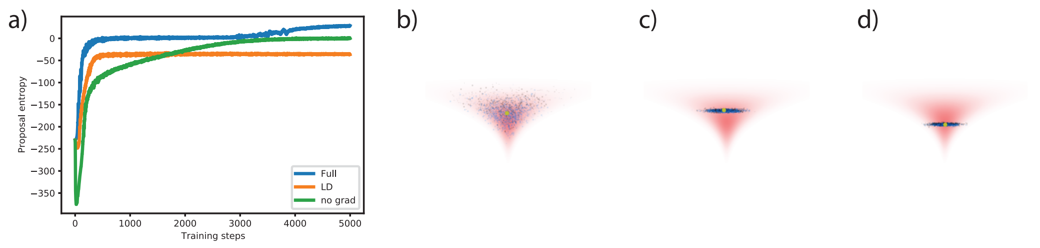


Figure A.1: Illustrating the role of gradient information. a) Comparison of proposal entropy (up to a common constant) during training. Full: full model using gradient, LD: modification 2. discussed in the text, no grad: modification 1 in the text. b), c), d): Example proposal distribution of Full, LD and no grad model.

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. Specifically we compare the full model to two 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 method 1, this variant is used in EBM training. (See below)

More specifically, variant 2 consist 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, T are neural networks. This model amounts to Langevin dynamics with a flexible noise and an element-wise affine transformed gradient. It is not flexible enough to express the full covariance Langevin dynamics (Titsias & Dellaportas, 2019), and does not perform well on low dimensional tasks, but we found it to be sufficient for energy-based model training. Variant 2 saves significant computation time because it only uses 2 gradient evaluation per step instead of $4N$ as in the original model.

We train on the 100d Funnel-1 distribution, in each model the learning rate is tuned to the maximum stable value, with the same learning rate schedule. As can be seen from the training curve and visualizations, the model that does not use gradient information learns more slowly and has difficulty with mixing between energy levels, resulting in proposal distributions with poor coverage.

A.2 Experimental details

Code for our model, as well as a small demo experiment, can be found under this link [link].

Architecture and training details We use a single network for S, Q and T. The network is a 5-layer MLP with ELU activation and constant width that depend on the dataset (See Table A.1). 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 z update need not 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 step number in the same way. Weights of all output layers of all networks are initialized at 0. We use random masks that are drawn independently for each 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.

	NN width	Steps	Accept rate target	Learning rate	Min learning rate
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}

Table A.1: Table for hyperparamters used in synthetic datasets and Bayesian Logistic regression. NN width: width of MLP network. Steps: Steps of updates in invertible model f. Min learning rate: terminal learning rate of cosine annealing schedule.

Other hyperparameters used in each experiments are listed in Table A.1. All models are trained with batch size of 8192, for 5000 steps, except for 20d Funnel-3, where the batch size is 1024, and is trained for 20000 steps. 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 section 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 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 10000 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 100000 steps, where the learning rate is decreased by a factor of 0.2 at step 80000 and 90000. All result using MALA sampler is generated by loading a checkpoint at 1000 steps with the trained sampler, since for reason unclear to us, MALA is unable to stably initialize the training, although training using MALA is stable later in the training process. The FID is calculated for model trained by learned sampler and trained by MALA at 82k steps.

Other datasets Recent work on 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 (Hoffman et al., 2019), without getting satisfactory result. Our model is able to learn the 2d SCG tasks which shows that it's 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 Hoffman et al. (2019) would be more appropriate.

(Levy et al., 2018) also presented the rough well distribution. We tried implementing it as described in the paper, but we found that 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 (Levy et al., 2018). 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 latter in our experiment.

Some other neural network MCMC studies consider 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 (Levy et al., 2018) 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

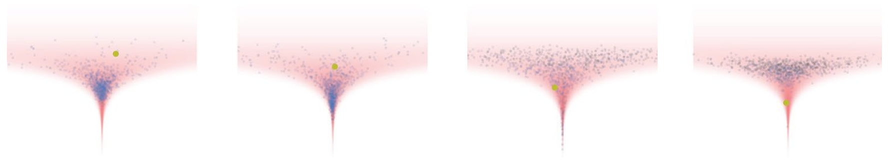


Figure A.2: Some visualizations of proposal distributions learned on the Funnel-3 distribution. Yellow dot: x, Blue dots: accepted x', Black dots: rejected x'. The sampler has an accept rate of 0.6. Although not perfectly covering the target distribution, the proposed samples travel far from the previous sample and in a manner that complies to the geometry of the target distribution

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

Table A.2: Comparing ESS/s between learned sampler and HMC on Bayesian Logistic regression task. Learned sampler is significantly more efficient.

didn't pursue it. It is our opinion the multi-modality of target distribution is a separate problem that probably should be solved by techniques such as independent M-H sampler (Neklyudov et al., 2018), not by our current model.

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 (Song et al., 2017) to use HMC with 40 Leapfrog steps and use the optimal step size reported in the same paper. The learned sampler and HMC are both executed on the same 2080Ti GPU with batch size 64. Result of this experiment is listed in Table A.2. As can be seen, the learned model outperforms HMC significantly.

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

Visualization of Proposal distributions on Funnel-3 distribution We show the visualization of proposal distributions learned on the 20d Funnel-3 distribution in Figure A.2. 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 significant speed up in terms of ESS/MH compared to HMC sampler, improvements comparable to Riemann Manifold HMC. However, the 100d funnel used in the manifold HMC paper still appears to be too difficult to our method, thus we used the easier 20d variant.

Further EBM results and discussion Figure A.3 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. c) shows the EBM remain stable with a mixture of positive and negative energy difference. d) Compares the L2 expected jump of MALA and 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 that shows the learned sampler do not use a trivial solution. In g) of Figure A.3, the pixel-wise standard deviation of variable $z = f(z_0; x)$ (note it does not use gradient here as we use variant 2 in Appendix A.1) is displayed after normalizing it into 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 sample to generate better proposals.

In our experiment MALA achieves similar proposal entropy if not slightly higher later during training, while the learned sampler helps training initialization and early training. This suggests that the optimal strategy could be to use the learned sampler initially and switch to MALA when MALA start to have similar proposal entropy. Investigating this is left for future work.

Our sampler provides more efficient exploration of the energy function *locally*, as measured by the proposal entropy. However, as reported previously Nijkamp et al. (2019), the learned energy landscape is highly multimodel with high energy barriers in between. Although a small level of mixing is visible in some sampling examples. Our sampler, being a local algorithm, cannot explore different modes efficiently, so we have the same non-mixing issue as previously reported. Additionally, as we do not use noise initialization, 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 (Yu et al., 2020; Grathwohl et al., 2019). The combination of none-mixing 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.

Check the correctness of 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 A.4.

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 A.4 a).

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 A.4 b), MALA generate plausible samples from the model and does not cause issue such as saturated image Nijkamp et al. (2019), indicating that the learned EBM is not biased by the learned sampler and is indeed a valid energy function of the data distribution.

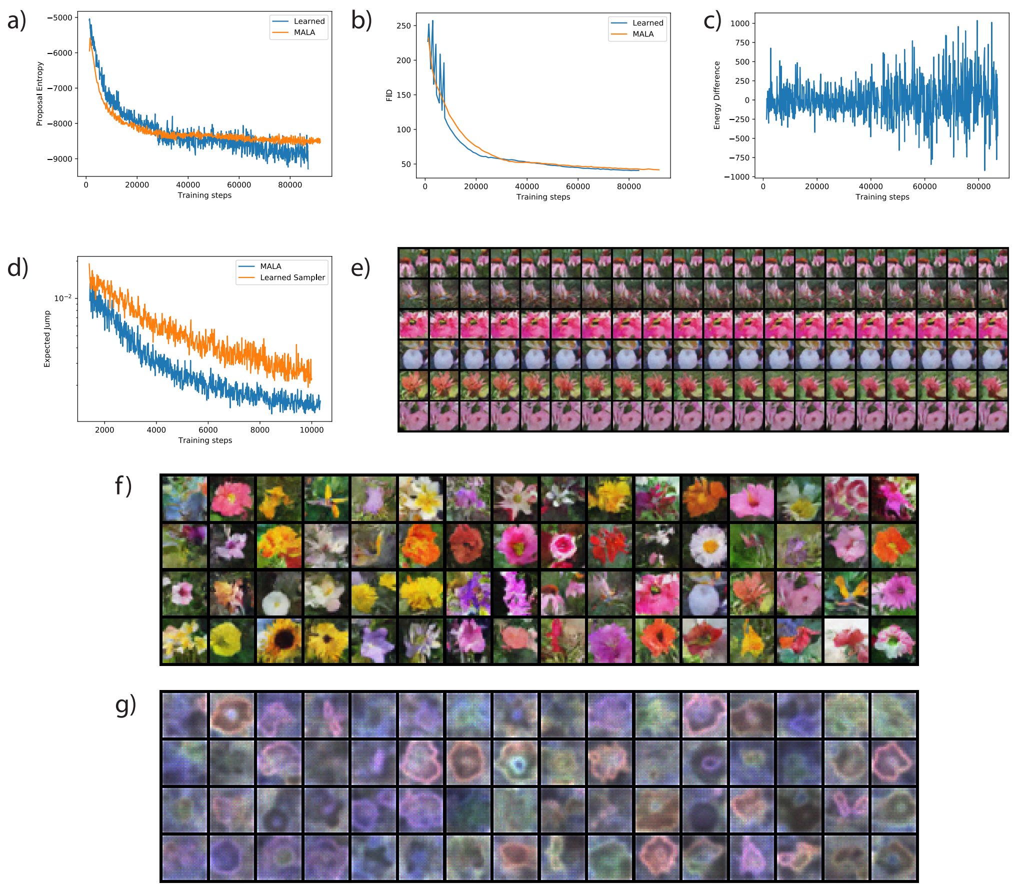
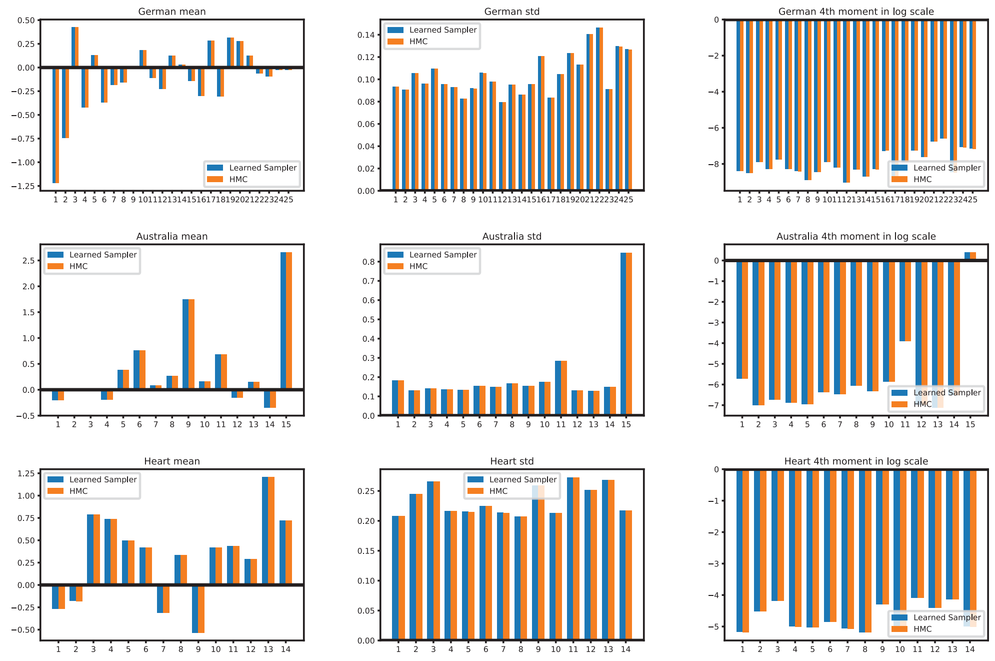


Figure A.3: Further results for Deep EBM. a), b) and c): Proposal Entropy, FID of replay buffer and energy difference during training. Result for MALA is also included in a) and b). a) shows that the learned sampler achieves better proposal entropy early during training. b) shows that the learned sampler converges faster than MALA. c) shows the EBM remain stable with a mixture of positive and negative energy difference. d) Compares L2 expected jump of MALA and learned sampler, plotted in log scale. It has almost the exact same shape as the proposal entropy plot in the main text. e) More samples from sampling process of 100k 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) and g) are arranged in the same order. Image-like structures that depends on the sample of origin are clearly visible in g). A MALA sampler would give uniform variance.

a)



b)

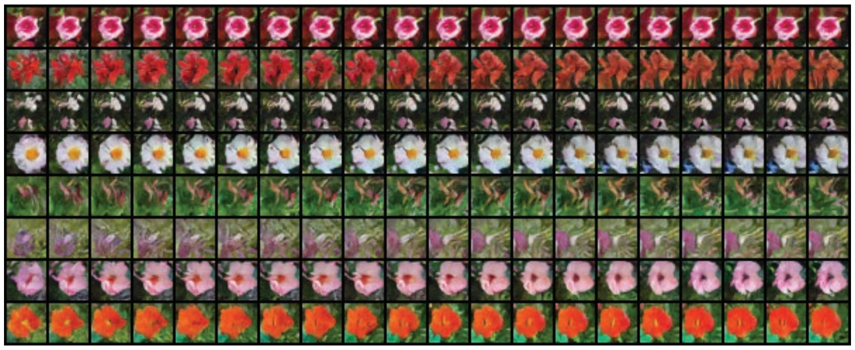


Figure A.4: Checking correctness of samples and EBM training process. a) Comparing dimension-wise mean, standard deviation and 4th moment of samples obtained from HMC and learned sampler for the Bayesian Logistic Regression datasets. Moments are matching very closely, indicating the learned sampler generate samples from the correct target distribution. b) 100k sampling steps by MALA sampler on an EBM energy function trained by the adaptive sampler, samples are initialized from the replay buffer. Samples looks plausible throughout the sampling process. This indicates that stable attractor basins formed are not specific to the learned sampler, and that EBM training is not biased by the adaptive sampler.