Generalizing Hamiltonian Monte Carlo with Neural Networks

Daniel Levy[*]
Stanford University
danilevy@cs.stanford.edu

Matthew D. Hoffman, Jascha Sohl-Dickstein
Google
{mhoffman, jaschasd}@google.com

Abstract

We present a general-purpose method to train Markov Chain Monte Carlo kernels, parameterized by deep neural networks, that converge and mix quickly to their target distribution. Our method generalizes Hamiltonian Monte Carlo and is trained to maximize expected squared jumped distance, a proxy for mixing speed. We demonstrate large empirical gains (up to $50\times$ greater effective sample size) on a collection of simple but challenging distributions. Finally, we show quantitative and qualitative gains on a real-world task: latent-variable generative modeling.

1 Introduction

High-dimensional distributions that are only analytically tractable up to a normalizing constant are ubiquitous in many fields. For instance, they arise in protein folding [18], physics simulations [16], and machine learning [1]. Sampling from such distributions is a critical task for learning and inference [13], however it is an extremely hard problem in general. Markov Chain Monte Carlo (MCMC) methods promise a solution to this problem.

For densities over continuous spaces, Hamiltonian Monte Carlo [HMC; 4, 15] introduces independent, auxiliary momentum variables, and computes a new state by integrating Hamiltonian dynamics. This method can traverse long distances in state space with a single Metropolis-Hastings test. This is the state-of-the-art method for sampling in many domains. However, HMC can perform poorly in a number of settings. While HMC mixes quickly spatially, it struggles at mixing across energy levels due to its volume-preserving dynamics. HMC also does not work well with multi-modal distributions, as the probability of sampling a large enough momentum to traverse a very low-density region is negligibly small. Furthermore, HMC struggles with ill-conditioned energy landscapes [5] and deals poorly with rapidly changing gradients [19].

Recently, probabilistic models parameterized by deep neural networks have achieved great success at *approximately* sampling from highly complex, multi-modal *empirical* distributions [11, 6, 3, 20]. Building on these successes, we present a method that, given an analytically described distribution, automatically returns an *exact* sampler with good convergence and mixing properties, from a class of highly expressive parametric models. The proposed family of samplers is a generalization of HMC; it transforms the HMC trajectory using parametric functions (deep networks in our experiments), while retaining theoretical guarantees with a tractable Metropolis-Hastings accept/reject step. The sampler is trained to minimize a variation of expected squared jumped distance (similar in spirit to [17]). Our parameterization reduces easily to standard HMC. It is further capable of emulating several common extensions of HMC such as within-trajectory tempering [15] and diagonal mass matrices [5].

We evaluate our method on distributions where HMC usually struggles, as well as on a real-world task of training latent-variable generative models.

[*]Work done while interning at Google Brain.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

2 L2HMC: Training MCMC Samplers

Background materials on MCMC methods and HMC [15, 4] are available in the Appendix. In this section, we describe our proposed method L2HMC. Given access to only an energy function U (and not samples), L2HMC learns a parametric leapfrog operator $\mathbf{L}_\theta$ over an augmented state space. We begin by describing what desiderata we have for $\mathbf{L}_\theta$, then go into detail on how we parameterize our sampler. Finally, we conclude this section by describing our training procedure.

Having observed the failure modes of HMC in the Introduction, we establish the list of desiderata for our learned MCMC kernel: *fast mixing*, *fast burn-in*, *mixing across energy levels*, and *mixing between modes*. While pursuing these goals, we must take care to ensure that our proposal operator retains two key features of the leapfrog operator used in HMC: it must be invertible, and the determinant of its Jacobian must be tractable. The leapfrog operator satisfies these properties by ensuring that each sub-update only affects a subset of the variables, and that no sub-update depends nonlinearly on any of the variables being updated. We are free to generalize the leapfrog operator in any way that preserves these properties. In particular, we are free to translate and rescale each sub-update of the leapfrog operator, so long as we are careful to ensure that these translation and scale terms do not depend on the variables being updated.

As in HMC, we begin by augmenting the current state $x \in \mathbb{R}^n$ with a continuous momentum variable $v \in \mathbb{R}^n$ drawn from a standard normal. We also introduce a binary direction variable $d \in \{-1, 1\}$, drawn from a uniform distribution. Finally, to each step t of the operator $\mathbf{L}_\theta$ we assign a fixed random binary mask $m^t \in \{0, 1\}^n$ that will determine which variables are affected by each sub-update. We draw m^t uniformly from the set of binary vectors satisfying $\sum_{i=1}^n m_i^t = \lfloor \frac{n}{2} \rfloor$, that is, half of the entries of m^t are 0 and half are 1. For convenience, we write $\bar{m}^t = 1 - m^t$ and $x_{m^t} = x \odot m^t$ ($\odot$ denotes element-wise multiplication). We will denote the complete augmented state as $\xi \triangleq \{x, v, d\}$, with probability density $p(\xi) = p(x)p(v)p(d)$.

We now describe the details of our augmented leapfrog integrator $\mathbf{L}_\theta$, for a single time-step t, and for direction $d = 1$.

We first update the momenta v. This update can only depend on a subset $\zeta_1 \triangleq \{x, \partial_x U(x), t\}$ of the full state, which excludes v. It takes the form

$$v' = v \odot \exp(\tfrac{\epsilon}{2} S_v(\zeta_1)) - \tfrac{\epsilon}{2} \left(\partial_x U(x) \odot \exp(Q_v(\zeta_1)) + T_v(\zeta_1)\right). \tag{1}$$

We have introduced three new functions of ζ_1: T_v, Q_v, and S_v. T_v is a translation, $\exp(Q_v)$ rescales the gradient, and $\exp(\frac{\epsilon}{2} S_v)$ rescales the momentum. The determinant of the Jacobian of this transformation is $\exp\left(\frac{\epsilon}{2}\mathbf{1} \cdot S_v(\zeta_1)\right)$. Note that if T_v, Q_v, and S_v are all zero, then we recover the standard leapfrog momentum update.

We now update x. As hinted above, to make our transformation more expressive, we first update a subset of the coordinates of x, followed by the complementary subset. The first update, which affects only x_{m^t}, depends on the state subset $\zeta_2 \triangleq \{x_{\bar{m}}, v, t\}$. Conversely, with x' defined below, the second update only affects $x_{\bar{m}^t}$ and depends only on $\zeta_3 \triangleq \{x'_m, v, t\}$:

$$\begin{aligned} x' &= x_{\bar{m}^t} + m^t \odot [x \odot \exp(\epsilon S_x(\zeta_2)) + \epsilon(v' \odot \exp(Q_x(\zeta_2)) + T_x(\zeta_2))] \\ x'' &= x'_{m^t} + \bar{m}^t \odot [x' \odot \exp(\epsilon S_x(\zeta_3)) + \epsilon(v' \odot \exp(Q_x(\zeta_3)) + T_x(\zeta_3))]. \end{aligned} \tag{2}$$

Again, T_x is a translation, $\exp(Q_x)$ rescales the effect of the momenta, and $\exp(\epsilon S_x)$ rescales the positions x, and we recover the original leapfrog position update if $T_x = Q_x = S_x = 0$. The determinant of the Jacobian of the first transformation is $\exp\left(\epsilon m^t \cdot S_x(\zeta_2)\right)$, and the determinant of the Jacobian of the second transformation is $\exp\left(\epsilon \bar{m}^t \cdot S_x(\zeta_3)\right)$.

Finally, we update v again, based on the subset $\zeta_4 \triangleq \{x'', \partial_x U(x''), t\}$:

$$v'' = v' \odot \exp(\tfrac{\epsilon}{2} S_v(\zeta_4)) - \tfrac{\epsilon}{2}(\partial_x U(x'') \odot \exp(Q_v(\zeta_4)) + T_v(\zeta_4)). \tag{3}$$

This update has exactly the same form as the momentum update in equation 1.

To give intuition into these terms, the scaling applied to the momentum can enable, among other things, acceleration in low-density zones, to facilitate mixing between modes. The scaling term applied to the gradient of the energy may allow better conditioning of the energy landscape (e.g., by learning a diagonal inertia tensor), or partial ignoring of the energy for rapidly changing gradients.

The corresponding integrator for $d = -1$ is given in Appendix C; it just inverts the updates in equations 1, 2 and 3. For all experiments, the functions Q, S, T are implemented using multi-layer perceptrons, with shared weights. We encode the current time step in the MLP input.

Our leapfrog operator $\mathbf{L}_\theta$ corresponds to running M steps of this modified leapfrog, $\mathbf{L}_\theta\xi = \mathbf{L}_\theta\{x, v, d\} = \{x''^{\times M}, v''^{\times M}, d\}$, and our flip operator $\mathbf{F}$ reverses the direction variable d, $\mathbf{F}\xi = \{x, v, -d\}$. Written in terms of these modified operators, our proposal and acceptance probability are identical to those for standard HMC. Note, however, that this parameterization enables learning non-volume-preserving transformations, as the determinant of the Jacobian is a function of S_x and S_v that does not necessarily evaluate to 1. This quantity is derived in Appendix D.

For convenience, we denote by $\mathbf{R}$ an operator that re-samples the momentum and direction – i.e., given $\xi = \{x, v, d\}$, $\mathbf{R}\xi = \{x, v', d'\}$ where $v' \sim \mathcal{N}(0, I), d' \sim \mathcal{U}\{-1, 1\}$. Sampling thus consists of alternating application of the $\mathbf{FL}$ and $\mathbf{R}$, in the following two steps each of which is a Markov transition for p: **1.** $\xi' = \mathbf{FL}_\theta\xi$ with probability $A(\mathbf{FL}_\theta\xi|\xi)$, otherwise $\xi' = \xi$. Here $A(\mathbf{FL}_\theta\xi|\xi)$ is the Metropolis-Hastings acceptance probability, given in Appendix Equation 8. **2.** $\xi' = \mathbf{R}\xi$.

This parameterization is effectively a generalization of standard HMC as it is non-volume preserving, with learnable parameters, and easily reduces to standard HMC for $Q, S, T = 0$.

2.1 Loss and Training Procedure

We need some criterion to tune the parameters θ that control the functions Q, S, and T. We choose a loss designed to reduce mixing time. Specifically, we aim to minimize lag-one autocorrelation. This is equivalent to maximizing expected squared jumped distance [17]. For ξ, ξ' in the extended state space, we define $\delta(\xi', \xi) = \delta(\{x', v', d'\}, \{x, v, d\}) = ||x - x'||_2^2$. Expected squared jumped distance is thus $\mathbb{E}_{\xi\sim p(\xi)}\left[\delta(\mathbf{FL}_\theta\xi, \xi)A(\mathbf{FL}_\theta\xi|\xi)\right]$. However, this loss need not encourage mixing across the entire state space. Indeed, maximizing this objective can lead to regions of state space where almost no mixing occurs, so long as the average squared distance traversed remains high. To optimize both for typical and worst case behavior, we include a reciprocal term in the loss,

$$\ell_\lambda(\xi, \xi', A(\xi'|\xi)) = \frac{\lambda^2}{\delta(\xi,\xi')A(\xi'|\xi)} - \frac{\delta(\xi,\xi')A(\xi'|\xi)}{\lambda^2}, \tag{4}$$

where λ is a scale parameter, capturing the dimension of the problem. The second term encourages typical moves to be large, while the first term strongly penalizes the sampler if it is ever in a state where it cannot move far. We train our sampler by minimizing this loss over both the target distribution and initialization distribution. Formally, given an initial distribution π_0 over $\mathcal{X}$, we define $q(\xi) = \pi_0(x)\mathcal{N}(v; 0, I)p(d)$, and minimize

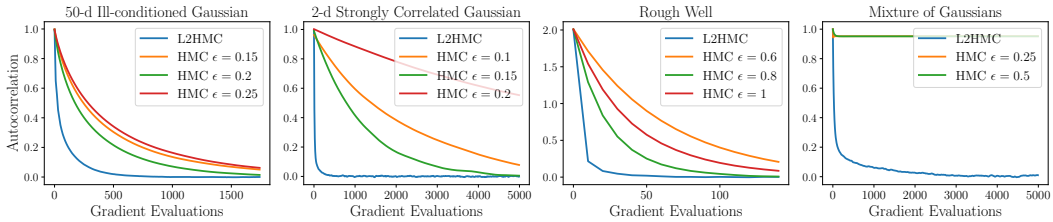
$$\mathcal{L}(\theta) \triangleq \mathbb{E}_{p(\xi)}\left[\ell_\lambda(\xi, \mathbf{FL}_\theta\xi, A(\mathbf{FL}_\theta\xi|\xi))\right] + \mathbb{E}_{q(\xi)}\left[\ell_\lambda(\xi, \mathbf{FL}_\theta\xi, A(\mathbf{FL}_\theta\xi|\xi))\right]. \tag{5}$$

The first term of this loss encourages mixing as it considers our operator applied on elements of the distribution; the second term is meant to reward fast burn-in. Given this loss, we exactly describe our training procedure in Algorithm 1, in the Appendix. It is important to note that each training iteration can be done with only one pass through the network and can be efficiently batched. We further emphasize that this training procedure can be applied to any learnable operator whose Jacobian's determinant is tractable, making it a general framework for training MCMC proposals.

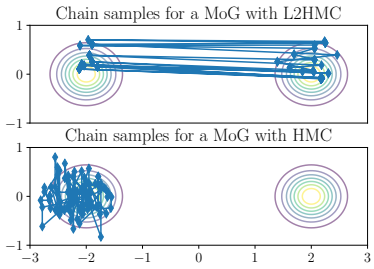
3 Experiments

We present an empirical evaluation of our trained sampler on a diversity of energy functions. We first present results on a collection of toy distributions capturing common pathologies of energy landscapes, followed by results on a task from machine learning: maximum-likelihood training of deep generative models. For each, we compare against a well-tuned HMC and show significant gains in mixing time.

Varied Collections of Energy Functions We evaluate our L2HMC sampler on a diverse collection of energy functions, each providing different challenges for standard HMC. Namely, we evaluate on a 50-d ill-conditioned Gaussian (ICG), a 2-d strongly correlated Gaussian (SCG), a mixture of two Gaussians (MoG) and a Rough Well. We observe that samplers trained with L2HMC show greatly improved autocorrelation and ESS on the presented tasks. In addition, for the Mixture of Gaussians, we show that L2HMC can easily mix between modes. We report our results in Figure 1, details for each distributions are in the Appendix.



(a) Autocorrelation



(b) Chain samples

Distribution	ESS-L2HMC	ESS-HMC	Ratio
50-d ICG	7.30×10^{-2}	1.65×10^{-2}	4.4
Rough Well	6.25×10^{-1}	1.16×10^{-1}	5.4
2-d SCG	2.32×10^{-1}	4.69×10^{-3}	49.5
MoG	3.24×10^{-2}	$\ll 1$	$\gg 1$

(c) ESS per Metropolis-Hastings step

Figure 1: L2HMC mixes faster than well-tuned HMC on a collection of toy distributions.

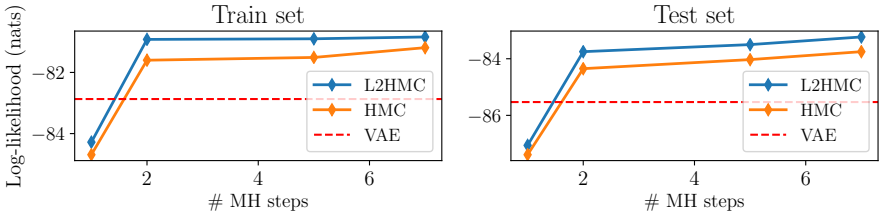
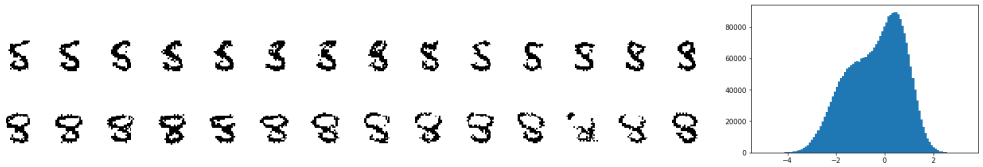


Figure 2: Training and held-out log-likelihood for models trained with L2HMC, HMC and the ELBO (VAE).



(a) *Top:* L2HMC as a posterior sampler. *Bottom:* q_ψ as a posterior sampler. (b) Non-Gaussian posterior

Figure 3: Demonstrations of the value of an expressive posterior approximation.

Exact Training Of Generative Models We use L2HMC as a posterior sampler for training generative models on MNIST [12]. We observe qualitative improvements (sharpness of the samples shown in the Appendix) as well as quantitative improvement (log-likelihood of the data compared to HMC, see Figure 2). Furthermore, this approach enables greater complexity of the posterior, which we show with projected posterior samples (Figure 3b) and Block Gibbs-Sampling (Figure 3a). We expand upon the details of our training procedure, as well as our motivations in the Appendix.

4 Conclusion

In this work, we presented a general method to train expressive MCMC kernels parameterized with deep neural networks. Given a target distribution p, analytically known up to a constant, our method provides a fast-mixing sampler, able to efficiently explore the state space. Our hope is that our method can be utilized in a "black-box" manner, in domains where sampling constitutes a huge bottleneck such as protein foldings [18] or physics simulations [16].

References

[1] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[3] Yoshua Bengio, Eric Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. In *International Conference on Machine Learning*, pages 226–234, 2014.

[4] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

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

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Ernst Hairer, Christian Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the Störmer–Verlet method. *Acta numerica*, 12:399–450, 2003.

[8] W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[9] Matthew D Hoffman. Learning deep latent Gaussian models with Markov chain Monte Carlo. In *International Conference on Machine Learning*, pages 1510–1519, 2017.

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

[13] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[14] Radford M Neal. Probabilistic inference using Markov chain Monte Carlo methods. 1993.

[15] Radford M Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2011.

[16] Peter Olsson. Two phase transitions in the fully frustrated XY model. *Physical review letters*, 75(14):2758, 1995.

[17] Cristian Pasarica and Andrew Gelman. Adaptively scaling the Metropolis algorithm using expected squared jumped distance. *Statistica Sinica*, pages 343–364, 2010.

[18] Ch Schütte, Alexander Fischer, Wilhelm Huisinga, and Peter Deuflhard. A direct approach to conformational dynamics based on hybrid Monte Carlo. *Journal of Computational Physics*, 151(1):146–168, 1999.

[19] Jascha Sohl-Dickstein, Mayur Mudigonda, and Michael R DeWeese. Hamiltonian Monte Carlo without detailed balance. pages 719–726, 2014.

[20] Jascha Sohl-Dickstein, Eric A Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *arXiv preprint arXiv:1503.03585*, 2015.

[21] Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-NICE-MC: Adversarial training for MCMC. *arXiv preprint arXiv:1706.07561*, 2017.

[22] Serena Yeung, Anitha Kannan, Yann Dauphin, and Li Fei-Fei. Tackling over-pruning in variational autoencoders. *arXiv preprint arXiv:1706.03643*, 2017.

Appendix

A Background

A.1 MCMC methods and Metropolis-Hastings

Let p be a target distribution, analytically known up to a constant, over a space $\mathcal{X}$. MCMC methods [14] aim to provide samples from p. To that end, MCMC methods construct a Markov Chain whose stationary distribution is the target distribution p. Obtaining samples then corresponds to simulating a Markov Chain, i.e., given an initial distribution π_0 and a transition kernel K, constructing the following sequence of random variables:

$$X_0 \sim \pi_0, \quad X_{t+1} \sim K(\cdot|X_t) \tag{6}$$

In order for p to be the stationary distribution of the chain, there are two conditions: K must be irreducible and aperiodic and, more importantly, p has to be a fixed point of K. The second condition can be expressed as: $p(x') = \int K(x'|x)p(x)\mathrm{d}x$. This condition is most often satisfied by satisfying the stronger *detailed balance* condition, which can be written as: $p(x')K(x|x') = p(x)K(x'|x)$.

Given any proposal distribution q, satisfying mild conditions, we can easily construct a transition kernel that respects detailed balance using Metropolis-Hastings [8] accept/reject rules. More formally, starting from $x_0 \sim \pi_0$, at each step t, we sample $x' \sim q(\cdot|X_t)$, and with probability $A(x'|x_t) = \min\left(1, \frac{p(x_t)q(x'|x_t)}{p(x')q(x_t|x')}\right)$, accept x' as the next sample x_{t+1} in the chain. If we reject x', then we retain the previous state and $x_{t+1} = x_t$. For typical proposals this algorithm has strong asymptotic guarantees. But in practice one must often choose between very low acceptance probabilities and very cautious proposals, both of which lead to slow mixing. For continuous state spaces, Hamiltonian Monte Carlo [HMC; 15] tackles this problem by proposing updates that move far in state space while staying roughly on iso-probability contours of p.

A.2 Hamiltonian Monte Carlo

Without loss of generality, we assume $p(x)$ to be defined by an energy function $U(x)$, s.t. $p(x) \propto \exp(-U(x))$, and where the state $x \in \mathbb{R}^n$. HMC extends the state space with an additional momentum vector $v \in \mathbb{R}^n$, where v is distributed independently from x, as $p(v) \propto \exp(-\frac{1}{2}v^T v)$ (i.e., identity-covariance Gaussian). From an augmented state $\xi = \{x, v\}$, HMC produces a proposed state $\xi' = \{x', v'\}$ by approximately integrating Hamiltonian dynamics jointly on x and v, with $U(x)$ taken to be the potential energy, and $\frac{1}{2}v^T v$ the kinetic. Since Hamiltonian dynamics conserve the total energy of a system, their approximate integration moves along approximate iso-probability contours of $p(x, v) = p(x)p(v)$.

Hamiltonian dynamics are typically integrated using the leapfrog integrator [7], which for a single time step consists of:

$$v^{\frac{1}{2}} = v - \tfrac{\epsilon}{2}\partial_x U(x); \quad x' = x + \epsilon v^{\frac{1}{2}}; \quad v' = v - \tfrac{\epsilon}{2}\partial_x U(x'). \tag{7}$$

Following [19], we write the action of the leapfrog integrator in terms of an operator $\mathbf{L}$, $\mathbf{L}\xi = \mathbf{L}\{x, v\} = \{x', v'\}$, and introduce a momentum flip operator $\mathbf{F}$, $\mathbf{F}\{x, v\} = \{x, -v\}$. It is important to note two properties of these operators. First, the transformation $\mathbf{FL}$ is an involution, i.e. $\mathbf{FLFL}\{x, v\} = \mathbf{FL}\{x', -v'\} = \{x, v\}$. Second, the transformations from $\{x, v\}$ to $\{x, v^{\frac{1}{2}}\}$, from $\{x, v^{\frac{1}{2}}\}$ to $\{x', v^{\frac{1}{2}}\}$, and from $\{x', v^{\frac{1}{2}}\}$ to $\{x', v'\}$ are all volume-preserving *shear* transformations i.e., only one of the variables (x or v) changes, by an amount determined by the other one. The determinant of the Jacobian, $\left|\frac{\partial[\mathbf{FL}\xi]}{\partial\xi^T}\right|$, is thus easy to compute. For vanilla HMC $\left|\frac{\partial[\mathbf{FL}\xi]}{\partial\xi^T}\right| = 1$, but we will leave it in symbolic form for use in Section 2. The Metropolis-Hastings acceptance probability for the HMC proposal is made simple by these two properties, and is

$$A(\mathbf{FL}\xi|\xi) = \min\left(1, \frac{p(\mathbf{FL}\xi)}{p(\xi)}\left|\frac{\partial[\mathbf{FL}\xi]}{\partial\xi^T}\right|\right). \tag{8}$$

Algorithm 1 Training L2HMC

Input: Energy function $U : \mathcal{X} \to \mathbb{R}$ and its gradient $\nabla_x U : \mathcal{X} \to \mathcal{X}$, initial distribution over the augmented state space q, number of iterations n_{iters}, number of leapfrogs M, learning rate schedule $(\alpha_t)_{t \leq n_{\text{iters}}}$, batch size N and scale parameter λ.
Initialize the parameters of the sampler θ.
Initialize $\{\xi_p^{(i)}\}_{i \leq N}$ from $q(\xi)$.
for $t = 0$ **to** $n_{\text{iters}} - 1$ **do**
  Sample a minibatch $\{\xi_q^{(i)}\}_{i \leq N}$ from $q(\xi)$.
  $\mathcal{L} \leftarrow 0$
  for $i = 1$ to N **do**
    $\xi_p^{(i)} \leftarrow \mathbf{R}\xi_p^{(i)}$
    $\mathcal{L} \leftarrow \mathcal{L} + \ell_\lambda\left(\xi_p^{(i)}, \mathbf{FL}_\theta \xi_p^{(i)}, A(\mathbf{FL}_\theta \xi_p^{(i)} | \xi_p^{(i)})\right) + \ell_\lambda\left(\xi_q^{(i)}, \mathbf{FL}_\theta \xi_q^{(i)}, A(\mathbf{FL}_\theta \xi_q^{(i)} | \xi_q^{(i)})\right)$
    $\xi_p^{(i)} \leftarrow \mathbf{FL}_\theta \xi_p^{(i)}$ with probability $A(\mathbf{FL}_\theta \xi_p^{(i)} | \xi_p^{(i)})$.
  end for
  $\theta \leftarrow \theta + \alpha_t \nabla_\theta \mathcal{L}$
end for

B Training Algorithm

We describe in Algorithm 1 our exact training procedure.

C Reverse leapfrog operator

Let's $\xi = \{x, v, d\}$ in the extended state space with $d = -1$. Here, we describe the leapfrog updates for a single time step t, this consists of inverting the equations presented in the corresponding section.

Let $\zeta_1 = \{x, v, t\}$, we have:

$$v' = \left[v + \frac{\epsilon}{2}\left(\partial_x U(x) \odot \exp(Q_v(\zeta_1)) + T_v(\zeta_1)\right)\right] \odot \exp(-S_v(\zeta_1)). \tag{9}$$

With the notation from Section 2, let $\zeta_2 \triangleq \{x_{m^t}, v, t\}$

$$x' = x_{m^t} + \bar{m}^t \odot [(x - \epsilon(\exp(Q_x(\zeta_2)) \odot v' + T_x(\zeta_2))] \odot \exp(-\epsilon S_v(\zeta_2)). \tag{10}$$

Let us denote $\zeta_3 \triangleq \{x'_{\bar{m}^t}, v, t\}$:

$$x' = x_{\bar{m}^t} + m^t \odot [(x' - \epsilon(\exp(Q_x(\zeta_3)) \odot v' + T_x(\zeta_3))] \odot \exp(-\epsilon S_v(\zeta_3)). \tag{11}$$

Finally, the last update, with $\zeta_4 \triangleq \{x'', \partial_x U(x''), t\}$:

$$v' = \left[v + \frac{\epsilon}{2}\left(\partial_x U(x'') \odot \exp(Q_v(\zeta_4)) + T_v(\zeta_4)\right)\right] \odot \exp(-S_v(\zeta_4)). \tag{12}$$

It is important to note that to invert $\mathbf{L}_\theta$, these steps should be ran for t from M to 1.

D Determinant of the Jacobian

Given the derivations (and notations) from Section 2, for the forward operator $\mathbf{L}_\theta$, we can immediately compute the Jacobian:

$$\log\left|\frac{\partial[\mathbf{FL}_\theta \xi]}{\partial \xi^T}\right| = d \sum_{t \leq M}\left[\frac{\epsilon}{2}\mathbf{1} \cdot S_v(\zeta_1^t) + \epsilon m^t \cdot S_x(\zeta_2^t) + \epsilon \bar{m}^t \cdot S_x(\zeta_3^t) + \frac{\epsilon}{2}\mathbf{1} \cdot S_v(\zeta_4^t)\right]. \tag{13}$$

Where ζ_i^t denotes the intermediary variable ζ_i at time step t and d is the direction of ξ i.e. $\xi = \{x, v, d\}$.

E Implementation details of L2HMC

E.1 Varied Collection of Energy Functions

We evaluate our L2HMC sampler on a diverse collection of energy functions, each providing different challenges for standard HMC.

Ill-Conditioned Gaussian (ICG): Gaussian distribution with diagonal covariance spaced log-linearly between 10^{-2} and 10^{2}. This is representative of L2HMC learning a diagonal inertia tensor.

Strongly correlated Gaussian (SCG): We rotate a diagonal Gaussian with covariance $[10^2, 10^{-2}]$ by $\frac{\pi}{4}$. This is an extreme version of an example from [15]. This problem shows that, although our parametric sampler only applies element-wise transformations, it can learn rotations.

Mixture of Gaussians (MoG): Mixture of two isotropic Gaussians with $\sigma^2 = 0.1$, and centroids separated by distance 4. The means are thus about 12 standard deviations apart, making it almost impossible for HMC to mix between modes. This experiment shows that L2HMC can learn to do within-trajectory tempering [15].

Rough Well: Similar to an example from [19], for a given $\eta > 0, U(x) = \frac{1}{2}x^T x + \eta \sum_i \cos(\frac{x_i}{\eta})$. While the energy itself is altered negligibly, its gradient is perturbed by a high frequency noise oscillating between -1 and 1. In our experiments, we choose $\eta = 10^{-2}$.

E.2 Architecture

First of all, we keep separate parameters for the network responsible for updating v and those updating x. The architectures are the same. Let us take the example of Q_v, S_v, T_v. The time step t is given as input to the MLP, encoded as $\tau(t) = \big(\cos(\frac{2\pi t}{M}), \sin(\frac{2\pi t}{M})\big)$. $\sigma(\cdot)$ denotes the ReLU non-linearity.

For n_h hidden units per layer:

- We first compute $h_1 = \sigma(W_1 x + W_2 v + W_3 \tau(t) + b)$ ($h \in \mathbb{R}^{n_h}$).
- $h_2 = \sigma(W_4 h + b_4) \in \mathbb{R}^{n_h}$
- $S_v = \lambda_s \texttt{tanh}(W_s h_2 + b_s), Q_v = \lambda_q \texttt{tanh}(W_q h_2 + b_q), T_v = W_t h_2 + b_t$.

In Section E.1,the Q, S, T are neural networks with 2 hidden layers with 10 (100 for the 50-d ICG) units and ReLU non-linearities. We train with Adam [10] and a learning rate $\alpha = 10^{-3}$. We train for $5,000$ iterations with a batch size of 200.

For the MoG tasks, we train our sampler with a temperature parameter that we continuously anneal; we evaluate the trained sampler without using temperature.

E.3 Comparison to A-NICE-MC

In addition to the well known issues of adversarial training [2], we note that parameterization using a volume-preserving operator can dramatically fail on simple energy landscapes. We build off of the *mog2* experiment presented in [21], which is a 2-d mixture of isotropic Gaussians separated by a distance of 10 with variances 0.5. We consider that setup but increase the ratio of variances: $\sigma_1^2 = 3, \sigma_2^2 = 0.05$. We show in Figure 4 sample chains trained with L2HMC and A-NICE-MC; NICE cannot effectively mix between the two modes as only a fraction of the volume of the large mode can be mapped to the small one, making it highly improbable to traverse. This is also an issue for HMC. On the other hand, L2HMC can both traverse the low-density region between modes, and map a larger volume in the left mode to a smaller volume in the right mode. It is important to note that the distance between both clusters minus their standard deviations is less than in the *mog2* case, and it is thus a good diagnostic of the shortcomings of volume-preserving transformations.

F L2HMC-DGLM

F.1 Latent-Variable Generative Model

We apply our learned sampler to the task of training, and sampling from the posterior of, a latent-variable generative model. The model consists of a latent variable $z \sim p(z)$, where we choose

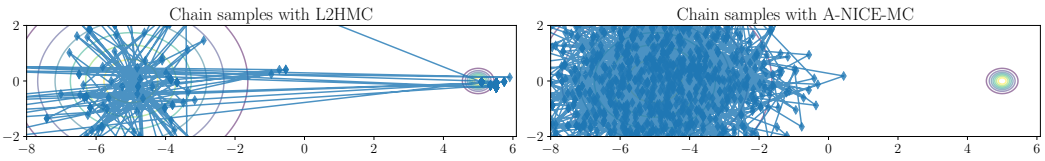


Figure 4: L2HMC can mix between modes for a MoG with different variances, contrary to A-NICE-MC.

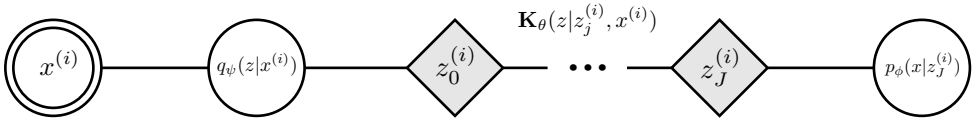


Figure 5: Diagram of our L2HMC-DGLM model. Nodes are functions of their parents. Round nodes are deterministic, diamond nodes are stochastic and the doubly-circled node is observed.

$p(z) = \mathcal{N}(z; 0, I)$, and a conditional distribution $p(x|z)$ which generates the image x. Given a family of parametric 'decoders' $\{z \mapsto p(x|z;\phi), \phi \in \Phi\}$, and a set of samples $\mathcal{D} = \{x^{(i)}\}_{i \leq N}$, training involves finding $\phi^* = \arg\max_{\phi \in \Phi} p(\mathcal{D};\phi)$. However, the log-likelihood is intractable as $p(x;\phi) = \int p(x|z;\phi)p(z)\mathrm{d}z$. To remedy that problem, [11] proposed jointly training an approximate posterior q_ψ that maximizes a tractable lower-bound on the log-likelihood:

$$\mathcal{L}_{\mathrm{ELBO}}(x,\phi,\psi) = \mathbb{E}_{q_\psi(z|x)}\left[p(x|z;\phi)\right] - \mathrm{KL}(q_\psi(z|x)||p(z)) \leq p(x) \tag{14}$$

Recently, to improve upon well-known pitfalls like over-pruning [22] of the VAE, [9] proposed HMC-DLGM. For a data sample $x^{(i)}$, after obtaining a sample from the approximate posterior $q_\psi(\cdot|x^{(i)})$, [9] runs a MCMC algorithm with energy function $U(z, x^{(i)}) = -\log p(z) - \log p(x^{(i)}|z;\phi)$ to obtain a more exact posterior sample from $p(z|x^{(i)};\phi)$. Given that better posterior sample z', the algorithm maximizes $p(x^{(i)}|z';\phi)$.

To show the benefits of L2HMC, we borrow the method from [9], but replace HMC by jointly training an L2HMC sampler to improve the efficiency of the posterior sampling. We call this model **L2HMC-DLGM**. A diagram of our model and a formal description of our training procedure are presented in Appendix F.We define, for $\xi = \{z, v, d\}$, $r(\xi|x;\psi) \triangleq q_\psi(z|x)\mathcal{N}(v; 0, I)\, p(d)$.

In the subsequent sections, we compare our method to the standard VAE model from [11] and HMC-DGLM from [9]. It is important to note that, since our sampler is trained jointly with p_ϕ and q_ψ, *it performs exactly the same number of gradient computations of the energy function as HMC*. We first show that training a latent variable generative model with L2HMC results in better generative models both qualitatively and quantitatively. We then show that our improved sampler enables a more expressive, non-Gaussian, posterior.

Implementation details: Our decoder (p_ϕ) is a neural network with 2 fully connected layers, with 1024 units each and softplus non-linearities, and outputs Bernoulli activation probabilities for each pixel. The encoder (q_ψ) has the same architecture, returning mean and variance for the approximate posterior. Our model was trained for 300 epochs with Adam [10] and a learning rate $\alpha = 10^{-3}$. All experiments were done on the dynamically binarized MNIST dataset [12].

F.2 Training algorithm

In this section is presented our training algorithm as well as a diagram explaining L2HMC-DGLM. For conciseness, given our operator $\mathbf{L}_\theta$, we denote by $\mathbf{K}_\theta(\cdot|x)$ the distribution over next state given sampling of a momentum and direction and the Metropolis-Hastings step.

Algorithm 2 L2HMC for latent variable generative models

Input: dataset $\mathcal{D}$, number of iterations n_{iters}, number of Metropolis-Hastings step J, number of leapfrogs M, and learning rate schedule $(\alpha_t)_{t \leq n_{\text{iters}}}$.
Randomly initialize the decoder's parameters ϕ and the approximate posterior ψ. Initialize the parameters of the sampler θ with M leapfrog steps.
for $t = 0$ **to** $n_{\text{iters}} - 1$ **do**
  Randomly sample a minibatch $\mathcal{B}$ from the dataset $\mathcal{D}$.
  $\mathcal{L}_{\text{ELBO}}, \mathcal{L}_{\text{Sampler}}, \mathcal{L}_{\text{Decoder}} \leftarrow 0$
  for $x^{(b)} \in \mathcal{B}$ **do**
    Sample $\xi_0^{(b)} \sim r(\cdot|x^{(b)}; \psi)$.
    $\mathcal{L}_{\text{ELBO}} \leftarrow p(x^{(b)}|z_0^{(b)}; \phi) - \text{KL}(q_\psi(z|x^{(b)})||p(z))$ ▷ With $\xi_0^{(b)} = \{z_0^{(b)}, v_0^{(b)}, d_0^{(b)}\}$
    Define the energy function $U_{x^{(b)}}(z) = -\log p(x^{(b)}|z; \theta) - \log p(z)$
    $\mathcal{L}_{\text{Sampler}} \leftarrow 0$
    $\lambda \leftarrow \sqrt{\text{Var}(q_\psi(z_0^{(b)}|x^{(b)})}$
    for $j = 0$ **to** $J - 1$ **do**
      $\xi_j^{(b)} \leftarrow \mathbf{R}\xi_j^{(b)}$
      $\mathcal{L}_{\text{Sampler}} \leftarrow \mathcal{L}_{\text{Sampler}} + \ell_\lambda(\xi_j^{(b)}, \mathbf{FL}_\theta \xi_j^{(b)}, A(\mathbf{FL}_\theta \xi_j^{(b)}|\xi_j^{(b)}))$
      Set $\xi_{j+1}^{(b)}$ to $\mathbf{FL}_\theta \xi_j^{(b)}$ with probability $A(\mathbf{FL}_\theta \xi_j^{(b)}|\xi_j^{(b)})$.
    end for
    $\mathcal{L}_{\text{Decoder}} \leftarrow \mathcal{L}_{\text{Decoder}} + \log p(x^{(b)}|z_J^{(s)}; \phi)$ ▷ With $\xi_J^{(b)} = \{z_J^{(b)}, v_J^{(b)}, d_J^{(b)}\}$
  end for
  $\phi \leftarrow \phi + \alpha_t \nabla_\phi \mathcal{L}_{\text{Decoder}}$
  $\psi \leftarrow \psi + \alpha_t \nabla_\psi \mathcal{L}_{\text{ELBO}}$
  $\theta \leftarrow \theta + \alpha_t \nabla_\theta \mathcal{L}_{\text{Sampler}}$
end for

F.3 Implementation details of L2HMC-DGLM

Similar to our L2HMC training on unconditional sampling, we share weights across Q, S and T. In addition, the auxiliary variable x (here the image from MNIST) is first passed through a 2-layer neural network, with softplus non-linearities and 512 hidden units. This input is given to both networks $\{\cdot\}_x$ and $\{\cdot\}_v$. The architecture then consists of 2 hidden layers of 200 units and ReLU non-linearities. For λ (scale parameter of the loss), we use the standard deviation of the approximate posterior.

AIS Evaluation For each data point, we run 20 Markov Chains in parallel, $10,000$ annealing steps with 10 leapfrogs per step and choose the step size for an acceptance rate of 0.65.

F.4 MNIST Samples

(a) L2HMC (b) HMC (c) VAE

Figure 6: L2HMC-DGLM decoder produces sharper mean activations.