

---

# Stochastic Normalizing Flows

---

**Hao Wu**

Tongji University  
Shanghai, P.R. China  
wtian@gmail.com

**Jonas Köhler**

FU Berlin  
Berlin, Germany  
jonas.koehler@fu-berlin.de

**Frank Noé**

FU Berlin  
Berlin, Germany  
frank.noe@fu-berlin.de

## Abstract

The sampling of probability distributions specified up to a normalization constant is an important problem in both machine learning and statistical mechanics. While classical stochastic sampling methods such as Markov Chain Monte Carlo (MCMC) or Langevin Dynamics (LD) can suffer from slow mixing times there is a growing interest in using normalizing flows in order to learn the transformation of a simple prior distribution to the given target distribution. Here we propose a generalized and combined approach to sample target densities: Stochastic Normalizing Flows (SNF) – an arbitrary sequence of deterministic invertible functions and stochastic sampling blocks. We show that stochasticity overcomes expressivity limitations of normalizing flows resulting from the invertibility constraint, whereas trainable transformations between sampling steps improve efficiency of pure MCMC/LD along the flow. By invoking ideas from non-equilibrium statistical mechanics we derive an efficient training procedure by which both the sampler’s and the flow’s parameters can be optimized end-to-end, and by which we can compute exact importance weights without having to marginalize out the randomness of the stochastic blocks. We illustrate the representational power, sampling efficiency and asymptotic correctness of SNFs on several benchmarks including applications to sampling molecular systems in equilibrium.

## 1 Introduction

A common problem in machine learning and statistics with important applications in physics is the generation of asymptotically unbiased samples from a target distribution defined up to a normalization constant by means of an energy model  $u(\mathbf{x})$ :

$$\mu_X(\mathbf{x}) \propto \exp(-u(\mathbf{x})). \quad (1)$$

Sampling of such unnormalized distributions is often done with Markov Chain Monte Carlo (MCMC) or other stochastic sampling methods [13]. This approach is asymptotically unbiased, but suffers from the sampling problem: without knowing efficient moves, MCMC approaches may get stuck in local energy minima for a long time and fail to converge in practice.

Normalizing flows (NFs) [41, 40, 5, 35, 6, 33] combined with importance sampling methods are an alternative approach that enjoys growing interest in molecular and material sciences and nuclear physics [28, 25, 32, 22, 1, 30]. NFs are learnable invertible functions, usually represented by a neural network, pushing forward a probability density over a latent or “prior” space  $Z$  towards the target space  $X$ . Utilizing the change of variable rule these models provide exact densities of generated samples allowing them to be trained by either maximizing the likelihood on data (ML) or minimizing the Kullback-Leibler divergence (KL) towards a target distribution.

Let  $F_{ZX}$  be such a map and its inverse  $F_{XZ} = F_{ZX}^{-1}$ . We can consider it as composition of  $T$  invertible transformation layers  $F_0, \dots, F_T$  with intermediate states  $\mathbf{y}_t$  given by:

$$\mathbf{y}_{t+1} = F_t(\mathbf{y}_t) \quad \mathbf{y}_t = F_t^{-1}(\mathbf{y}_{t+1}) \quad (2)$$

By calling the samples in  $Z$  and  $X$  also  $\mathbf{z}$  and  $\mathbf{x}$ , respectively, the flow structure is as follows:

$$\begin{array}{ccccc} \mathbf{z} = \mathbf{y}_0 & \xrightarrow{F_0} & \mathbf{y}_1 \xrightarrow{\dots} \mathbf{y}_{T-1} & \xrightarrow{F_{T-1}} & \mathbf{y}_T = \mathbf{x} \\ & F_0^{-1} & & F_{T-1}^{-1} & \end{array} \quad (3)$$

We suppose each transformation layer is differentiable with a Jacobian determinant  $|\det \mathbf{J}_t(\mathbf{y})|$ . This allows to apply the *change of variable* rule:

$$p_{t+1}(\mathbf{y}_{t+1}) = p_{t+1}(F_t(\mathbf{y}_t)) = p_t(\mathbf{y}_t) |\det \mathbf{J}_t(\mathbf{y}_t)|^{-1}. \quad (4)$$

As we often work with log-densities, we abbreviate the log Jacobian determinant as:

$$\Delta S_t = \log |\det \mathbf{J}_t(\mathbf{y})|. \quad (5)$$

The log Jacobian determinant of the entire flow is defined by  $\Delta S_{ZX} = \sum_t \Delta S_t(\mathbf{y}_t)$  and correspondingly  $\Delta S_{XZ}$  for the inverse flow.

**Unbiased sampling with Boltzmann Generators.** Unbiased sampling is particularly important for applications in physics and chemistry where unbiased expectation values are required [25, 32, 1, 30]. A Boltzmann generator [32] utilizing NFs achieves this by (i) generating one-shot samples  $\mathbf{x} \sim p_X(\mathbf{x})$  from the flow and (ii) using a reweighing/resampling procedure respecting weights

$$w(\mathbf{x}) = \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})} \propto \exp(-u_X(\mathbf{x}) + u_Z(\mathbf{z}) + \Delta S_{ZX}(\mathbf{z})), \quad (6)$$

turning these one-shot samples into asymptotically unbiased samples. Reweighting/resampling methods utilized in this context are e.g. *Importance Sampling* [28, 32] or *Neural MCMC* [25, 1, 30].

**Training NFs.** NFs are trained in either “forward” or “reverse” mode, e.g.:

1. Density estimation – given data samples  $\mathbf{x}$ , train the flow such that the back-transformed samples  $\mathbf{z} = F_{XZ}(\mathbf{x})$  follow a latent distribution  $\mu_Z(\mathbf{z})$ , e.g.  $\mu_Z(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ . This is done by maximizing the likelihood – equivalent to minimizing the KL divergence  $KL[\mu_X \| p_X]$ .
2. Sampling of a given target density  $\mu_X(\mathbf{x})$  – sample from the simple distribution  $\mu_Z(\mathbf{z})$  and minimize a divergence between the distribution generated by the forward-transformation  $\mathbf{x} = F_{XZ}(\mathbf{z})$  and  $\mu_X(\mathbf{x})$ . A common choice is the reverse KL divergence  $KL[p_X \| \mu_X]$ .

We will use densities interchangeably with energies, defined by the negative logarithm of the density. The exact prior and target distributions are:

$$\mu_Z(\mathbf{z}) = Z_Z^{-1} \exp(-u_Z(\mathbf{z})) \quad \mu_X(\mathbf{x}) = Z_X^{-1} \exp(-u_X(\mathbf{x})) \quad (7)$$

with generally unknown normalization constants  $Z_Z$  and  $Z_X$ . As can be shown (Suppl. Material Sec. 1) minimizing  $KL[p_X \| \mu_X]$  or  $KL[\mu_X \| p_X]$  corresponds to maximizing the forward or backward weights of samples drawn from  $p_X$  or  $\mu_X$ , respectively.

**Topological problems of NFs.** A major caveat of sampling with exactly invertible functions for physical problems are topological constraints. While these can be strong manifold results, e.g., if the sample space is restricted to a non-trivial Lie group [11, 12], another practical problem are induced Bi-Lipschitz constraints resulting from mapping uni-modal base distributions onto well-separated multi-modal target distributions[4]. For example, when trying to map a unimodal Gaussian distribution to a bimodal distribution with affine coupling layers, a connection between the modes remains (Fig. 1a). This representational insufficiency poses serious problems during optimization – in the bimodal distribution example, the connection between the density modes seems largely determined by the initialization and does not move during optimization, leading to very different results in multiple runs (Suppl. Material, Fig. S1). More powerful coupling layers, e.g., [9], can mitigate this effect. Yet, as they are still diffeomorphic, strong Bi-Lipschitz requirements can make optimization difficult. This problem can be resolved when relaxing bijectivity of the flow by adding noise as we show in our results. Other proposed solutions are real-and-discrete mixtures of flows [7] or augmentation of the bases space [8, 18] at the cost of losing asymptotically unbiased sampling.

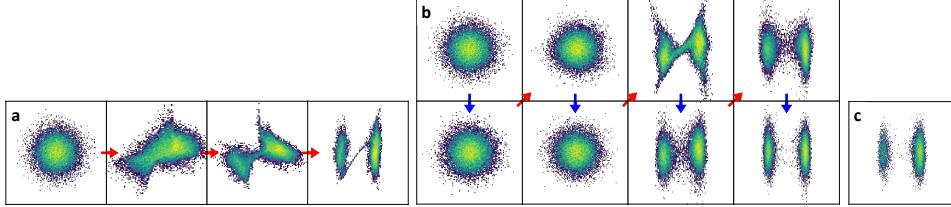


Figure 1: **Deterministic versus stochastic normalizing flow for the double well.** Red arrows indicate deterministic transformations, blue arrows indicate stochastic dynamics. **a)** 3 RealNVP blocks (2 layers each). **b)** Same with 20 BD steps before or after RealNVP blocks. **c)** Unbiased sample from true distribution.

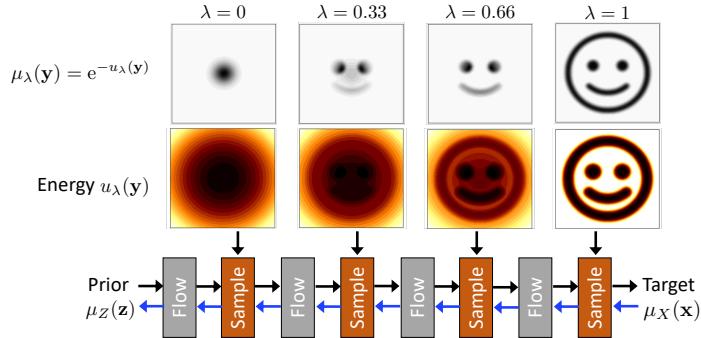


Figure 2: **Schematic for Stochastic Normalizing Flow (SNF).** An SNF transforms a tractable prior  $\mu_Z(\mathbf{z}) \propto \exp(-u_0(\mathbf{z}))$  to a complicated target distribution  $\mu_X(\mathbf{x}) \propto \exp(-u_1(\mathbf{x}))$  by a sequence of deterministic invertible transformations (flows, grey boxes) and stochastic dynamics (sample, ochre) that sample with respect to a guiding potential  $u_\lambda(\mathbf{x})$ . SNFs can be trained and run in forward mode (black) and reverse mode (blue).

**Contributions.** We show that NFs can be interwoven with stochastic sampling blocks into arbitrary sequences, that together overcome topological constraints and improve expressivity over deterministic flow architectures (Fig. 1a, b). Furthermore, NSF have improved sampling efficiency over pure stochastic sampling as the flow’s and sampler’s parameters can be optimized jointly.

Our main result is that NSF can be trained in a similar fashion as NFs and exact importance weights for each sample ending in  $\mathbf{x}$  can be computed, facilitating asymptotically unbiased sampling from the target density. The approach avoids explicitly computing  $p_X(\mathbf{x})$  which would require solving the intractable integral over all stochastic paths ending in  $\mathbf{x}$ .

We apply the model to the recently introduced problem of asymptotically unbiased sampling of molecular structures with flows [32] and show that it significantly improves sampling the multi-modal torsion angle distributions which are the relevant degrees of freedom in the system. We further show the advantage of the method over pure flow-based sampling / MCMC by quantitative comparison on benchmark data sets and on sampling from a VAE’s posterior distribution.

Code is available at [github.com/noegroup/stochastic\\_normalizing\\_flows](https://github.com/noegroup/stochastic_normalizing_flows)

## 2 Stochastic normalizing flows

A SNF is a sequence of  $T$  stochastic and deterministic transformations. We sample  $\mathbf{z} = \mathbf{y}_0$  from the prior  $\mu_Z$ , and generate a forward path  $(\mathbf{y}_1, \dots, \mathbf{y}_T)$  resulting in a proposal  $\mathbf{y}_T$  (Fig. 2). Correspondingly, latent space samples can be generated by starting from a sample  $\mathbf{x} = \mathbf{y}_T$  and invoking the backward path  $(\mathbf{y}_{T-1}, \dots, \mathbf{y}_0)$ . The conditional forward / backward path probabilities are

$$\mathbb{P}_f(\mathbf{z} = \mathbf{y}_0 \rightarrow \mathbf{y}_T = \mathbf{x}) = \prod_{t=0}^{T-1} q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}), \quad \mathbb{P}_b(\mathbf{x} = \mathbf{y}_T \rightarrow \mathbf{y}_0 = \mathbf{z}) = \prod_{t=0}^{T-1} \tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t) \quad (8)$$

where

$$\mathbf{y}_{t+1} | \mathbf{y}_t \sim q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}) \quad \mathbf{y}_t | \mathbf{y}_{t+1} \sim \tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t) \quad (9)$$

denote the forward / backward sampling density at step  $t$  respectively. If step  $t$  is a deterministic transformation  $F_t$  this simplifies as

$$\mathbf{y}_{t+1} \sim \delta(\mathbf{y}_{t+1} - F_t(\mathbf{y}_t)), \quad \mathbf{y}_t \sim \delta(\mathbf{y}_t - F_t^{-1}(\mathbf{y}_{t+1})).$$

In contrast to NFs, the probability that an SNF generates a sample  $\mathbf{x}$  cannot be computed by Eq. (4) but instead involves an integral over all paths that end in  $\mathbf{x}$ :

$$p_X(\mathbf{x}) = \int \mu_Z(\mathbf{y}_0) \mathbb{P}_f(\mathbf{y}_0 \rightarrow \mathbf{y}_T) d\mathbf{y}_0 \cdots d\mathbf{y}_{T-1}. \quad (10)$$

This integral is generally intractable, thus a feasible training method must avoid using Eq. (10). Following [31], we can draw samples  $\mathbf{x} \sim \mu_X(\mathbf{x})$  by running Metropolis-Hastings moves in the path-space of  $(\mathbf{z} = \mathbf{y}_0, \dots, \mathbf{y}_T = \mathbf{x})$  if we select the backward path probability  $\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})$  as the target distribution and the forward path probability  $\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})$  as the proposal density. Since we sample paths independently, it is simpler to assign an unnormalized importance weight proportional to the acceptance ratio to each sample path from  $\mathbf{z} = \mathbf{y}_0$  to  $\mathbf{x} = \mathbf{y}_T$ :

$$w(\mathbf{z} \rightarrow \mathbf{x}) = \exp \left( -u_X(\mathbf{x}) + u_Z(\mathbf{z}) + \sum_t \Delta S_t(\mathbf{y}_t) \right) \propto \frac{\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})}{\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})}, \quad (11)$$

where

$$\Delta S_t = \log \frac{\tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} \quad (12)$$

denotes the forward-backward probability ratio of step  $t$ , and corresponds to the usual change of variable formula in NF for deterministic transformation steps (Suppl. Material Sec. 3). These weights allow asymptotically unbiased sampling and training of SNFs while avoiding Eq. (10). By changing denominator and numerator in (11) we can alternatively obtain the backward weights  $w(\mathbf{x} \rightarrow \mathbf{z})$ .

**SNF training.** As in NFs, the parameters of a SNF can be optimized by minimizing the Kullback-Leibler divergence between the forward and backward path probabilities, or alternatively maximizing forward and backward path weights as long as we can compute  $\Delta S_t$  (Suppl. Material Sec 1):

$$J_{KL} = \mathbb{E}_{\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})} [-\log w(\mathbf{z} \rightarrow \mathbf{x})] = KL(\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x}) || \mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})) + \text{const.} \quad (13)$$

In the ideal case of  $J_{KL} = 0$ , all paths have the same weight  $w(\mathbf{z} \rightarrow \mathbf{x}) = 1$  and the independent and identically distributed sampling of  $\mu_X$  can be achieved. Accordingly, we can maximize the likelihood of the generating process on data drawn from  $\mu_X$  by minimizing:

$$J_{ML} = \mathbb{E}_{\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})} [-\log w(\mathbf{x} \rightarrow \mathbf{z})] = KL(\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z}) || \mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})) + \text{const.} \quad (14)$$

**Variational bound.** Minimization of the reverse path divergence  $J_{KL}$  minimizes an upper bound on the reverse KL divergence between the marginal distributions:

$$KL(p_X(\mathbf{x}) || \mu_X(\mathbf{x})) \leq KL(\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x}) || \mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})) \quad (15)$$

And the same relationship exists between the forward path divergence  $J_{ML}$  and the forward KL divergence. While invoking this variational approximation precludes us from explicitly computing  $p_X(\mathbf{x})$  and  $KL(p_X(\mathbf{x}) || \mu_X(\mathbf{x}))$ , we can still generate asymptotically unbiased samples from the target density  $\mu_X$ , unlike in variational inference.

**Asymptotically unbiased sampling.** As stated in the theorem below (Proof in Suppl. Material Sec. 2), SNFs are Boltzmann Generators: We can generate asymptotically unbiased samples of  $\mathbf{x} \sim \mu_X(\mathbf{x})$  by performing importance sampling or Neural MCMC using the path weight  $w(\mathbf{z}_k \rightarrow \mathbf{x}_k)$  of each path sample  $k$ .

**Theorem 1.** Let  $O$  be a function over  $X$ . An asymptotically unbiased estimator is given by

$$\mathbb{E}_{\mathbf{x} \sim \mu_X} [O(\mathbf{x})] \approx \frac{\sum_k w(\mathbf{z}_k \rightarrow \mathbf{x}_k) O(\mathbf{x}_k)}{\sum_k w(\mathbf{z}_k \rightarrow \mathbf{x}_k)}, \quad (16)$$

if paths are drawn from the forward path distribution  $\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})$ .

### 3 Implementing SNFs via Annealed Importance Sampling

In this paper we focus on the use of SNFs as samplers of  $\mu_X(\mathbf{x})$  for problems where the target energy  $u_X(\mathbf{x})$  is known, defining the target density up to a constant, and provide an implementation of stochastic blocks via MCMC / LD. These blocks make local stochastic updates of the current state  $\mathbf{y}$  with respect to some potential  $u_\lambda(\mathbf{y})$  such that they will asymptotically sample from  $\mu_\lambda(\mathbf{y}) \propto \exp(-u_\lambda(\mathbf{y}))$ . While such potentials  $u_\lambda(\mathbf{y})$  could be learned, a straightforward strategy is to interpolate between prior and target potentials

$$u_\lambda(\mathbf{y}) = (1 - \lambda)u_Z(\mathbf{y}) + \lambda u_X(\mathbf{y}), \quad (17)$$

similarly as it is done in *annealed importance sampling* [29]. Our implementation for SNFs is thus as follows: deterministic flow layers in-between only have to approximate the partial density transformation between adjacent  $\lambda$  steps while the stochastic blocks anneal with respect to the given intermediate potential  $u_\lambda$ . The parameter  $\lambda$  could again be learned – in this paper we simply choose a linear interpolation along the SNF layers:  $\lambda = t/T$ .

**Langevin dynamics.** Overdamped Langevin dynamics, also known as Brownian dynamics, using an Euler discretization with time step  $\Delta t$ , are given by [10]:

$$\mathbf{y}_{t+1} = \mathbf{y}_t - \epsilon_t \nabla u_\lambda(\mathbf{y}_t) + \sqrt{2\epsilon_t/\beta} \boldsymbol{\eta}_t, \quad (18)$$

where  $\boldsymbol{\eta}_t \sim \mathcal{N}(0, \mathbf{I})$  is Gaussian noise. In physical systems, the constant  $\epsilon_t$  has the form  $\epsilon_t = \Delta t/\gamma m$  with time step  $\Delta t$ , friction coefficient  $\gamma$  and mass  $m$ , and  $\beta$  is the inverse temperature (here set to 1). The backward step  $\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t$  is realized under these dynamics with the backward noise realization (Suppl. Material Sec. 4 and [31]):

$$\tilde{\boldsymbol{\eta}}_t = \sqrt{\frac{\beta\epsilon_t}{2}} [\nabla u_\lambda(\mathbf{y}_t) + \nabla u_\lambda(\mathbf{y}_{t+1})] - \boldsymbol{\eta}_t. \quad (19)$$

The log path probability ratio is (Suppl. Material Sec. 4):

$$\Delta S_t = -\frac{1}{2} (\|\tilde{\boldsymbol{\eta}}_t\|^2 - \|\boldsymbol{\eta}_t\|^2). \quad (20)$$

We also give the results for non-overdamped Langevin dynamics in Suppl. Material Sec. 5.

**Markov Chain Monte Carlo.** Consider MCMC methods with a proposal density  $q_t$  that satisfies the detailed balance condition w.r.t. the interpolated density  $\mu_\lambda(\mathbf{y}) \propto \exp(-u_\lambda(\mathbf{y}))$ :

$$\exp(-u_\lambda(\mathbf{y}_t)) q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}) = \exp(-u_\lambda(\mathbf{y}_{t+1})) q_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t) \quad (21)$$

We show that for all  $q_t$  satisfying (21), including Metropolis-Hastings and Hamiltonian MC moves, the log path probability ratio is (Suppl. Material Sec. 6 and 7):

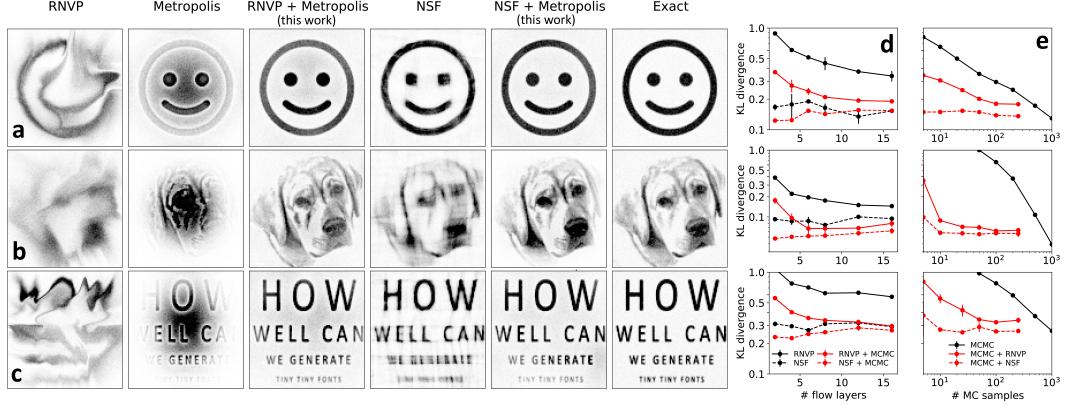
$$\Delta S_t = u_\lambda(\mathbf{y}_{t+1}) - u_\lambda(\mathbf{y}_t), \quad (22)$$

if the backward sampling density satisfies  $\tilde{q}_t = q_t$ .

## 4 Results

**Representational power versus sampling efficiency.** We first illustrate that SNFs can break topological constraints and improve the representational power of deterministic normalizing flows at a given network size and at the same time beat direct MCMC in terms of sampling efficiency. To this end we use images to define complex two-dimensional densities (Fig. 3a-c, ‘‘Exact’’) as target densities  $\mu_X(\mathbf{x})$  to be sampled. Note that a benchmark aiming at generating high-quality images would instead represent the image as a high-dimensional pixel array. We compare three types of flows with 5 blocks each trained by samples from the exact density (details in Suppl. Material Sec. 9):

1. Normalizing flow with 2 swapped coupling layers (RealNVP or neural spline flow) per block
2. Non-trainable stochastic flow with 10 Metropolis MC steps per block
3. SNF with both, 2 swapped coupling layers and 10 Metropolis MC steps per block.



**Figure 3: Sampling of two-dimensional densities.** **a-c)** Sampling of smiley, dog and text densities with different methods. Columns: (1) Normalizing Flow with RealNVP layers, (2) Metropolis MC sampling, (3) Stochastic Normalizing Flow combining (1+2), (4) neural spline flow (NSF), (5) Stochastic Normalizing Flow combining (1+4), (6) Unbiased sample from exact density. **d-e)** Compare representative power and statistical efficiency of different flow methods by showing KL divergence (mean and standard deviation over 3 training runs) between flow samples and true density for the three images from Fig. 3. **d)** Comparison of deterministic flows (black) and SNF (red) as a function of the number of RealNVP or Neural Spline Flow transformations. Total number of MC steps in SNF is fixed to 50. **e)** Comparison of pure Metropolis MC (black) and SNF (red, solid line RealNVP, dashed line Neural spline flow) as a function of the number of MC steps. Total number of RealNVP or NSF transformations in SNF is fixed to 10.

The pure Metropolis MC flow suffers from sampling problems – density is still concentrated in the image center from the prior. Many more MC steps would be needed to converge to the exact density (see below). The RealNVP normalizing flow architecture [6] has limited representational power, resulting in a “smeared out” image that does not resolve detailed structures (Fig. 3a-c, RNVP). As expected, neural spline flows perform significantly better on the 2D-images than RealNVP flows, but at the chosen network architecture their ability to resolve fine details and round shapes is still limited (See dog and small text in Fig. 3c, NSF). Note that the representational power for all flow architectures tend to increase with depth - here we compare the performance of different architectures at fixed depth and similar computational cost.

In contrast, SNFs achieve high-quality approximations although they simply combine the same deterministic and stochastic flow components that fail individually in the SNF learning framework (Fig. 3a-c, RNVP+Metropolis and NSF+Metropolis). This indicates that the SNF succeeds in performing the large-scale probability mass transport with the trainable flow layers and sampling the details with Metropolis MC.

Fig. 3d-e quantifies these impressions by computing the KL divergence between generated densities  $p_X(\mathbf{x})$  and exact densities  $\mu_X(\mathbf{x})$ . Both normalizing flows and SNFs improve with greater depth, but SNFs achieve significantly lower KL divergence at a fixed network depth (Fig. 3d). Note that both RealNVP and NSF improve significantly when stochasticity is added.

Moreover, SNFs have higher statistical efficiency than pure Metropolis MC flows. Depending on the example and flow architecture, 1-2 orders of magnitude more Metropolis MC steps are needed to achieve similar KL divergence as with an SNF. This demonstrates that the large-scale probability transport learned by the trainable deterministic flow blocks in SNFs significantly helps with the sampling.

Importantly, adding stochasticity is very inexpensive. Although every MCMC or Langevin integration step adds a neural network layer, these layers are very lightweighted, and have only linear computational complexity in the number of dimensions. As an example, for our SNF implementation of the examples in Fig. 3 we can add 10-20 stochastic layers to each trainable normalizing flow layer before the computational cost increases by a factor of 2 (Suppl. Material Fig. S2).

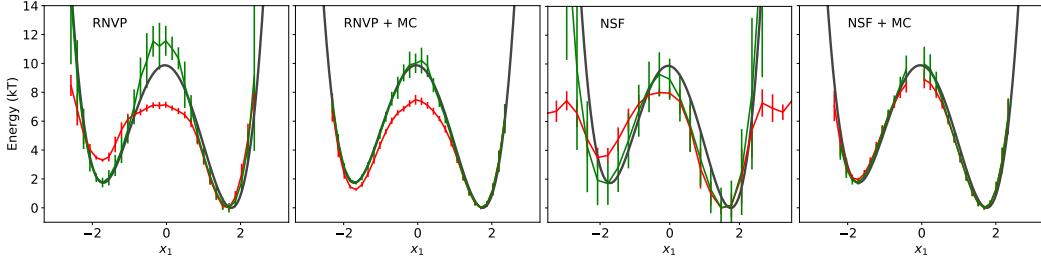


Figure 4: **Reweighting results for the double well potential** (see also Fig. 1). Free energy along  $x_1$  (negative log of marginal density) for deterministic normalizing flows (RNVP, NSF) and SNFs (RNVP+MC, NSF+MC). Black: exact energy, red: energy of proposal density  $p_X(\mathbf{x})$ , green: reweighted energy using importance sampling.

**SNFs as asymptotically unbiased samplers.** We demonstrate that SNFs can be used as Boltzmann Generators, i.e., to sample target densities without asymptotic bias by revisiting the double-well example (Fig. 1). Fig. 4 (black) shows the free energies (negative marginal density) along the double-well coordinate  $x_1$ . Flows with 3 coupling layer blocks (RealNVP or neural spline flow) are trained summing forward and reverse KL divergence as a joint loss using either data from a biased distribution, or with the unbiased distribution (Details in Suppl. Material Sec. 9). Due to limitations in representational power the generation probability  $p_X(\mathbf{x})$  will be biased – even when explicitly minimizing the KL divergence w.r.t. the true unbiased distribution in the joint loss. By relying on importance sampling we can turn the flows into Boltzmann Generators [32] in order to obtain unbiased estimates. Indeed all generator densities  $p_X(\mathbf{x})$  can be reweighted to an estimate of the unbiased density  $\mu_X(\mathbf{x})$  whose free energies are within statistical error of the exact result (Fig. 4, red and green).

We inspect the bias, i.e. the error of the mean estimator, and the statistical uncertainty ( $\sqrt{\text{var}}$ ) of the free energy in  $x_1 \in \{-2.5, 2.5\}$  with and without reweighting using a fixed number of samples (100,000). Using SNFs with Metropolis MC steps, both biases and uncertainties are reduced by half compared to purely deterministic flows (Table 1). Note that neural spline flows perform better than RealNVP without reweighting, but significantly worse with reweighting - presumably because the sharper features representable by splines can be detrimental for reweighting weights. With stochastic layers, both RealNVP and neural spline flows perform approximately equally well.

The differences between multiple runs (see standard deviations of the uncertainty estimate) also reduce significantly, i.e. SNF results are more reproducible than RealNVP flows, confirming that the training problems caused by the density connection between both modes (Fig. 1, Suppl. Material Fig. S1) can be reduced. Moreover, the sampling performance of SNF can be further improved by optimizing MC step sizes based on loss functions  $J_{KL}$  and  $J_{ML}$  (Suppl. Material Table S1).

Reweighting reduces the bias at the expense of a higher variance. Especially in physics applications, a small or asymptotically zero bias is often very important, and the variance can be reduced by generating more samples from the trained flow, which is relatively cheap and parallel.

Table 1: **Unbiased sampling for double well potential:** mean uncertainty of the reweighted energy along  $x_1$  averaged over 10 independent runs ( $\pm$  standard deviation).

	not reweighted			reweighted		
	bias	$\sqrt{\text{var}}$	$\sqrt{\text{bias}^2 + \text{var}}$	bias	$\sqrt{\text{var}}$	$\sqrt{\text{bias}^2 + \text{var}}$
RNVP	$1.4 \pm 0.6$	$0.4 \pm 0.1$	$1.5 \pm 0.5$	$0.3 \pm 0.2$	$1.1 \pm 0.4$	$1.2 \pm 0.4$
<b>RNVP + MC</b>	$1.5 \pm 0.2$	$0.3 \pm 0.1$	$1.5 \pm 0.2$	$0.2 \pm 0.1$	$0.6 \pm 0.1$	$0.6 \pm 0.1$
NSF	$0.8 \pm 0.4$	$1.0 \pm 0.2$	$1.3 \pm 0.3$	$0.6 \pm 0.2$	$2.1 \pm 0.4$	$2.2 \pm 0.5$
<b>NSF + MC</b>	$0.4 \pm 0.3$	$0.5 \pm 0.1$	$0.7 \pm 0.2$	$0.1 \pm 0.1$	$0.6 \pm 0.2$	$0.6 \pm 0.2$

**Alanine dipeptide.** We further evaluate SNFs on density estimation and sampling of molecular structures from a simulation of the alanine dipeptide molecule in vacuum (Fig. 5). The molecule has 66 dimensions in  $\mathbf{x}$ , and we augment it with 66 auxiliary dimensions in a second channel  $\mathbf{v}$ , similar

to “velocities” in a Hamiltonian flow framework [42], resulting in 132 dimensions total. The target density is given by  $\mu_X(\mathbf{x}, \mathbf{v}) = \exp\left(-u(\mathbf{x}) - \frac{1}{2}\|\mathbf{v}\|^2\right)$ , where  $u(\mathbf{x})$  is the potential energy of the molecule and  $\frac{1}{2}\|\mathbf{v}\|^2$  is the kinetic energy term.  $\mu_Z$  is an isotropic Gaussian normal distribution in all dimensions. We utilize the invertible coordinate transformation layer introduced in [32] in order to transform  $\mathbf{x}$  into normalized bond, angle and torsion coordinates. RealNVP transformations act between the  $\mathbf{x}$  and  $\mathbf{v}$  variable groups (Details in Suppl. Material Sec. 9).

We compare deterministic normalizing flows using 5 blocks of 2 RealNVP layers with SNFs that additionally use 20 Metropolis MC steps in each block totalling up to 100 MCMC steps in one forward pass. Fig. 5a shows random structures sampled by the trained SNF. Fig. 5b shows marginal densities in all five multimodal torsion angles (backbone angles  $\phi, \psi$  and methyl rotation angles  $\gamma_1, \gamma_2, \gamma_3$ ). While the RealNVP networks that are state of the art for this problem miss many of the modes, the SNF resolves the multimodal structure and approximates the target distribution better, as quantified in the KL divergence between the generated and target marginal distributions (Table 2).

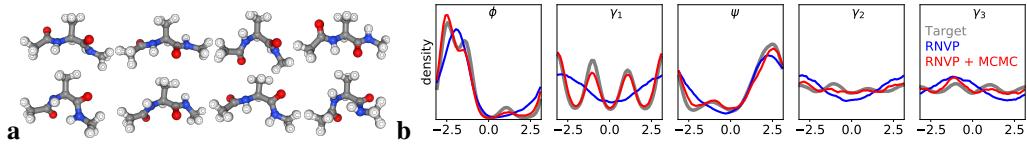


Figure 5: **Alanine dipeptide** sampled with deterministic normalizing flows and stochastic normalizing flows. **a)** One-shot SNF samples of alanine dipeptide structures. **b)** Energy (negative logarithm) of marginal densities in 5 unimodal torsion angles (top) and all 5 multimodal torsion angles (bottom).

Table 2: **Alanine dipeptide:** KL-divergences of RNVP flow and SNF (RNVP+MCMC) between generated and target distributions for all multimodal torsion angles. Mean and standard deviation from 3 independent runs.

KL-div.	$\phi$	$\gamma_1$	$\psi$	$\gamma_2$	$\gamma_3$
RNVP	$1.69 \pm 0.03$	$3.82 \pm 0.01$	$0.98 \pm 0.03$	$0.79 \pm 0.03$	$0.79 \pm 0.09$
SNF	<b><math>0.36 \pm 0.05</math></b>	<b><math>0.21 \pm 0.01</math></b>	<b><math>0.27 \pm 0.03</math></b>	<b><math>0.12 \pm 0.02</math></b>	<b><math>0.15 \pm 0.04</math></b>

**Variational Inference.** Finally, we use normalizing flows to model the latent space distribution of a variational autoencoder (VAE), as suggested in [35]. Table 3 shows results for the variational bound and the log likelihood on the test set for MNIST [23] and Fashion-MNIST [44]. For a 50-dimensional latent space we compare a six-layer RNVP to MCMC using overdamped Langevin dynamics as proposal (MCMC) and a SNF combining both (RNVP+MCMC). Both sampling and the deterministic flow improve over a naive VAE using a reparameterized diagonal Gaussian variational posterior distribution, while the SNF outperforms both, RNVP and MCMC. See Suppl. Material Sec. 8 for details.

Table 3: **Variational inference using VAEs with stochastic normalizing flows:**  $J_{KL}$ : variational bound of the KL-divergence computed during training. NLL: negative log likelihood of test set.

	MNIST		Fashion-MNIST	
	$J_{KL}$	NLL	$J_{KL}$	NLL
Naive (Gaussian)	$108.4 \pm 24.3$	$98.1 \pm 4.2$	$241.3 \pm 7.4$	$238.0 \pm 2.9$
RNVP	$91.8 \pm 0.4$	$87.0 \pm 0.2$	$233.7 \pm 0.1$	$231.4 \pm 0.2$
MCMC	$102.1 \pm 8.0$	$96.2 \pm 1.9$	$234.7 \pm 0.4$	$235.2 \pm 2.4$
SNF	<b><math>89.7 \pm 0.1</math></b>	<b><math>86.8 \pm 0.1</math></b>	<b><math>232.4 \pm 0.2</math></b>	<b><math>230.9 \pm 0.2</math></b>

## 5 Related work

The cornerstone of our work is nonequilibrium statistical mechanics. Particularly important is Nonequilibrium Candidate Monte Carlo (NCMC) [31], which provides the theoretical framework to compute SNF path likelihood ratios. However, NCMC is for fixed deterministic and stochastic protocols, while we generalize this into a generative model by substituting fixed protocols with trainable layers and deriving an unbiased optimization procedure.

Neural stochastic differential equations learn optimal parameters of designed stochastic processes from observations along the path [43, 19, 27, 26], but are not designed for marginal density estimation or asymptotically unbiased sampling. It has been demonstrated that combining learnable proposals/transformations with stochastic sampling techniques can improve expressiveness of the proposals [36, 24, 39, 17, 16]. Yet, these contributions do not provide an exact reweighing scheme based on a tractable model likelihood and do not provide efficient algorithms to optimize arbitrary sequences of transformation or sampling steps end-to-end efficiently. These methods can be seen as instances of SNFs with specific choice of deterministic transformations and / or stochastic blocks and model-specific optimizations - see Suppl. Material Table S2) for a categorization. While our experiments focus on nontrainable stochastic blocks, the proposal densities of MC steps can also be optimized within the framework of SNFs as shown in Suppl. Material Table S1.

An important aspect of SNFs compared to trainable Monte-Carlo kernels such as A-NICE-MC [39] is the use of detailed balance (DB). While Monte-Carlo frameworks are usually designed to use DB in each step, SNFs rely on path-based detailed balance between the prior and the target density. This means that SNFs can also perform nonequilibrium moves along the transformation, as done by Langevin dynamics without acceptance step and by the deterministic flow transformations such as RealNVP and neural spline flows.

More closely related is [37] which uses of stochastic flows for density estimation and trains diffusion kernels by maximizing a variational bound of the model likelihood. Their derivation using stochastic paths is similar to ours and this work can be seen as a special instance of SNFs, but it does not consider more general stochastic and deterministic building blocks and does not discuss the problem of asymptotically unbiased sampling of a target density. Ref. [2] proposes a learnable stochastic process by integrating Langevin dynamics with learnable drift and diffusion term. This approach is in a spirit similar as our proposed method, but requires variational approximation of the generative distribution and it has not been worked out how it could be used as a building block within a NF. The approach of [14] combines NF layers with Langevin dynamics, yet approximates the intractable integral with MC samples which we can avoid utilizing the path-weight derivation. Finally, [15] propose a stochastic extension to neural ODEs [3] which can then be trained as samplers. This approach to sampling is very general yet requires costly integration of a SDE which we can avoid by combining simple NFs with stochastic layers.

## 6 Conclusions

We have introduced stochastic normalizing flows (SNFs) that combine both stochastic processes and invertible deterministic transformations into a single learning framework. By leveraging nonequilibrium statistical mechanics we show that SNFs can efficiently be trained to sample asymptotically unbiased from target densities. This can be done by utilizing path probability ratios and avoiding intractable marginalization. Besides possible applicability in classical machine learning domains such as variational and Bayesian inference, we believe that the latter property can make SNFs a key component in the efficient sampling of many-body physics systems. In future research we aim to apply SNFs with many stochastic sampling steps to accurate large-scale sampling of molecules.

## Broader Impact

The sampling of probability distributions defined by energy models is a key step in the rational design of pharmacological drug molecules for disease treatment, and the design of new materials, e.g., for energy storage. Currently such sampling is mostly done by Molecular Dynamics (MD) and MCMC simulations, which is in many cases limited by computational resources and generates extremely high energy costs. For example, the direct simulation of a single protein-drug binding and dissociation event could require the computational time of an entire supercomputer for a year. Developing machine learning (ML) approaches to solve this problem more efficiently and go beyond existing enhanced sampling methods is therefore of importance for applications in medicine and material science and has potentially far-reaching societal consequences for developing better treatments and reducing energy consumption. Boltzmann Generators, i.e. the combination of Normalizing Flows (FNs) and resampling/reweighting are a new and promising ML approach to this problem and the current paper adds a key technology to overcome some of the previous limitations of NFs for this task.

A risk of the method is that flow-based sampling bears the risk that non-ergodic samplers can be constructed, i.e. samplers that are not guaranteed to sample from the target distribution even in the limit of long simulation time. From such an incomplete sample, wrong conclusions can be drawn. While incomplete sampling is also an issue with MD/MCMC, it is well understood how to at least ensure ergodicity of these methods in the asymptotic limit, i.e. in the limit of generating enough data. Further research is needed to obtain similar results with normalizing flows.

**Acknowledgements.** Special thanks to José Miguel Hernández Lobarto (University of Cambridge) for his valuable input on the path probability ratio for MCMC and HMC. We acknowledge funding from the European Commission (ERC CoG 772230 ScaleCell), Deutsche Forschungsgemeinschaft (GRK DAEDALUS, SFB1114/A04), the Berlin Mathematics center MATH+ (Project AA1-6 and EF1-2) and the Fundamental Research Funds for the Central Universities of China (22120200276).

## References

- [1] M. S. Albergo, G. Kanwar, and P. E. Shanahan. “Flow-based generative models for Markov chain Monte Carlo in lattice field theory”. In: *Phys. Rev. D* 100 (2019), p. 034515.
- [2] Changyou Chen et al. “Continuous-time flows for efficient inference and density estimation”. In: *arXiv preprint arXiv:1709.01179* (2017).
- [3] Tian Qi Chen et al. “Neural ordinary differential equations”. In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.
- [4] Rob Cornish et al. “Relaxing bijectivity constraints with continuously indexed normalising flows”. In: *arXiv preprint arXiv:1909.13833* (2019).
- [5] Laurent Dinh, David Krueger, and Yoshua Bengio. “Nice: Non-linear independent components estimation”. In: *arXiv preprint arXiv:1410.8516* (2014).
- [6] Laurent Dinh, Jasha Sohl-Dickstein, and Samy Bengio. “Density estimation using Real NVP”. In: *arXiv:1605.08803* (2016).
- [7] Laurent Dinh et al. “A RAD approach to deep mixture models”. In: *arXiv preprint arXiv:1903.07714* (2019).
- [8] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. “Augmented neural odes”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 3134–3144.
- [9] Conor Durkan et al. “Neural spline flows”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 7509–7520.
- [10] D. L. Ermak and Y. Yeh. “Equilibrium electrostatic effects on the behavior of polyions in solution: polyion-mobile ion interaction”. In: *Chem. Phys. Lett.* 24 (1974), pp. 243–248.
- [11] Luca Falorsi et al. “Explorations in homeomorphic variational auto-encoding”. In: *arXiv preprint arXiv:1807.04689* (2018).
- [12] Luca Falorsi et al. “Reparameterizing Distributions on Lie Groups”. In: *arXiv preprint arXiv:1903.02958* (2019).
- [13] Daan Frenkel and Berend Smit. *Understanding molecular simulation*. Academic Press, 2001.
- [14] Minghao Gu, Shiliang Sun, and Yan Liu. “Dynamical Sampling with Langevin Normalization Flows”. In: *Entropy* 21.11 (2019), p. 1096.

- [15] Liam Hodgkinson et al. “Stochastic Normalizing Flows”. In: *arXiv preprint arXiv:2002.09547* (2020).
- [16] Matthew D. Hoffman. “Learning deep latent Gaussian models with Markov chain Monte Carlo”. In: *International Conference on Machine Learning* (2017), pp. 1510–1519.
- [17] Matthew Hoffman et al. “Neutra-lizing bad geometry in Hamiltonian Monte Carlo using neural transport”. In: *arXiv preprint arXiv:1903.03704* (2019).
- [18] Chin-Wei Huang, Laurent Dinh, and Aaron Courville. “Augmented normalizing flows: Bridging the gap between generative flows and latent variable models”. In: *arXiv preprint arXiv:2002.07101* (2020).
- [19] Junteng Jia and Austin R. Benson. “Neural Jump Stochastic Differential Equations”. In: *arXiv preprint arXiv:1905.10403* (2019).
- [20] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980* (2014).
- [21] D. P. Kingma and M. Welling. “Auto-Encoding Variational Bayes”. In: *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, *arXiv:1312.6114*. 2014.
- [22] Jonas Köhler, Leon Klein, and Frank Noé. “Equivariant Flows: sampling configurations for multi-body systems with symmetric energies”. In: *arXiv preprint arXiv:1910.00753* (2019).
- [23] Y. LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proc. IEEE 86* (1998), pp. 2278–2324.
- [24] Daniel Levy, Matthew D Hoffman, and Jascha Sohl-Dickstein. “Generalizing hamiltonian monte carlo with neural networks”. In: *arXiv preprint arXiv:1711.09268* (2017).
- [25] Shuo-Hui Li and Lei Wang. “Neural Network Renormalization Group”. In: *Phys. Rev. Lett.* 121 (2018), p. 260601.
- [26] Xuechen Li et al. “Scalable Gradients for Stochastic Differential Equations”. In: *arXiv preprint arXiv:2001.01328* (2020).
- [27] Xuanqing Liu et al. “Neural SDE: Stabilizing neural ode networks with stochastic noise”. In: *arXiv preprint arXiv:1906.02355* (2019).
- [28] Thomas Müller et al. “Neural importance sampling”. In: *arXiv preprint arXiv:1808.03856* (2018).
- [29] Radford M. Neal. “Annealed Importance Sampling”. In: *arXiv preprint arXiv:physics/9803008* (1998).
- [30] Kim A. Nicoli et al. “Asymptotically unbiased estimation of physical observables with neural samplers”. In: *arXiv:1910.13496* (2019).
- [31] Jerome P. Nilmeier et al. “Nonequilibrium candidate Monte Carlo is an efficient tool for equilibrium simulation”. In: *Proc. Natl. Acad. Sci. USA* 108 (2011), E1009–E1018.
- [32] Frank Noé et al. “Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning”. In: *Science* 365.6457 (2019), eaaw1147.
- [33] George Papamakarios et al. “Normalizing Flows for Probabilistic Modeling and Inference”. In: *arXiv preprint arXiv:1912.02762* (2019).
- [34] D. A. Pearlman et al. “AMBER, a computer program for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to elucidate the structures and energies of molecules”. In: *Comp. Phys. Commun.* 91 (1995), pp. 1–41.
- [35] Danilo Jimenez Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *arXiv preprint arXiv:1505.05770* (2015).
- [36] Tim Salimans, Diederik Kingma, and Max Welling. “Markov chain monte carlo and variational inference: Bridging the gap”. In: *International Conference on Machine Learning*. 2015, pp. 1218–1226.
- [37] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *arXiv preprint arXiv:1503.03585* (2015).
- [38] J. Song, S. Zhao, and S. Ermon. “A-NICE-MC: Adversarial Training for MCMC”. In: *NIPS* (2017).
- [39] Jiaming Song, Shengjia Zhao, and Stefano Ermon. “A-nice-mc: Adversarial training for mcmc”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5140–5150.
- [40] Esteban G Tabak and Cristina V Turner. “A family of nonparametric density estimation algorithms”. In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164.

- [41] Esteban G Tabak, Eric Vanden-Eijnden, et al. “Density estimation by dual ascent of the log-likelihood”. In: *Communications in Mathematical Sciences* 8.1 (2010), pp. 217–233.
- [42] Peter Toth et al. “Hamiltonian Generative Networks”. In: *arXiv preprint arXiv:1909.13789* (2019).
- [43] Belinda Tzen and Maxim Raginsky. “Neural stochastic differential equations: Deep latent Gaussian models in the diffusion limit”. In: *arXiv preprint arXiv:1905.09883* (2019).
- [44] Han Xiao, Kashif Rasul, and Roland Vollgraf. “Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms”. In: *arXiv preprint arXiv:cs.LG/1708.07747* (2017).

## Supplementary Material

### 1. Training normalizing flows

**Energy-based training and forward weight maximization.** If the target density  $\mu_X$  is known up to a constant  $Z_X$ , we minimize the forward KL divergence between the generated and the target distribution.

$$\begin{aligned} & \text{KL}(p_X \parallel \mu_X) \\ &= \mathbb{E}_{\mathbf{x} \sim p_X(\mathbf{x})} [\log p_X(\mathbf{x}) - \log \mu_X(\mathbf{x})] \\ &= \mathbb{E}_{\mathbf{z} \sim \mu_Z(\mathbf{z})} [u_X(F_{ZX}(\mathbf{z})) - \Delta S_{ZX}(\mathbf{z})] + \text{const.} \end{aligned} \quad (23)$$

The importance weights wrt the target distribution can be computed as:

$$w_X(\mathbf{x}) = \exp(-u_X(F_{ZX}(\mathbf{z})) + u_Z(\mathbf{z}) + \Delta S_{ZX}(\mathbf{z})) \propto \frac{\mu_X(\mathbf{x})}{p_X(\mathbf{x})}. \quad (24)$$

As  $\mathbb{E}_{\mathbf{z} \sim p_Z(\mathbf{z})} [u_Z(\mathbf{z})]$  is a constant, we can equivalently minimize KL or maximize log weights:

$$\max \mathbb{E}_{\mathbf{z} \sim p_Z(\mathbf{z})} [\log w_X(\mathbf{x})] = \min \text{KL}(p_X \parallel \mu_X), \quad (25)$$

**Maximum likelihood and backward weight maximization.** The backward KL divergence  $\text{KL}(\mu_X \parallel p_X)$  is not always tractable as  $\mu_X(\mathbf{x})$  can be difficult to sample from. Replacing  $\mu_X(\mathbf{x})$  by the empirical data distribution  $\rho_X(\mathbf{x})$ , the KL becomes a negative log-likelihood:

$$\begin{aligned} & \text{NLL}(\rho_X \parallel p_X) \\ &= \mathbb{E}_{\mathbf{x} \sim \rho_X(\mathbf{x})} [u_Z(F_{XZ}(\mathbf{x})) - \Delta S_{XZ}(\mathbf{x})] + \text{const.} \end{aligned} \quad (26)$$

Using  $\mathbb{E}_{\mathbf{x} \sim \rho_X(\mathbf{x})} [-\log \rho_X(\mathbf{x})] = \text{const}$  and the weights:

$$w_Z(\mathbf{z}) = \exp(-u_Z(F_{XZ}(\mathbf{x})) - \log \rho_X(\mathbf{x}) + \Delta S_{XZ}(\mathbf{x})) \propto \frac{\mu_Z(\mathbf{z})}{p_Z(\mathbf{z})},$$

maximum likelihood equals log weight maximization:

$$\max \mathbb{E}_{\mathbf{x} \sim \rho_X(\mathbf{x})} [\log w_Z(\mathbf{z})] = \min \text{NLL}(\rho_X \parallel p_X). \quad (27)$$

### 2. Proof of theorem 1 (unbiased sampling with SNF importance weights)

Considering

$$\begin{aligned} \mathbb{E}_{\mu_X}[O] &= \int \mu_X(\mathbf{x}) O(\mathbf{x}) d\mathbf{x} \\ &= \iint \mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z}) O(\mathbf{x}) d\mathbf{z} d\mathbf{x} \\ &= \iint \mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x}) \left[ \frac{\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})}{\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})} O(\mathbf{x}) \right] d\mathbf{z} d\mathbf{x} \\ &= \mathbb{E}_f \left[ \frac{\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})}{\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})} O(\mathbf{x}) \right], \end{aligned}$$

where  $\mathbb{E}_f$  denotes the expectation over forward path realizations. In practice, we do not know the normalization constant of  $\mu_X$  and we therefore replace  $\frac{\mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})}{\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x})}$  by the unnormalized path weights in Eq. (11). Then we must normalize the estimator for expectation values, obtaining:

$$\frac{\sum_{k=1}^N \mathbf{w}(\mathbf{z}_k \rightarrow \mathbf{x}_k) O(\mathbf{x}_k)}{\sum_{k=1}^N \mathbf{w}(\mathbf{z}_k \rightarrow \mathbf{x}_k)} \xrightarrow{p} \mathbb{E}_\mu[O]$$

which converges towards  $\mathbb{E}_\mu[O]$  with  $N \rightarrow \infty$  according to the law of large numbers.

### 3. Derivation of the deterministic layer probability ratio

In order to work with delta distributions, we define  $\delta^\sigma(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \sigma\mathbf{I})$ , i.e. a Gaussian normal distribution with mean  $\mathbf{0}$  and variance  $\sigma$  and then consider the limit  $\sigma \rightarrow 0^+$ . In the case where  $\sigma > 0$ , by defining

$$q_t^\sigma(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}) = \delta^\sigma(\mathbf{y}_{t+1} - F_t(\mathbf{y}_t)),$$

and

$$\begin{aligned}\tilde{q}_t^\sigma(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t) &= \frac{p_t(\mathbf{y}_t)q_t^\sigma(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})}{\int p_t(\mathbf{y})q_t^\sigma(\mathbf{y} \rightarrow \mathbf{y}_{t+1})d\mathbf{y}} \\ &= \frac{p_t(\mathbf{y}_t)\delta^\sigma(\mathbf{y}_{t+1} - F_t(\mathbf{y}_t))}{\int p_t(\mathbf{y})\delta^\sigma(\mathbf{y}_{t+1} - F_t(\mathbf{y}))d\mathbf{y}},\end{aligned}$$

we have

$$\frac{\tilde{q}_t^\sigma(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t^\sigma(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} = \frac{p_t(\mathbf{y}_t)}{\int p_t(\mathbf{y})\delta^\sigma(\mathbf{y}_{t+1} - F_t(\mathbf{y}))d\mathbf{y}},$$

where  $p_t(y_t)$  denotes the marginal distribution of  $\mathbf{y}_t$ . By considering

$$\begin{aligned}\lim_{\sigma \rightarrow 0^+} \int p_t(\mathbf{y})\delta^\sigma(\mathbf{y}_{t+1} - F_t(\mathbf{y}))d\mathbf{y} &= \lim_{\sigma \rightarrow 0^+} \int p_t(F_t^{-1}(\mathbf{y}'))\delta^\sigma(\mathbf{y}_{t+1} - \mathbf{y}') \left| \det \left( \frac{\partial F_t^{-1}(\mathbf{y}')}{\partial \mathbf{y}'} \right) \right| d\mathbf{y}' \\ &= p_t(F_t^{-1}(\mathbf{y}_{t+1})) \left| \det \left( \frac{\partial F_t^{-1}(\mathbf{y}_{t+1})}{\partial \mathbf{y}_{t+1}} \right) \right| \\ &= p_t(\mathbf{y}_t) |\det \mathbf{J}_t(\mathbf{y}_t)|^{-1}\end{aligned}$$

and using the definition of  $\Delta S_t$  in terms of path probability ratios, we obtain:

$$\begin{aligned}\exp(\Delta S_t) &= \frac{\tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} = \lim_{\sigma \rightarrow 0^+} \frac{\tilde{q}_t^\sigma(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t^\sigma(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} \\ &= \lim_{\sigma \rightarrow 0^+} \frac{p_t(\mathbf{y}_t)}{\int p_t(\mathbf{y})\delta^\sigma(\mathbf{y}_{t+1} - F_t(\mathbf{y}))d\mathbf{y}} \\ &= |\det \mathbf{J}_t(\mathbf{y}_t)|\end{aligned}$$

and thus

$$\Delta S_t = \log |\det \mathbf{J}_t(\mathbf{y}_t)|.$$

### 4. Derivation of the overdamped Langevin path probability ratio

These results follow [31]. The backward step is realized by

$$\mathbf{y}_t = \mathbf{y}_{t+1} - \epsilon_t \nabla u_\lambda(\mathbf{y}_{t+1}) + \sqrt{\frac{2\epsilon_t}{\beta}} \tilde{\boldsymbol{\eta}}_t. \quad (28)$$

Combining Equations (18) and (28):

$$-\epsilon_t \nabla u_\lambda(\mathbf{y}_t) + \sqrt{\frac{2\epsilon_t}{\beta}} \boldsymbol{\eta}_t = \epsilon_t \nabla u_\lambda(\mathbf{y}_{t+1}) - \sqrt{\frac{2\epsilon_t}{\beta}} \tilde{\boldsymbol{\eta}}_t.$$

and thus

$$\tilde{\boldsymbol{\eta}}_t = \sqrt{\frac{\epsilon_t \beta}{2}} [\nabla u_\lambda(\mathbf{y}_t) + \nabla u_\lambda(\mathbf{y}_{t+1})] - \boldsymbol{\eta}_t.$$

Resulting in the path probability ratio:

$$\begin{aligned}\exp(\Delta S_t) &= \frac{q_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} = \frac{p(\tilde{\boldsymbol{\eta}}_t) \left| \frac{\partial \mathbf{y}_t}{\partial \tilde{\boldsymbol{\eta}}_t} \right|}{p(\boldsymbol{\eta}_t) \left| \det \left( \frac{\partial \mathbf{y}_{t+1}}{\partial \boldsymbol{\eta}_t} \right) \right|} \\ &= \frac{p(\tilde{\boldsymbol{\eta}}_t)}{p(\boldsymbol{\eta}_t)} = e^{-\frac{1}{2}(\|\tilde{\boldsymbol{\eta}}_t\|^2 - \|\boldsymbol{\eta}_t\|^2)}.\end{aligned}$$

and thus

$$-\Delta S_t = \frac{1}{2} (\|\tilde{\boldsymbol{\eta}}_t\|^2 - \|\boldsymbol{\eta}_t\|^2)$$

## 5. Derivation of the Langevin probability ratio

These results follow [31]. We define constants:

$$\begin{aligned} c_1 &= \frac{\Delta t}{2m} \\ c_2 &= \sqrt{\frac{4\gamma m}{\Delta t \beta}} \\ c_3 &= 1 + \frac{\gamma \Delta t}{2} \end{aligned}$$

Then, the forward step of Brooks-Brünger-Karplus (BBK, leap-frog) Langevin dynamics are defined as:

$$\mathbf{v}' = \mathbf{v}_t + c_1 [-\nabla u_\lambda(\mathbf{x}_t) - \gamma m \mathbf{v}_t + c_2 \boldsymbol{\eta}_t] \quad (29)$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta t \mathbf{v}' \quad (30)$$

$$\mathbf{v}_{t+1} = \frac{1}{c_3} [\mathbf{v}' + c_1 (-\nabla u_\lambda(\mathbf{x}_{t+1}) + c_2 \boldsymbol{\eta}'_t)] \quad (31)$$

Note that the factor 4 in sqrt is different from [31] – this factor is needed as we employ  $\Delta t/2$  in both half-steps. The backward step with reversed momenta,  $(\mathbf{x}_{t+1}, -\mathbf{v}_{t+1}) \rightarrow (\mathbf{x}_t, -\mathbf{v}_t)$  is then defined by:

$$\mathbf{v}'' = -\mathbf{v}_{t+1} + c_1 [-\nabla u_\lambda(\mathbf{x}_{t+1}) + \gamma m \mathbf{v}_{t+1} + c_2 \tilde{\boldsymbol{\eta}}_t] \quad (32)$$

$$\mathbf{x}_t = \mathbf{x}_{t+1} + \Delta t \mathbf{v}'' \quad (33)$$

$$-\mathbf{v}_t = \frac{1}{c_3} [\mathbf{v}'' + c_1 (-\nabla u_\lambda(\mathbf{x}_t) + c_2 \tilde{\boldsymbol{\eta}}'_t)] \quad (34)$$

To compute the momenta  $\tilde{\boldsymbol{\eta}}_t, \tilde{\boldsymbol{\eta}}'_t$  that realize the reverse step, we first combine Eqs. (30-33) to obtain:

$$\mathbf{v}' = -\mathbf{v}'' \quad (35)$$

Combining Eqs. (31), (32) and (35), we obtain:

$$\begin{aligned} \left(1 + \frac{\gamma \Delta t}{2}\right) \mathbf{v}_{t+1} &= \mathbf{v}' + c_1 (-\nabla u_\lambda(\mathbf{x}_{t+1}) + c_2 \boldsymbol{\eta}'_t) \\ \left(1 - \frac{\gamma \Delta t}{2}\right) \mathbf{v}_{t+1} &= \mathbf{v}'' + c_1 (-\nabla u_\lambda(\mathbf{x}_t) + c_2 \tilde{\boldsymbol{\eta}}_t), \end{aligned}$$

and:

$$\tilde{\boldsymbol{\eta}}_t = \boldsymbol{\eta}'_t - \sqrt{\gamma \Delta t m \beta} \mathbf{v}_{t+1}$$

Combining Eqs. (29), (34) and (35), we obtain:

$$\begin{aligned} -\mathbf{v}_t \left(1 - \frac{\gamma \Delta t}{2}\right) &= \mathbf{v}'' + c_1 (-\nabla u_\lambda(\mathbf{x}_t) + c_2 \boldsymbol{\eta}_t) \\ -\mathbf{v}_t \left(1 + \frac{\gamma \Delta t}{2}\right) &= \mathbf{v}' + c_1 (-\nabla u_\lambda(\mathbf{x}_{t+1}) + c_2 \tilde{\boldsymbol{\eta}}'_t), \end{aligned}$$

and:

$$\begin{aligned} -\mathbf{v}_t \left(1 - \frac{\gamma \Delta t}{2}\right) - c_2 \boldsymbol{\eta}_t &= -\mathbf{v}_t \left(1 + \frac{\gamma \Delta t}{2}\right) - c_2 \tilde{\boldsymbol{\eta}}'_t \\ \tilde{\boldsymbol{\eta}}'_t &= \boldsymbol{\eta}_t - \sqrt{\gamma \Delta t m \beta} \mathbf{v}_t \end{aligned}$$

To compute the path probability ratio we introduce the Jacobian

$$J(\boldsymbol{\eta}_t, \boldsymbol{\eta}'_t) = \det \begin{bmatrix} \frac{\partial \mathbf{x}_{t+1}}{\partial \boldsymbol{\eta}_t} & \frac{\partial \mathbf{v}_{t+1}}{\partial \boldsymbol{\eta}_t} \\ \frac{\partial \mathbf{x}_{t+1}}{\partial \boldsymbol{\eta}'_t} & \frac{\partial \mathbf{v}_{t+1}}{\partial \boldsymbol{\eta}'_t} \end{bmatrix}$$

and find:

$$\begin{aligned}\exp(\Delta S_t) &= \frac{\tilde{q}_t((\mathbf{x}_{t+1}, -\mathbf{v}_{t+1}) \rightarrow (\mathbf{x}_t, \mathbf{v}_t))}{q_t((\mathbf{x}_t, \mathbf{v}_t) \rightarrow (\mathbf{x}_{t+1}, -\mathbf{v}_{t+1}))} \\ &= \frac{p(\tilde{\boldsymbol{\eta}}_t)p(\tilde{\boldsymbol{\eta}}'_t)J(\tilde{\boldsymbol{\eta}}_t, \tilde{\boldsymbol{\eta}}'_t)}{p(\boldsymbol{\eta}_t)p(\boldsymbol{\eta}'_t)J(\boldsymbol{\eta}_t, \boldsymbol{\eta}'_t)} \\ -\Delta S_t &= \frac{1}{2} \left( \left( \|\tilde{\boldsymbol{\eta}}_t\|^2 + \|\tilde{\boldsymbol{\eta}}'_t\|^2 \right) - \left( \|\boldsymbol{\eta}_t\|^2 + \|\boldsymbol{\eta}'_t\|^2 \right) \right)\end{aligned}$$

where the Jacobian ratio cancels as the Jacobians are independent of the noise variables.

## 6. Derivation of the probability ratio for Markov Chain Monte Carlo

For MCMC,  $q_t$  satisfies the detailed balance condition

$$\exp(-u_\lambda(\mathbf{y}_t)) \cdot q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}) = \exp(-u_\lambda(\mathbf{y}_{t+1})) \cdot \tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)$$

with respect to the potential function  $u_\lambda$ . We have

$$\begin{aligned}\Delta S_t &= \log \frac{\tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} \\ &= \log \frac{\exp(-u_\lambda(\mathbf{y}_t))}{\exp(-u_\lambda(\mathbf{y}_{t+1}))} \\ &= u_\lambda(\mathbf{y}_{t+1}) - u_\lambda(\mathbf{y}_t)\end{aligned}$$

## 7. Derivation of the probability ratio for Hamiltonian MC with Metropolis acceptance

Hamiltonian MC with Metropolis acceptance defines a forward path density

$$q_t((\mathbf{y}_t, \mathbf{v}) \rightarrow (\mathbf{y}_{t+1}, \mathbf{v}^K))$$

which satisfies the joint detailed balance condition

$$\begin{aligned}\exp(-u_\lambda(\mathbf{y}_t))\mathcal{N}(\mathbf{v}|\mathbf{0}, \mathbf{I}) \cdot q_t((\mathbf{y}_t, \mathbf{v}) \rightarrow (\mathbf{y}_{t+1}, \mathbf{v}^K)) \\ = \exp(-u_\lambda(\mathbf{y}_{t+1}))\mathcal{N}(\mathbf{v}^K|\mathbf{0}, \mathbf{I}) \cdot \tilde{q}_t((\mathbf{y}_{t+1}, \mathbf{v}^K) \rightarrow (\mathbf{y}_t, \mathbf{v})).\end{aligned}\quad (36)$$

Considering the velocity  $\mathbf{v}$  is independently drawn from  $\mathcal{N}(\mathbf{v}|\mathbf{0}, \mathbf{I})$ , the “marginal” forward path density of  $\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}$  is

$$q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}) = \iint \mathcal{N}(\mathbf{v}|\mathbf{0}, \mathbf{I}) \cdot q_t((\mathbf{y}_t, \mathbf{v}) \rightarrow (\mathbf{y}_{t+1}, \mathbf{v}^K)) d\mathbf{v} d\mathbf{v}^K.$$

Then, it can be obtained from (36) that

$$\begin{aligned}\exp(-u_\lambda(\mathbf{y}_t))q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1}) &= \iint \exp(-u_\lambda(\mathbf{y}_t))\mathcal{N}(\mathbf{v}|\mathbf{0}, \mathbf{I})((\mathbf{y}_t, \mathbf{v}) \rightarrow (\mathbf{y}_{t+1}, \mathbf{v}^K)) d\mathbf{v} d\mathbf{v}^K \\ &= \iint \exp(-u_\lambda(\mathbf{y}_{t+1}))\mathcal{N}(\mathbf{v}^K|\mathbf{0}, \mathbf{I})\tilde{q}_t((\mathbf{y}_{t+1}, \mathbf{v}^K) \rightarrow (\mathbf{y}_t, \mathbf{v})) d\mathbf{v} d\mathbf{v}^K \\ &= \exp(-u_\lambda(\mathbf{y}_{t+1}))\tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t),\end{aligned}$$

and

$$\begin{aligned}\Delta S_t &= \log \frac{\tilde{q}_t(\mathbf{y}_{t+1} \rightarrow \mathbf{y}_t)}{q_t(\mathbf{y}_t \rightarrow \mathbf{y}_{t+1})} \\ &= u_\lambda(\mathbf{y}_{t+1}) - u_\lambda(\mathbf{y}_t)\end{aligned}$$

## 8. Details on using SNFs for variational inference

Here we elaborate on the details of using SNFs as a variational approximation of the posterior distribution of a *variational autoencoder* (VAE) [21] as presented in our last results section. In contrast to the usual notation used in common VAE literature, we choose  $\mathbf{x}$  to indicate the *latent*

variable, while we call the *observed* variable  $\mathbf{s}$ . This is due to being consistent with the use of  $\mathbf{x}$  as the sampled variable of interest throughout our former discussions.

For a given data set  $\{\mathbf{s}_1, \dots, \mathbf{s}_N\}$ , the decoder  $D$  of a VAE characterizes each  $\mathbf{s}$  as a random variable with a tractable distribution  $\mathbb{P}_D(\mathbf{s}|\mathbf{x})$  depending on a unknown latent variable  $\mathbf{x}$ . Furthermore, the prior distribution is assumed to be tractable as well (e.g. an isotropic normal distribution). Here we take the prior

$$\mathbb{P}(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mathbf{0}, \mathbf{I}).$$

Together, this defines the joint distribution

$$\mathbb{P}_D(\mathbf{x}, \mathbf{s}) = \mathbb{P}(\mathbf{x}) \cdot \mathbb{P}_D(\mathbf{s}|\mathbf{x}).$$

Conditioned on a given  $\mathbf{s}$ , we can utilize a SNF to approximate the posterior distribution

$$\mathbb{P}_D(\mathbf{x}|\mathbf{s}) = \frac{\mathbb{P}_D(\mathbf{x}, \mathbf{s})}{\mathbb{P}_D(\mathbf{s})}.$$

For convenience and consistency with the former discussion, we define  $\mu_X(\mathbf{x}) = \mathbb{P}_D(\mathbf{x}|\mathbf{s})$  and  $u_X(\mathbf{x}) = -\log \mathbb{P}_D(\mathbf{x}, \mathbf{s})$ . Thus, the parameters of the SNF and the decoder  $D$  can be trained by minimizing  $J_{KL}$  which provides an upper bound of the negative log-likelihood of  $\mathbf{s}$  as follows:

$$\begin{aligned} J_{KL}(\mathbf{s}) &= \mathbb{E}_{\mathbf{z} \sim \mu_Z, \mathbf{y}_1, \dots, \mathbf{y}_T} [u_X(\mathbf{y}_T) - \sum_{t=0}^{T-1} \Delta S_t] \\ &= \mathbb{E}_{\mathbf{z} \sim \mu_Z, \mathbf{y}_1, \dots, \mathbf{y}_T} [-\log \mu_X(\mathbf{y}_T) - \sum_{t=0}^{T-1} \Delta S_t] - \log \mathbb{P}_D(\mathbf{s}) \\ &= \text{KL}(\mu_Z(\mathbf{z}) \mathbb{P}_f(\mathbf{z} \rightarrow \mathbf{x}) \parallel \mu_X(\mathbf{x}) \mathbb{P}_b(\mathbf{x} \rightarrow \mathbf{z})) - \log \mathbb{P}_D(\mathbf{s}) \\ &\geq -\log \mathbb{P}_D(\mathbf{s}) \end{aligned}$$

If the SNF consists of only deterministic transformations,  $J_{KL}$  is equivalent to  $\mathcal{F}$  in [35].

We estimate  $J_{KL}(\mathbf{s})$  on samples as

$$\hat{J}_{KL}(\mathbf{s}) = \frac{1}{M} \sum_{i=1}^M u_X(\mathbf{y}_T^{(i)}) - \sum_{t=0}^{T-1} \Delta S_t^{(i)}, \quad (37)$$

by sampling  $M$  paths  $\{(\mathbf{y}_0^{(i)}, \dots, \mathbf{y}_T^{(i)})\}_{i=1}^M$  for each  $\mathbf{s}$  and setting  $M = 5$ .

**Estimating the evidence.** After training we approximate  $-\log \mathbb{P}_D(\mathbf{s})$  by marginalizing out the latent variable  $\mathbf{x}$  via Monte Carlo sampling. In order to improve sampling efficiency and have a fair comparison among the three different SNF instantiations, we approximate the posterior distribution  $\mathbb{P}_D(\mathbf{x}|\mathbf{s})$  of the trained model using the same variational approximation:

1. We define a simple base distribution  $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ , together with a conditional diffeomorphism  $F_{LL}(\mathbf{x}|\mathbf{s})$  transforming  $\mathbf{z}$  to  $\mathbf{x}$  and vice-versa conditioned on  $\mathbf{s}$ :

$$\begin{array}{ccc} & F_{LL}^{-1}(\cdot|\mathbf{s}) & \\ \mathbf{x} & \xleftrightarrow{=} & \mathbf{z} \\ & F_{LL}(\cdot|\mathbf{s}) & \end{array}$$

We realize such a conditional flow via RealNVP transformations, where coupling layers are additionally conditioned on  $\mathbf{s}$  and only  $\mathbf{x}/\mathbf{z}$  is transformed during the flow. Together with  $q(\mathbf{z})$  this defines the conditional distribution

$$q_{LL}(\mathbf{x}|\mathbf{s}) = q(F_{LL}^{-1}(\mathbf{x}|\mathbf{s})) \left| \det \left( \frac{\partial F_{LL}^{-1}(\mathbf{x}|\mathbf{s})}{\partial \mathbf{x}} \right) \right|$$

which we use as variational approximation to the true posterior. We then train  $q_{LL}$  by minimizing the KL divergence

$$\mathbb{E}_{\mathbf{s}, \mathbf{z} \sim q(\mathbf{z})} [\log q_{LL}(F_{LL}(\mathbf{z}|\mathbf{s})|\mathbf{s}) - \log \mathbb{P}(F_{LL}(\mathbf{z}|\mathbf{s})) - \log \mathbb{P}_D(\mathbf{s}|F_{LL}(\mathbf{z}|\mathbf{s}))] + \text{const.}$$

until convergence. This loss is minimized iff  $q_{LL}(\mathbf{x}|\mathbf{s}) = \mathbb{P}_D(\mathbf{x}|\mathbf{s})$ .

2. Now considering

$$\begin{aligned}
\mathbb{P}_D(\mathbf{s}) &= \int \mathbb{P}(\mathbf{x}) \mathbb{P}_D(\mathbf{s}|\mathbf{x}) d\mathbf{x} \\
&= \int q_{LL}(\mathbf{x}|\mathbf{s}) \frac{\mathbb{P}(\mathbf{x}) \mathbb{P}_D(\mathbf{s}|\mathbf{x})}{q_{LL}(\mathbf{x}|\mathbf{s})} d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x} \sim q_{LL}(\mathbf{x}|\mathbf{s})} \left[ \frac{\mathbb{P}(\mathbf{x}) \mathbb{P}_D(\mathbf{s}|\mathbf{x})}{q_{LL}(\mathbf{x}|\mathbf{s})} \right] \\
&= \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})} \left[ \frac{\mathbb{P}(F_{LL}(\mathbf{z}|\mathbf{s})) \mathbb{P}_D(\mathbf{s}|F_{LL}(\mathbf{z}|\mathbf{s}))}{q(\mathbf{z}) \left| \det \left( \frac{\partial F_{LL}^{-1}(\mathbf{x}|\mathbf{s})}{\partial \mathbf{x}} \right) \right|^{-1}} \right],
\end{aligned}$$

we can draw  $N$  samples  $\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(N)}$  and approximate  $\mathbb{P}_D(\mathbf{s})$  by

$$\hat{\mathbb{P}}_D(\mathbf{s}) = \frac{1}{N} \sum_{i=1}^N \frac{\mathbb{P}(F_{LL}(\mathbf{z}^{(i)}|\mathbf{s})) \mathbb{P}_D(\mathbf{s}|F_{LL}(\mathbf{z}^{(i)}|\mathbf{s}))}{q(\mathbf{z}^{(i)}) \left| \det \left( \frac{\partial F_{LL}^{-1}(\mathbf{x}|\mathbf{s})}{\partial \mathbf{x}} \right) \right|^{-1}}.$$

In experiments,  $N$  is set to be 2000.

In table 3, the first column is the mean value of  $\hat{J}_{KL}(\mathbf{s})$  on the test data set as a variational bound of the mean of  $-\log p(\mathbf{s})$  (related to Fig. 4a in [35]). The second column is the mean value of  $-\log \hat{\mathbb{P}}_D(\mathbf{s})$  on the test data set (related to Fig. 4c in [35]).

## 9. Hyper-parameters and other benchmark details

All experiments were run using PyTorch 1.2 and on GTX1080Ti cards. Optimization uses Adam [20] with step-size 0.001 and otherwise default parameters. All deterministic flow transformations use RealNVP [6]. A RealNVP block is defined by two subsequent RealNVP layers that are swapped such that each channel gets transformed once as a function of the other channel. The affine transformation of each RealNVP layer is given by a fully connected ReLU network. For the NSF layers we substitute the simple affine transformations used in RealNVP by the rational-quadratic (RQ) spline transformation implemented in <https://github.com/bayesiains/nflows>. As before the width, height and slope of the RQ transformations are given by fully connected ReLU networks. Again a NSF block consists of two subsequent NSF layers with intermediate swap layers.

### Double well examples in Figures 1 and 4

- Both normalizing flow and SNF networks use 3 RealNVP blocks with three hidden layers of dimension 64. The SNF additionally uses 20 Metropolis MC steps per block using a Gaussian proposal density with standard deviation 0.25.
- Training is done by minimizing  $J_{ML}$  for 300 iterations and  $\frac{1}{2}J_{ML} + \frac{1}{2}KL$  for 300 iterations using a batch-size of 128.
- “Biased data” is defined by running local Metropolis MC in each of the two wells. These simulations do not transition to the other well and we use 1000 data points in each well for training.
- “Unbiased data” is produced by running Metropolis MC with a large proposal step (standard deviation 1.5) to convergence and retaining 10000 data points for training.
- In Table S1, the sampling results of SNFs with RealNVP blocks and Metropolis MC steps. MC step sizes of the first SNF is fixed to be 0.25 as before, and all step sizes of the second one are trainable parameters in  $[0.01, 0.3]$ . The other settings are the same as in Table 1.

### Two-dimensional image densities in Figure 3

- RealNVP and NSF flows both use 5 blocks. All involved transformation parameters (translation/scale in RealNVP layers, width/height/slope in NSF layers) use three hidden layers of dimension 64. For the NSF layers we used 20 knot points in the RQ-spline transformation. Training was done by minimizing  $J_{ML}$  for 2000 iterations with batch-size 250.

Table S1: Unbiased sampling for double well potential by SNFs with nontrainable/trainable MC step sizes.

	not reweighted			reweighted		
	bias	$\sqrt{\text{var}}$	$\sqrt{\text{bias}^2 + \text{var}}$	bias	$\sqrt{\text{var}}$	$\sqrt{\text{bias}^2 + \text{var}}$
RNVP + MC	$1.5 \pm 0.2$	$0.3 \pm 0.1$	$1.5 \pm 0.2$	$0.2 \pm 0.1$	$0.6 \pm 0.1$	$0.6 \pm 0.1$
<b>RNVP + MC with trainable step sizes</b>	$1.0 \pm 0.2$	$0.2 \pm 0.1$	$1.0 \pm 0.2$	$0.1 \pm 0.1$	$0.4 \pm 0.1$	$0.4 \pm 0.1$

- Purely stochastic flow (column 2) uses five blocks with 10 Metropolis MC steps each using a Gaussian proposal density with standard deviation 0.1.
- SNF (column 3/5) uses 5 blocks (RNVP/NSF block and 10 Metropolis MC steps with same parameters as above). Training was done by minimizing  $J_{ML}$  for 6000 iterations with batch-size 250.

### Alanine dipeptide in Fig. 5

- Normalizing flow uses 3 RealNVP blocks with 3 hidden layers and [128, 128, 128] nodes in their transformers. Training was done by minimizing  $J_{ML}$  for 1000 iterations with batch-size 256.
- SNF uses the same architecture and training parameters, but additionally 20 Metropolis MC steps each using a Gaussian proposal density with standard deviation 0.1.
- As a last flow layer before  $\mathbf{x}$ , we used an invertible transformation between Cartesian coordinates and internal coordinates (bond lengths, angles, torsion angles) following the procedure described in [32]. The internal coordinates were normalized by removing the mean and dividing by the standard deviation of their values in the training data.
- Training data: We set up Alanine dipeptide in vacuum using OpenMMTools. Parameters are defined by the force field ff96 of the AMBER program [34]. Simulations are run at standard OpenMMTools parameters with no bond constraints, 1 femtosecond time-step for  $10^6$  time-steps (1 nanosecond) at a temperature of 1000 K in order to facilitate rapid exploration of the  $\phi/\psi$  torsion angles and a few hundred transitions between metastable states.  $10^5$  atom positions were saved as training data.

### MNIST and Fashion-MNIST VAE in Table 3

- The latent space dimension was set to 50. The decoder consists of 2 fully connected hidden layers, with 1024 units and ReLU non-linearities for each hidden layer. The activation function of the output layer is sigmoid function.  $\mathbb{P}_D(\mathbf{s}|\mathbf{x})$  is defined as

$$\begin{aligned} \log \mathbb{P}_D(\mathbf{s}|\mathbf{x}) &= \log \mathbb{P}_D(\mathbf{s}|D(\mathbf{x})) \\ &= \sum_{i=1}^{784} [\mathbf{s}]_i \log [D(\mathbf{x})]_i + (1 - [\mathbf{s}]_i) \log (1 - [D(\mathbf{x})]_i), \end{aligned}$$

where  $[\mathbf{s}]_i$ ,  $[D(\mathbf{x})]_i$  denote the  $i$ th pixel of  $\mathbf{s}$  and the  $i$ th output of  $D$ .

- Adam algorithm is used to train all models. Training was done by minimizing  $\hat{J}_{KL}$  (see (37)) for 40 epochs with batch-size 128 and step size  $10^{-3}$  unless otherwise stated.
- In simple VAE, the encoder  $E$  consists of 2 fully connected hidden layers, with 1024 nodes and ReLU non-linearities for each hidden layer. The encoder has 100 outputs, where the activation function of the first 50 outputs is the linear function and the activation function of the last 50 outputs is the absolute value function. The transformation from  $\mathbf{z}$  to  $\mathbf{x}$  is given by

$$[\mathbf{x}]_i = [E(\mathbf{s})]_i + [\mathbf{z}]_i \cdot [E(\mathbf{s})]_{i+50}. \quad (38)$$

- MCMC uses 30 Metropolis MC steps each using a overdamped Langevin proposal, where the interpolated potential are used. The interpolation coefficients and the step size of the proposal are both trained as parameters of the flow.

- Normalizing flow uses 6 RealNVP blocks with 2 hidden layers and [64, 64] nodes in their transformers.
- SNF uses three units with each unit consisting of 2 RealNVP blocks + 10 Metropolis MC steps, where architectures are the same as the above. During the training procedure, we first train parameters of the 6 RealNVP blocks without the Metropolis MC steps for 20 epochs, and then train all parameters for another 20 epochs. The training step size is  $10^{-3}$  for the first 20 epochs and  $10^{-4}$  for the last 20 epochs.
- For calculating the marginal likelihood  $\mathbb{P}_D(\mathbf{s})$ ,  $F_{LL}$  uses 12 RealNVP blocks with 2 hidden layers and [256, 256] nodes in their transformers.

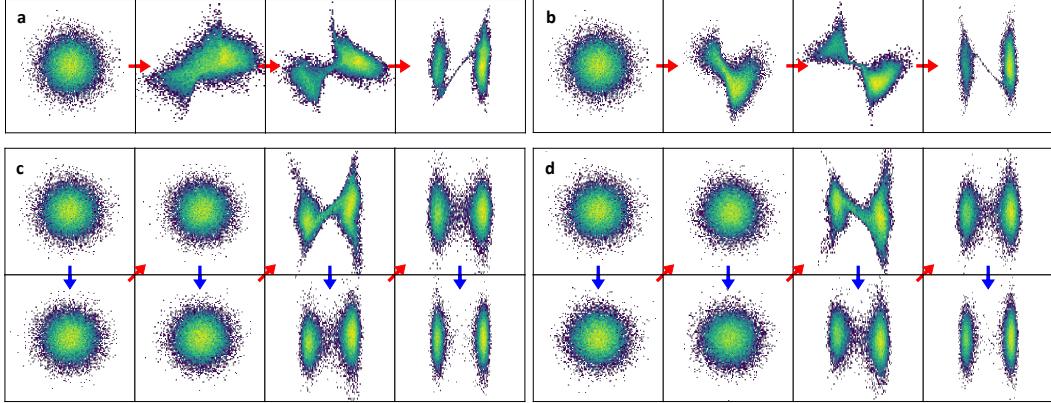
## 10. Comparison with related sampling methods

A brief comparison of the proposed SNF and selected sampling methods with learnable proposals and transformations is provided in Table S2. Most previous sampling methods are developed based on the detailed balance in each step, except that HVI presented in [36] can perform nonequilibrium sampling steps by using annealed target distributions. Furthermore, some sampling techniques [16] and [17] also improve the sampling efficiency by linear or nonlinear deterministic transformation, where the transformation is performed only once. It can be seen from the comparison that SNF provides a universal framework for sampling, where the deterministic and stochastic blocks can be flexibly designed and combined.

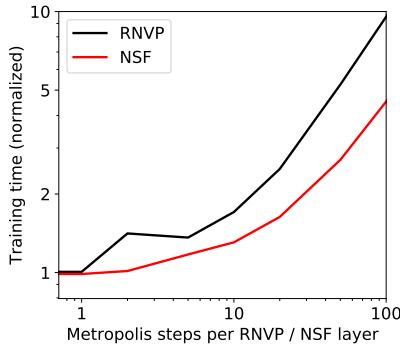
Table S2: A comparison of samplers with learnable proposals/transformations.

Method	Containing nonequilibrium sampling steps	Combining with learnable deterministic transformation
HVI [36]	✓	✗
L2HMC [24]	✗	✗
A-NICE-MC [38]	✗	✗
HMC for DLGMs [16]	✗	✓
NeuTra [17]	✗	✓
SNF	✓	✓

## Supplementary Figures



**Figure S1: Reproducibility of normalizing flows for the double well.** Red arrows indicate deterministic transformations (perturbations), blue arrows indicate stochastic dynamics (relaxations). **a-b)** Two independent runs of 3 RealNVP blocks (6 layers). **c-d)** Two independent runs of same architecture with 20 BD steps before/after RealNVP blocks.



**Figure S2: Computational cost of adding stochastic layers.** Time required for training SNFs of images shown in 3 with a fixed number of steps, as a function of the number of stochastic layers per RNVP or NSF layer. Timings are normalized to one RNVP or NSF layer. While details of these timings depend on hyperparameters, implementation and compute platform, the main feature is that deterministic flow layers are much more computationally expensive than stochastic flow layers, and therefore a few stochastic flow layers can be added to each deterministic flow layer without significant increase in computational cost.