{2m_i} \forall i \in [1, \dots, d]$$

where m_i is the mass associated with dimension i . In the synchronized leapfrog integration scheme, the position in dimension i (q_i) is first updated by:

(7)

$$q_i(t + \Delta t) = q_i(t) + \frac{\Delta t}{m_i} p_i(t) + \frac{\Delta t^2}{2m_i} \dot{p}_i(t) = q_i(t) + \frac{\Delta t}{m_i} p_i(t) - \frac{\Delta t^2}{2m_i} \frac{\partial H}{\partial q_i(t)}$$

where Δt is the integration time step and $\dot{p}_i(t)$ is replaced with $-\frac{\partial H}{\partial q_i(t)}$ from Equation (4). Next, the momentum in dimension i (p_i) is updated by:

(8)

$$p_i(t + \Delta t) = p_i(t) + \frac{\Delta t}{2} (\dot{p}_i(t) + \dot{p}_i(t + \Delta t)) = p_i(t) - \frac{\Delta t}{2} \left(\frac{\partial H}{\partial q_i(t)} + \frac{\partial H}{\partial q_i(t + \Delta t)} \right)$$

where $p_i(t)$ is again replaced with $-\frac{\partial H}{\partial q_i(t)}$ from Equation (4). Positions and momenta in all dimensions are iteratively updated at each time step until a specified end time T is reached.

2.2. Hamiltonian Monte Carlo

The fundamentals of HMC are thoroughly discussed in several resources (e.g., [4], [21], [22]). The HMC algorithm aims to sample from a joint probability distribution of the following form:

$$f(\mathbf{z} = \{\mathbf{q}, \mathbf{p}\}) \propto \exp(-H(\mathbf{z} = \{\mathbf{q}, \mathbf{p}\})) \quad (9)$$

Using the definition of the Hamiltonian from Equation (1), the joint distribution can be explicitly expressed to include contributions from the potential and kinetic energies as follows:

$$f(\{\mathbf{q}, \mathbf{p}\}) \propto \exp(-U(\mathbf{q}))\exp(-K(\mathbf{p})) \quad (10)$$

The potential energy $U(\mathbf{q})$ is defined as the negative logarithm of the likelihood function times the prior distribution:

$$U(\mathbf{q}) \propto -\log [\mathcal{L}(\mathbf{q}|D)g(\mathbf{q})] \quad (11)$$

where $\mathcal{L}(\mathbf{q}|D)$ is the likelihood function of the data D , and $g(\mathbf{q})$ is the prior distribution. $K(\mathbf{p})$ is also defined to simulate Hamiltonian dynamics. In the basic HMC algorithm, \mathbf{p} is assumed to follow a multivariate Gaussian with a diagonal covariance matrix \mathbf{M} . \mathbf{M} has d diagonal

components: $m_i \forall i \in [1, \dots, d]$ is then defined as the negative logarithm of a multivariate Gaussian and is given by Equation (6). This definition of \mathbf{M} implies that the Hamiltonian dynamics operates in a Euclidean space. In contrast, if \mathbf{M} is nondiagonal but symmetric positive definite and depends on \mathbf{q} , then the Hamiltonian dynamics operates in a Riemannian space [23], and the associated Monte Carlo algorithm is called Riemannian HMC.

In the basic HMC algorithm, \mathbf{p} is drawn from a multivariate Gaussian with a diagonal \mathbf{M} , the Hamiltonian dynamics is simulated given T and the starting position vector \mathbf{q}^0 , and the vector $\{\mathbf{q}, \mathbf{p}\}$ at the end of the time integration serves as the Markov chain proposal $\{\mathbf{q}^*, \mathbf{p}^*\}$. This proposed sample is accepted as the next sample with the following probability:

(12)

$$\alpha = \min[1, \exp(H(\{\mathbf{q}^{i-1}, \mathbf{p}^{i-1}\}) - H(\{\mathbf{q}^*, \mathbf{p}^*\}))]$$

where $\{\mathbf{q}^{i-1}, \mathbf{p}^{i-1}\}$ is an accepted sample from the previous iteration of HMC.

3. Sampling with L-HNNs

Greydanus et al. [13] introduced HNNs as unsupervised machine learning architectures to learn Hamiltonian systems. Specifically, HNNs learn the gradients of the Hamiltonian, which can be further used in a time integration scheme to predict the Hamiltonian trajectories. Given training data $\{\mathbf{q}, \mathbf{p}\}$, numerical gradients are first calculated to give numerical time derivatives $\{\frac{\Delta \mathbf{q}}{\Delta t}, \frac{\Delta \mathbf{p}}{\Delta t}\}$, according to Hamilton's equations in (4). For some network parameters θ , the Hamiltonians predicted by HNN (i.e., H_θ) and their corresponding gradients (i.e., $\frac{\partial H_\theta}{\partial \mathbf{p}}$ and $\frac{\partial H_\theta}{\partial \mathbf{q}}$) are also calculated. HNNs optimize the network parameters θ to minimize the following loss function:

(13)

$$L = \left\| \frac{\partial H_\theta}{\partial \mathbf{p}} - \frac{\Delta \mathbf{q}}{\Delta t} \right\| + \left\| -\frac{\partial H_\theta}{\partial \mathbf{q}} - \frac{\Delta \mathbf{p}}{\Delta t} \right\|$$

Algorithm 1 summarizes the procedure for training HNNs. During the evaluation phase, the HNNs predict the numerical gradients of the Hamiltonian $\{\frac{\partial H_\theta}{\partial \mathbf{p}}, \frac{\partial H_\theta}{\partial \mathbf{q}}\}$, without explicit calls to the joint density $f(\{\mathbf{q}, \mathbf{p}\}) \propto \exp(-H(\{\mathbf{q}, \mathbf{p}\}))$. Therefore, during evaluation, HNNs require neither the numerical gradients of the potential energy function $U(\mathbf{q}) \propto -\log[\mathcal{L}(\mathbf{q}|D)g(\mathbf{q})]$ nor explicit calls to it. This, by itself, leads to computational gains since these gradients only need to be evaluated during HNN training, which requires comparatively small data sets. Moreover, evaluating the likelihood $\mathcal{L}(\cdot)$ in $U(\mathbf{q})$ can require calling an expensive computational model or a large data set whose numerical gradients are very expensive to evaluate. Thus, avoiding numerical gradients of $U(\mathbf{q})$ can lead to even greater computational gains. Finally, gradients predicted by HNNs can be used to solve the following initial value problem to predict Hamiltonian dynamics:

(14)

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T \mathbf{J}\nabla H_\theta(\mathbf{z})dt$$

where $\nabla H_\theta(\mathbf{z}) = \{\frac{\partial H_\theta}{\partial \mathbf{p}}, \frac{\partial H_\theta}{\partial \mathbf{q}}\}$ are predicted by HNNs. Equation (14) can be numerically solved using the synchronized leapfrog symplectic integrator, as presented in Algorithm 2. The Hamiltonian gradients are provided by HNNs (that is, line 5 in Algorithm 2) rather than relying on numerical gradient evaluations of the Hamiltonian function.

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

Algorithm 1. Hamiltonian neural network training.

```

1: Hamiltonian:  $H$ ; Initial conditions:  $\mathbf{z}(0) = \{\mathbf{q}(0), \mathbf{p}(0)\}$ ; 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  $\frac{\partial H_{\theta}}{\partial \mathbf{q}(t)}$ 
6:   for  $i = 1 : d$  do
7:      $q_i(t + \Delta t) = q_i(t) + \frac{\Delta t}{m_i} p_i(t) - \frac{\Delta t^2}{2m_i} \frac{\partial H_{\theta}}{\partial q_i(t)}$ 
8:   end for
9:   Compute HNN output gradient  $\frac{\partial H_{\theta}}{\partial \mathbf{q}(t + \Delta t)}$ 
10:  for  $i = 1 : d$  do
11:     $p_i(t + \Delta t) = p_i(t) - \frac{\Delta t}{2} \left( \frac{\partial H_{\theta}}{\partial q_i(t)} + \frac{\partial H_{\theta}}{\partial q_i(t + \Delta t)} \right)$ 
12:  end for
13: end for

```

Algorithm 2. Hamiltonian neural networks evaluation in leapfrog integration.

To generate the training data for HNNs and to further use them to solve Equation (14), Greydanus et al. [13] relied on the Runge-Kutta scheme. A noteworthy difference in our approach is that we rely on the synchronized leapfrog integrator. Being a symplectic integrator, synchronized leapfrog leads to stable Hamiltonian dynamics and better training data quality under relatively larger time steps. Furthermore, for the numerical evaluation of Equation (14) using HNNs, our initial studies indicated that synchronized leapfrog improves Hamiltonian conservation, which is key to an effective sampling scheme, as discussed in Section 3.3.

3.1. Latent outputs in HNNs

The HNNs proposed by Greydanus et al. [13] take as input the $\{\mathbf{q}, \mathbf{p}\}$ coordinates during the forward pass and return a scalar output value (i.e., the ‘‘Hamiltonian’’ $[H_{\theta}]$). Gradients are computed with respect to this Hamiltonian value and the loss function in Equation (13) is minimized. However, for Bayesian inference, it may be more beneficial for the HNNs to predict d latent variables $\lambda_i \{i \in 1, \dots, d\}$ denoted by the vector $\boldsymbol{\lambda}$, where d is the dimensionality of the uncertainty space. The architecture of L-HNNs is formally defined as:

$$\mathbf{u}_p = \phi(\mathbf{w}_{p-1} \mathbf{u}_{p-1} + \mathbf{b}_{p-1}), \{p \in 1, \dots, P\} \boldsymbol{\lambda} = \mathbf{w}_P \mathbf{u}_P + \mathbf{b}_P$$

(15)

where P is the number of hidden layers indexed by p , \mathbf{u}_p are the outputs of the hidden layer p , \mathbf{w}_p and \mathbf{b}_p are the weights and biases, respectively, $\phi(\cdot)$ is the nonlinear activation function, and \mathbf{u}_0 are the inputs $\mathbf{z} = \{\mathbf{q}, \mathbf{p}\}$. The Hamiltonian computed by L-HNNs is defined as:

(16)

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

Gradients are then calculated with respect to the Hamiltonian in Equation (16) and the loss function in Equation (13) is minimized. Through these latent variables, L-HNNs have more degrees of freedom and expressivity than traditional HNNs, potentially leading to reduced integration errors when predicting Hamiltonian dynamics. As a result, L-HNNs can show improved sampling efficiency compared to HNNs. A comparison between the performance of L-HNNs and HNNs is presented in Section 5 for a 3-D Rosenbrock density function.

The training cost of traditional HNNs proposed by Greydanus et al. [13] is very similar to that of deep neural networks, as we observed in some analytical case studies. The proposed L-HNNs, due to having additional neurons in the final layer, have slightly more model parameters to learn compared to the HNNs. As such, the training cost of L-HNNs can be slightly higher than that of HNNs, but still negligible in practice for posterior parameter spaces having around 100 dimensions.

3.2. Sampling

HMC sampling using L-HNNs proceeds in similar fashion to traditional HMC. The key difference is that, given the initial conditions $\{\mathbf{q}(0), \mathbf{p}(0)\}$, the Hamiltonian dynamics is approximated using L-HNNs via Algorithm 2 rather than through direct integration. Once the new proposal $\{\mathbf{q}^*, \mathbf{p}^*\}$ is simulated, the Metropolis acceptance criterion is used to decide whether to accept this proposal, based on the conservation of the Hamiltonian:

(17)

$$\{\mathbf{q}^i, \mathbf{p}^i\} \leftarrow \{\mathbf{q}^*, \mathbf{p}^*\} \text{ with probability } \alpha \text{ where } \alpha = \min\{1, \exp(H(\{\mathbf{q}^{i-1}, \mathbf{p}^{i-1}\}) - H(\{\mathbf{q}^*, \mathbf{p}^*\}))\}$$

This procedure is summarized in Algorithm 3.

```

1: Hamiltonian:  $H = U(\mathbf{q}) + K(\mathbf{p})$ ; Samples:  $M$ ; Starting sample:  $(\mathbf{q}^0, \mathbf{p}^0)$ ; End time for trajectory:  $T$ ; Steps:  $N$ ; Dimensions:  $d$ 
2: for  $i = 1 : M$  do
3:    $\mathbf{p}(0) \sim \mathcal{N}(\mathbf{0}, I_d)$ 
4:    $\mathbf{q}(0) = \mathbf{q}^{i-1}$ 
5:   Compute  $\{\mathbf{q}^*, \mathbf{p}^*\} = \{\mathbf{q}(T), \mathbf{p}(T)\}$  with Algorithm 2
6:    $\alpha = \min\{1, \exp(H(\{\mathbf{q}^{i-1}, \mathbf{p}^{i-1}\}) - H(\{\mathbf{q}^*, \mathbf{p}^*\}))\}$ 
7:   With probability  $\alpha$ , set  $\{\mathbf{q}^i, \mathbf{p}^i\} \leftarrow \{\mathbf{q}^*, \mathbf{p}^*\}$ 
8: end for

```

Algorithm 3. L-HNNs in Hamiltonian Monte Carlo.

For a simple demonstration of using L-HNNs in HMC, we consider the following 1-D Gaussian mixture density, which has modes at $+1$ and -1 , both having a standard deviation of 0.35 :

(18)

$$f(q) \propto 0.5 \exp\left(\frac{(q-1)^2}{2 \times 0.35^2}\right) + 0.5 \exp\left(\frac{(q+1)^2}{2 \times 0.35^2}\right)$$

The L-HNNs have three hidden layers with 100 neurons each. To generate the training data, we simulated the Hamiltonian dynamics for 20 samples, each with $T = 20$ units. The number of leap-frog steps was 20 per unit, which means that $\Delta t = 0.05$. The initial condition for the first sample was set to zeros. The initial conditions for the rest of the samples were defined as follows: (1) position $q(0)$ was the final position of the previous sample, and (2) momentum $p(0)$ was drawn randomly from a standard Gaussian distribution. In total, L-HNNs training took 8,000 numerical gradient evaluations.

Fig. 2a compares the phase space $\{q, p\}$ between L-HNNs and numerical gradients for several initial momentum values $p(0)$ and identical initial position $q(0) = 0$. Note that the phase-space plots predicted using numerical gradients and L-HNNs compare satisfactorily. We further performed HMC using numerical gradients and L-HNNs to generate 5,000 samples from the distribution in Equation (18), with the first 1,000 considered burn-in. Each sample had an end time of $T = 5$ units, each unit being discretized into 20 steps (i.e., $\Delta t = 0.05$). The histogram plots compare well between traditional HMC (Fig. 2b) and HMC using HNNs (Fig. 2c). Furthermore, for this simple example, using L-HNNs for HMC requires only the 8000 numerical gradients of the target density from the training, while HMC required 500,000 numerical gradients for the same number of samples (a reduction of 98.4%).

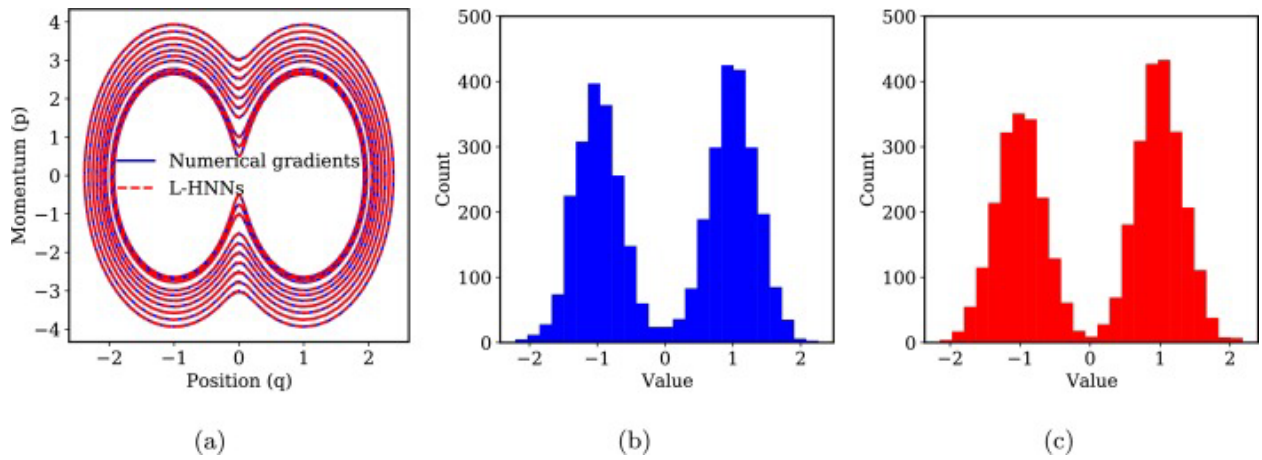


Fig. 2. (a) $\{q, p\}$ phase space plot corresponding to different energy levels. (b) and (c) Histogram of samples from a 1-D Gaussian mixture density using traditional HMC and L-HNNs in HMC, respectively. L-HNNs in HMC requires only 1.6% of the numerical gradients of target density required by traditional HMC.

3.3. Discussion of stationarity

MCMC samplers should be stationary with respect to the target density π . As discussed by Greydanus et al. [13], the Hamiltonian dynamics simulated by HNNs and L-HNNs are perfectly

time-reversible as a result of Liouville's theorem. This feature can also be seen in Fig. 3a, where two Hamiltonian trajectories for the density of the 1D Gaussian mixture are simulated in forward and reverse time using L-HNNs. Both trajectories have the same initial position in forward time, but different initial momenta. To simulate reverse-time trajectories, the final values of $\{q, p\}$ at the end time during forward integration were used as initial conditions. Backward time integration was then performed using L-HNNs.

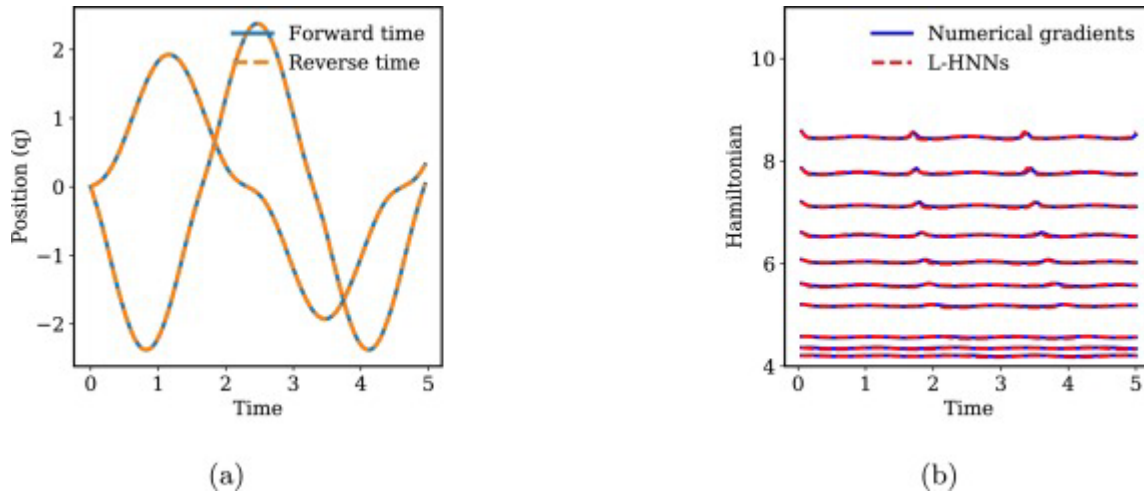


Fig. 3. (a) Time reversibility of the L-HNNs in predicting the Hamiltonian dynamics under two different sets of initial conditions, given a 1-D Gaussian mixture density. (b) Conservation of the Hamiltonian for different energy levels given a 1-D Gaussian mixture density.

By virtue of their loss function (Equation (13)), L-HNNs can also conserve the Hamiltonian in an approximate sense. The degree to which the Hamiltonian is conserved may depend on the target distribution of interest and the training data. For example, Fig. 3b presents the Hamiltonian evolution with time for the 1-D Gaussian mixture density, considering several initial conditions $\{q(0), p(0)\}$ using $\Delta t = 0.05$. Overall, the Hamiltonian values are seen to be approximately constant over time. For comparison, the Hamiltonian values obtained using traditional numerical gradients are also presented and are also approximately constant with time. Owing to some numerical errors in Hamiltonian conservation, and in line with the traditional

HMC procedure, the use of a Metropolis acceptance criterion in Equation (17) is important to ensure stationarity when using HNNs in HMC. The Metropolis acceptance criterion leaves the target density π invariant, as discussed in Roberts and Rosenthal [24].

4. No-U-Turn Sampling (NUTS) with L-HNNs

Traditional HMC requires that the end time T be specified. For all but simple problems, specifying T is not only challenging, but also has considerable impact on HMC sampler performance diagnostics (e.g., the serial autocorrelations between the samples). Hoffman and Gelman [5] introduced the NUTS algorithm to automatically decide T for each sample in HMC. In summary, NUTS works by building the set of candidate $\{\mathbf{q}, \mathbf{p}\}$ states \mathcal{B} , using a binary tree. At each iteration, the tree length is doubled by integrating randomly in either the forward or reverse direction. Of the states \mathcal{B} , a subset of states \mathcal{C} is selected so as not to violate the detailed balance condition. A slice variable u is often used in NUTS to facilitate the selection of subset \mathcal{C} . The doubling procedure for building the binary tree is stopped when a u-turn is made, as defined by the criterion:

(19)

$$(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^- < 0 \text{ or } (\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^+ < 0$$

where $\{\mathbf{q}^+, \mathbf{p}^+\}$ and $\{\mathbf{q}^-, \mathbf{p}^-\}$ are, respectively, the leftmost and rightmost states of a subtree in the binary tree. The doubling procedure may also be stopped when the integration error while simulating the Hamiltonian dynamics exceeds a threshold, as defined by:

(20)

$$\varepsilon \equiv H(\mathbf{q}, \mathbf{p}) + \ln u > \Delta_{max}$$

where $\{\mathbf{q}, \mathbf{p}\}$ is the current candidate state and Δ_{max} is a threshold recommended by Hoffman and Gelman to be set to 1,000 [5]. From the subset of states \mathcal{C} , the next state is selected randomly

with a uniform probability. This version of the NUTS algorithm is called naive NUTS, since it requires storing candidate states $O(2^j)$ (j is the number of doubling iterations in the binary tree), leading to large memory requirements. Naive NUTS also requires simulating the full tree—even when the stopping criteria are satisfied in the middle of the doubling iteration—leading to an expenditure of computational resources. Hoffman and Gelman [5] introduced an efficient version of NUTS to alleviate these problems by using a sophisticated transition kernel to build the binary tree, which only requires storing $O(j)$ candidate states in memory, but generating additional $O(2^j)$ random numbers (an insignificant cost). In this paper, the efficient version of NUTS is used.

4.1. L-HNNs in NUTS

Here, we use L-HNNs in place of the leapfrog integrator in the efficient NUTS algorithm to build the binary tree. As a simple demonstration, consider the 1-D Gaussian mixture density discussed in Section 3.2. The details of the L-HNN training remain the same. Using L-HNNs in efficient NUTS, 5,000 samples were simulated, the first 1,000 samples were considered burn-in and $\Delta t = 0.05$. Table 1 compares the Effective Sample Size (ESS) of traditional NUTS with that of L-HNNs in NUTS, where the ESS is normalized by the total number of gradient evaluations (either from the HMC or L-HNN training). Also presented are the normalized ESS values for traditional HMC and L-HNNs in HMC. Overall, the ESS per gradient of the target density is considerably better for L-HNNs in HMC and L-HNNs in NUTS than for traditional HMC and traditional NUTS. Although the ESS per gradient of the target density is similar between L-HNNs in HMC and L-HNNs in NUTS, to achieve a similar value of the ESS, while L-HNNs in HMC required 500,000 neural network gradients, L-HNNs in NUTS required only around 91,000. This difference demonstrates the efficacy of NUTS over HMC with L-HNNs for this simple problem. For more complex high-dimensional problems, the efficacy of L-HNNs in NUTS is expected to be even, since it is difficult to know the optimal end time for the Hamiltonian trajectory a priori.

Table 1. Comparison of the performance of traditional HMC, traditional NUTS, HNN-HMC, and HNN-NUTS, considering a 1-D Gaussian mixture density. ESS per gradient is used as the performance metric.

ESS per gradient of target density	
L-HNNs in HMC	$4.59E-3$
Traditional HMC	$8.42E-5$
L-HNNs in NUTS	$4.83E-3$
Traditional NUTS	$4.4E-4$

Hoffman and Gelman [5] also proposed a dual averaging scheme to automatically tune the step size Δt given a target acceptance ratio. While this automatic step-size determination will be interesting to explore for L-HNNs in NUTS, in this paper, we fix the step size to the same small value for both L-HNNs in NUTS and traditional NUTS. This helps facilitate a fair comparison between these algorithms, and step-size tuning will be covered in a future study.

4.2. Online error monitoring

While direct use of L-HNNs in NUTS works for a simple 1-D Gaussian mixture density, for more complex distributions care must be taken to avoid large integration errors when building the binary tree. Large integration errors can arise when sampling in the tails of a distribution, since limited L-HNN training data may be available in these regions of low probability density. When building the binary tree, when such a region of low probability density is reached and integration errors are large, NUTS prematurely terminates the tree-building procedure. As a result, the next sample will likely be in the same vicinity as the previous one. This cycle of generating a new sample near the previous sample continues, leading to degenerate clusters of samples in regions of low probability density. Fig. 4 presents an example of this sampling

degeneracy when using L-HNNs in NUTS for a 3-D Rosenbrock density with heavy tails [25]. Note that the sampler is unable to rapidly move to regions of high probability density after visiting a region of low probability density, as a result of large integration errors when building the binary tree. This sort of sampling degeneracy may be less concerning when using L-HNNs in HMC, due to the Metropolis acceptance criterion in Equation (17), which rejects a proposed sample when the difference in Hamiltonian values is very large. At the same time, the Metropolis criterion can also prevent L-HNNs in HMC from efficiently exploring the uncertainty space.

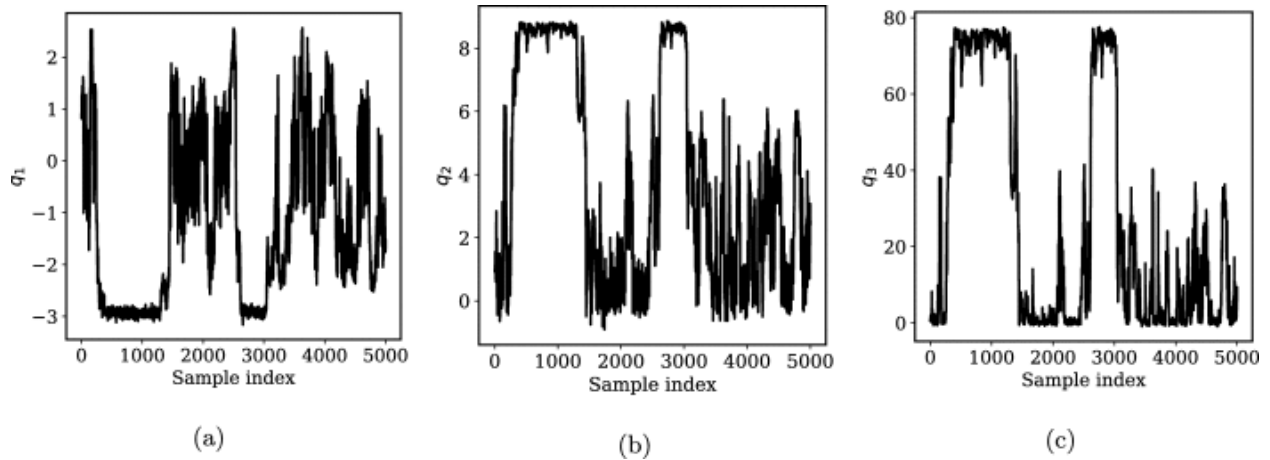


Fig. 4. Degeneracy in sampling when using L-HNNs in NUTS for a 3-D Rosenbrock density. These trace plots correspond to the dimensions (a) q_1 , (b) q_2 , and (c) q_3 .

To alleviate the above-mentioned issues caused by large integration errors, we propose an online error monitoring scheme for L-HNNs in NUTS. This error monitoring scheme relies on the stopping criterion in Equation (20), as introduced by Hoffman and Gelman [5]. This criterion monitors the integration errors when simulating the Hamiltonian trajectories and terminates the tree building procedure when they are large. In theory, if the integration errors are nonexistent, then the quantity ε in Equation (20) will always be less than zero. But for practical applications with non-negligible integration errors, the error metric $\varepsilon \equiv H(\mathbf{q}, \mathbf{p}) + \ln u$ can be greater than zero during a doubling iteration. We define a threshold Δ_{max}^{hnn} for the error metric ε using L-HNNs. When this quantity exceeds Δ_{max}^{hnn} , the L-HNN is abandoned and standard leapfrog integration using numerical gradients of the target density takes over. We also introduce a

parameter N_{lf} , which sets the number of leapfrog samples to take before reintroducing L-HNN. N_{lf} can be viewed as the number of “cooldown” samples needed to return the sampler to regions where the L-HNN is sufficiently trained. In the traditional version of efficient NUTS, Hoffman and Gelman [5] recommend that the error threshold for leapfrog integration be set to a large value (e.g., $\Delta_{max}^{lf} = 1,000$). When setting the threshold for L-HNNs, we recommend a more conservative value of around $\Delta_{max}^{hnn}=10$ to prevent sampling degeneracy in regions of low probability density. N_{lf} can be set to a low value of 5–20 samples to essentially return the sampler to regions of high probability density. These settings will contribute to the sampler exploring the uncertainty space well, while ensuring that L-HNNs integration errors in certain regions of the space do not cause sampling degeneracy.

Algorithm 4, Algorithm 5 present the pseudocode to use L-HNNs in efficient NUTS with online error monitoring. These algorithms differ somewhat from the traditional version of efficient NUTS proposed by Hoffman and Gelman [5]. The most notable difference is the use of L-HNNs to predict Hamiltonian dynamics in the **BuildTree(.)** function of Algorithm 5. In addition, the **BuildTree(.)** function takes and returns an indicator variable $\mathbb{1}_{lf}$ to determine whether L-HNNs or the leapfrog scheme with numerical gradients of the target density should be used to simulate Hamiltonian dynamics. The main loop (Algorithm 4) monitors the number of samples to which the leapfrog scheme with numerical gradients of the target density is applied, and facilitates the reversion to L-HNNs once this number equals N_{lf} . There are two error thresholds (i.e., Δ_{max}^{lf} and Δ_{max}^{hnn}) in the pseudocode. Δ_{max}^{lf} is the error threshold for the leapfrog scheme using numerical gradients of the target density. Δ_{max}^{hnn} is the error threshold for L-HNNs with $\Delta_{max}^{hnn} \ll \Delta_{max}^{lf}$.

```

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

```

Algorithm 4. L-HNNs in efficient NUTS with online error monitoring (main loop).

```

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

```

Algorithm 5. L-HNNs in efficient NUTS with online error monitoring (build tree function).

5. Sampling from a 3-D Rosenbrock density

A 3-D Rosenbrock density [25] is considered to demonstrate the performance differences between HNNs and L-HNNs, and to investigate the performance of L-HNNs considering

different amounts of training data. A 3-D Rosenbrock density makes for a particularly interesting case study, due to its heavy tails. The density function is defined by [25]:

(21)

$$f(\mathbf{q}) \propto \exp\left(-\sum_{i=1}^{N-1} [100(q_{i+1} - q_i^2)^2 + (1 - q_i)^2]/20\right)$$

We trained HNNs and L-HNNs with the same training data, which contained 40 HMC samples, each with an end time of $T = 40$ using a $\Delta t = 0.025$. Both HNNs and L-HNNs had three hidden layers with 100 neurons each. A total of 100,000 steps at a learning rate of $5E - 4$ were used to train HNNs and L-HNNs. A sine activation function was used, according to the recommendations of Mattheakis et al. [15]. Efficient NUTS with online error monitoring (i.e., Algorithm 4, Algorithm 5) was employed to simulate 35,000 samples of the target density. HNNs and L-HNNs were independently used to simulate Hamiltonian dynamics within the **BuildTree(.)** function. Δ_{max}^{hnn} was set to 10—and when exceeded, the leapfrog scheme with numerical gradients of the target density was set to take over—and N_{lf} was set to 20.

Figs. 5a and 5b present trace plots for the dimension q_1 , obtained using HNNs and L-HNNs, respectively. Note that, whether using HNNs or L-HNNs, efficient NUTS with online error monitoring mitigates degeneracy in sampling, compared to Fig. 4a. Perhaps more important are the integration errors when predicting the Hamiltonian dynamics. Figs. 6a and 6b present these errors—defined as $[\varepsilon \equiv H(\mathbf{q}', \mathbf{p}') + \ln u]$ in Algorithm 5—as obtained using HNNs and L-HNNs, respectively. The errors when using HNNs are substantially larger across the sample indices compared to when using L-HNNs. Large integration errors lead to an increase in the number of samples for which the leapfrog scheme with numerical gradients of the target density is called. When using HNNs, the number of samples for which traditional leapfrog is used is 4,706; using L-HNNs, this number is reduced to only 180. In total, HNNs and L-HNNs require an additional 1,437,900 and 63,598 numerical gradients of the target density, respectively (above what is required to generate the training data), a substantial difference.

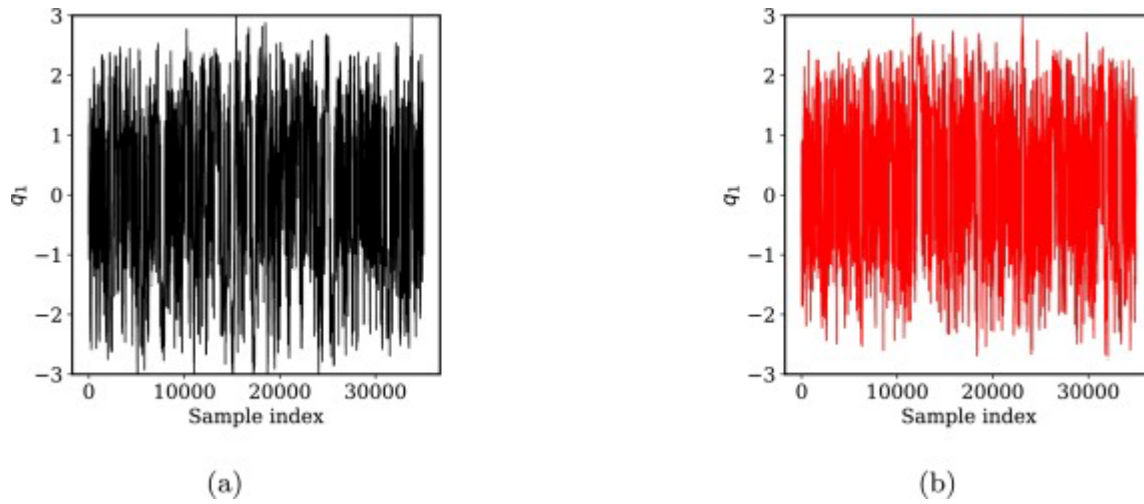


Fig. 5. Trace plots of dimension q_1 for a 3-D Rosenbrock density simulated via efficient NUTS with online error monitoring using (a) HNNs and (b) L-HNNs. Note that the sort of degeneracy presented in Fig. 4 is absent due to the online error monitoring scheme.

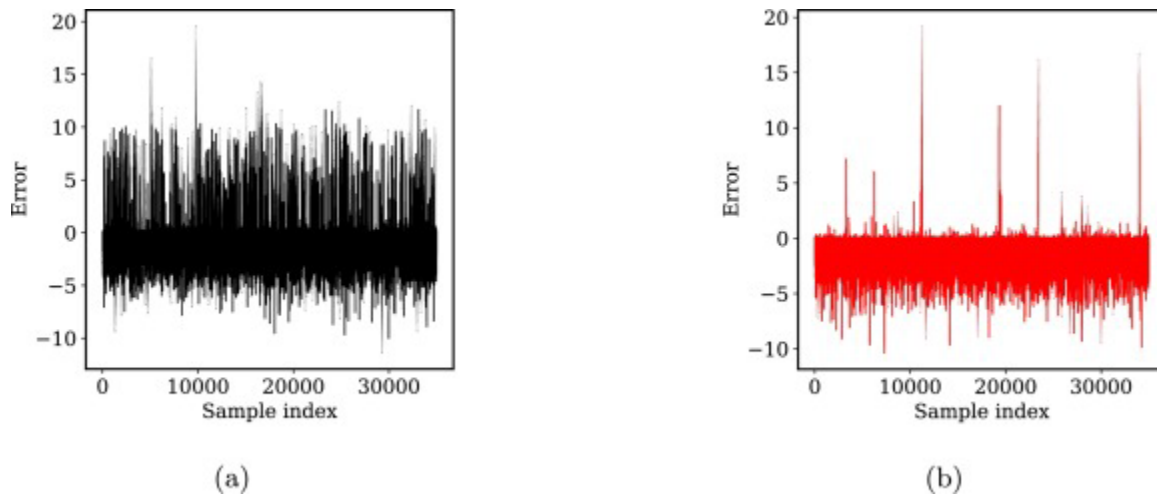


Fig. 6. L-HNNs error metric, as defined by the quantity $[\varepsilon \equiv H(\mathbf{q}', \mathbf{p}') + \ln u]$ in Algorithm 5, while generating samples from a 3-D Rosenbrock density using (a) HNNs and (b) L-HNNs in

efficient NUTS with online error monitoring. These error values are considerably lower when using L-HNNs than HNNs.

To investigate the influence of the size of the training data set, we consider three training datasets for L-HNNs. All three sets include $M_t = 40$ samples from the 3-D Rosenbrock density simulated using traditional HMC, but with different end times $T = 100, 150,$ and 250 units — and $\Delta t = 0.025$. We simulate 125,000 samples from the 3-D Rosenbrock density, with the first 5,000 samples considered burn-in. For efficient NUTS with online error monitoring, we set $\Delta_{max}^{hnn} = 10$ and $N_{lf} = 20$. Fig. 7 presents empirical cumulative distribution function (eCDF) plots for the three dimensions. In this figure, LHNN-NUTS 1, LHNN-NUTS 2, and LHNN-NUTS 3 correspond to end times of $T = 100, 150,$ and 250 units, respectively. Note that irrespective of the amount of training data, in general, the eCDFs of LHNN-NUTS with online error monitoring closely match the eCDFs simulated using traditional NUTS with numerical gradients of the target density. Furthermore, LHNN-NUTS 2 and LHNN-NUTS 3, which were trained on HMC samples with longer Hamiltonian trajectories, almost exactly match the reference eCDFs, compared to LHNN-NUTS 1, which shows slight deviations in dimensions q_2 (Fig. 7b) and q_3 (Fig. 7c).

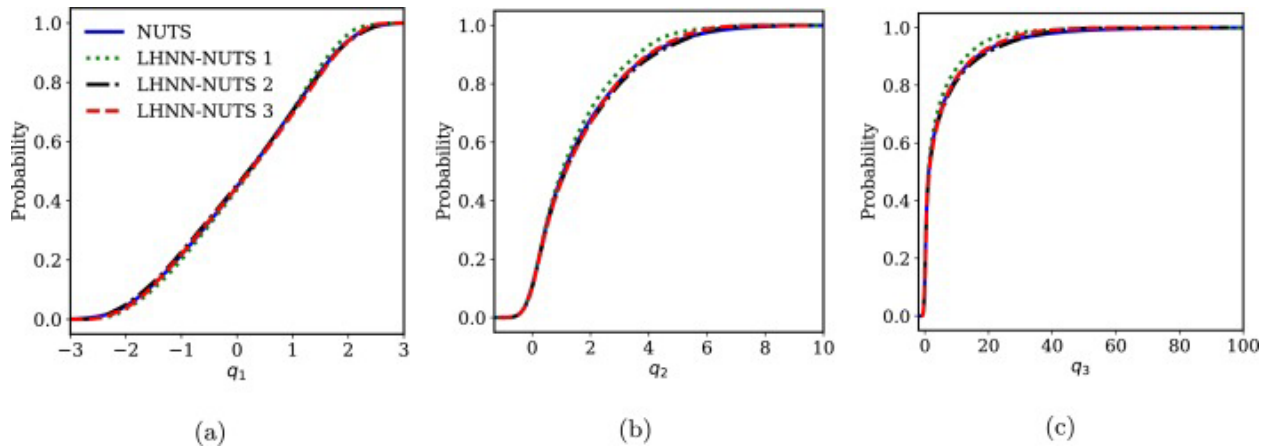


Fig. 7. Empirical CDF plots of samples from a 3-D Rosenbrock density simulated via NUTS and L-HNNs in NUTS with online error monitoring, using different training datasets. eCDF plots are

presented for the dimensions (a) q_1 , (b) q_2 , and (c) q_3 . Details of the L-HNN training data are given in Table 2.

Table 2 compares the performance of traditional NUTS with LHNN-NUTS with online error monitoring in terms of ESS, the number of gradients of the target density and the average ESS in all dimensions per gradient of the target density. In summary, all LHNN-NUTS results show a significant improvement over the traditional NUTS method in terms of required gradient evaluations, while accurately matching target eCDFs (Fig. 7) and producing samples with comparable ESS values. Even for short training trajectories with $T = 100$, the total number of gradient evaluations is reduced by 83% and the average ESS per gradient evaluation increases by 5 times. For longer training trajectories, the number of gradient evaluations reduces from approximately 16 million to only approximately 600,000 while the ESS per gradient evaluation improves by more than an order of magnitude relative to traditional NUTS. However, it should be noted that there is no noticeable improvement in performance by increasing T from 150 to 250 for training data generation, which implies that there is a critical length T for training data.

Table 2. Performance comparison between traditional NUTS and HNN-NUTS with different training data characteristics, considering a 3-D degenerate Rosenbrock density.

	ESS	# gradients of target density	Avg. ESS per gradient of target density
NUTS [5]	(1396.59, 2338.99, 2411.40)	15,899,976	0.000128
$M_t = 40; T = 100$ [LHNN-NUTS 1]	(1339.13, 1665.76, 1648.85)	2,711,240 Training: 160,000 Evaluation: 2,551,240	0.000572

	ESS	# gradients of target density	Avg. ESS per gradient of target density
$M_t = 40; T = 150$ [LHNN-NUTS 2]	(1217.63, 1710.97, 1654.11)	607,298 Training: 240,000 Evaluation: 367,298	0.00251
$M_t = 40; T = 250$ [LHNN-NUTS 3]	(1445.03, 1729.98, 1387.91)	642,414 Training: 400,000 Evaluation: 242,414	0.00236

- **Notations.** M_t : Number of training samples; T : End time for trajectory

- Avg. ESS represents the ESS values averaged across the three dimensions.

- Evaluation in the table implies the use of L-HNNs in NUTS with online error monitoring

For a small value of M_t , as expected, L-HNNs in NUTS would require more numerical gradients of the target density during the evaluation because the training data do not adequately capture the randomization of the momenta. But for a large value of M_t , although the additional numerical gradients of the target density may become small, the upfront cost of generating the training data may be high. We found from the 3-D Rosenbrock example that fixing $M_t = 40$ provides a balance between upfront costs to generate training data and additional numerical gradients required during evaluation. Therefore, we use $M_t = 40$ for the case study examples that will be discussed in Section 6.

6. Analytical case studies

We considered three analytical case studies to demonstrate LHNN-NUTS with online error monitoring and compared its performance with traditional NUTS. Table 3 summarizes the main performance metrics. In the following sections, the case studies are defined along with the

training details for L-HNNs, the results from Table 3 are discussed and plots of the samples from the target densities are presented.

Table 3. Summary of the performance of LHNN-NUTS with online error monitoring across the different analytical case studies.

	ESS	# gradients of target density	Avg. ESS per gradient of target density
2-D Neal's funnel			
LHNN-NUTS	(1019.87,666.73)	3,596,688 Training: 400,000 Evaluation: 3,196,688	0.000234
NUTS	(1012.45,718.07)	16,639,072	0.000052
5-D ill-conditioned Gaussian			
LHNN-NUTS	(20000,17741.21, 13906.42,8382.5, 2763.38)	408,800 Training: 400,000 Evaluation: 8,800	0.0307
NUTS	(20000,17877.90, 14645.54,7931.68, 2657.19)	11,817,469	0.00106
10-D degenerate Rosenbrock			

	ESS	# gradients of target density	Avg. ESS per gradient of target density
LHNN-NUTS	(15535.69,28217.40, 28965.00,27280.67, 21268.00,13532.28, 8418.80,5731.09, 3299.92,2045.02)	418,936 Training: 400,000 Evaluation: 18,936	0.0368
NUTS	(15771.46,27125.5, 27825.9,25107.02, 21963.9,13553.21, 8548.44,5631.39, 2921.78,1698.69)	7,015,025	0.00219

- Evaluation in the table implies the use of L-HNNs in NUTS with online error monitoring.

We considered the following three analytical case studies. Corresponding results are presented here. For all cases, except where otherwise noted, L-HNNs were trained using $M_t = 40$ HMC samples each, with an end time of $T = 250$ units and a step size of $\Delta t = 0.025$. The architecture of the L-HNNs was composed of three hidden layers, each with 100 neurons, and a sine activation function [15]. For training, a total of 100,000 steps were used at a learning rate of $5E - 4$ and Δ_{max}^{hnn} and N_{lf} in LHNN-NUTS with online error monitoring set to 10 and 20, respectively.

6.1. 2-D Neal's funnel density

The following 2-D Neal's funnel density was considered:

$$f(\mathbf{q}) \propto \begin{cases} q_1 = \mathcal{N}(0,3) \\ q_2 = \mathcal{N}(0, \exp^{q_1}) \end{cases} \quad (22)$$

This density can be complex to sample from because of its extreme curvature. L-HNNs had four inputs (two positions and two momenta) and predicted two latent variables, whose sum was treated as the Hamiltonian. A total of 25,000 samples from the target density were simulated, with the first 5,000 samples treated as burn-in.

In this case, the online error monitoring scheme was extensively used as seen in Table 3 due to the difficulty in training L-HNNs for this complex problem. However, we see a reduction of more than 13 million gradient evaluations (traditional NUTS requires a huge number of gradient evaluations) and an almost an order of magnitude improvement in the ESS per gradient evaluation. Fig. 8a presents a scatter plot of the samples generated using LHNN-NUTS and traditional NUTS. Figs. 8b and 8c compare the eCDF plots for the two dimensions. Note that the plots generated with LHNN-NUTS compare well with those obtained with traditional NUTS.

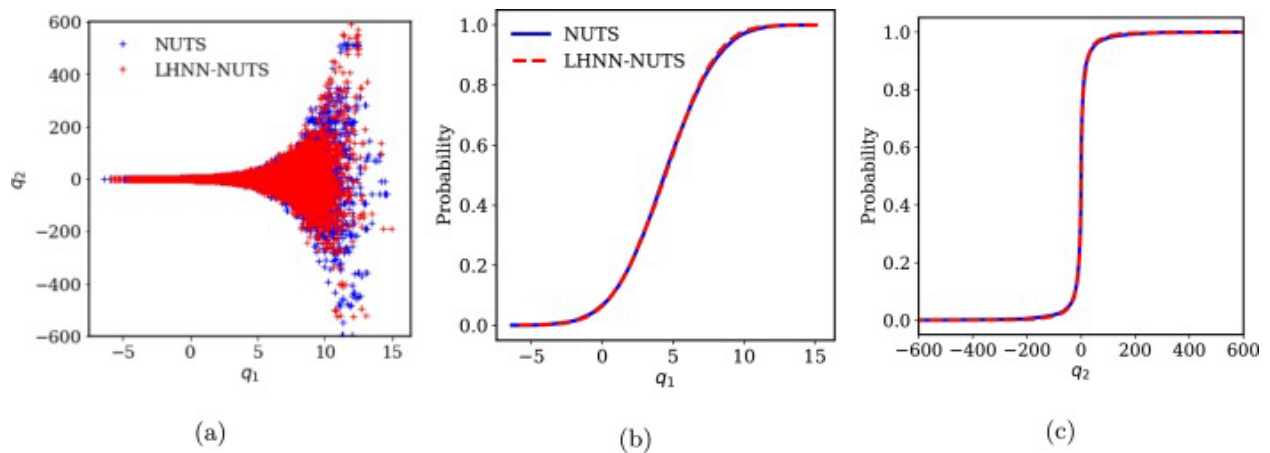


Fig. 8. (a) Scatter plot comparison of samples generated from a 2-D Neal's funnel density, using NUTS and L-HNNs in NUTS with online error monitoring. Comparison of the eCDF plots for the dimensions (b) q_1 and (c) q_2 .

6.2. 5-D ill-conditioned Gaussian

A 5-D ill-conditioned Gaussian distribution from Levy et al. [10] with a zero mean vector and a diagonal covariance matrix was considered. The diagonal elements of the covariance matrix scale in log-linear fashion, as presented below:

$$\text{diag}(\mathbf{\Sigma}) = [0.01, 0.1, 1.0, 10.0, 100.0]$$

In this example, the L-HNNs had 10 inputs (five position variables and five momenta) and predicted five latent variables, whose sum was treated as Hamiltonian, as discussed in Section 3.1. A total of 25,000 samples from the target density were simulated, with the first 5,000 samples treated as burn-in.

From Table 3, we see that the use of LHNN-NUTS reduces the number of gradient evaluations by a factor of nearly 30, while improving the ESS per gradient evaluation by more than an order of magnitude. Moreover, as we can see from Fig. 9, which shows scatter plots of the 5-D ill-conditioned Gaussian samples using LHNN-NUTS and traditional NUTS, LHNN-NUTS with online error monitoring satisfactorily samples from the target density. This is further verified by the ESS values in Table 3, which are consistent between the two methods.

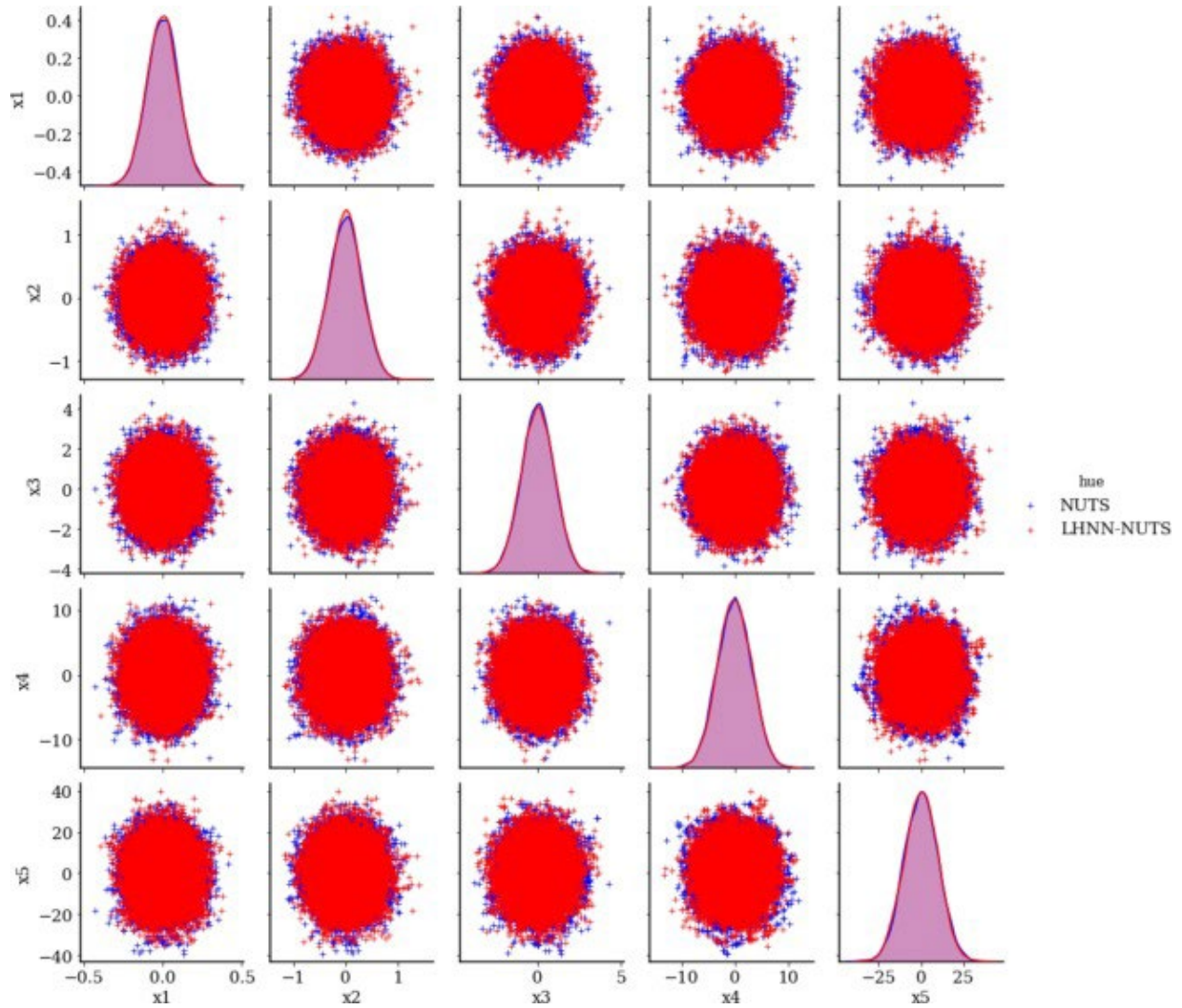


Fig. 9. Scatter plot comparison between NUTS and LHNN-NUTS with online error monitoring for a 5-D ill-conditioned Gaussian distribution. (Blue dots: traditional NUTS; Red dots: LHNN-NUTS with online error monitoring). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

6.3. 10-D degenerate Rosenbrock density

Here, a 10-D version of the Rosenbrock density in Equation (21) was considered. L-HNNs had 20 inputs (10 position variables and 10 momenta) and predicted 10 latent variables, whose sum was treated as the Hamiltonian. Approximately 125,000 samples were simulated from the target density, with the first 5,000 samples treated as burn-in.

From Table 3, we again see a drastic reduction in the number of gradient evaluations (approximately a factor of 17) and more than an order of magnitude reduction in ESS per gradient evaluation. Fig. 10 presents scatter plots of the samples generated with LHNN-NUTS and the traditional NUTS. Although for dimensions q_9 and q_{10} , traditional NUTS included a few more samples in the tails than LHNN-NUTS, the scatter plots overall match satisfactorily. Furthermore, the eCDF plots for dimensions q_9 and q_{10} from LHNN-NUTS closely matched the reference eCDFs.

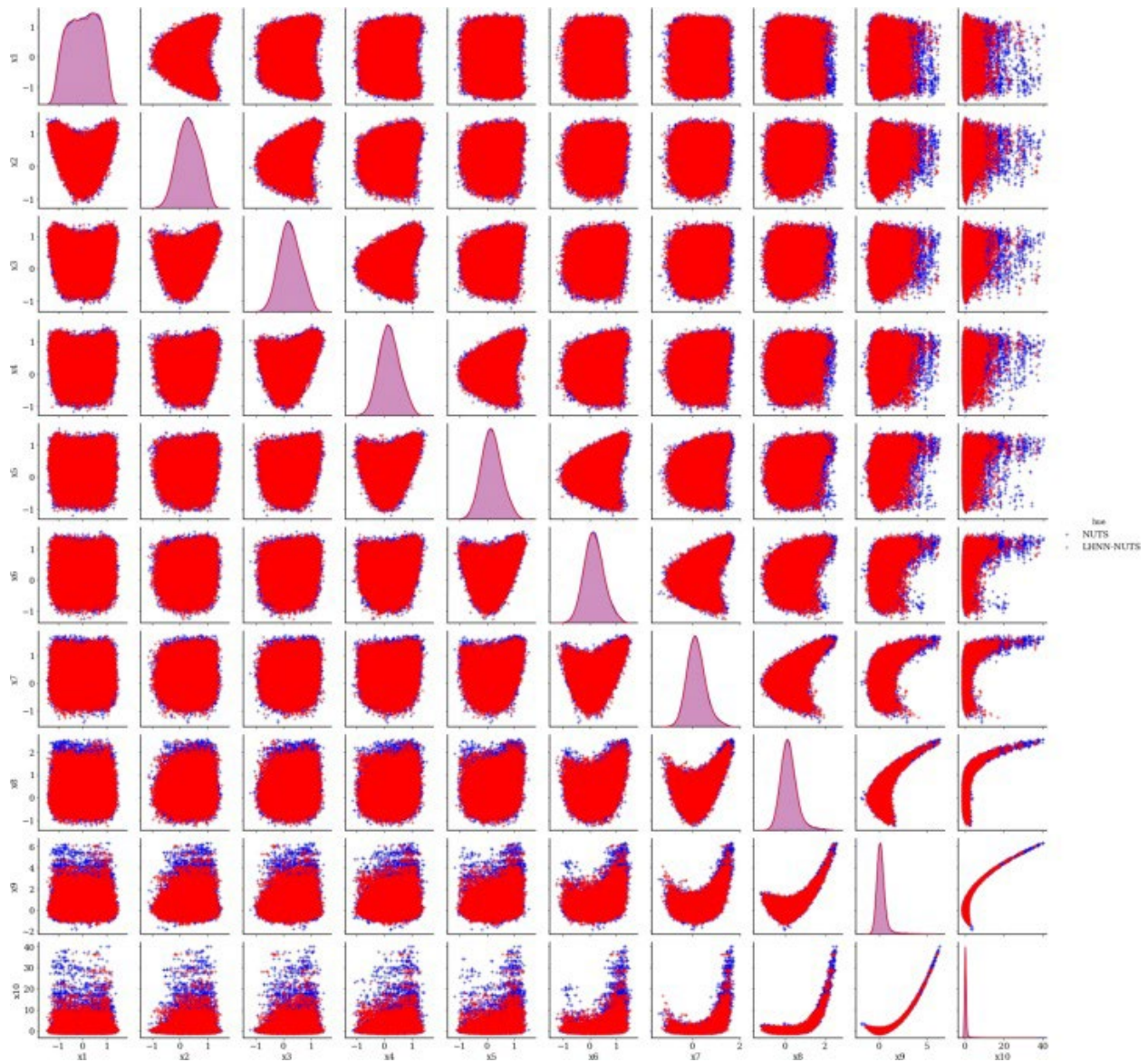


Fig. 10. Scatter plot comparison between NUTS and LHNN-NUTS with online error monitoring, considering a 10-D degenerate Rosenbrock distribution. (Blue dots: traditional NUTS; Red dots: LHNN-NUTS with online error monitoring).

7. Computational case studies

We considered two computational case studies to demonstrate LHNN-NUTS with online error monitoring and to compare its performance with traditional NUTS. These case studies are the Allen-Cahn stochastic partial differential equation and an elliptic partial differential equation with 25 and 50 inference parameters, respectively. Table 4 summarizes the main performance metrics. In the following sections, the case studies are defined along with the training details for L-HNNs, and the results of Table 4 are discussed.

Table 4. Summary of the performance of LHNN-NUTS with online error monitoring across the different computational case studies.

Empty Cell	ESS	# gradients of target density	Avg. ESS per gradient of target density
Allen-Cahn stochastic PDE (25 inference parameters)			
LHNN-NUTS	414.5	97,904 Training: 10,000 Evaluation: 87,904	0.00423
NUTS	217.88	681,905	0.00032
Elliptic PDE (50 inference parameters)			
LHNN-NUTS	3229.9	217,824 Training: 64,000 Evaluation: 153,824	0.015

Empty Cell	ESS	# gradients of target density	Avg. ESS per gradient of target density
NUTS	4048.98	621,713	0.0065

- The average ESS across all dimensions is reported.

- Evaluation in the table implies the use of L-HNNs in NUTS with online error monitoring.

7.1. Allen-Cahn stochastic partial differential equation with 25 inference parameters

The Allen-Cahn stochastic partial differential equation is widely used for multiphase flows [26].

It is defined as:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - V'(u) + \sqrt{2}\eta \text{ where } V(u) = (1 - u^2)^2 \quad (24)$$

with $\pi(\mathbf{u})$ being its stationary distribution which is further defined as:

$$\pi(\mathbf{u}) \propto \exp \left(- \int_0^1 \left(\frac{1}{2} \left(\frac{\partial u}{\partial x} \right)^2 + V(u(x)) \right) dx \right) \quad (25)$$

Discretizing $\pi(\mathbf{u})$ gives:

(26)

$$\pi(\mathbf{u}) = \exp \left(- \sum_{i=0}^{d-1} \left(\frac{1}{2\Delta x} (u(i\Delta x + \Delta x) - u(i\Delta x))^2 + \frac{\Delta x}{2} (V(u(i\Delta x + \Delta x)) - V(u(i\Delta x))) \right) \right)$$

where $d = 25$ is the number of discretized points. We wish to sample from this 25-dimensional stationary distribution, which is known to be a particularly difficult task for MCMC methods to handle [27]. We trained L-HNNs by first generating HMC trajectories, which required 10,000 gradient computations of the target density. Next, we sampled 5,000 samples from the target density using L-HNNs in NUTS with online error monitoring. We also sampled the target density using NUTS to provide reference results.

Fig. 11 presents results for four representative dimensions of the 25 dimensions, namely 1, 12, 13, and 25. We observe that the LHNN-NUTS and traditional NUTS results compare favorably with each other. Moreover, for nearby dimensions (i.e., 12 and 13), the correlations are strong, whereas the correlation diminishes as the separation grows. This correlation structure is characteristic of the time-dependent stochastic evolution of the Allen-Cahn equation. From Table 4, we also see that the ESS obtained using LHNN-NUTS compares well with that of traditional NUTS, which means that both methods sample the distribution in a similar way. But more importantly, the LHNN-NUTS method does so with far fewer gradient evaluations, such that the ESS per gradient for LHNN-NUTS (which includes both training and evaluation gradients during error monitoring) is more than an order of magnitude larger than traditional NUTS.

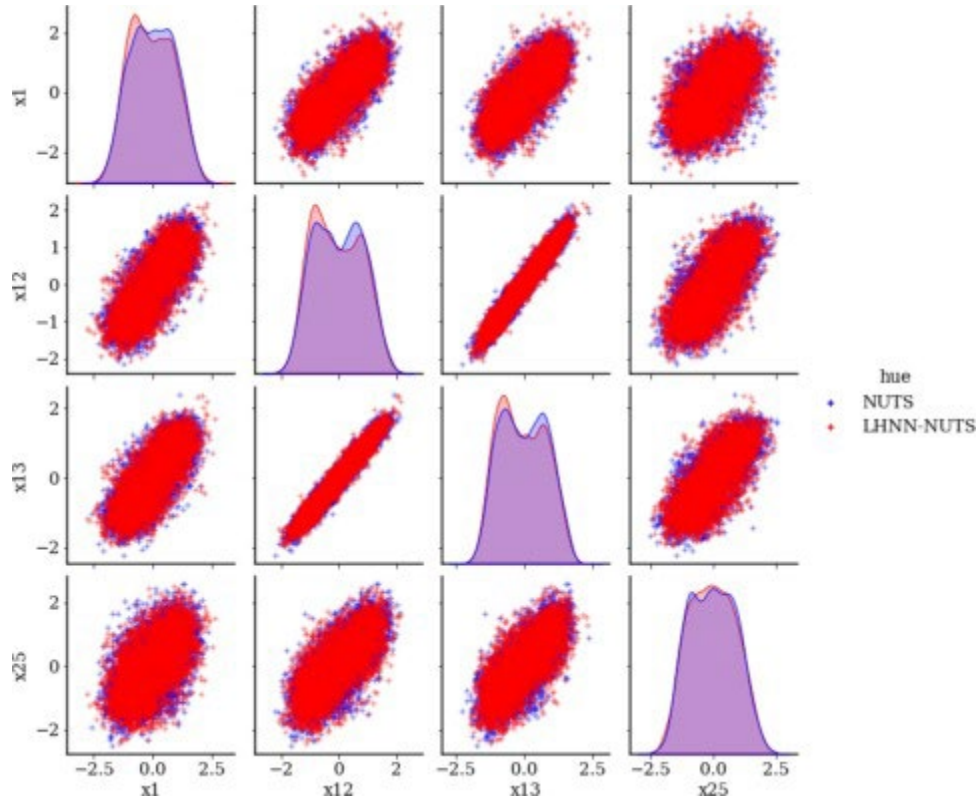


Fig. 11. Sampling results for four of the 25 dimensions, namely, 1, 12, 13, and 25, of the Allen-Cahn equation. It is observed that the LHNN-NUTS and traditional NUTS results compare well with each other.

7.2. An elliptic partial differential equation with 50 inference parameters

We consider an elliptic partial differential equation (PDE) in 2-D given by:

(27)

$$\frac{\partial}{\partial x} \left(k(x, y) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(k(x, y) \frac{\partial u}{\partial y} \right) = f(x, y)$$

where we specify $u(x, y) = \sin 2x + \sin 2y$, treat $k(x, y)$ as an unknown to be estimated using Bayesian inference, and observe $f(x, y)$. To generate the “experimental data”, $k(x, y)$ is set to $x + y$, the $f(x, y)$ field is numerically computed, and a Gaussian noise of zero mean unit variance is added to the $f(x, y)$ field. We record the noisy field $f(x, y)$ at 50 random “sensor”

locations in the spatial domain, as shown in Fig. 12a and aim to infer $k(x, y)$ at these 50 sensor locations so that the error in the observed $f(x, y)$ plus noise values is minimized.

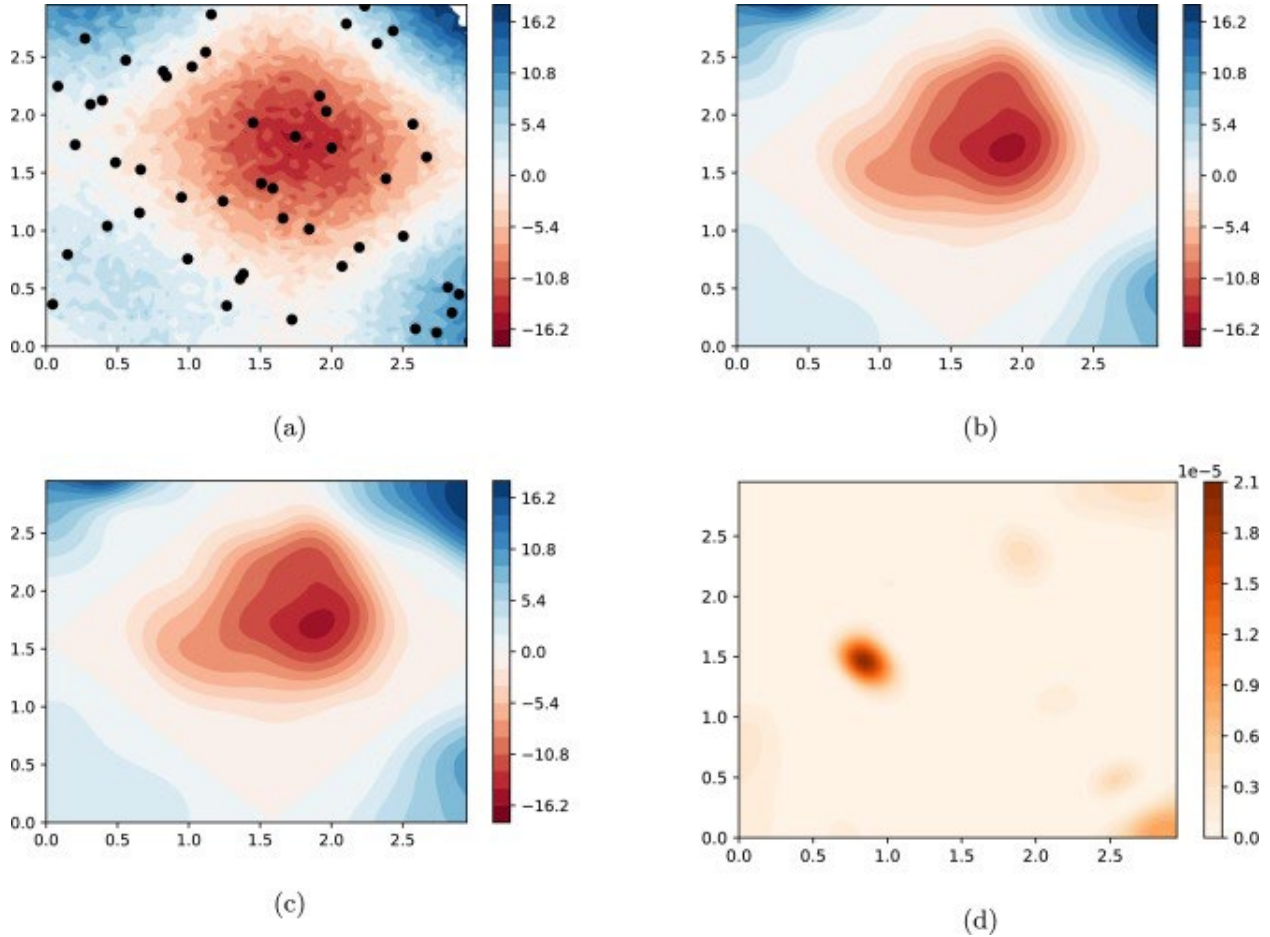


Fig. 12. (a) The $f(x,y)$ field corrupted by a Gaussian noise and the points at which $f(x,y)$ is recorded for treating as experimental data. (b) and (c) The estimated mean $f(x,y)$ field after performing Bayesian inference using LHNN-NUTS and NUTS, respectively. (d) Mean squared error of the mean $f(x,y)$ field between LHNN-NUTS and NUTS.

The L-HNNs are trained from HMC samples requiring a total of 64,000 gradient estimations of the elliptic PDE. Next, we draw 5,000 samples from the 50-dimensional target joint density of $k(x, y)$ at the sensor locations using L-HNNs in NUTS with online error monitoring and computed the corresponding $f(x, y)$. We also sampled the target density using NUTS to provide the reference results. We then calculate the mean field $f(x, y)$ from the 5,000 samples and plot

them in Figs. 12b and 12c for LHNN-NUTS and NUTS, respectively. Note that the full mean $f(x, y)$ field is estimated using spatial interpolation from the mean values at the 50 sensor locations. We clearly observe that the calculated mean field $f(x, y)$ matches closely between LHNN-NUTS and NUTS as also indicated by the mean squared error field in Fig. 12d, where $f(x, y)$ from NUTS is considered the “truth”. Furthermore, as seen in Table 4, LHNN-NUTS requires four times fewer gradient computations of the PDE and improves the ESS per gradient by more than a factor of 2 compared to NUTS.

8. Summary and conclusions

Neural networks that learn Hamiltonian systems (e.g., HNNs) have received a great deal of interest. We proposed the use of HNNs to solve Bayesian inference problems through the HMC framework. HNNs have important properties (e.g., time reversibility and Hamiltonian conservation) that make them amenable for use in HMC sampling. Furthermore, once trained, HNNs do not require numerical gradients of the target density—which can be computationally expensive—during sampling. To increase the expressivity of HNNs, we proposed HNNs with latent outputs (L-HNNs) and demonstrated that L-HNNs enable integration error reductions in predicting the Hamiltonian dynamics when compared to HNNs for a 3-D Rosenbrock density function. Although HMC is a state-of-the-art sampler for Bayesian inference, it requires the specification of an end time of the trajectory for each sample, and this can considerably influence the inference quality. NUTS automatically decides this end time, while also ensuring stationarity and that the next proposed sample is farther away from the previous one (i.e., lower correlations between samples). We further proposed using L-HNNs in NUTS with an online error monitoring scheme that reverts to the standard leapfrog integration for a few samples whenever the L-HNNs integration errors are high. L-HNNs integration errors can be high in regions of the uncertainty space that feature fewer training data (e.g., the tails). This proposed online error monitoring scheme ensures that L-HNNs rapidly move to regions of high probability density after visiting a region of low probability density, and prevents degeneracy of the sampling scheme.

L-HNNs in NUTS with online error monitoring was demonstrated for several example cases involving complex probability densities and computational applications with dimensions ranging from 5 to 50. The performance of L-HNNs in NUTS was compared with that of traditional

NUTS, which is based on numerical gradients of the target density. L-HNNs in NUTS satisfactorily inferred the distributions of these example cases, as evidenced by scatter plots and eCDFs, compared to traditional NUTS. More interestingly, L-HNNs in NUTS required 1–2 orders of magnitude fewer evaluations of the target density gradients and produced an ESS per gradient that was an order of magnitude better than that generated via traditional NUTS.

Several future opportunities exist for improving the performance of neural networks that identify Hamiltonian systems for Bayesian inference problems: (1) newer architectures (e.g., symplectic networks) better conserve the Hamiltonian than L-HNNs, and result in an efficient inference, as quantified by the ESS per gradient; and (2) training and evaluating L-HNNs in a Riemannian space rather than an Euclidian space may lead to significant computational efficiencies for densities with high local curvatures. In addition, although the error monitoring scheme introduced in this paper calls the standard leapfrog with gradients of the target density when the integration errors of the L-HNNs are large, it does not retrain the L-HNNs. Such retraining, when performed on-the-fly within the error-monitoring scheme, is similar in concept to active or online learning. Owing to the computational expense involved in training neural networks for high-dimensional problems, quantifying the optimal schedule for retraining that is agnostic to the complexity and properties of the target density will also be an interesting avenue of exploration. Additionally, high-dimensional physics-based neural networks can be expensive to train on-the-fly. Use of dimensionality reduction to perform inference on low-dimensional manifolds is also an interesting avenue for future exploration. Gradient-based MCMC methods have important applications for forward UQ problems, such as rare event estimation. In principle, the proposed L-HNNs in NUTS with online error monitoring should be able to characterize rare events accurately. To reduce the computational cost of requiring many gradient estimations of the computational model, L-HNNs in NUTS can be integrated with variance reduction methods such as importance sampling and subset simulation. This will be pursued as part of a future study.

References

- [1] Y. Xia, N. Zabarar, Bayesian multiscale deep generative model for the solution of high-dimensional inverse problems, *J. Comput. Phys.* 455 (2022) 111008, <https://doi.org/10.1016/j.jcp.2022.111008>.

- [2] D.J. Warne, T.P. Prescott, R.E. Baker, M.J. Simpson, Multifidelity multilevel Monte Carlo to accelerate approximate Bayesian parameter inference for partially observed stochastic processes, *J. Comput. Phys.* 469 (2022) 111543, <https://doi.org/10.1016/j.jcp.2022.111543>.
- [3] D. Rossat, J. Baroth, M. Briffaut, F. Dufour, Bayesian inversion using adaptive polynomial chaos Kriging within subset simulation, *J. Comput. Phys.* 455 (2022) 110986, <https://doi.org/10.1016/j.jcp.2022.110986>.
- [4] R.M. Neal, MCMC using Hamiltonian dynamics, arXiv:1206.1901, Jun. 2012.
- [5] M.D. Hoffman, A. Gelman, The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo, *J. Mach. Learn. Res.* 15 (1) (2014) 1593–1623, <https://doi.org/10.5555/2627435.2638586>.
- [6] Y. Che, X. Wu, G. Pastore, W. Li, K. Shirvan, Application of Kriging and variational Bayesian Monte Carlo method for improved prediction of doped UO₂ fission gas release, *Ann. Nucl. Energy* 153 (2021) 108046, <https://doi.org/10.1016/j.anucene.2020.108046>.
- [7] S.L.N. Dhulipala, M.D. Shields, B.W. Spencer, C. Bolisetti, A.E. Slaughter, V.M. Laboure, P. Chakroborty, Active learning with multifidelity modeling for efficient rare event simulation, *J. Comput. Phys.* 468 (2022) 111506, <https://doi.org/10.1016/j.jcp.2022.111506>.
- [8] M. Gu, S. Sun, Neural Langevin dynamical sampling, *IEEE Access* 8 (2020) 31595–31605, <https://doi.org/10.1109/ACCESS.2020.2972611>.
- [9] L. Li, A. Holbrook, B. Shahbaba, P. Baldi, Neural network gradient Hamiltonian Monte Carlo, *Comput. Stat.* 34 (2019) 281–299, <https://doi.org/10.1007/s00180-018-00861-z>.
- [10] D. Levy, M.D. Hoffman, J. Sohl-Dickstein, Generalizing Hamiltonian Monte Carlo with neural networks, arXiv:1711.09268, Mar. 2017.
- [11] A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, P.-C. Bürkner, Rank-normalization, folding, and localization: an improved R-hat for assessing convergence of MCMC, *Bayesian Anal.* 16 (2) (2021) 667–718, <https://doi.org/10.1214/20-BA1221>.
- [12] M. Nishio, A. Arakawa, Performance of Hamiltonian Monte Carlo and No-U-

- Turn sampler for estimating genetic parameters and breeding values, *Genet. Sel. Evol.* 51 (2019) 1–12, <https://doi.org/10.1186/s12711-019-0515-1>.
- [13] S. Greydanus, M. Dzamba, J. Yosinski, Hamiltonian neural networks, [arXiv:1906.01563](https://arxiv.org/abs/1906.01563), Sep. 2019.
- [14] Y. Tong, S. Xiong, X. He, G. Pan, B. Zhu, Symplectic neural networks in Taylor series form for Hamiltonian systems, *J. Comput. Phys.* 437 (2021) 110325, <https://doi.org/10.1016/j.jcp.2021.110325>.
- [15] M. Mattheakis, D. Sondak, A.S. Dogra, P. Protopapas, Hamiltonian neural networks for solving equations of motion, *Phys. Rev. E* 105 (2022) 065305, <https://doi.org/10.1103/PhysRevE.105.065305>.
- [16] Y.D. Zhong, B. Dey, A. Chakraborty, Symplectic ode-net: learning Hamiltonian dynamics with control, [arXiv:1909.12077](https://arxiv.org/abs/1909.12077), Apr. 2019.
- [17] P. Jin, Z. Zhang, A. Zhu, Y. Tang, G.E. Karniadakis, SympNets: intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems, *Neural Netw.* 132 (2020) 166–179, <https://doi.org/10.1016/j.neunet.2020.08.017>.
- [18] P. Mann, *Lagrangian and Hamiltonian Dynamics*, 1st edition, Oxford University Press, United Kingdom, 2018.
- [19] S. Blanes, F. Casas, J.M. Sanz-Serna, Numerical integrators for the hybrid Monte Carlo method, *SIAM J. Sci. Comput.* 36 (4) (2014) A1556–A1580, <https://doi.org/10.1137/130932740>.
- [20] M.P. Calvo, J.M. Sanz-Serna, High-order symplectic Runge–Kutta–Nyström methods, *SIAM J. Sci. Comput.* 14 (5) (1993) 1237–1252, <https://doi.org/10.1137/0914073>.
- [21] M. Betancourt, A conceptual introduction to Hamiltonian Monte Carlo, [arXiv:1701.02434](https://arxiv.org/abs/1701.02434), Jan. 2017.
- [22] N.K. Vishnoi, An introduction to Hamiltonian Monte Carlo method for sampling, [arXiv:2108.12107](https://arxiv.org/abs/2108.12107), Aug. 2021.
- [23] M. Girolami, B. Calderhead, Riemann manifold Langevin and Hamiltonian Monte Carlo methods, *J. R. Stat. Soc., Ser. B, Stat. Methodol.* 73 (2) (2011) 123–214, <https://doi.org/10.1111/j.1467-9868.2010.00765.x>.
- [24] G.O. Roberts, J.S. Rosenthal, General state space Markov chains and MCMC

algorithms, *Probab. Surv.* 1 (2004) 20–71, <https://doi.org/10.1214/154957804100000024>.

- [25] F. Pagani, M. Wiegand, S. Nadarajah, An n-dimensional Rosenbrock distribution for MCMC testing, arXiv:1903.09556, May 2019.
- [26] L. Yang, X. Meng, G.E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, *J. Comput. Phys.* 425 (2021) 109913, <https://doi.org/10.1016/j.jcp.2020.109913>.
- [27] J. Goodman, J. Weare, Ensemble samplers with affine invariance, *Commun. Appl. Math. Comput. Sci.* 5 (1) (2010) 65–80, <https://doi.org/10.2140/camcos.2010.5.65>.