entropy

MDPI

Article

A Neural Network MCMC Sampler That Maximizes Proposal Entropy

Zengyi Li [1,2,*], Yubei Chen [1,3] and Friedrich T. Sommer [1,4,5]

1 Redwood Center for Theoretical Neuroscience, Berkeley, CA 94720, USA; yubeic@berkeley.edu (Y.C.); fsommer@berkeley.edu (F.T.S.)
2 Department of Physics, University of California Berkeley, Berkeley, CA 94720, USA
3 Berkeley AI Research, University of California Berkeley, Berkeley, CA 94720, USA
4 Helen Wills Neuroscience Institute, University of California Berkeley, Berkeley, CA 94720, USA
5 Neuromorphic Computing Group, Intel Labs, 2200 Mission College Blvd., Santa Clara, CA 95054-1549, USA
* Correspondence: zengyi_li@berkeley.edu

Abstract: Markov Chain Monte Carlo (MCMC) methods sample from unnormalized probability distributions and offer guarantees of exact sampling. However, in the continuous case, unfavorable geometry of the target distribution can greatly limit the efficiency of MCMC methods. Augmenting samplers with neural networks can potentially improve their efficiency. Previous neural network-based samplers were trained with objectives that either did not explicitly encourage exploration, or contained a term that encouraged exploration but only for well structured distributions. Here we propose to maximize proposal entropy for adapting the proposal to distributions of any shape. To optimize proposal entropy directly, we devised a neural network MCMC sampler that has a flexible and tractable proposal distribution. Specifically, our network architecture utilizes the gradient of the target distribution for generating proposals. Our model achieved significantly higher efficiency than previous neural network MCMC techniques in a variety of sampling tasks, sometimes by more than an order magnitude. Further, the sampler was demonstrated through the training of a convergent energy-based model of natural images. The adaptive sampler achieved unbiased sampling with significantly higher proposal entropy than a Langevin dynamics sample. The trained sampler also achieved better sample quality.

Keywords: MCMC; neural network sampler; maximum entropy; energy-based model

check for updates

Citation: Li, Z.; Chen, Y.; Sommer, F.T. A Neural Network MCMC Sampler That Maximizes Proposal Entropy. *Entropy* **2021**, *23*, 269. https://doi.org/10.3390/e23030269

Academic Editor: Philip Broadbridge

Received: 10 February 2021
Accepted: 22 February 2021
Published: 25 February 2021

1. Introduction

Sampling from unnormalized distributions is important for many applications, including statistics, simulations of physical systems and machine learning. However, the inefficiency of state-of-the-art sampling methods remains a main bottleneck for many challenging applications, such as protein folding [1] and energy-based model training [2].

A prominent strategy for sampling is the Markov Chain Monte Carlo (MCMC) method [3]. In MCMC, one chooses a transition kernel that leaves the target distribution invariant and constructs a Markov Chain by applying the kernel repeatedly. The MCMC method relies only on the ergodicity assumption. Other than that, it is general: if enough computation is performed, the Markov Chain generates correct samples from any target distribution, no matter how complex the distribution is. However, the performance of MCMC depends critically on how well the chosen transition kernel explores the state space of the problem. If exploration is ineffective, samples will be highly correlated and of very limited use for downstream applications. Thus, despite the theoretical guarantee that MCMC algorithms are exact, practically they may still suffer from inefficiencies.

Take, for example, Hamiltonian Monte Carlo (HMC) sampling [4], a type of MCMC technique. HMC is regarded as the state-of-the-art for sampling in continuous spaces [5]. It uses a set of auxiliary momentum variables and generates new samples by simulating

Hamiltonian dynamics starting from the previous sample. This allows the sample to travel in state space much further than possible with other techniques, most of whom have more pronounced random walk behavior. Theoretical analysis shows that the cost of traversing in d-dimensional state space and generating an uncorrelated proposal is $O(d^{\frac{1}{4}})$ for HMC, which is lower than $O(d^{\frac{1}{3}})$ for Langevin Monte Carlo, and $O(d)$ for random walk [6]. However, unfavorable geometry of a target distribution may still render HMC ineffective because the Hamiltonian dynamics have to be simulated numerically. Numerical errors in the simulation are corrected by a Metropolis–Hastings (MH) accept–reject step of a proposed sample. If the the target distribution has unfavorable geometric properties, for example, large differences in variance along different directions, the numerical integrator in HMC will produce high errors, leading to a very low accept probability [7]. For simple distributions, this inefficiency can be mitigated by an adaptive re-scaling matrix [4]. For analytically tractable distributions, one can also use the Riemann manifold HMC method [8]. However, in most other cases, the Hessian required in the Riemann manifold HMC algorithm is often intractable or expensive to compute, preventing its application.

Recently, approaches have been proposed that inherit the exact sampling property from the MCMC method, while potentially mitigating the described issues of unfavorable geometry. One approach is MCMC samplers augmented with neural networks [9–11]; the other approach is neural transport MCMC techniques [12,13]. A disadvantage of these recent techniques is that their objectives optimize the quality of proposed samples, but do not explicitly encourage the exploration speed of the sampler. One notable exception is L2HMC [10], a method whose objective includes the size of the expected L2 jump, thereby encouraging exploration. However, the L2 expected jump objective is not very general; it only works for simple distributions (see Figure 1, and below).

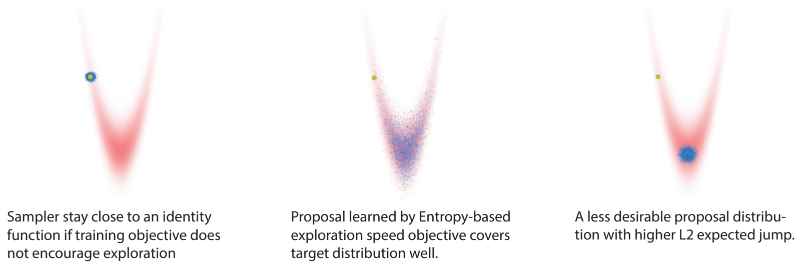


Figure 1. Illustration of learning for exploring a state space. Left panel: a Langevin sampler that has poor exploration. Middle panel: our proposed method—samples travel far within the target distribution. Right panel: a sampler with a higher L2 jump than ours—the exploration is still worse. In each panel, the yellow dot on the top left is the initial point x; blue and black dots are accepted and rejected samples, respectively.

In another recent work [14], exploration speed was encouraged by a quite general objective: the entropy of the proposal distribution. In continuous space, the entropy of a distribution is essentially the logarithm of its volume in state space. Thus, the entropy objective naturally encourages the proposal distribution to “fill up” the target state space and possible independent of the geometry of the target distribution. The authors demonstrated the effectiveness of this objective on samplers with simple linear adaptive parameters.

We highlight the difference between L2 expected jump objective and entropy objective in Figure 1. A neural network sampler, trained with the entropy-based objective, generates samples that explore the target distribution quite well. In contrast, sampling by constructing proposals with higher expected L2 jumps leads to a less desirable result (right panel).

In the paper we make the following contributions:

1. Here, we employed the entropy-based objective in a neural network MCMC sampler for optimizing exploration speed. To build the model, we designed a novel flexible proposal distribution wherein the optimization of the entropy objective is tractable.

2. Inspired by the HMC and the L2HMC [10] algorithm, the proposed sampler uses a special architecture that utilizes the gradient of the target distribution to aid sampling.
3. We demonstrate a significant improvement in sampling efficiency over previous techniques, sometimes by an order of magnitude. We also demonstrate energy-based model training with the proposed sampler and demonstrate higher exploration and higher resultant sample quality.

The reminder of the paper is organized as follows. Section 2 briefly introduces MCMC methods. In Section 3 we formulate the model. Section 4 discusses relationships and differences of the new model and HMC-based models from the literature. In Section 5, we provide experimental results. The paper concludes with a discussion in Section 6.

2. Preliminaries: MCMC Methods from Vanilla to Learned

Consider the problem of sampling from a target distribution $p(x) = e^{-U(x)}/Z$ defined by the energy function $U(x)$ in a continuous state space. MCMC methods solve this problem by constructing and running a Markov Chain with a transition probability $p(x'|x)$ that leaves $p(x)$ invariant. The most general invariance condition is: $p(x') = \int p(x'|x)p(x)dx$ for all x', which is typically enforced by the simpler but more restrictive condition of detailed balance: $p(x)p(x'|x) = p(x')p(x|x')$.

For a general distribution $p(x)$ it is difficult to directly construct a $p(x'|x)$ that satisfies detailed balance. However, one can easily (Up to ergodic and aperiodic assumptions) make any transition probability satisfy detailed balance by adding a Metropolis–Hastings (M–H) accept–reject step [15]. When we sample x' at step t from an arbitrary proposal distribution $q(x'|x^t)$, the M–H accept–reject process accepts the new sample $x^{t+1} = x'$ with probability

$$A(x', x) = \min\left(1, \frac{p(x')q(x^t|x')}{p(x^t)q(x'|x^t)}\right). \tag{1}$$

If x' is rejected, the new sample is set to the previous state $x^{t+1} = x^t$. This transition kernel $p(x'|x)$ constructed from $q(x'|x)$ and $A(x', x)$ leaves any target distribution $p(x)$ invariant.

Most popular MCMC techniques use the described M–H accept–reject step to enforce detailed balance, for example, the random walk Metropolis (RWM), the metropolis-adjusted Langevin algorithm (MALA) and the Hamiltonian Monte Carlo (HMC). For brevity, we will focus on MCMC methods that use the M–H step, although some alternatives do exist [16]. These methods share the requirement that the accept probability in the M–H step (Equation (1)) must be tractable to compute. For two of the mentioned MCMC methods this is indeed the case. In the Gaussian random-walk sampler, the proposal distribution is a Gaussian around the current position: $x' = x + \epsilon * \mathcal{N}(0, \mathbf{I})$, which has the form $x' = x + z$. Thus, forward and reverse proposal probabilities are given by $q(x'|x) = p_{\mathcal{N}}[(x' - x)/\epsilon]$ and $q(x|x') = p_{\mathcal{N}}[-(x' - x)/\epsilon]$, where $p_{\mathcal{N}}$ denotes the density function of a Gaussian with a 0 mean and unit diagonal variance. The probability ratio $\frac{q(x^t|x')}{q(x'|x^t)}$ used in the M–H step (Equation (1)) is therefore equal to 1. In MALA the proposal distribution is a single step of Langevin dynamics with step size ϵ: $x' = x + z$ with $z = -\frac{\epsilon^2}{2}\partial_x U(x) + \epsilon N(0, \mathbf{I})$. We then have $q(x'|x) = p_{\mathcal{N}}\left[(x' - x)/\epsilon + \frac{\epsilon}{2}\partial_x U(x)\right]$ and $q(x|x') = p_{\mathcal{N}}\left[-(x' - x)/\epsilon + \frac{\epsilon}{2}\partial_{x'} U(x')\right]$. Both the forward and reverse proposal probabilities are tractable since they are the densities of Gaussians evaluated at a known location.

Next we introduce the HMC sampler and show how it can be formulated as an M–H sampler. Basic HMC involves a Gaussian auxiliary variable v of the same dimension as x, which plays the role of the momentum in physics. HMC sampling consists of two steps: (1) The momentum is sampled from a normal distribution $\mathcal{N}(v; 0, \mathbf{I})$. (2) The Hamiltonian dynamics are simulated for a certain duration with initial condition x and v, typically by running a few steps of the leapfrog integrator [4]. Then, an M–H accept–reject process with accept probability $A(x', v', x, v) = \min\left(1, \frac{p(x',v')q(x,v|x',v')}{p(x,v)q(x',v'|x,v)}\right) = \min\left(1, \frac{p(x')p_{\mathcal{N}}(v')}{p(x)p_{\mathcal{N}}(v)}\right)$ is performed to correct for error in the integration process. We have $\frac{q(x,v|x',v')}{q(x',v'|x,v)} = 1$ since the Hamiltonian transition is volume-preserving over (x, v). Both HMC steps leave the joint distribution $p(x, v)$ invariant; therefore, HMC samples form the correct distribution $p(x)$ after marginalizing over v. To express basic HMC in the standard M–H scheme, steps 1 and 2 can be aggregated into a single proposal distribution on x with the proposal probability: $q(x'|x) = p_{\mathcal{N}}(v)$ and $q(x|x') = p_{\mathcal{N}}(v')$. Note, although the probability $q(x'|x)$ can be calculated after the Hamiltonian dynamics are simulated, this term is intractable for general x and x'. The reason is that it is difficult to solve for the v at x to make the transition to x' using the Hamiltonian dynamics. This issue is absent in RWM and MALA, where $q(x'|x)$ is tractable for any x and x'.

Previous work on augmenting MCMC sampler with neural networks also relied on the M–H procedure to ensure asymptotic correctness of the sampling process, for example [9,10]. They used HMC style accept–reject probabilities that lead to intractable $q(x'|x)$. Here, we strive for a flexible sampler for which $q(x'|x)$ is tractable. This maintains the tractable M–H step while allowing us to train this sampler to explore the state space by directly optimizing the proposal entropy objective, which is a function of $q(x'|x)$.

3. A Gradient-Based Sampler with Tractable Proposal Probability

We "abuse" the power of neural networks to design a sampler that is flexible and has tractable proposal probability $q(x'|x)$ between any two points. However, without any extra help, the sampler would be modeling a conditional distribution $q(x'|x)$ with brute force, which might be possible but requires a large model capacity. Thus, our method uses the gradient of the target distribution to guide proposal distribtion. Specifically, we use an architecture similar to L2HMC [10], which itself was inspired by the HMC algorithm and RealNVP [17]. To quantify the benefit of using the target distribution gradient, we provide ablation studies of our model in the Appendix A.1.

3.1. Proposal Model and How to Use Gradient Information

We restrict our sampler to the simple general form $x' = x + z$. As discussed in Section 2, the sampler will have tractable proposal probability if one can calculate the probability of any given z. To fulfill this requirement, we model vector z by a flow model (For more details on flow models, see [18,19]): $z = f(z_0; x, U)$, with inverse $z_0 = f^{-1}(z; x, U)$. Here z_0 is sampled from a fixed Gaussian base distribution. The flow model f is a flexible and trainable invertible function of z conditioned on x, U, and it has tractable Jacobian determinant w.r.t. z. The flow model f can be viewed as a change of variable from the Gaussian base distribution $z0$ to z. The proposed sampler then has tractable forward and reverse proposal probabilities: $q(x'|x) = p_Z(x' - x; x)$, $q(x|x') = p_Z(x - x'; x')$, where $p_Z(z; x) = p_{\mathcal{N}}(z_0)|\frac{\partial z}{\partial z_0}|^{-1}$ is the density defined by the flow model f. Note, this sampler is ergodic and aperiodic, since $q(x'|x) \neq 0$ for any x and x', which follows from the invertibility of f. Thus, combined with the M–H step, the sampling process will be asymptotically correct. The sampling process first consists of drawing from $p_{\mathcal{N}}(z_0)$ and then evaluating $z = f(z_0; x, U)$ and $q(x'|x)$. Next, the reverse $z_0' = f^{-1}(-z; x + z, U)$ is evaluated at $x' = x + z$ to obtain the reverse proposal probability $q(x|x')$. Finally, the sample is accepted with the standard M–H rule.

For the flow model f, we use an architecture similar to a non-volume preserving coupling-based flow RealNVP [17]. In the coupling-based flow, half of the components of the state vector are kept fixed and used to update the other half through an affine transform parameterized by a neural network. The gradient of the target distribution enters our model in those affine transformations. To motivate this particular model choice, we take a closer look at the standard HMC algorithm. Basic HMC starts with drawing a random initial momentum v^0, followed by several steps of leapfrog integration. In the nth leapfrog step, the integrator first updates v with a half step of the gradient: $v^{n\prime} = v^{n-1} - \frac{\epsilon}{2}\partial_x U(x^{n-1})$, followed by a full step of x update: $x^n = x^{n-1} + \epsilon v^{n\prime}$ and another half step of v update: $v^n = v^{n\prime} - \frac{\epsilon}{2}\partial_x U(x^n)$. After several steps, the overall update of x can be written as: $x^n = x^0 + \sum_{i=0}^n v^{i\prime}$, which has the form $x' = x + z$ with $z = \sum_i^n v^{i\prime} = -nv^0 - \frac{n\epsilon}{2}\left[\partial_x U(x^0)\right] - \epsilon\left[\sum_{i=1}^n (n-i)\partial_x U(x^i)\right]$. This equation suggests that when generating z through affine transformations, gradient should enter through the shift term with a negative sign.

3.2. Model Formulation

To formulate our model (Equations (2) and (3)), we use a binary mask vector m and its complement $\overline{m}$ to select half of z's dimensions for update at a time. As discussed above, we include the gradient term with a negative sign in the shift term of the affine transform. We also use an element-wise scaling on the gradient term as in [10]. However, two issues remain. First, as required by the coupling-based architecture, the gradient term can only depend on the masked version of vector z. Second, it is unclear where the gradient should be evaluated to sample effectively. As discussed above, the sampler should evaluate the gradient at points far away from x, similar as in HMC, to travel long distances in the state space. To handle these issues, we use another neural network R which receives x and the masked z as input, and evaluates the gradient at $x + R$. During training, R learns where the gradient should be evaluated based on the masked z.

We denote the input to network R by $\zeta_m^n = (x, m \odot z^n)$ and the input to the other networks by $\xi_m^n = (x, m \odot z^n, \partial U(x + R(\zeta_m^n)))$, where $\odot$ is the Hadamard product (element wise multiply). Further, we denote the neural network outputs that parameterize the affine transform by $S(\xi_m^n)$, $Q(\xi_m^n)$ and $T(\xi_m^n)$, where $\exp[S]$ and T parameterize the element-wise scaling term and shift term in the affine transform, and $\exp[Q]$ gives the element-wise scaling term for the gradient. For notation clarity we omit dependencies of the mask m and all neural network terms on the step number n.

Additionally, we introduce a scale parameter ϵ, which modifies the x update to $x' = x + \epsilon z$. We also define $\epsilon' = \epsilon/(2N)$, with N the total number of z update steps. This parameterization makes our sampler equivalent to the MALA algorithm with step size ϵ at initialization, where the neural network outputs are zero. The resulting update rule is:

$$z^{n\prime} = m \odot z^{n-1} + \overline{m} \odot \left(z^{n-1} \odot \exp[S(\xi_m^{n-1})] - \epsilon'\{\partial U[x + R(\zeta_m^{n-1})] \odot \exp[Q(\xi_m^{n-1})] + T(\xi_m^{n-1})\}\right) \quad (2)$$

$$z^n = \overline{m} \odot z^{n\prime} + m \odot \left(z^{n\prime} \odot \exp[S(\xi_{\overline{m}}^{n\prime})] - \epsilon'\{\partial U[x + R(\zeta_{\overline{m}}^{n\prime})] \odot \exp[Q(\xi_{\overline{m}}^{n\prime})] + T(\xi_{\overline{m}}^{n\prime})\}\right) \quad (3)$$

The log determinant of N steps of transformation is:

$$\log\left|\frac{\partial x'}{\partial z^0}\right| = \epsilon \mathbf{1} * \mathbf{1} + \sum_{n=1}^{N} \mathbf{1} * \left[\overline{m} \odot S(\xi_m^{n-1})\right] + \mathbf{1} * \left[m \odot S(\xi_{\overline{m}}^{n\prime})\right] \quad (4)$$

where $\mathbf{1}$ is the vector of 1-entries with the same dimension as z, $*$ denotes the dot product. This follows from the simple fact that the log determinant of affine transform $z' = z \odot \exp(S) + T$ is $\mathbf{1} * S$.

3.3. Optimizing the Proposal Entropy Objective

The proposal entropy can be expressed as:

$$H(X'|X = x) = -\int dx' q(x'|x) \log[q(x'|x)] = -\int dz^0 p_{\mathcal{N}}\left(z^0\right)\left[\log\left(p_{\mathcal{N}}\left(z^0\right)\right) - \log\left|\frac{\partial z^N}{\partial z^0}\right|\right] \quad (5)$$

For each x, we aim to optimize $S(x) = \exp[\beta H(X'|X = x)] \times a(x)$, where $a(x) = \int A(x', x) q(x'|x) dx'$ is the average accept probability of the proposal distribution at x. Following [14], we transform this objective into log space and use Jensen's inequality to obtain a lower bound:

$$\begin{aligned} \log S(x) &= \log \int A(x', x) q(x'|x) dx' + \beta H(X'|X = x) \\ &\geq \int \log[A(x'x)] q(x'|x) dx' + \beta H(X'|X = x) = L(x) \end{aligned}$$

The distribution $q(x'|x)$ is reparameterizable; therefore, the expectation over $q(x'|x)$ can be expressed as expectation over $p_{\mathcal{N}}(z_0)$. By expanding the lower bound $L(x)$ and omitting the (constant) entropy of the base distribution $p_{\mathcal{N}}(z_0)$, we arrive at:

$$L(x) = \int dz^0 p_{\mathcal{N}}(z^0) \left[\min\left(0, \log \frac{p(x')}{p(x)} + \log \frac{q(x|x')}{q(x'|x)}\right) - \beta \log \left| \frac{\partial x'}{\partial z^0} \right| \right] \tag{6}$$

During training we maximize $L(x)$ with x sampled from the target distribution $p(x)$ if it is available, or with x obtained from the bootstrapping process [9] which maintains a buffer of samples and updates them continuously. Typically, only one sample of z^0 is used for each x.

A curious feature of our model is that during training one has to back-propagate over the gradient of the target distribution multiple times to optimize R. In [14] the authors avoid multiple back-propagation by stopping the derivative calculation at the density gradient term. In our experiment we did not use this trick and performed full back-propagation without encountering any issue. We found that stopping the derivative computation instead harms performance.

The entropy-based exploration objective contains a parameter β that controls the balance between acceptance rate and proposal entropy. As in [14], we use a simple adaptive scheme to adjust β to maintain a constant accept rate close to a target accept rate. The target accept rate is chosen empirically. As expected, we find that the target accept rate needs to be lower for more complicated distributions.

4. Related Work: Other Samplers Inspired by HMC

Here we discuss other neural network MCMC samplers and how they differ from our method. Methods we compare ours to in the results are marked with bold font.

A-NICE-MC [9], which was generalized in [20], used the same accept probability as HMC, but replaced the Hamiltonian dynamics with a flexible volume-preserving flow [21]. A-NICE-MC matches samples from $q(x'|x)$ directly to samples from $p(x)$, using adversarial loss. This permits training the sampler on empirical distributions, i.e., in cases where only samples, but not the density function, are available. The problem with this method is that samples from the resulting sampler can be highly correlated because the adversarial objective only optimizes for the quality of the proposed sample. If the sampler produces a high quality sample x, the learning objective does not encourage the next sample x' to be substantially different from x. The authors used a pairwise discriminator that empirically mitigated this issue but the benefit in exploration speed is limited.

Another related sampling approach is **neural transport MCMC** [12,13,22], which fits a distribution defined by a flow model $p_g(x)$ to the target distribution using $\mathbf{KL}[p_g(x)||p(x)]$. Sampling is then performed with HMC in the latent space of the flow model. Due to the invariance of the KL-divergence with respect to a change of variables, the "transported distribution" in z space $p_{g^{-1}}(z)$ will be fitted to resemble the Gaussian prior $p_{\mathcal{N}}(z)$. Samples of x can then be obtained by passing z through the transport map. Neural transport MCMC improves sampling efficiency compared to sampling in the original space because a distribution closer to a Gaussian is easier to sample. However, the sampling cost is not a monotonic function of the KL-divergence used to optimize the transport map [23].

Another line of work connects the MCMC method to Variational Inference [24–27]. Simply put, they improve the variational approximation by running several steps of MCMC transitions initialized from a variational distribution. The MCMC steps are optimized by minimizing the KL-divergence between the resulting distribution and the true posterior. This amounts to optimizing a "burn in" process in MCMC. In our setup however, the exact sampling is guaranteed by the M–H process, thus the KL divergence loss is no longer applicable. Like in variational inference, the **normalizing flow Langevin MC** (NFLMC) [11] also used a KL divergence loss. Strictly speaking, this model is a normalizing flow but not an MCMC method. We compare our method to it, because the model architecture, like ours, uses the gradient of the target distribution.

Another related technique is [28], where the authors trained an independent M–H sampler by minimizing $\mathbf{KL}[p(x)q(x'|x)||p(x')q(x|x')]$. This objective can be viewed as a lower bound of the M–H accept rate. However, as discussed in [14], this type of objective is not applicable for samplers that condition on the previous state.

All the mentioned techniques have in common that their objective does not encourage exploration speed. In contrast, **L2HMC** [10,29] does encourage fast exploration of the state space by employing a variant of the expected square jump objective [30]: $L(x) = \int dx' q(x'|x)A(x',x)||x'-x||^2$. This objective provides a learning signal even when x is drawn from the exact target distribution $p(x)$. L2HMC generalized the Hamiltonian dynamics with a flexible non-volume-preserving transformation [17]. The architecture of L2HMC is very flexible and uses gradient of target distribution. However, the L2 expected jump objective in L2HMC improves exploration speed only in well-structured distributions (see Figure 1).

The shortcomings of the discussed methods led us to consider the use of an entropy-based objective. However, L2HMC does not have tractable proposal probability $p(x'|x)$, preventing the direct application of the entropy-based objective. In principle, the proposal entropy objective could be optimized for the L2HMC sampler with variational inference [31,32], but our preliminary experiments using this idea were not promising. Therefore, we designed our sampler to possess tractable proposal probability and investigated tractable optimization of the proposal entropy objective.

5. Experimental Result

5.1. Synthetic Dataset and Bayesian Logistic Regression

First we demonstrate that our technique accelerates sampling of the funnel distribution, a particularly challenging example from [33]. We then compare our model with A-NICE-MC [9], L2HMC [10], normalizing flow Langevin MC (NFLMC) [11] and NeuTra [12] on several other synthetic datasets and a Bayesian logistic regression task. We additionally compare to gradMALA [14] to show the benefit of using neural network over linear adaptive sampler. For all experiments, we report effective sample size [34] per M–H step (ESS/MH) and/or ESS per target density gradient evaluation (ESS/grad). All results are given in minimum ESS over all dimensions unless otherwise noted. In terms of these performance measures, larger numbers are better.

Here we provide brief descriptions of the datasets used in our experiments:

Ill Conditioned Gaussian: a 50-dimensional ill-conditioned Gaussian task described in [10]; a Gaussian with diagonal covariance matrix with log-linearly distributed entries between $[10^{-2}, 10^2]$.

Strongly correlated Gaussian: a 2-dimensional Gaussian with variance $[10^2, 10^{-1}]$ rotated by $\frac{\pi}{4}$; same as in [10].

Funnel distribution: The density function is $p_{funnel}(x) = \mathcal{N}(x_0; 0, \sigma^2)\mathcal{N}(x_{1:n}; 0, \mathbf{I}\exp(-2x_0))$. This is a challenging distribution because the spatial scale of $x_{1:n}$ varies drastically depending on the value of x_0. This geometry causes problems to adaptation algorithms that rely on a spatial scale. An important detail is that earlier work, such as [35] used $\sigma = 3$, while some recent works used $\sigma = 1$. We ran experiments with $\sigma = 1$ for comparison with recent techniques and also used our method on a 20 dimensional funnel distribution with $\sigma = 3$. We denote the two variants by Funnel-1 and Funnel-3.

Bayesian logistic regression: we follow the setup in [34] and used the German, Heart and Australian datasets from the UCI data registry.

In Figure 2, we compare our method with HMC on the 20d Funnel-3 distribution. As discussed in [35], the stepsize of HMC needs to be manually tuned down to allow traveling into the neck of the funnel, otherwise the sampling process will be biased. We thus tuned the stepsize of HMC to be the largest that still allows traveling into the neck. Each HMC proposal was set to use the same number of gradient steps as each proposal of the trained sampler. As can be seen, the samples proposed by our method travel significantly further than the HMC samples. Our method achieved 0.256 (ESS/MH), compared to 0.0079 (ESS/MH) with HMC.

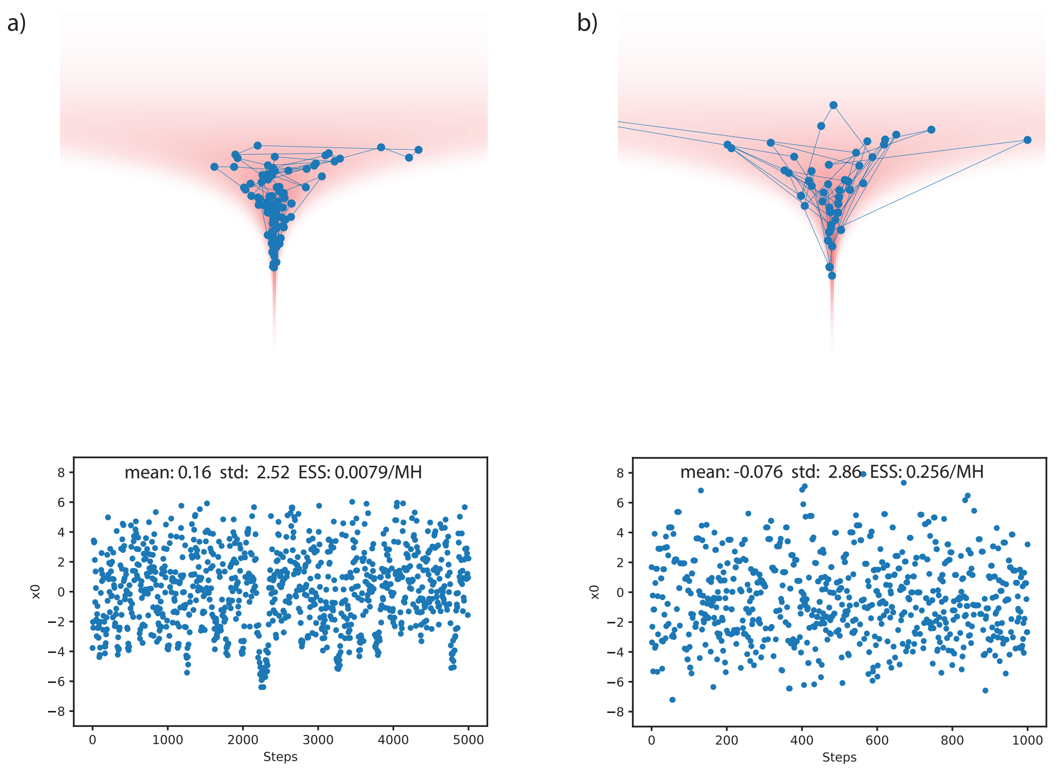


Figure 2. Comparison of our method with Hamiltonian Monte Carlo (HMC) on the 20d Funnel-3 distribution. (**a**) Chain and samples of x_0 (from neck to base direction) for HMC. (**b**) Same as (**a**) but for our learned sampler. Note, samples in (**a**) look significantly more correlated than those in (**b**), although they are plotted over a longer time scale.

As a demonstration we provide a visualization of the resulting chain of samples in Figure 2 and the learned proposal distributions in Appendix Figure A2. The energy value for the neck of the funnel can be very different than for the base, which makes it hard for methods such as HMC to mix between them [35]. In contrast, our model can produce very asymmetric $q(x'|x)$ and $q(x|x')$, making mixing between different energy levels possible.

Performances on other synthetic datasets and the Bayesian logistic regression are shown in Table 1. With all these datasets our method outperformed previous neural network-based MCMC approaches by significant margins. Our model also outperformed gradMALA [14], which uses the same objective but only use linear adaptive parameters. The experiments used various parameter settings, as detailed in Appendix A.2. Results of other models were adopted or converted from numbers reported in the original papers. The Appendix provides further experimental results, ablation studies, visualizations and details on the implementation of the model.

Table 1. Performance comparisons. SCG: strongly correlated Gaussian. ICG: ill-conditioned Gaussian. German, Australian, Heart: Datasets for Bayesian logistic regression. ESS: effective sample size (a correlation measure).

Dataset (Measure)	**L2HMC**		**Ours**
50d ICG (ESS/MH)	0.783		**0.86**
2d SCG (ESS/MH)	0.497		**0.89**
50d ICG (ESS/grad)	7.83×10^{-2}		$\mathbf{2.15 \times 10^{-1}}$
2d SCG (ESS/grad)	2.32×10^{-2}		$\mathbf{2.2 \times 10^{-1}}$
Dataset (Measure)	**Neutra**		**Ours**
Funnel-1 x_0 (ESS/grad)	8.9×10^{-3}		$\mathbf{3.7 \times 10^{-2}}$
Funnel-1 $x_{1\ldots99}$(ESS/grad)	4.9×10^{-2}		$\mathbf{7.2 \times 10^{-2}}$
Dataset (Measure)	**A-NICE-MC**	**NFLMC**	**Ours**
German (ESS/5k)	926.49	1176.8	**3150**
Australian (ESS/5k)	1015.75	1586.4	**2950**
Heart (ESS/5k)	1251.16	2000	**3600**

5.2. Training a Convergent Deep Energy-Based Model

A very challenging application of the MCMC method is training a deep energy-based model (EBM) of images [2,36,37]. We demonstrate stable training of a convergent EBM, and that the learned sampler achieves better proposal entropy early during training, as well as better sample quality at convergence, compared to the MALA algorithm. An added benefit is that, like in adaptive MALA, tuning the Langevin dynamics step size is no longer needed, instead, one only needs to specify a target accept rate. This contrasts earlier work using unadjusted Langevin dynamics, where step size needs to be carefully tuned [2].

Following [2], we used the Oxford flowers dataset of 8189 28 $\times$ 28 colored images. We dequantize the images to 5 bits by adding uniform noise and use logit transform [17]. Sampling was performed in the logit space with variant 2 of our sampler (without the R network; see Appendix A.1). During training, we used persistent contrastive divergence (PCD) [38] with a replay buffer size of 10,000. We alternated between training the sampler and updating samples for the EBM training. Each EBM training step uses 40 sampling steps, with a target accept rate of 0.6.

Figure 3 depicts samples from the trained EBM replay buffer, as well as samples from a 100,000 step sampling process—for demonstrating that in general the sampling sequence converges to a fixed low energy state. We also show that early during training, the proposal entropy of the learned sampler is higher than that of an adaptive MALA algorithm with the same accept rate target. Later during training, the proposal entropy is not significantly different (See Figure A3a). This is likely because the explorable volume around samples becomes too small for the learned sampler to make a difference. Additionally, we show that the model trained with the learned sampler achieved better sample quality by evaluating the Fréchet Inception Distance (FID) [39] between the replay buffer and ground truth data. A model trained with the learned sampler achieved 38.1 FID, while a model trained with MALA achieved 43.0 FID (evaluated at a late checkpoint, lower is better). We provide a plot that tracks the FID during training in Appendix A.3 Figure A3.

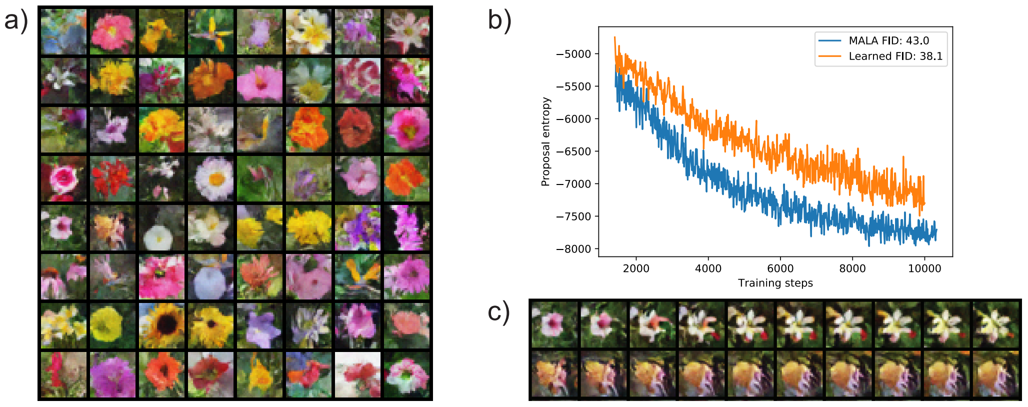


Figure 3. Training of the convergent energy-based model (EBM) with pixel space sampling. (**a**) Samples from replay buffer after training. (**b**) Proposal entropy of trained sampler vs. Metropolis-adjusted Langevin algorithm (MALA) early during training—note that the entropy of the learned sampler is significantly higher. (**c**) Samples from 100,000 sampling steps by the learned sampler, initialized at samples from replay buffer. Large transitions like the one in the first row are rare; this atypical example was selected for display.

6. Discussion

In this paper we propose a gradient based neural network MCMC sampler with tractable proposal probability. The training is based on the entropy-based exploration speed objective. Thanks to an objective that explicitly encourages exploration, our method achieves better performance than previous neural network-based MCMC samplers on a variety of tasks, sometimes by an order magnitude. We also improved the FID of samples from the trained EBM from 41 to 38. Compared to the manifold HMC [35] methods, our model provides a more scalable alternative for mitigating unfavorable geometry in the target distribution.

There are many potential applications of our method beyond what is demonstrated in this paper—for example, training latent-variable models [40], latent sampling in GANs [41] and others applications outside machine learning, such as molecular dynamics simulations [1]. For future research, it would be interesting to investigate other architectures, such as an auto-regressive architecture, or alternative masking strategies to improve the expressiveness of the proposed model. Another promising direction could be to combine our technique with a neural transport MCMC.

Our proposed sampler provides more efficient exploration of the target distribution, as measured by the ESS results and proposal entropy. However, our method still has the limitation that the exploration is local. In EBM training, for example, as reported previously [2], the learned energy landscape is highly multi-modal with high energy barriers in between the minims. A sample proposal cannot cross those high barriers since it will result in high rejection probability. Our sampler achieves a small level of mixing as is visible in some sampling examples. However, our sampler, being a local algorithm, cannot explore different modes efficiently—it exhibits the same shortcoming in mixing as Langevin dynamics sampler.

Author Contributions: Conceptualization, Z.L.; methodology, Z.L.; software, Z.L.; writing—original draft preparation, Z.L.; writing—review and editing, Z.L., Y.C. and F.T.S.; visualization, Z.L.; supervision, F.T.S.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by NSF award 1718991, NIH grant R01-EB026955 and by an INRC research grant from Intel Corporation.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We thank Hongye Hu for helpful suggestions on the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

Sample Availability: Code of this project is available.

Abbreviations

The following abbreviations are used in this manuscript:

MCMC	Markov Chain Monte Carlo
HMC	Hamiltonian Monte Carlo
EBM	Energy-based Model
MALA	Metropolis-Adjusted Langevin Algorithm
FID	Fréchet Inception Distance
GAN	Generative Adversarial Network
ESS	Effective Sample Size
MH	Metropolis-Hastings
A-NICE-MC	Adversarial Nonlinear Independent Component Estimation Monte Carlo
L2HMC	Learn to HMC
NFLMC	Normalizing Flow Langevine Monte Carlo
RealNVP	Real-Valued Non-Volume-Preserving flow

Appendix A

Appendix A.1. Ablation Study: Effect of Gradient Information

In the main text we propose to use gradient information in sampling, inspired by the success of gradient-based sampling techniques. Here we present an ablation study that shows the impact of gradient information and of some architectural simplifications. We compare the full model to two ablated model variants: 1. The original model with data distribution gradient set to 0, this is equivalent to generating proposals with a Real-NVP flow model conditioned on x. 2. A Langevin dynamics with noise generated by model variant 1. Specifically, variant 2 consists of the following proposal process $x' = x + \epsilon z - \frac{\epsilon^2}{2}\partial_x U(x) \odot \exp\left[S(x)\right] + T(x)$, where $z = f(z_0; x)$ is modeled by the original architecture with the gradient turned off, and S and T are neural networks. Thus, variant 2 implements Langevin dynamics with flexible noise and an element-wise affine transformed gradient. It is not flexible enough to express the full covariance Langevin dynamics [14], but we found it to be sufficient for energy-based model training. The computation time is significantly smaller than for our full model because it only uses per step only 2 gradient evaluations instead of $4N$ evaluations.

For training on the 100d Funnel-1 distribution, the learning rate of each model is tuned to the maximum stable value, with both models using the same learning rate schedule. As can be seen in Figure A1, both ablated models learn more slowly and have difficulty with mixing between energy levels, resulting in proposal distributions with poor coverage.

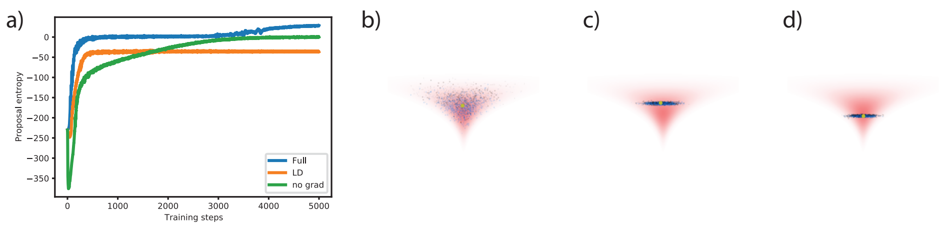


Figure A1. Illustrating the role of gradient information. (**a**) Comparison of the proposal entropy (up to a common constant) during training. Full: full model with gradient; LD: model variant 2; no grad: model variant 1. (**b**–**d**): Example proposal distribution of full, LD and no grad models.

Appendix A.2. Experimental Details

Code for our model, as well as a small demo experiment, can be found under this link https://github.com/NEM-MC/Entropy-NeuralMC (accessed on 10 February 2021).

Architecture and training details We use a single network for S, Q and T. The network is a 5-layer MLP with ELU activation function and constant width that depends on the dataset (See Table A1). The weights of both input and output layer are indexed with step number as a means to condition on the step number. The two substeps of the z update need not to be indexed as they use disjoint sets of input and output units indicated by the masks. The weights of all other layers are shared across different steps. We use a separate network with the same architecture and size for R and we condition on the step number in the same way. Weights of the output layers of all networks are initialized at 0. We use random masks that are drawn independently for each z update step.

When training the sampler, we use gradient clipping with L2 norm of 10. Additionally, we use a cosine annealing learning rate schedule. For training the Bayesian logistic regression task, we use a sample buffer of size equal to the batch size. For ESS evaluation we sample for 2000 steps and calculate the correlation function for samples from the later 1000 steps. One exception is the HMC 20d Funnel-3 result, where we sampled for 50000 steps. For the ESS/5k result used in Bayesian logistic regression task, we simply multiply the obtained ESS/MH value by 5000.

Other hyperparameters used in each experiments are listed in Table A1. All models are trained with batch size of 8192, for 5000 steps, except for 20d Funnel-3, where the batch size is 1024, and the number of training steps is $20,000$. The momentum parameters in the Adam optimizer are always set to $(0.9, 0.999)$. The scale parameter ϵ is chosen to be 0.01 for EBM training and 0.1 for all other experiments.

For deep EBM training we use architecture variant 2, described in Appendix A.1 because it uses less gradient computation. We use a small 4-layer convnet with 64 filters and ELU activation for S, Q and T. In the invertible network we use a fixed checkerboard mask. For the energy function in the EBM we used a 6 layer convnet with 64 filters and ELU activation function. In the EBM experiment we use 4 steps of z updates. We use a replay buffer with 10,000 samples, in each training step we first draw 64 samples randomly from the replay buffer and train the sampler for 5 consecutive steps. Updated samples are put back into the replay buffer. If the average accept rate of the batches is higher than target accept rate -0.1, another 128 samples are draw from the replay buffer and are updated 40 times using the sampler. These samples are then used as negative samples in the Maximum Likelihood training of the EBM. The EBM uses the Adam optimizer with a learning rate of 10^{-4} and Adam momentum parameters of $(0.5, 0.9)$. The sampler uses the same learning parameter, and has an accept rate target of 0.6. The EBM was trained for a total of 100,000 steps, where the learning rate is decreased by a factor of 0.2 at steps 80,000 and 90,000. All results with the MALA sampler are generated by loading a checkpoint at 1000 steps with the trained sampler. For reasons unclear to us, MALA is unable to stably initialize the training, although the later training process with MALA is stable. The FID is calculated for a model trained by learned sampler and trained by MALA at 82k steps.

Other datasets Recent studies of neural network-based samplers used datasets other than those reported in the Results. We experimented with applying our model on some of those datasets. In particular, we attempted applying our method to the rotated 100d ill-conditioned Gaussian task in [12], without getting satisfactory result. Our model is able to learn the 2d SCG tasks which shows that it is able to learn a non-diagonal covariance, but in this task it learns very slowly and does not achieve good performance. We believe this is because coupling-based architectures do not have the right inductive bias to efficiently learn the strongly non-diagonal covariance structure, perhaps the autoregressive architecture used in [12] would be more appropriate.

Table A1. Table for hyperparamters used in synthetic datasets and Bayesian logistic regression. Width: width of MLP networks. Steps: steps of updates in invertible model f. AR: target acceptance rate. LR: learning rate. Min LR: terminal learning rate of cosine annealing schedule.

	Width	**Steps**	**AR**	**LR**	**Min LR**
50d ICG	256	1	0.9	10^{-3}	10^{-5}
2d SCG	32	1	0.9	10^{-3}	10^{-5}
100d Funnel-1	512	3	0.7	10^{-3}	10^{-5}
20d Funnel-3	1024	4	0.6	5×10^{-4}	10^{-7}
German	128	1	0.7	10^{-3}	10^{-5}
Australian	128	1	0.8	10^{-3}	10^{-5}
Heart	128	1	0.9	10^{-3}	10^{-5}

Ref. [10] also presented the rough well distribution. We tried implementing it as described in the paper, but found that already a well-tuned HMC can easily sample from this distribution. Therefore we did not proceed to train our sampler on it. We should also add a comment regarding the 2d SCG task in [10]: In the paper the authors stated that the covariance is $[10^2, 10^{-2}]$, but in the provided code it is $[10^2, 10^{-1}]$, so we use the latter in our experiment.

Some neural network MCMC studies considered the mixture of Gaussian distribution. Our model optimizes a local exploration speed objective starting from small initialization. It is therefore inherently local and not able to explore modes far away and separated by high energy barriers. The temperature annealing trick in the L2HMC paper [10] does result in a sampler that can partially mix between the modes in a 2d mixture of Gaussian distribution. However, this approach cannot be expected to scale up to higher dimensions and more complicated datasets; therefore, we did not pursue it. We believe multi-modal target distributions are a separate problem that might be solvable with techniques such as the independent M–H sampler [28], but not with our current model.

Appendix A.3. Additional Experimental Results

Comparing computation time with HMC Here we compare the efficiency of our method to HMC in terms of actual execution speed (ESS/s) on Bayesian logistic regression tasks.

We follow [9] in using HMC with 40 Leapfrog steps and the optimal step size reported in this paper. The learned sampler and HMC are both executed on the same 2080Ti GPU with batch size 64. Results of this experiment are listed in Table A2. As can be seen, the learned model outperforms HMC significantly.

We did not compare run time with other neural network-based MCMC methods due to the difficulty of reproducing the previous models. In our experiment, the run time does not significantly depend on the batch size for batches as large as 10000, indicating that the execution speed is likely limited by other factors than the FLOPs (floating point operation per second) of the GPU. This makes execution speed a poor indicator of the true computation budget of the algorithm, but we still provide some execution speed results to comply with the community standard.

Visualization of proposal distributions on Funnel-3 distribution The proposal distributions learned on the 20d Funnel-3 distribution ars visualized in Figure A2. This demonstrates the ability of the sampler to adapt to the geometry of the target distribution. Further, it shows that the sampler can generate proposal points across regions of very different probability density, since the neck and base of the funnel have very different densities but proposals can travel between them easily.

Our sampler can achieve a significant speed up in terms of ESS/MH compared to HMC sampler, the improvement on the 20d funnel distribution is comparable to the one obtained by the Riemann Manifold HMC. However, the 100d funnel used in the manifold HMC paper could not be handled by our method.

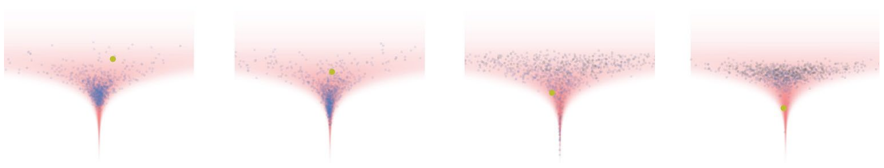


Figure A2. Visualizations of proposal distributions learned on the Funnel-3 distribution. Yellow dot: x. Blue dots: accepted x'. Black dots: rejected x'. The sampler has an accept rate of 0.6. Although not perfectly covering the target distribution, the proposed samples travel far from the previous sample and in a manner that complies with the geometry of the target distribution

Table A2. Comparing ESS/s between the learned sampler and HMC on the Bayesian logistic regression task. The learned sampler is significantly more efficient.

Dataset (Measure)	HMC	Ours
German (ESS/s)	772.7	**3651**
Australian (ESS/s)	127.2	**3433**
Heart (ESS/s)	997.1	**4235**

Further EBM results Figure A3 displays further results from the deep EBM training. (a) shows that the learned sampler achieves better proposal entropy early during training. (b) shows that the learned sampler converges faster than MALA, as indicated by the lower FID. (c) shows the EBM remains stable with a mixture of positive and negative energy differences between batch of real samples and a batch of samples from replay buffer. (d) Compares the L2 expected jump of MALA and the learned sampler, plotted in log scale. It has almost the exact same shape as the proposal entropy plot in the main text. (f) and (g) provide a sanity check showing that the learned sampler does not use a trivial solution. In g) the pixel-wise standard deviation of variable $z = f(z_0; x)$ (note we use variant 2 in Appendix A.1 which does not employ the gradient) is displayed after normalizing it into the image range. One can clearly see image-like structures similar to the sample of which the proposal is generated from. A MALA sampler would produce uniform images in this test, as z is just a Gaussian in MALA. This shows that the learned sampler is utilizing the structures of the samples to generate better proposals.

As shown in Figure A3a, MALA achieves similar proposal entropy if not slightly higher later during EBM training, while the learned sampler helps training initialization and early training. This suggests for future research that the optimal strategy could be to use the learned sampler initially and then switch to MALA once it produces similar proposal entropy.

Since we do not use noise initialization during EBM training, our model does not provide a meaningful gradient to re-sample from noise after the model has converged. This is quite different from what was reported in for example [42,43]. The combination of non-mixing as discussed earlier, and inability of sampling from noise brings the problem of not being able to obtain new samples of the model distribution after the EBM is trained, replay buffer is all we have to work with. Resolving this difficulty is left for future study.

Checks of correctness for the proposed sampler and the EBM training process We run some simple experiments to check the correctness of samples generated from the proposed sampler and show that the EBM training process is not biased by the learned sampler, see Figure A4.

To check if the learned sampler generates correct samples for the Bayesian logistic regression task, we compare dimension-wise mean, standard deviation and 4th moment of samples from the learned sampler and samples from the HMC sampler. We average over large amount of samples to obtain the moments, and the result matches very closely, indicating that the learned sampler samples from the correct target distribution, see Figure A4a.

Second, we sample an EBM energy function trained by the learned sampler with the MALA sampler, samples are initialized with samples from the replay buffer. As shown in

Figure A4b, MALA generate plausible samples from the model and does not cause issues such as saturated image [2], indicating that the learned EBM is not biased by the learned sampler and is indeed a valid energy function of the data distribution.

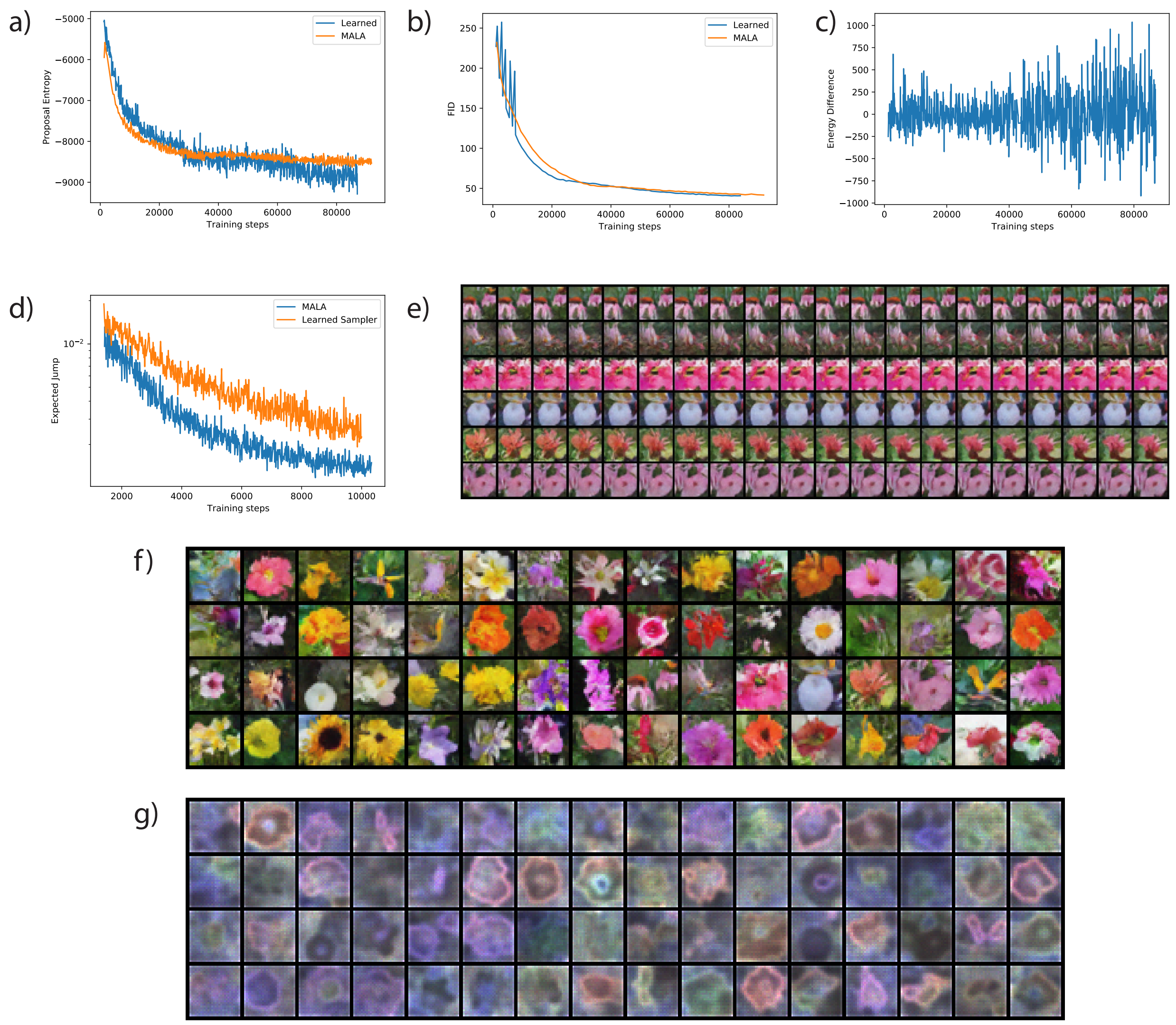
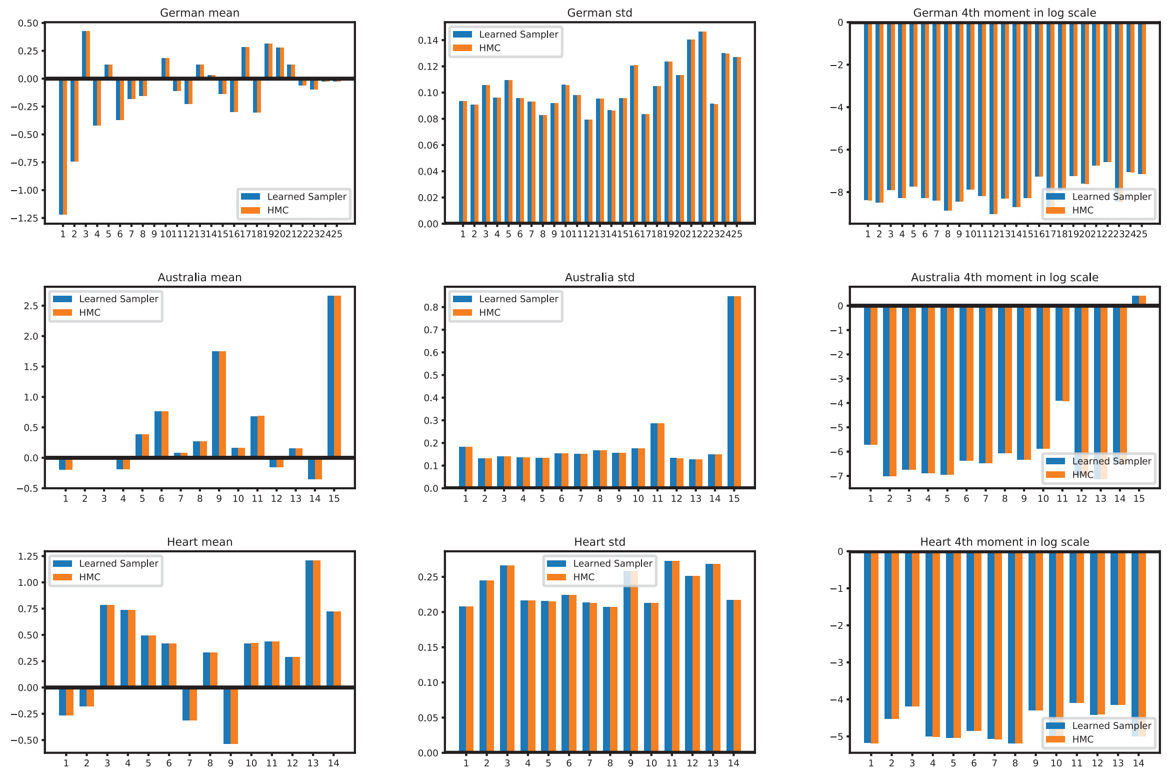


Figure A3. Further results for Deep EBM. (**a**–**c**) Proposal entropy, Fréchet Inception Distance (FID) of replay buffer and energy difference during training. Results for MALA are also included in (**a**,**b**). (**a**) shows that the learned sampler achieves better proposal entropy early during training. (**b**) shows that the learned sampler converges faster than MALA. (**c**) shows the EBM remains stable with a mixture of positive and negative energy difference. (**d**) Compares the L2 expected jump of MALA and the learned sampler, plotted in log scale. It has almost the exact same shape as the proposal's entropy plot in the main text. (**e**) More samples from sampling process of 100,000 steps with the learned sampler. (**f**,**g**) Samples from the replay buffer and the corresponding visualization of the pixel-wise variance of displacement vector z evaluated at the samples. Images in (**f**,**g**) are arranged in the same order. Image-like structures that depend on the sample of origin are clearly visible in (**g**). A MALA sampler would give uniform variance.

a)


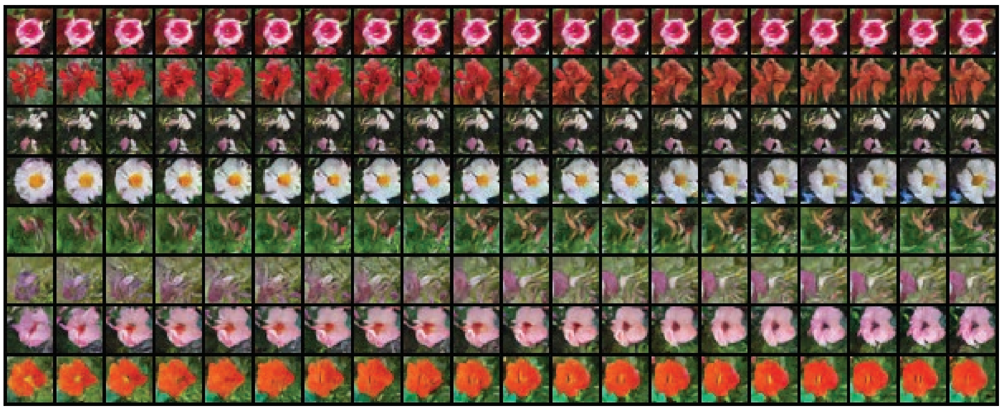
b)


Figure A4. Checking the correctness of samples and the EBM training process. (**a**) Comparing the dimension-wise means, standard deviations and 4th moments of samples obtained from HMC and the learned sampler on the Bayesian logistic regression datasets. Moments are matching very closely, indicating the learned sampler generates samples from the correct target distribution. (**b**) One-hundred-thousand sampling steps by MALA sampler on an EBM energy function trained by the adaptive sampler; samples are initialized from the replay buffer. Samples look plausible throughout the sampling process. This indicates that stable attractor basins are formed that are not specific to the learned sampler, and that EBM training is not biased by the adaptive sampler.

References

1. Noé, F.; Olsson, S.; Köhler, J.; Wu, H. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science* **2019**, *365*, eaaw1147. [CrossRef] [PubMed]
2. Nijkamp, E.; Hill, M.; Han, T.; Zhu, S.C.; Wu, Y.N. On the Anatomy of MCMC-based Maximum Likelihood Learning of Energy-Based Models. In Proceedings of the Conference on Artificial Intelligence (AAAI), New York, NY, USA, 7–12 February 2019.
3. Neal, R.M. *Probabilistic Inference Using Markov Chain Monte Carlo Methods*; Department of Computer Science, University of Toronto: Toronto, ON, Canada, 1993.
4. Neal, R.M. MCMC using Hamiltonian dynamics. *Handb. Markov Chain Monte Carlo* **2011**, *2*, 2.
5. Radivojević, T.; Akhmatskaya, E. Modified Hamiltonian Monte Carlo for Bayesian Inference. *Stat. Comput.* **2020**, *30*, 377–404. [CrossRef]

6. Beskos, A.; Pillai, N.; Roberts, G.; Sanz-Serna, J.M.; Stuart, A. Optimal tuning of the hybrid Monte Carlo algorithm. *Bernoulli* **2013**, *19*, 1501–1534. [CrossRef]
7. Betancourt, M.; Byrne, S.; Livingstone, S.; Girolami, M. The geometric foundations of hamiltonian monte carlo. *Bernoulli* **2017**, *23*, 2257–2298. [CrossRef]
8. Girolami, M.; Calderhead, B. Riemann manifold langevin and hamiltonian monte carlo methods. *J. R. Stat. Soc. Ser. B (Stat. Methodol.)* **2011**, *73*, 123–214. [CrossRef]
9. Song, J.; Zhao, S.; Ermon, S. A-nice-mc: Adversarial training for mcmc. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5140–5150.
10. Levy, D.; Hoffman, M.D.; Sohl-Dickstein, J. Generalizing Hamiltonian Monte Carlo with Neural Networks. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
11. Gu, M.; Sun, S.; Liu, Y. Dynamical Sampling with Langevin Normalization Flows. *Entropy* **2019**, *21*, 1096. [CrossRef]
12. Hoffman, M.; Sountsov, P.; Dillon, J.V.; Langmore, I.; Tran, D.; Vasudevan, S. Neutra-lizing bad geometry in hamiltonian monte carlo using neural transport. *arXiv* **2019**, arXiv:1903.03704.
13. Nijkamp, E.; Gao, R.; Sountsov, P.; Vasudevan, S.; Pang, B.; Zhu, S.C.; Wu, Y.N. Learning Energy-based Model with Flow-based Backbone by Neural Transport MCMC. *arXiv* **2020**, arXiv:2006.06897.
14. Titsias, M.; Dellaportas, P. Gradient-based Adaptive Markov Chain Monte Carlo. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 15730–15739.
15. Hastings, W. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **1970**, *57*, 97–109. [CrossRef]
16. Sohl-Dickstein, J.; Mudigonda, M.; DeWeese, M.R. Hamiltonian Monte Carlo without detailed balance. *arXiv* **2014**, arXiv:1409.5191.
17. Dinh, L.; Sohl-Dickstein, J.; Bengio, S. Density estimation using real nvp. *arXiv* **2016**, arXiv:1605.08803.
18. Kobyzev, I.; Prince, S.; Brubaker, M.A. Normalizing flows: Introduction and ideas. *arXiv* **2019**, arXiv:1908.09257.
19. Papamakarios, G.; Nalisnick, E.; Rezende, D.J.; Mohamed, S.; Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *arXiv* **2019**, arXiv:1912.02762.
20. Spanbauer, S.; Freer, C.; Mansinghka, V. Deep Involutive Generative Models for Neural MCMC. *arXiv* **2020**, arXiv:2006.15167.
21. Dinh, L.; Krueger, D.; Bengio, Y. Nice: Non-linear independent components estimation. *arXiv* **2014**, arXiv:1410.8516.
22. Marzouk, Y.; Moselhy, T.; Parno, M.; Spantini, A. An introduction to sampling via measure transport. *arXiv* **2016**, arXiv:1602.05023.
23. Langmore, I.; Dikovsky, M.; Geraedts, S.; Norgaard, P.; Von Behren, R. A Condition Number for Hamiltonian Monte Carlo. *arXiv* **2019**, arXiv:1905.09813.
24. Salimans, T.; Kingma, D.; Welling, M. Markov chain monte carlo and variational inference: Bridging the gap. In Proceedings of the International Conference on Machine Learning, Lille, France, 7–9 July 2015; pp. 1218–1226.
25. Zhang, Y.; Hernández-Lobato, J.M.; Ghahramani, Z. Ergodic measure preserving flows. *arXiv* **2018**, arXiv:1805.10377.
26. Postorino, M.N.; Versaci, M. A geometric fuzzy-based approach for airport clustering. *Adv. Fuzzy Syst.* **2014**, *2014*, 201243. [CrossRef]
27. Tkachenko, R.; Izonin, I.; Kryvinska, N.; Dronyuk, I.; Zub, K. An approach towards increasing prediction accuracy for the recovery of missing IoT data based on the GRNN-SGTM ensemble. *Sensors* **2020**, *20*, 2625. [CrossRef]
28. Neklyudov, K.; Egorov, E.; Shvechikov, P.; Vetrov, D. Metropolis-hastings view on variational inference and adversarial training. *arXiv* **2018**, arXiv:1810.07151.
29. Thin, A.; Kotelevskii, N.; Durmus, A.; Panov, M.; Moulines, E. Metropolized Flow: From Invertible Flow to MCMC. In Proceedings of the ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models, 12–18 July 2020; virtual event.
30. Pasarica, C.; Gelman, A. Adaptively scaling the Metropolis algorithm using expected squared jumped distance. *Stat. Sin.* **2010**, *20*, 343–364.
31. Poole, B.; Ozair, S.; Oord, A.V.d.; Alemi, A.A.; Tucker, G. On variational bounds of mutual information. *arXiv* **2019**, arXiv:1905.06922.
32. Song, J.; Ermon, S. Understanding the limitations of variational mutual information estimators. *arXiv* **2019**, arXiv:1910.06222.
33. Neal, R.M. Slice sampling. *Ann. Stat.* **2003**, *31*, 705–741. [CrossRef]
34. Hoffman, M.D.; Gelman, A. The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.* **2014**, *15*, 1593–1623.
35. Betancourt, M. A general metric for Riemannian manifold Hamiltonian Monte Carlo. In *Lecture Notes in Computer Science, Proceedings of the International Conference on Geometric Science of Information, Paris, France, 28–30 August 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 327–334.
36. Xie, J.; Lu, Y.; Zhu, S.C.; Wu, Y. A theory of generative convnet. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 2635–2644.
37. Du, Y.; Mordatch, I. Implicit generation and generalization in energy-based models. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Vancouver, BC, Canada, 8–14 December 2019.
38. Tieleman, T. Training restricted Boltzmann machines using approximations to the likelihood gradient. In Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, 5–9 July 2008; pp. 1064–1071.

39. Heusel, M.; Ramsauer, H.; Unterthiner, T.; Nessler, B.; Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 6626–6637.
40. Hoffman, M.D. Learning deep latent Gaussian models with Markov chain Monte Carlo. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 1510–1519.
41. Che, T.; Zhang, R.; Sohl-Dickstein, J.; Larochelle, H.; Paull, L.; Cao, Y.; Bengio, Y. Your GAN is Secretly an Energy-based Model and You Should use Discriminator Driven Latent Sampling. *arXiv* **2020**, arXiv:2003.06060.
42. Yu, L.; Song, Y.; Song, J.; Ermon, S. Training Deep Energy-Based Models with f-Divergence Minimization. In Proceedings of the International Conference on Machine Learning, Virtual Event, Vienna, Austria, 12–18 July 2020.
43. Grathwohl, W.; Wang, K.C.; Jacobsen, J.H.; Duvenaud, D.; Norouzi, M.; Swersky, K. Your classifier is secretly an energy based model and you should treat it like one. In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.