Diffusion Normalizing Flow

Qinsheng Zhang
Georgia Institute of Technology
qzhang419@gatech.edu

Yongxin Chen
Georgia Institute of Technology
yongchen@gatech.edu

Abstract

We present a novel generative modeling method called diffusion normalizing flow based on stochastic differential equations (SDEs). The algorithm consists of two neural SDEs: a forward SDE that gradually adds noise to the data to transform the data into Gaussian random noise, and a backward SDE that gradually removes the noise to sample from the data distribution. By jointly training the two neural SDEs to minimize a common cost function that quantifies the difference between the two, the backward SDE converges to a diffusion process the starts with a Gaussian distribution and ends with the desired data distribution. Our method is closely related to normalizing flow and diffusion probabilistic models and can be viewed as a combination of the two. Compared with normalizing flow, diffusion normalizing flow is able to learn distributions with sharp boundaries. Compared with diffusion probabilistic models, diffusion normalizing flow requires fewer discretization steps and thus has better sampling efficiency. Our algorithm demonstrates competitive performance in both high-dimension data density estimation and image generation tasks.

1 Introduction

Generative model is a class of machine learning models used to estimate data distributions and sometimes generate new samples from the distributions [8, 35, 16, 37, 7]. Many generative models learn the data distributions by transforming a latent variable $\mathbf{z}$ with a tractable prior distribution to the data space [8, 35, 32]. To generate new samples, one can sample from the latent space and then follow the transformation to the data space. There exist a large class of generative models where the latent space and the data space are of the same dimension. The latent variable and the data are coupled through trajectories in the same space. These trajectories serve two purposes: in the forward direction $\mathbf{x} \to \mathbf{z}$, the trajectories infer the posterior distribution in the latent space associated with a given data sample $\mathbf{x}$, and in the backward direction $\mathbf{z} \to \mathbf{x}$, it generates new samples by simulating the trajectories starting from the latent space. This type of generative model can be roughly divided into two categories, depending on whether these trajectories connecting the latent space and the data space are deterministic or stochastic.

When deterministic trajectories are used, these generative models are known as flow-based models. The latent space and the data space are connected through an invertible map, which could either be realized by the composition of multiple invertible maps [35, 8, 20] or a differential equation [4, 14]. In these models, the probability density at each data point can be evaluated explicitly using the change of variable theorem, and thus the training can be carried out by minimizing the negative log-likelihood (NLL) directly. One limitation of the flow-based model is that the invertible map parameterized by neural networks used in it imposes topological constraints on the transformation from $\mathbf{z}$ to $\mathbf{x}$. Such limitation affects the performance significantly when the prior distribution on $\mathbf{z}$ is a simple unimodal distribution such as Gaussian while the target data distribution is a well-separated multi-modal distribution, i.e., its support has multiple isolated components. In [6], it is shown that there are some fundamental issues of using well-conditioned invertible functions to approximate such complicated multi-modal data distributions.

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

When stochastic trajectories are used, the generative models are often known as the diffusion model [38]. In a diffusion model, a prespecified stochastic forward process gradually adds noise into the data to transform the data samples into simple random variables. A separate backward process is trained to revert this process to gradually remove the noise from the data to recover the original data distributions. When the forward process is modeled by a stochastic differential equation (SDE), the optimal backward SDE [1] can be retrieved by learning the score function [39, 40, 17, 2]. When the noise is added to the data sufficiently slow in the forward process, the backward diffusion can often revert the forward one reasonably well and is able to generate high fidelity samples. However, this also means that the trajectories have to be sufficiently long with a large number of time-discretization steps, which leads to slow training and sampling. In addition, since the forward process is fixed, the way noise is added is independent of the data distribution. As a consequence, the learned model may miss some complex but important details in the data distribution, as we will explain later.

In this work, we present a new generative modeling algorithm that resembles both the flow-based models and the diffusion models. It extends the normalizing flow method by gradually adding noise to the sampling trajectories to make them stochastic. It extends the diffusion model by making the forward process from $\mathbf{x}$ to $\mathbf{z}$ trainable. Our algorithm is thus termed *Diffusion Normalizing Flow (DiffFlow)*. The comparisons and relations among DiffFlow, normalizing flow, and diffusion models are shown in Figure 1. When the noise in DiffFlow shrinks to zero, DiffFlow reduces to a standard normalizing flow. When the forward process is fixed to some specific type of diffusion, DiffFlow reduces to a diffusion model.

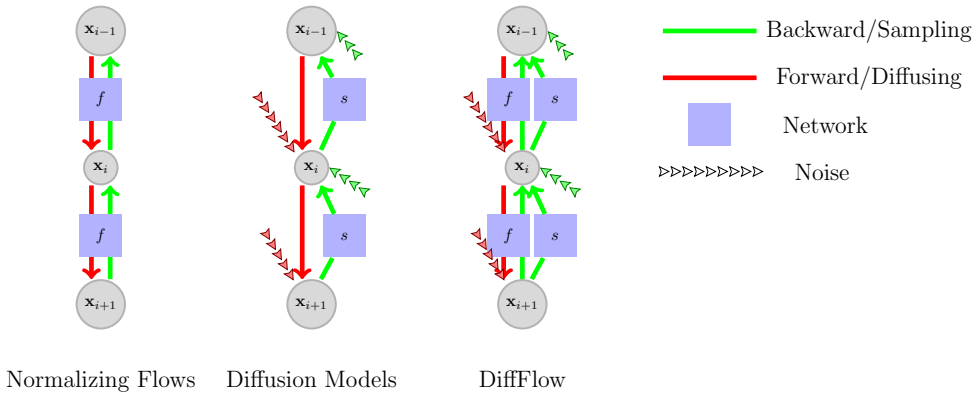


Figure 1: The schematic diagram for normalizing flows, diffusion models, and DiffFlow. In normalizing flow, both the forward and the backward processes are deterministic. They are the inverse of each other and thus collapse into a single process. The diffusion model has a fixed forward process and trainable backward process, both are stochastic. In DiffFlow, both the forward and the backward processes are trainable and stochastic.

In DiffFlow, the forward and backward diffusion processes are trained simultaneously by minimizing the distance between the forward and the backward process in terms of the Kullback-Leibler (KL) divergence of the induced probability measures [42]. This cost turns out to be equivalent to (see Section 3 for a derivation) the (amortized) negative evidence lower bound (ELBO) widely used in variational inference [21]. One advantage to use the KL divergence directly is that it can be estimated with no bias using sampled trajectories of the diffusion processes. The KL divergence in the trajectory space also bounds the KL divergence of the marginals, providing an alternative method to bound the likelihood (see Section 3 for details). To summarize, we have made the following contributions.

1. We propose a novel density estimation model termed diffusion normalizing flow (DiffFlow) that extends both the flow-based models and the diffusion models. The added stochasticity in DiffFlow boosts the expressive power of the normalizing flow and results in better performance in terms of sampling quality and likelihood. Compared with diffusion models, DiffFlow is able to learn a forward diffusion process to add noise to the data adaptively and more efficiently. This avoids adding noise to regions where noise is not so desirable. The learnable forward process also shortens the trajectory length, making the sampling much faster than standard diffusion models (We observe a 20 times speedup over diffusion models without decreasing sampling quality much).
2. We develop a stochastic adjoint algorithm to train the DiffFlow model. This algorithm evaluates the objective function and its gradient sequentially along the trajectory. It avoids storing all the

intermediate values in the computational graph, making it possible to train DiffFlow for high-dimensional problems.
3. We apply the DiffFlow model to several generative modeling tasks with both synthetic and real datasets, and verify the performance of DiffFlow and its advantages over other methods.

2 Background

Below we provide a brief introduction to normalizing flows and diffusion models. In both of these models, we use $\tau = \{\mathbf{x}(t), 0 \leq t \leq T\}$ to denote trajectories from the data space to the latent space in the continuous-time setting, and $\tau = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_N\}$ in the discrete-time setting.

Normalizing Flows: The trajectory in normalizing flows is modeled by a differential equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t, \theta), \tag{1}$$

parameterized by θ. This differential equation starts from random $\mathbf{x}(0) = \mathbf{x}$ and ends at $\mathbf{x}(T) = \mathbf{z}$. Denote by $p(\mathbf{x}(t))$ the probability distribution of $\mathbf{x}(t)$, then under mild assumptions, it evolves following [4]

$$\frac{\partial \log p(\mathbf{x}(t))}{\partial t} = -\text{tr}(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}). \tag{2}$$

Using this relation (1) (2) one can compute the likelihood of the model at any data point $\mathbf{x}$.

In the discrete-time setting, the map from $\mathbf{x}$ to $\mathbf{z}$ in normalizing flows is a composition of a collection of bijective functions as $F = F_N \circ F_{N-1} \cdots F_2 \circ F_1$. The trajectory $\tau = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_N\}$ satisfies

$$\mathbf{x}_i = F_i(\mathbf{x}_{i-1}, \theta), \quad \mathbf{x}_{i-1} = F_i^{-1}(\mathbf{x}_i, \theta) \tag{3}$$

for all i. Similar to Equation (2), based on the rule for change of variable, the log-likelihood of any data samples $\mathbf{x}_0 = \mathbf{x}$ can be evaluated as

$$\log p(\mathbf{x}_0) = \log p(\mathbf{x}_N) - \sum_{i=1}^{N} \log |\det(\frac{\partial F_i^{-1}(\mathbf{x}_i)}{\partial \mathbf{x}_i})|. \tag{4}$$

Since the exact likelihood is accessible in normalizing flows, these models can be trained by minimizing the negative log-likelihood directly.

Diffusion Models: The trajectories in diffusion models are modeled by stochastic differential equations. More explicitly, the forward process is of the form

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w}, \tag{5}$$

where the *drift* term $\mathbf{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is a vector-valued function, and the *diffusion* coefficient $g : \mathbb{R} \to \mathbb{R}$ is a scalar function (in fact, g is often chosen to be a constant). Here $\mathbf{w}$ denotes the standard Brownian motion. The forward process is normally a simple linear diffusion process [38, 16]. The forward trajectory τ can be sampled using (5) initialized with the data distribution. Denote by p_F the resulting probability distribution over the trajectories. With a slight abuse of notation, we also use p_F to denote the marginal distribution of the forward process.

The backward diffusion from $\mathbf{z} = \mathbf{x}(T)$ to $\mathbf{x} = \mathbf{x}(0)$ is of the form

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\mathbf{s}(\mathbf{x}, t, \theta)]dt + g(t)d\mathbf{w}. \tag{6}$$

It is well-known that when $\mathbf{s}$ coincides with the score function $\nabla \log p_F$, and $\mathbf{x}(T)$ in the forward and backward processes share the same distribution, the distribution p_B induced by the backward process (6) is equal to p_F [1],[29, Chapter 13]. To train the score network $\mathbf{s}(\mathbf{x}, t, \theta)$, one can use the KL divergence between p_F and p_B as an objective function to reduce the difference between p_F and p_B. When the difference is sufficiently small, p_F and p_B should have similar distribution over $\mathbf{x}(0)$, and one can then use the backward diffusion (6) to sample from the data distribution.

In the discrete setting, the trajectory distributions can be more explicitly written as

$$p_F(\tau) = p_F(\mathbf{x}_0) \prod_{i=1}^{N} p_F(\mathbf{x}_i|\mathbf{x}_{i-1}), \quad p_B(\tau) = p_B(\mathbf{x}_T) \prod_{i=1}^{N} p_B(\mathbf{x}_{i-1}|\mathbf{x}_i). \tag{7}$$

The KL divergence between p_F and p_B can be decomposed according to this expression (7). Most diffusion models use this decomposition, and meanwhile take advantage of the simple structure of the forward process (5), to evaluate the objective function in training [39, 40, 41].

3 Diffusion normalizing flow

We next present our diffusion normalizing flow models. Similar to diffusion models, the DiffFlow models also has a forward process

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t, \theta)dt + g(t)d\mathbf{w}, \tag{8}$$

and a backward process

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t, \theta) - g^2(t)\mathbf{s}(\mathbf{x}, t, \theta)]dt + g(t)d\mathbf{w}. \tag{9}$$

The major difference is that, instead of being a fixed linear function as in most diffusion models, the drift term $\mathbf{f}$ is also learnable in DiffFlow. The forward process is initialized with the data samples at $t = 0$ and the backward process is initialized with a given noise distribution at $t = T$. Our goal is to ensure the distribution of the backward process at time $t = 0$ is close to the real data distribution. That is, we would like the difference between $p_B(\mathbf{x}(0))$ and $p_F(\mathbf{x}(0))$ to be small.

To this end, we use the KL divergence between $p_B(\tau)$ and $p_F(\tau)$ over the trajectory space as the training objective function. Since

$$KL(p_F(\mathbf{x}(t))|p_B(\mathbf{x}(t))) \leq KL(p_F(\tau)|p_B(\tau)) \tag{10}$$

for any $0 \leq t \leq T$ by data processing inequality, small difference between $p_B(\tau)$ and $p_F(\tau)$ implies small difference between $p_B(\mathbf{x}(0))$ and $p_F(\mathbf{x}(0))$ in terms of KL divergence (more details are included in Appendix B).

3.1 Implementation

In real implementation, we discretize the forward process (8) and the backward process (9) as

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mathbf{f}_i(\mathbf{x}_i)\Delta t_i + g_i \delta_i^F \sqrt{\Delta t_i} \tag{11}$$

$$\mathbf{x}_i = \mathbf{x}_{i+1} - [\mathbf{f}_{i+1}(\mathbf{x}_{i+1}) - g_{i+1}^2 \mathbf{s}_{i+1}(\mathbf{x}_{i+1})]\Delta t_i + g_{i+1}\delta_i^B \sqrt{\Delta t_i}, \tag{12}$$

where $\delta_i^F, \delta_i^B \sim \mathcal{N}(0, \mathbf{I})$ are unit Gaussian noise, $\{t_i\}_{i=0}^N$ are the discretization time points, and $\Delta t_i = t_{i+1} - t_i$ is the step size at the i-th step. Here we have dropped the dependence on the parameter θ to simplify the notation. With this discretization, the KL divergence between trajectory distributions becomes

$$KL(p_F(\tau)|p_B(\tau)) = \underbrace{\mathbb{E}_{\tau\sim p_F}[\log p_F(\mathbf{x}_0)]}_{L_0} + \underbrace{\mathbb{E}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N)]}_{L_N} + \sum_{i=1}^{N-1} \underbrace{\mathbb{E}_{\tau\sim p_F}[\log \frac{p_F(\mathbf{x}_i|\mathbf{x}_{i-1})}{p_B(\mathbf{x}_{i-1}|\mathbf{x}_i)}]}_{L_i}. \tag{13}$$

The term L_0 in (13) is a constant determined by entropy of the dataset as

$$\mathop{\mathbb{E}}_{\tau\sim p_F}[\log p_F(\mathbf{x}_0)] = \mathop{\mathbb{E}}_{\mathbf{x}_0\sim p_F}[\log p_F(\mathbf{x}_0)] =: -H(p_F(x(0))).$$

The term L_N is easy to calculate since $p_B(x_N)$ is a simple distribution, typically standard Gaussian distribution.

To evaluate $L_{1:N-1}$, we estimate it over sampled trajectory from the forward process p_F. For a given trajectory τ sampled from $p_F(\tau)$, we need to calculate $p_B(\tau)$ along the same trajectory. To this end, a specific group of $\{\delta_i^B\}$ is chosen such that the same trajectory can be reconstructed from the backward process. Thus, δ_i^B satisfies

$$\delta_i^B(\tau) = \frac{1}{g_{i+1}\sqrt{\Delta t}}\left[\mathbf{x}_i - \mathbf{x}_{i+1} + [\mathbf{f}_{i+1}(\mathbf{x}_{i+1}) - g_{i+1}^2 \mathbf{s}_{i+1}(\mathbf{x}_{i+1})]\Delta t\right]. \tag{14}$$

Since δ_i^B is a Gaussian noise, the negative log-likelihood term $p_B(\mathbf{x}_i|\mathbf{x}_{i+1})$ is equal to $\frac{1}{2}(\delta_i^B(\tau))^2$ after dropping constants (see more details in Appendix B). In view of the fact that the expectation of $\sum_i \frac{1}{2}(\delta_i^F(\tau))^2$ remains a constant, minimizing Equation (13) is equivalent to minimizing the following loss (see Appendix C for the full derivation):

$$L := \mathbb{E}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N) + \sum_i \frac{1}{2}(\delta_i^B(\tau))^2] = \mathbb{E}_{\delta^F;\mathbf{x}_0\sim p_0}[-\log p_B(\mathbf{x}_N) + \sum_i \frac{1}{2}(\delta_i^B(\tau))^2], \tag{15}$$

where the last equality is based on a reparameterization trick [21]. We can minimize the loss in Equation (15) with Monto Carlo gradient estimation as in Algorithm 1.

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim$ Real data distribution
  $\delta^F_{1:N} \sim \mathcal{N}(0, \mathbf{I})$
  Discrete timestamps: $t^N_{i=0}$
  Sample: $\tau = \{\mathbf{x}_i\}^N_{i=0}$ based on $\delta^F_{1:N}$
  Gradient descent step $\nabla_\theta[-\log p_B(\mathbf{x}_N) + \sum_i \frac{1}{2}(\delta^B_i(\tau))^2]$
until converged

Algorithm 2 Stochastic Adjoint Algorithm for DiffFlow

Input: Forward trajectory $\{\mathbf{x}_i\}^N_{i=0}$
$\frac{\partial L}{\partial \mathbf{x}_N} = \frac{1}{2}\frac{\partial(\delta^B_N(\tau))^2}{\partial \mathbf{x}_N} - \frac{\partial \log p_B(\mathbf{x}_N)}{\mathbf{x}_N}$
$\frac{\partial L}{\partial \theta} = 0$
for $i = N, N-1, \cdots, 1$ **do**
  $\frac{\partial L}{\partial \mathbf{x}_{i-1}} = (\frac{\partial L}{\partial \mathbf{x}_i} + \frac{1}{2}\frac{\partial(\delta^B_i(\tau))^2}{\partial \mathbf{x}_i})\frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} + \frac{1}{2}\frac{\partial(\delta^B_i(\tau))^2}{\partial \mathbf{x}_{i-1}}$
  $\frac{\partial L}{\partial \theta} += \frac{1}{2}\frac{\partial(\delta^B_i(\tau))^2}{\partial \theta} + (\frac{\partial L}{\partial \mathbf{x}_i} + \frac{1}{2}\frac{\partial(\delta^B_i(\tau))^2}{\partial \mathbf{x}_i})\frac{\partial \mathbf{x}_i}{\partial \theta}$
end for

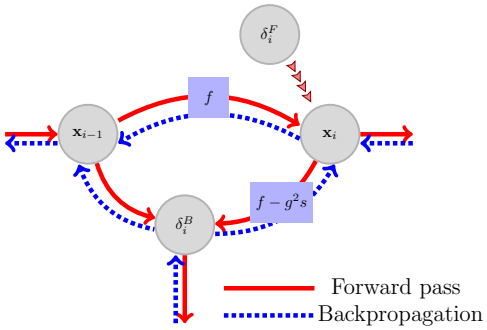


Figure 2: Gradient Flowchart.

3.2 Stochastic Adjoint method

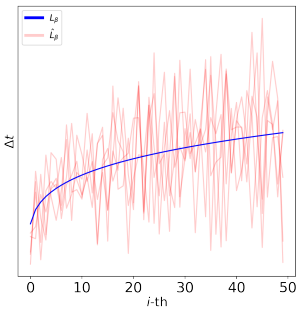
One challenge in training DiffFlow is memory consumption. When a naive backpropagation strategy is used, the memory consumption explodes quickly. Indeed, differentiating through the operations of the forward pass requires unrolling networks N times and caching all network intermediate values for every step, which prevents this naive implementation of DiffFlow from being applied in high dimensional applications. Inspired by the adjoint method in Neural ODE [4], we propose the adjoint variable $\frac{\partial L}{\partial \mathbf{x}_i}$ and a stochastic adjoint algorithm that allows training the DiffFlow model with reduced memory consumption. In this adjoint method, we cache intermediate states $\mathbf{x}_i$ and, based on these intermediate states, reproduce the whole process, including δ^F_i, δ^B_i as well as f_i, s_i exactly.

We note another similar approach [26] of training SDEs caches random noise $d\mathbf{w}$ and further takes advantage of the pseudo-random generator to save memory for the intermediate noises, resulting in constant memory consumption However, the approach can not reproduce exact trajectories due to time discretization error and requires extra computation to recover $d\mathbf{w}$ from the pseudo-random generator. We found that in our image experiments in Section 4, the cached $\{\mathbf{x}_i\}$ consumes only about 2% memory out of all the memory being used during training. Due to the accuracy and acceptable memory overhead, the introduced stochastic adjoint approach is a better choice for DiffFlow. We summarize the method in Algorithm 2 and Figure 2. We also include the PyTorch [34] implementation in the supplemental material.

3.3 Time discretization and progressive training

We propose two time discretization schemes for training DiffFlow: fixed timestamps L_β and flexible timestamps $\hat{L}_\beta$. For fixed timestamps, the time interval $[0, T]$ is discretized with fixed schedule $t_i = (\frac{i}{N})^\beta T$. With such fixed time discretization over batches, we denote the loss function as L_β. Empirically, we found $\beta = 0.9$ works well. This choice of β increases stepsize Δt_i when the forward process approaches $\mathbf{z} = \mathbf{x}_N$ and provides higher resolution when the backward process is close to $\mathbf{x}_0$. We found such discretization generates samples with good quality and high fidelity details. The choice of polynomial function is arbitrary; other functions of similar sharps may work as well.

Figure 3: Δt_i of L_β and $\hat{L}_\beta$



In the flexiable timestamps scheme, we train different batches with different time discretization points. Specifically, t_i is sampled uniformly from the interval $[(\frac{i-1}{N-1})^\beta T, (\frac{i}{N-1})^\beta T]$ for each batch. We denote the training objective function with such random time discretization as $\hat{L}_\beta$. We empirically found such implementation results in lower loss and better stability when we conduct progressive training where we increase N gradually as training progresses. In progressive training, we refine the forward and backward processes as N increase. This training scheme can significantly save training time compared with the other method that uses a fixed large N during the whole process. Empirically, we found that progressive training can speed up the training up to 16 times.

To understand why such random time discretization scheme is more stable, we hypothesis that this choice encourages a smoother $\mathbf{f}, \mathbf{s}$ since it seeks functions $\mathbf{f}, \mathbf{s}$ to reduce objective loss under different sets of $\{t_i\}$ instead of a specific $\{t_i\}$. We illustrate fixed timestamps in L_β and a realization of random discretization in $\hat{L}_\beta$ in Figure 3 with $\beta = 0.9$.

3.4 Learnable forward process

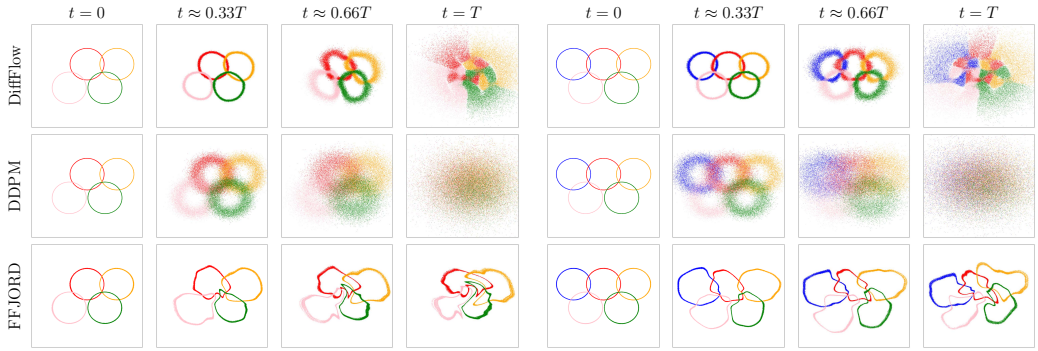


Figure 4: Illustration of forwarding trajectories of DiffFlow, DDPM, and FFJORD. Each row shows two trajectories of transforming data distributions, four rings, and Olympics rings, to a base distribution. Different modes of densities are in different colors. Though FFJORD adjusts forward process based on data, its bijective property prevents the approach from expanding density support to the whole space. DDPM can transform data distributions into Gaussian distribution but a data-invariant way of adding noise can corrupt the details of densities, e.g., the densities at the intersections of the rings. DiffFlow not only transforms data into base distribution but also keeps the topological information of the original datasets. Points from the same ring are transformed into continental plates instead of being distributed randomly.

The forward process not only is responsible for driving data into latent space, but also provides enough supervised information to learning backward process. Thanks to bijective property, NFs can reconstruct data exactly but there is no guarantee that it can reach the standard Gaussian. At the other extreme, Denoising diffusion probabilistic models (DDPM) [17] adopt a data-invariant forward diffusing schema, ensuring that $\mathbf{x}_N$ is Gaussian. DDPM can even reach Gaussian in one step with $N = 1$, which output noise disregarding data samples. However, backward process will be difficult to learn if data is destroyed in one step. Therefore, DDPM adds noise slowly and often needs more than one thousand steps for diffusion.

The forward module of DiffFlow is a combination of normalizing flow and diffusion models. We show the comparision in fitting toy 2D datasets in Figure 4. We are especially interested in data with well-separated modes and sharp density boundaries. Those properties are believed to appear in various datasets. As stated by manifold hypothesis [36], real-world data lie on low-dimensional manifold [28] embedded in a high-dimensional space. To construct the distributions in Figure 4, we rotate the 1-d Gaussian distribution $\mathcal{N}(1, 0.001^2)$ around the center to form a ring and copy the rings to different locations.

As a bijective model, FFJORD [14] struggles to diffuse the concentrated density mass into a Gaussian distribution. DiffFlow overcomes expressivity limitations of the bijective constraint by adding noise. As added noise shrinks to zero, the DiffFlow has no stochasticity and degrades to a flow-based model. Based on this fact, we present the following theorem with proof in Appendix A.

Theorem 1. *As diffusion coefficients $g_i \to 0$, DiffFlow reduces to Normalizing Flow. Moreover, minimizing the objective function in Equation* (13) *is equivalent to minimizing the negative log-likelihood as in Equation* (4)*.*

DDPM [17] uses a fixed noising transformation. Due to the data-invariant approach $p(\mathbf{x}_T|\mathbf{x}_0) = p(\mathbf{x}_T)$, points are diffused in the same way even though they may appear in different modes or different datasets. We observe that sharp details are destroyed quickly in DDPM diffusion, such as the intersection regions between rings. However, with the help of learnable transformation, DiffFlow diffuses in a much efficient way. The data-dependent approach shows different diffusion strategies on different modes and different datasets. Meanwhile, similar to NFs, it keeps some topological information for learning backward processes. We include more details about the toy sample in Section 4.

4 Experiments

We evaluate the performance of DiffFlow in sample quality and likelihood on test data. To evaluate the likelihood, we adopt the marginals distribution equivalent SDEs

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t, \theta) - \frac{1+\lambda^2}{2} g^2(t)\mathbf{s}(\mathbf{x}, t, \theta)]dt + \lambda g(t) d\mathbf{w}, \tag{16}$$

with $\lambda \geq 0$ (Proof see Appendix H). When $\lambda = 0$, it reduces to probability ODE [41]. The ODE provides an efficient way to evaluate the density and negative log-likelihood. For any $0 \leq \lambda \leq 1$, the above SDE can be used for sampling. Empirically, we found $\lambda = 1$ has the best performance.

4.1 Synthetic 2D examples

We compare the performance of DiffFlow and existing diffusion models and NFs on estimating the density of 2-dimensional data. We compare the forward trajectories of DiffFlow, DDPM [17] and FFJORD [14][1] in Figure 4 and its sampling performance in Figure 5. To make a fair comparison, we build models with comparable network sizes, around 90k learnable parameters. We include more training and model details in Appendix E.

All three algorithms have good performance on datasets whose underlying distribution has smooth density, such as 2 spirals. However, when we shrink the support of samples or add complex patterns, performance varies significantly. We observe that FFJORD leaks many samples out of the main modes and datasets with complex details and sharp density exacerbates the disadvantage.

DDPM has higher sample quality but blurs density details, such as intersections between rings, areas around leaves of the Fractal tree, and boxes in Sierpiński Carpet. The performance is within expectation given that details are easy to be destroyed and ignored with the data-invariant noising schema. On the less sharp dataset, such as 2 Spirals and Checkerboard, its samples align with data almost perfectly.

DiffFlow produces the best samples (according to a human observer). We owe the performance to the flexible noising forward process. As illustrated in Figure 4, DiffFlow provides more clues and retains detailed patterns longer for learning its reverse process. We also report a comprehensive comparison of the negative likelihood and more analysis in Appendix E. DiffFlow has a much lower negative likelihood, especially on sharp datasets.

4.2 Density estimation on real data

We perform density estimation experiments on five tabular datasets [33]. We employ the probability flow to evaluate the negative log-likelihood. We find that our algorithm has better performance in most datasets than most approaches trained by directly minimizing negative log-likelihood, including NFs and autoregressive models. DiffFlow outperforms FFJORD by a wide margin on all datasets except HEPMASS. Compared with autoregressive models, it excels NAF [18] on all but GAS. Those models require $\mathcal{O}(d)$ computations to sample from. Meanwhile, DiffFlow is quite effective in achieving such performance with MLPs that have less than 5 layers. We include more details in Appendix F

4.3 Image generation

In this section, we report the quantitative comparison and qualitative performance of our method and existing methods on common image datasets, MNIST [24] and CIFAR-10 [23]. We use the same

[1] Implementation of FFJORD and DDPM are based on official codebases.

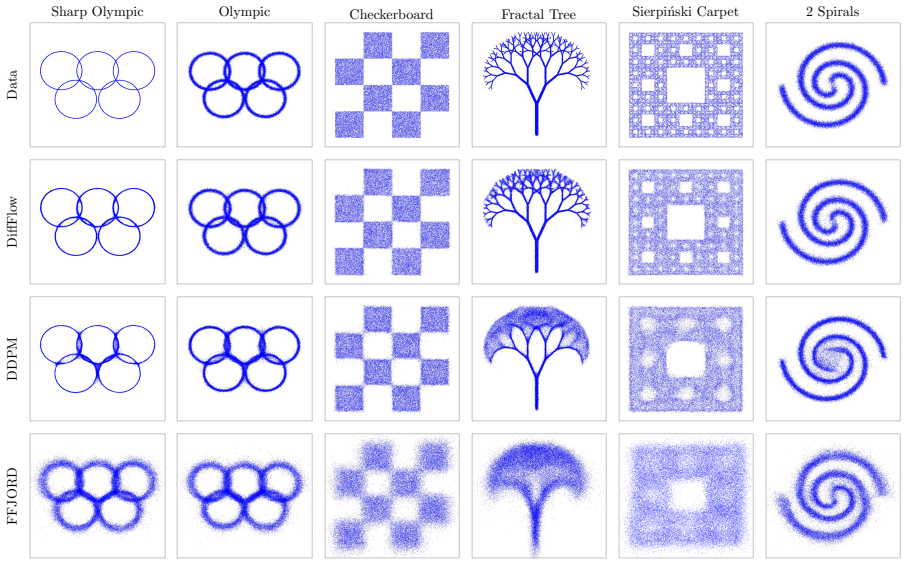


Figure 5: Samples from DiffFlow, DDPM and FFJORD on 2-D datasets. All three models have reasonable performance on datasets that have smooth underlying distributions. But only DiffFlow is capable to capture complex patterns and provides sharp samples when dealing with more challenging datasets.

Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
RealNVP [8]	-0.17	-8.33	18.71	13.55	-153.28
FFJORD [14]	-0.46	-8.59	**14.92**	10.43	-157.40
DiffFlow (ODE)	**-1.04**	-10.45	15.04	**8.06**	**-157.80**
MADE [11]	3.08	-3.56	20.98	15.59	-148.85
MAF [33]	-0.24	-10.08	17.70	11.75	-155.69
TAN [31]	-0.48	-11.19	15.12	11.01	-157.03
NAF [18]	-0.62	**-11.96**	15.09	8.86	-157.73

Table 1: Average negative log-likelihood (in nats) on tabular datasets [33] for density estimation (lower is better).

unconstrained U-net style model as used successfully by Ho et al. [17] for drift and score network. We reduce the network size to half of the original DDPM network so that the total number of trainable parameters of DiffFlow and DDPM are comparable. We use small $N = 10$ at the beginning of training and slowly increase to large N as training proceeds. The schedule of N reduces the training time greatly compared with using large N all the time. We use constants $g_i = 1$ and $T = 0.05$ for MNIST and CIFAR10, and $N = 30$ for sampling MNIST data and $N = 100$ for sampling CIFAR10. As it is reported by Jolicoeur-Martineau et al. [19], adding noise at the last step will significantly lower sampling quality, we use one single denoising step at the end of sampling with Tweedie's formula [10].

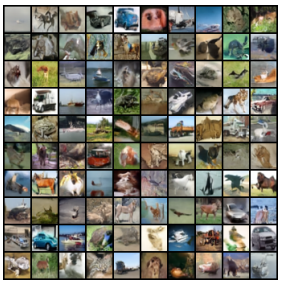
We report negative log-likelihood (NLL) in bits per dimension or negative ELBO if NLL is unavailable. On MNIST, we achieve competitive performance on NLL as in Table 2. We show the uncurated samples from DiffFlow in Figure 6 and Figure 7. On CIFAR-10, DiffFlow also achieves competitive NLL performance as shown in Table 3. DiffFlow performs better than normalizing flows and DDPM models, but is slightly worse than DDPM++(sub, deep, sub-vp) and Improved DDPM. However, these approaches conduct multiple architectural improvements and use much deeper and wider networks. We also report the popular sample metric, Fenchel Inception Distance (FID) [15]. DiffFLow has a lower FID score than normalizing flows and has competitive performance compared with DDPM trained with unweighted variational bounds, DDPM and Improved DDPM. It is worse than DDPM trained with reweighted loss, DDPM (L_s), DDPM cont, and DDPM++ [17, 41, 30]. Besides, sampling

Table 2: NLL on MNIST

Model	NLL (↓)
RealNVP [8]	1.06
Glow [20]	1.05
FFJORD [14]	0.99
ResFlow [5]	0.97
DiffFlow	0.93

Figure 6: MNIST Samples

Figure 7: CIFAR10 Samples



quality with different sampling steps N are compared in Table 4 [2]. The advantage of DiffFlow is clear when we compare relative FIDs degeneracy ratio with $N = 100$ respectively. DiffFlow is able to retain better sampling quality when decreasing N. Full details on architectures used, training setup details, and more samples can be found in Appendix G.

Table 3: NLLs and FIDs on CIFAR-10.

Model	NLL(↓)	FID (↓)
RealNVP [8]	3.49	-
Glow [20]	3.35	46.90
Flow++ [16]	3.29	-
FFJORD [14]	3.40	-
ResFlow [5]	3.28	46.37
DDPM (L) [17]	$\leq$ 3.70	13.51
DDPM (L_s) [17]	$\leq$ 3.75	3.17
DDPM (L_s)(ODE) [41]	3.28	3.37
DDPM cont. (sub-VP) [41]	3.05	3.56
DDPM++ (sub-VP) [41]	3.02	3.16
DDPM++ (deep, sub-VP) [41]	2.99	2.94
Improved DDPM [30]	$\leq$2.94	11.47
DiffFlow (L_β)	$\leq$ 3.71	13.87
DiffFlow ($\hat{L}_\beta$)	$\leq$ 3.67	13.43
DiffFlow ($\hat{L}_\beta$, ODE)	3.04	14.14

Table 4: FIDs with various N

N	DiffFlow	DDPM (L)	DDPM (L_s)	DDIM
5	28.31	373.51	370.23	44.69
10	22.56	364.64	365.12	18.62
20	17.98	138.84	135.44	10.89
50	14.72	47.12	34.56	7.01
100	13.43	22.23	10.04	5.63

Table 5: Relative FIDs degeneracy ratio

N	DiffFlow	DDPM (L)	DDPM (L_s)	DDIM
5	2.12	16.80	37.02	7.94
10	1.68	16.40	36.12	3.31
20	1.34	6.24	13.54	1.93
50	1.10	2.12	3.45	1.24
100	1.0	1.0	1.0	1.0

5 Related work

Normalizing flows [8, 35] have recently received lots of attention due to its exact density evaluation and ability to model high dimensional data [20, 9]. However, the bijective requirement poses limitations on modeling complex data, both empirically and theoretically [42, 6]. Some works attempt to relax the bijective requirement; discretely index flows [9] use domain partitioning with only locally invertible functions. Continuously indexed flows [6] extend discretely indexing to a continuously indexing approach. As pointed out in Stochastic Normalizing Flows (SNF) [42], stochasticity can effectively improve the expressive power of the flow-based model in low dimension applications. The architecture used in SNF, which requires known underlying energy models, presents challenges for density learning tasks; SNF is designed for sampling from unnormalized probability distribution instead of density estimation. Besides, even with ideal networks and infinite amount of data, due to the predefined stochstic block being used, SNF cannot find models with aligned forward and backward distribution as DiffFlow.

When it comes to stochastic trajectories, minimizing the distance between trajectory distributions has been explored in existing works. Denoising diffusion model [38] uses a fixed linear forward diffusion schema and reparameterizes the KL divergence such that minimizing loss is possible without computing whole trajectories. Diffusion models essentially corrupt real data iteratively and learn to remove the noise when sampling. Recently, Diffusion models have shown the capability to model

[2] The performance of DDPM is evaluated based on the officially released checkpoint with L_s denotes for L_{simple} in the original paper.

high-dimensional data distribution, such as images [17, 40], shapes [3], text-to-speech [22]. Lately, the Score-based model [41] provides a unified framework for score-matching methods and diffusion models based on stochastic calculus. The diffusion processes and sampling processes can be viewed as forwarding SDE and reverse-time SDE. Thanks to the linear forward SDE being used in DDPM, the forward marginal distributions have a closed-form and are suitable for training score functions on large-scale datasets. Also, due to the reliance on fixed linear forward process, it takes thousands of steps to diffuse data and generate samples. DiffFlow considers general SDEs and nosing and sampling are more efficient.

Existing Neural SDE approaches suffer from poor scaling properties. Backpropagating through solver [12] has a linear memory complexity with the number of steps. The pathwise approach [13] scales poorly in computation complexity. Our stochastic adjoint approach shares a similar spirit with SDE adjoint sensitivity [25]. The choice of caching noise requires high resolution of time discretization and prevents the approach from scaling to high dimension applications. By caching the trajectory states, DiffFlow can use a coarser discretization and deploy on larger dimension problems and problems with more challenging densities. The additional memory footprint is negligible compared with the other network memory consumption in DiffFlow.

6 Limitations

While DiffFlow gains more flexibility due to the introduction of a learnable forward process, it loses the analytical form for $p_F(\mathbf{x}_t|\mathbf{x}_0)$ and thus the training less efficient compared with score-based loss [41]. Training DiffFlow relies on backpropagation through trajectories and is thus significantly slower than diffusion models with affine drift. Empirically, we found DiffFlow is about 6 times slower than DDPM in 2d toy examples, 55 times in MNIST, and 160 times in CIFAR10 without progressive training in Section 3.3. Though the stochastic adjoint method and progressive training help save memory footprint and reduce training time, the training of DiffFlow is still more expensive than DDPM and its variants. On the other hand, compared with normalizing flows, the extra noise in DiffFlow boosts the expressive power of the model with little extra cost. Though DiffFlow trained based on SDE, its marginal distribution equivalent ODE 2 shows much better performance than its counterpart trained with ODE [14]. It is interesting to investigate, both empirically and theoretically, the benefit in terms of expressiveness improvement caused by stochastic noise for training normalizing flows.

7 Conclusions

We proposed a novel algorithm, the diffusion normalizing flow (DiffFlow), for generative modeling and density estimation. The proposed method extends both the normalizing flow models and the diffusion models. Our DiffFlow algorithm has two trainable diffusion processes modeled by neural SDEs, one forward and one backward. These two SDEs are trained jointly by minimizing the KL divergence between them. Compared with most normalizing flow models, the added noise in DiffFlow relaxes the bijectivity condition in deterministic flow-based models and improves their expressive power. Compared with diffusion models, DiffFlow learns a more flexible forward diffusion that is able to transform data into noise more effectively and adaptively. In our experiments, we observed that DiffFlow is able to model distributions with complex details that are not captured by representative normalizing flow models and diffusion models, including FFJORD, DDPM. For CIFAR10 dataset, our DiffFlow method has worse performance than DDPM in terms of FID score. We believe our DiffFlow algorithm can be improved further by using different neural network architectures, different time discretizing method and different choices of time interval. We plan to explore these options in the near future.

Our algorithm is able to learn the distribution of high-dimensional data and then generate new samples from it. Like many other generative modeling algorithms, it may be potentially used to generate misleading data such as fake images or videos.

Acknowledgements

The authors would like to thank the anonymous reviewers for useful comments. This work is supported in part by grants NSF CAREER ECCS-1942523 and NSF CCF-2008513.

References

[1] Anderson, B. D. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12 (3):313–326, 1982.

[2] Bordes, F., Honari, S., and Vincent, P. Learning to generate samples from noise through infusion training. *arXiv preprint arXiv:1703.06975*, 2017.

[3] Cai, R., Yang, G., Averbuch-Elor, H., Hao, Z., Belongie, S., Snavely, N., and Hariharan, B. Learning gradient fields for shape generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[4] Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

[5] Chen, R. T. Q., Behrmann, J., Duvenaud, D., and Jacobsen, J. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, 2019.

[6] Cornish, R., Caterini, A., Deligiannidis, G., and Doucet, A. Relaxing bijectivity constraints with continuously indexed normalising flows. In *International Conference on Machine Learning*, pp. 2133–2143. PMLR, 2020.

[7] Deng, Y., Bakhtin, A., Ott, M., Szlam, A., and Ranzato, M. Residual energy-based models for text generation. *arXiv preprint arXiv:2004.11714*, 2020.

[8] Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[9] Dinh, L., Sohl-Dickstein, J., Larochelle, H., and Pascanu, R. A rad approach to deep mixture models. *arXiv preprint arXiv:1903.07714*, 2019.

[10] Efron, B. Tweedie's formula and selection bias. *Journal of the American Statistical Association*, 106(496): 1602–1614, 2011.

[11] Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889. PMLR, 2015.

[12] Giles, M. and Glasserman, P. Smoking adjoints: Fast monte carlo greeks. *Risk*, 19(1):88–92, 2006.

[13] Gobet, E. and Munos, R. Sensitivity analysis using itô–malliavin calculus and martingales, and application to stochastic optimal control. *SIAM Journal on control and optimization*, 43(5):1676–1713, 2005.

[14] Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[15] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[16] Ho, J., Chen, X., Srinivas, A., Duan, Y., and Abbeel, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pp. 2722–2730. PMLR, 2019.

[17] Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *arXiv preprint arXiv:2006.11239*, 2020.

[18] Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. In *International Conference on Machine Learning*, pp. 2078–2087. PMLR, 2018.

[19] Jolicoeur-Martineau, A., Piché-Taillefer, R., Combes, R. T. d., and Mitliagkas, I. Adversarial score matching and improved sampling for image generation. *arXiv preprint arXiv:2009.05475*, 2020.

[20] Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[21] Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[22] Kong, Z., Ping, W., Huang, J., Zhao, K., and Catanzaro, B. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020.

[23] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

[24] LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[25] Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pp. 3870–3882. PMLR, 2020.

[26] Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pp. 3870–3882. PMLR, 2020.

[27] Meng, C., Song, J., Song, Y., Zhao, S., and Ermon, S. Improved autoregressive modeling with distribution smoothing. *arXiv preprint arXiv:2103.15089*, 2021.

[28] Narayanan, H. and Mitter, S. Sample complexity of testing the manifold hypothesis. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems-Volume 2*, pp. 1786–1794, 2010.

[29] Nelson, E. *Dynamical theories of Brownian motion*. Princeton university press, 2020.

[30] Nichol, A. and Dhariwal, P. Improved denoising diffusion probabilistic models. *arXiv preprint arXiv:2102.09672*, 2021.

[31] Oliva, J., Dubey, A., Zaheer, M., Poczos, B., Salakhutdinov, R., Xing, E., and Schneider, J. Transformation autoregressive networks. In *International Conference on Machine Learning*, pp. 3898–3907. PMLR, 2018.

[32] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[33] Papamakarios, G., Pavlakou, T., and Murray, I. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

[34] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[35] Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pp. 1530–1538. PMLR, 2015.

[36] Roweis, S. T. and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[37] Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[38] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

[39] Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, pp. 11895–11907, 2019.

[40] Song, Y. and Ermon, S. Improved techniques for training score-based generative models. *arXiv preprint arXiv:2006.09011*, 2020.

[41] Song, Y., Sohl-Dickstein, J., Kingma, D., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=PxTIG12RRHS`.

[42] Wu, H., Köhler, J., and Noe, F. Stochastic normalizing flows. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 5933–5944. Curran Associates, Inc., 2020.

A Proof of Theorem 1: Normalizing Flow as a special case of DiffFlow

Proof. As $g(t) \to 0$, the stochastic trajectories become deterministic. The forward SDE in DiffFlow reduces to the ODE in Normalizing Flow. In the discrete implementation, DiffFlow reduces to a normalizing flow consisting of the discrete layers

$$F_i(\mathbf{x}, \theta) = \mathbf{x} + \mathbf{f}(\mathbf{x}, t_i, \theta)\Delta t_i \quad \mathbf{x}_i = F_i(\mathbf{x}_{i-1}, \theta). \tag{17}$$

In the following, we show that minimizing the KL divergence between trajectory distributions is equivalent to minimizing the negative log-likelihood in Normalizing flow as $g(t) \to 0^+$. The derivation is essentially the same as that in [42] except that our model is for density estimation instead of sampling [42].

The discrete forward dynamics of DiffFlow is $\mathbf{x}_i = F_i(\mathbf{x}_{i-1}) + g_i \delta_i^F \sqrt{\Delta t_i}$, and it is associated with the conditional distribution

$$p_F(\mathbf{x}_i|\mathbf{x}_{i-1}) = \mathcal{N}(\mathbf{x}_i - F_i(\mathbf{x}_{i-1}); \mathbf{0}, g_i^2 \mathbf{I}\Delta t_i). \tag{18}$$

Denote the conditional distribution for the backward process by

$$p_B(\mathbf{x}_{i-1}|\mathbf{x}_i) = \frac{p_B(\mathbf{x}_{i-1})p_B(\mathbf{x}_i|\mathbf{x}_{i-1})}{\int p_B(\mathbf{x})p_B(\mathbf{x}_i|\mathbf{x})d\mathbf{x}}. \tag{19}$$

For a fixed forward process, minimizing the KL divergence implies that $p_B(\mathbf{x}_{i-1}|\mathbf{x}_i)$ is the posterior of $p_F(\mathbf{x}_i|\mathbf{x}_{i-1})$, that is, $p_F(\mathbf{x}_i|\mathbf{x}_{i-1}) = p_B(\mathbf{x}_i|\mathbf{x}_{i-1})$. This is possible when the diffusion intensity $g(t)$ is small. It follows that

$$\frac{p_F(\mathbf{x}_i|\mathbf{x}_{i-1})}{p_B(\mathbf{x}_{i-1}|\mathbf{x}_i)} = \frac{\int p_B(\mathbf{x})p_B(\mathbf{x}_i|\mathbf{x})d\mathbf{x}}{p_B(\mathbf{x}_{i-1})}. \tag{20}$$

In the deterministic limit $g_i \to 0^+$,

$$\begin{aligned}
\lim_{g\to 0^+} \int p_B(\mathbf{x})p_B(\mathbf{x}_i|\mathbf{x}) &= \lim_{g\to 0^+} \int p_B(\mathbf{x})\mathcal{N}(\mathbf{x}_i - F_i(\mathbf{x}); \mathbf{0}, g_i^2\mathbf{I}\Delta t_i)d\mathbf{x} \\
&= \lim_{g\to 0^+} \int p_B(F_i^{-1}(\mathbf{x}'))\mathcal{N}(\mathbf{x}_i - \mathbf{x}'; \mathbf{0}, g_i^2\mathbf{I}\Delta t_i)|\det(\frac{\partial F_i^{-1}(\mathbf{x}')}{\partial \mathbf{x}'})|d\mathbf{x}' \\
&= p_B(F_i^{-1}(\mathbf{x}_i))|\det(\frac{\partial F_i^{-1}(\mathbf{x}_i)}{\partial \mathbf{x}_i})| \\
&= p_B(\mathbf{x}_{i-1})|\det(\frac{\partial F_i^{-1}(\mathbf{x}_i)}{\partial \mathbf{x}_i})|.
\end{aligned}$$

The second equality is based on the rule of change variable for the map $\mathbf{x}' = F_i^{-1}(\mathbf{x})$. Plugging the above and Equation (20) into Equation (13), we arrive at

$$\mathbb{E}_{\tau\sim p_F}[\log p_F(\mathbf{x}_0) - \log p_B(\mathbf{x}_N) + \sum_{i=1}^{N} \log|\det \frac{\partial F_i^{-1}(\mathbf{x}_i)}{\partial \mathbf{x}_i}|]. \tag{21}$$

Therefore minimizing the KL divergence between forward and backward trajectory distributions is equivalent to maximizing the log probability as in Equation (4). □

B Proofs for Eq (10) and Eq (14)

B.1 Proof of Eq (10)

Eq (10) follows from the disintegration of measure theorem and the non-negativity of KL divergence. More specifically, by disintegration of measure,

$$p_F(\tau) = \int p_F(\tau|\mathbf{x}(t))p_F(\mathbf{x}(t))d\mathbf{x}(t) \quad p_B(\tau) = \int p_B(\tau|\mathbf{x}(t))p_B(\mathbf{x}(t))d\mathbf{x}(t).$$

Eq (10) follows that

$$KL(p_F(\tau)|p_B(\tau)) = KL(p_F(\mathbf{x}(t))|p_B(\mathbf{x}(t)) + \int KL(p_F(\tau|\mathbf{x}(t))|p_B(\tau|\mathbf{x}(t)))d\mathbf{x}(t) \geq KL(p_F(\mathbf{x}(t))|p_B(\mathbf{x}(t)).$$

B.2 Proof for Eq (14) and log-likelihood of $p_B(\mathbf{x}_i|\mathbf{x}_{i+1})$

For a given trajectory $\tau = \{\mathbf{x}_i\}$ generated from forward dynamics, the density of such trajectory in the distribution induced by backward dynamics

$$p_B(\tau) = p_B(\mathbf{x}_T)\prod_{i=1}^{N} p_B(\mathbf{x}_{i-1}|\mathbf{x}_i).$$

And term $p_B(\mathbf{x}_i|\mathbf{x}_{i+1})$ can be written as

$$p_B(\mathbf{x}_i|\mathbf{x}_{i+1}) = \int p_B(\mathbf{x}_i|\mathbf{x}_{i+1}, \delta_i^B)\mathcal{N}(\delta_i^B)d\delta_i^B,$$

where $\mathcal{N}$ denotes the standard normal distribution. The distribution $p_B(\mathbf{x}_i|\mathbf{x}_{i+1})$ and $p_B(\mathbf{x}_i|\mathbf{x}_{i+1}, \delta_i^B)$ encode dynamics in Eq (9). Therefore density can be reformulated as

$$p_B(\mathbf{x}_i|\mathbf{x}_{i+1}, \delta_i^B) = \mathbb{I}_{\mathbf{x}_i = \mathbf{x}_{i+1} - [\mathbf{f}_{i+1}(\mathbf{x}_{i+1}) - g_{i+1}^2 \mathbf{s}_{i+1}(\mathbf{x}_{i+1})]\Delta t_i + g_{i+1}\delta_i^B\sqrt{\Delta t_i}}$$

$$p_B(\mathbf{x}_i|\mathbf{x}_{i+1}) = \mathcal{N}(\frac{1}{g_{i+1}\sqrt{\Delta t}}\left[\mathbf{x}_i - \mathbf{x}_{i+1} + [\mathbf{f}_{i+1}(\mathbf{x}_{i+1}) - g_{i+1}^2\mathbf{s}_{i+1}(\mathbf{x}_{i+1})]\Delta t\right])$$

Thus, the negative log-likelihood term $p_B(\mathbf{x}_i|\mathbf{x}_{i+1})$ is equal to $\frac{1}{2}(\delta_i^B(\tau))^2$ where $\delta_i^B(\tau)$ is given by Eq 14.

C Derivation of the loss function

The KL divergence between the distributions induced by the forward process and the backward process is

$$\begin{aligned} KL(p_F(\tau)|p_B(\tau)) &= KL(p_F(\mathbf{x}_0)p_F(\tau|\mathbf{x}_0)|p_B(\mathbf{x}_N)p_B(\tau|\mathbf{x}_N)) \\ &= \mathop{\mathbb{E}}_{p_F(\mathbf{x}_0)p_F(\tau|\mathbf{x}_0)}[\log p_F(\mathbf{x}_0) - \log p_B(\mathbf{x}_N) + \log \frac{p_F(\tau|\mathbf{x}_0)}{p_B(\tau|\mathbf{x}_N)}] \end{aligned} \tag{22}$$

In discrete implementation, the density ratio between conditional forward and reverse process can be reformulated as

$$\frac{p_F(\tau|\mathbf{x}_0)}{p_B(\tau|\mathbf{x}_N)} = \frac{p_F(\mathbf{x}_N|\mathbf{x}_{N-1})\cdots p_F(\mathbf{x}_2|\mathbf{x}_1)p_F(\mathbf{x}_1|\mathbf{x}_0)}{p_B(\mathbf{x}_{N-1}|\mathbf{x}_N)\cdots p_B(\mathbf{x}_1|\mathbf{x}_2)p_B(\mathbf{x}_0|\mathbf{x}_1)}.$$

Since

$$\frac{p_F(\mathbf{x}_i|\mathbf{x}_{i-1})}{p_B(\mathbf{x}_{i-1}|\mathbf{x}_i)} = \frac{g_i\mathcal{N}(\delta_i^F|0,1)}{g_{i+1}\mathcal{N}(\delta_i^B|0,1)}.$$

and

$$\log\frac{p(\delta_i^F)}{p(\delta_i^B)} = \frac{1}{2}[(\delta_i^B)^2 - (\delta_i^F)^2],$$

we can rewrite the loss function as

$$\begin{aligned} KL(p_F(\tau)|p_B(\tau)) &= -H(p_F(\mathbf{x}_0)) + \mathop{\mathbb{E}}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N) + \sum_{i=1}^N \log\frac{p_F(\mathbf{x}_i|\mathbf{x}_{i-1})}{p_B(\mathbf{x}_{i-1}|\mathbf{x}_i)}] \\ &= -H(p_F(\mathbf{x}_0)) + \mathop{\mathbb{E}}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N) + \sum_{i=1}^N \log\frac{p_F(\mathbf{x}_i|\mathbf{x}_{i-1})}{p_B(\mathbf{x}_{i-1}|\mathbf{x}_i)}] \\ &= -H(p_F(\mathbf{x}_0)) + \mathop{\mathbb{E}}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N) + \log\frac{g_0}{g_N} + \sum_{i=1}^N \log\frac{p(\delta_i^F)}{p(\delta_i^B)}]. \end{aligned}$$

The first and third terms are constant, so minimizing KL divergence is equivalent to minimizing

$$\mathop{\mathbb{E}}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N) + \sum_{i=0}^N \frac{1}{2}((\delta_i^B)^2 - (\delta_i^F)^2)].$$

We sample the trajectory τ from the forward process and δ_i^F is Gaussian random noise that is independent of the model. Thus, its expectation is a constant and the objective function can be simplified to

$$\mathop{\mathbb{E}}_{\tau\sim p_F}[-\log p_B(\mathbf{x}_N) + \sum_{i=1}^N \frac{1}{2}(\delta_i^B)^2]. \tag{23}$$

Additionally, for a given trajectory τ, the following equations holds

$$f_i\Delta t + g_i\delta_i^F\sqrt{\Delta t} = [f_{i+1} - g_{i+1}^2 s_{i+1}]\Delta t_i + g_{i+1}\delta_i^B\sqrt{\Delta t_i}.$$

Hence, the backward noise can be evaluated as

$$\delta_i^B = \frac{g_i}{g_{i+1}}\delta_i^F - [\frac{f_{i+1} - f_i}{g_{i+1}} + g_{i+1}s_{i+1}]\sqrt{\Delta t_i}. \tag{24}$$

D Implemtation of Stochastic Adjoint in PyTorch [34]

Below is the implementation of stochastic adjoint method with `torch.autograd.Function` in PyTorch. The roles of most helper functions and variables can be informed from their names and comments.

```
class SdeF(torch.autograd.Function):
    @staticmethod
    @amp.custom_fwd
    def forward(ctx, x, model, times, diffusion, condition,*m_params):
        shapes = [y0_.shape for y0_ in m_params]

        def _flatten(parameter):
            # flatten the gradient dict and parameter dict
            ...

        def _unflatten(tensor, length):
            # return object like parameter groups
            ...

        history_x_state = x.new_zeros(len(times) - 1, *x.shape)
        rtn_logabsdet = x.new_zeros(x.shape[0])
        delta_t = times[1:] - times[:-1]
        new_x = x
        with torch.no_grad():
            for i_th, cur_delta_t in enumerate(delta_t):
                history_x_state[i_th] = new_x
                new_x, new_logabsdet = model.forward_step(
                    new_x,
                    cur_delta_t,
                    condition[i_th],
                    condition[i_th + 1],
                    diffusion[i_th],
                    diffusion[i_th + 1],
                )
                rtn_logabsdet += new_logabsdet
        ctx.model = model
        ctx._flatten = _flatten
        ctx._unflatten = _unflatten
        ctx.nparam = np.sum([shape.numel() for shape in shapes])
        ctx.save_for_backward(
            history_x_state.clone(), new_x.clone(), times,
            diffusion, condition
        )
        return new_x, rtn_logabsdet

    @staticmethod
    @amp.custom_bwd
    def backward(ctx, dL_dz, dL_logabsdet):
        history_x_state, z, times, diffusion, condition = ctx.
                                                saved_tensors
        dL_dparameter = dL_dz.new_zeros((1, ctx.nparam))

        model, _flatten, _unflatten = ctx.model, ctx._flatten, ctx.
                                                _unflatten
        m_params = tuple(model.parameters())
        delta_t = times[1:] - times[:-1]
        with torch.no_grad():
            for bi_th, cur_delta_t in enumerate( \
                torch.flip(delta_t, (0,))):
                bi_th += 1
                with torch.set_grad_enabled(True):
                    x = history_x_state[-bi_th].requires_grad_(True)
                    z = z.requires_grad_(True)
                    noise_b = model.cal_backnoise(
                        x, z, cur_delta_t,
```

```
                    condition[-bi_th], diffusion[-bi_th]
                )

                cur_delta_s = -0.5 * (
                    torch.sum(noise_b.flatten(start_dim=1) ** 2,
                        dim=1)
                )
                dl_dprev_state, dl_dnext_state, *dl_model_b =
                    torch.autograd.grad(
                        (cur_delta_s),
                        (x, z) + m_params,
                        grad_outputs=(dL_logabsdet),
                        allow_unused=True,
                        retain_graph=True,
                    )
                dl_dx, *dl_model_f = torch.autograd.grad(
                    (
                        model.cal_next_nodiffusion(
                            x, cur_delta_t, condition[-bi_th - 1]
                        )
                    ),
                    (x,) + m_params,
                    grad_outputs=(dl_dnext_state + dL_dz),
                    allow_unused=True,
                    retain_graph=True,
                )
                del x, z, dl_dnext_state
            z = history_x_state[-bi_th]
            dL_dz = dl_dx + dl_dprev_state
            dL_dparameter += _flatten(dl_model_b).unsqueeze(0)
                        + _flatten(dl_model_f).unsqueeze(0)

    return (dL_dz, None, None, None, None,
            *_unflatten(dL_dparameter, (1,)))
```

E 2-D sythetic examples

E.1 Time discretization for DDPM

DDPM uses a fixed noising schema, $p(\mathbf{x}_i|\mathbf{x}_{i-1}) = \mathcal{N}(\mathbf{x}_i; \sqrt{1-\beta_i}\mathbf{x}_{i-1}, \beta_i \mathbf{I})$. The discretization can be reformulated as

$$\mathbf{x}_i = \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\mathbf{w}_{i-1}.$$

Since β_i is very small in DDPM implementation, the corresponding SDE is

$$d\mathbf{x} = -\frac{1}{2}\mathbf{x}dt + d\mathbf{w}, \tag{25}$$

and DDPM follows discretization $\Delta t_i = \beta_i$. Clearly, DDPM has a fixed forward diffusion $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\mathbf{x}$ with no learnable parameters. We can compare trajectory evolution of DDPM with that of DiffFlow in the following examples.

E.2 Trajectories on 2D points

We use similar architectures for DiffFlow and DDPM. To capture the dependence of the drift as well as the score networks on time, we use the Fourier feature embeddings with learned frequencies. Points positions embeddings and the embedded time signals are added before standard MLP layers. In various points datasets, we use 3 layers MLP with 128 hidden neurons for DiffFlow. We increase the width for DDPM so that the two models have comparable sizes. We adopt the official code for evaluating FFJORD. To further illustrate the backward process of all those three models, we include the comparison in Figure 8. We use 30 steps sampling for DiffFlow and 500 steps for DDPM. It can be observed from Figure 8 that DiffFlow diffuses data efficiently and meanwhile keeps the details of the distributions better.

E.3 NLL comparison on 2D points and Sharp Datasets

The negative log-likelihood results on 2-D data are reported in Table 6. We use `torchdiffeq`[3] for integral over time. We use `atol`$= 10^{-5}$ and `rtol`$= 10^{-5}$ for FFJORD and DiffFlow (ODE). Clearly, DiffFlow has

[3] `https://github.com/rtqichen/torchdiffeq`

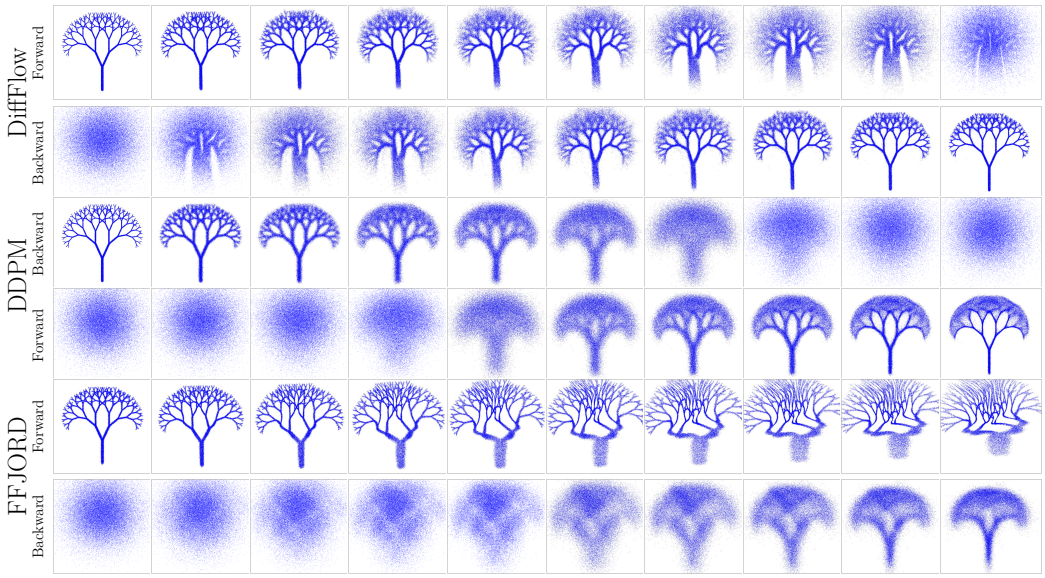


Figure 8: We compare the forward and backward sequences of DiffFlow, DDPM and FFJORD. Due to the unsatisfying forward process, the backward sequence and forward sequence for FFJORD are mismatched. The backward process of DDPM misses the details presented in the forward process. In contrast, DiffFlow has its backward trajectory distribution in alignment with the forward process.

.

much better performance on all these examples. The advantage of DiffFlow is even more obvious when the distribution concentrates on a 1d submanifold. Indeed, in the absence of noise as in FFJORD, it is easier to transform a smooth distribution than a sharp one to a Gaussian distribution. This phenomenon is better illustrated by comparing performance between Sharp Olympics and Olympics and their trajectories in Figure 9.

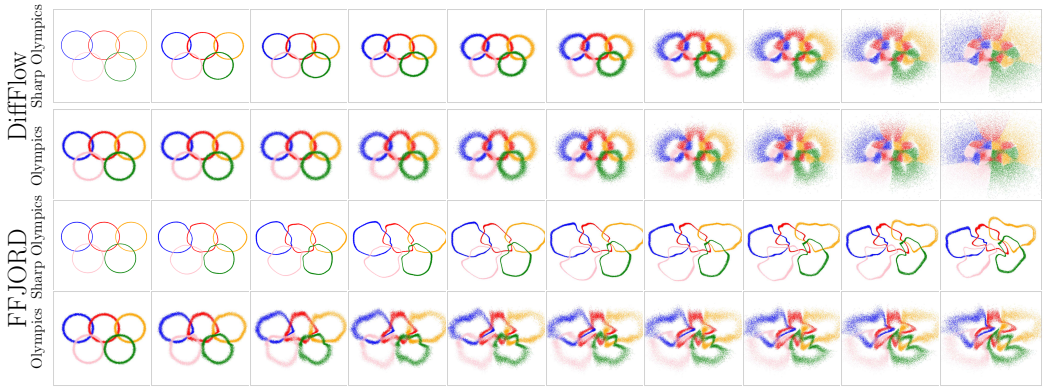


Figure 9: Forward sequences of DiffFlow and FFJORD on Sharp Olympics and Olympics. For FFJORD, it is more challenging to diffuse Sharp Olympics than Olympics. Thanks to the added noise, DiffFlow is more powerful in transforming data into Gaussian.

.

F Density estimation

We use similar network architectures with the following modifications. Since the ReZero [17] network shows better convergence property and often leads to lower negative log-likelihood, we use it instead of the MLP with the cost of additional number of layers parameters for ReZero. We find $N = 30$ works well for training on tabular datasets. We normalize the data with zero mean and standard deviation before feeding them into networks. We search over terminal time $T = 0.1, 0.25, 0.5, 0.75, 1.0$. We also try different widths and depths of the network. We stop training once the loss plateaus for five epochs. The architecture choices for different datasets can be found in Table 7. We rerun the density estimation with different random seeds and report the

Table 6: Negative log-likelihood on 2-d synthetic datasets.

Dataset	RealNVP	FFJORD	CIF-ResFlow	DiffFlow (ODE)
Sharp Olympics	2.12	1.52	1.32	**-0.63**
Olympics	2.19	1.64	1.59	**1.24**
Checkerboard	2.35	2.05	1.95	**1.82**
Fractal Tree	2.43	2.16	2.17	**1.02**
Carpet	2.47	2.37	3.32	**2.08**
2 Spirals	2.04	1.84	1.83	**1.73**

Dataset	layers	hidden dims	T	batchsize
POWER	3	128	1.0	20000
GAS	4	128	0.1	20000
HEPMASS	4	128	0.5	20000
MINIBOONE	3	128	0.75	2500
BSDS300	3	256	0.1	20000

Table 7: DiffFlow architectures for density estimation on tabular data.

standard deviation over 3 runs in Table 8. DiffFlow is trained on one RTX 2080 Ti and the training takes between 14 minutes and 2 hours depending on the datasets and the model size.

G Image generation

G.1 Settings

We rescale the images into $[-1, 1]$ for all experiments on images. We adopt U-net style model as used successfully in DDPM for both the drift and the score networks and they are trained with training data. We use the Adam optimizer with a learning rate 2×10^{-4} for all three datasets. On MNIST, we use batchsize 128 and train 6k iterations. It takes 20 hours on one RTX 3090Ti. On CelebA, we train with $N = 10$ for 10k iterations, $N = 30$ for next 10k iterations and $N = 50$ for 10k iterations. It takes around 40 hours on 8 RTX 2080 Ti. On CIFAR10, we follow the similar scheduling over $N = \{10, 30, 50, 75\}$ for a total of 100k iterations. It takes around 4 days in 8 RTX 2080 Ti GPUs. We found that the exponential moving average (EMA) with a decay factor of 0.999 stabilizes the updating of the parameters of models. As it is reported in existing works [17, 41], we also found EMA dramatically improves the sampling quality. We save checkpoints for every 500 steps and report the best NLL and sampling quality among those checkpoints. We tested performance with three different random seeds and report mean in Table 2 and Table 3. The FIDs and NLL reported in this works are based on the selected checkpoints that achieve the best performance.

Among all hyperparameters, we find that the parameter T is very crucial for training and sampling quality. The length of time interval deals with the trade-off between diffusion and learning backward process from the reconstruction. Larger T injects more noise into the processes. On one hand, it helps diffuse data into Gaussian. On the other hand, larger T requires backward process remove more noise to recover the original data or generate high-quality samples. Figure 10 shows the impacts of different T values. We sweep the choice of $0.05, 0.1, 0.2, 0.5, 1.0$. $T = 0.05$ works best for CIFAR10 and $T = 0.1$ for CelebA.

Dataset	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
RealNVP [8]	-0.17± 0.01	-8.33± 0.14	18.71± 0.02	13.55± 0.49	-153.28± 1.78
FFJORD [14]	-0.46± 0.01	-8.59± 0.12	14.92± 0.08	10.43± 0.04	-157.40± 0.19
MADE [11]	3.08± 0.03	-3.56± 0.04	20.98± 0.02	15.59± 0.50	-148.85± 0.28
MAF [33]	-0.24± 0.01	-10.08± 0.02	17.70± 0.02	11.75± 0.44	-155.69± 0.28
TAN [31]	-0.48± 0.01	-11.19± 0.02	15.12± 0.02	11.01± 0.48	-157.03± 0.07
NAF [18]	-0.62± 0.01	-11.96± 0.33	15.09± 0.40	8.86± 0.15	-157.73± 0.04
DiffFlow (ODE)	-1.04± 0.01	-10.45± 0.15	15.04± 0.04	8.06± 0.13	-157.80± 0.17

Table 8: Average negative log-likelihood (in nats) on tabular datasets [33] for density estimation (lower is better).

G.2 Image Completion

We also perform image completion tasks using our methods. The task is to complete the full images given partial masked images. The task is similar to image inpainting [27] but we use the whole masked forward trajectory instead of masked image [41, 17]. For a given real image $\mathbf{x}$, we denote the trajectories $\{\mathbf{x}_i^F\}_{i=0}^N$ by running forward process started with $\mathbf{x}_0 = \mathbf{x}$. For a given mask Γ, $\hat{\mathbf{x}}_i^F = \mathbf{x}_i^F \cdot \Gamma$ is the masked image, with $\cdot$ denotes element wise multiplication. $\{\hat{\mathbf{x}}_i^F\}_{i=0}^N$ gives information for backward process to complete original images. Instead of simulating reverse-time SDE, we incorporate the signals of $\{\hat{\mathbf{x}}_i^F\}_{i=0}^N$ by

$$\bar{\mathbf{x}}_{i-1}^B = \hat{\mathbf{x}}_i^B - [\mathbf{f}_i(\hat{\mathbf{x}}_i^B) - g_i^2 \mathbf{s}_i(\hat{\mathbf{x}}_i^B)]\Delta t_i + g_i \delta_i^B \sqrt{\Delta t_i} \tag{26a}$$

$$\hat{\mathbf{x}}_{i-1}^B = \bar{\mathbf{x}}_{i-1}^B \cdot (1-\Gamma) + \hat{\mathbf{x}}_{i-1}^F. \tag{26b}$$

Compared with sampling through the backward process, the Equation (26) iteratively refine the masked regions based on the unmasked image regions. We present the image completion results in Figure 11 and Figure 12.

G.3 More samples

We present more generated image samples in Figure 13, Figure 14 and Figure 15.

H Marginal Equivalent SDEs

Below we provide a derivation for the marginal equivalent SDEs in Equation (16). The reverse time SDE in DiffFlow is

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\mathbf{s}(\mathbf{x}, t)]dt + g(t)d\mathbf{w}, \tag{27}$$

where the drift $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^d$, the diffusion coefficients $g : \mathbb{R} \to \mathbb{R}$, and $\mathbf{w}(t) \in \mathbb{R}^d$ is a standard Brownian motion. According to Fokker-Planck-Kolmogorov (FPK) Equation for reverse time,

$$\frac{\partial p_1(\mathbf{x}, t)}{\partial t} = -\sum_i \frac{\partial}{\partial x_i}\{[f_i(\mathbf{x}, t) - g^2(t)s_i(\mathbf{x}, t)]p_1(\mathbf{x}, t)\} - \frac{g^2(t)}{2}\sum_i \frac{\partial^2}{\partial x_i^2}[p_1(\mathbf{x}, t)], \tag{28}$$

with $p_1(\mathbf{x}, t)$ being the marginal distribution induced by (27).

Now consider the SDE

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - \frac{1+\lambda^2}{2} g^2(t)\mathbf{s}(x, t)]dt + \lambda g(t)d\mathbf{w} \tag{29}$$

with $\lambda \geq 0$. To simplify notation, we denote $\hat{\mathbf{f}}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}, t) - \frac{1+\lambda^2}{2} g^2(t)\mathbf{s}(x, t)$. According to Fokker-Planck-Kolmogorov (FPK) Equation, its marginal distribution evolves as

$$\frac{\partial p_2(\mathbf{x}, t)}{\partial t} = -\sum_i \frac{\partial}{\partial x_i}[\hat{f}_i(\mathbf{x}, t)p_2(\mathbf{x}, t)] - \frac{\lambda^2 g^2(t)}{2}\sum_i \frac{\partial^2}{\partial x_i^2}[p_2(\mathbf{x}, t)]. \tag{30}$$

Since

$$\sum_i \frac{\partial^2}{\partial x_i^2}[p(\mathbf{x}, t)] = \sum_i \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_i}[p(\mathbf{x}, t)] = \sum_i \frac{\partial}{\partial x_i}[p(\mathbf{x}, t)\frac{\partial \log p(\mathbf{x}, t)}{\partial x_i}],$$

in view of $\mathbf{s}(\mathbf{x}, t) = \nabla \log p_2(\mathbf{x}, t)$, we can rewrite Equation (30) as

$$\begin{aligned}
&-\sum_i \frac{\partial}{\partial x_i}[\hat{f}_i(\mathbf{x}, t)p_2(\mathbf{x}, t)] - \frac{\lambda^2 g^2(t)}{2}\sum_i \frac{\partial^2}{\partial x_i^2}[p_2(\mathbf{x}, t)] \\
=&-\sum_i \frac{\partial}{\partial x_i}[\hat{f}_i(\mathbf{x}, t)p_2(\mathbf{x}, t)] - \frac{(\lambda^2-1)g^2(t)}{2}\sum_i \frac{\partial}{\partial x_i}[p_2(\mathbf{x}, t)\frac{\partial \log p_2(\mathbf{x}, t)}{\partial x_i}] - \frac{g^2(t)}{2}\sum_i \frac{\partial^2}{\partial x_i^2}[p_2(\mathbf{x}, t)] \\
=&-\sum_i \frac{\partial}{\partial x_i}\{[\hat{f}_i(\mathbf{x}, t) + \frac{\lambda^2-1}{2}g^2\frac{\partial \log p_2(\mathbf{x}, t)}{\partial x_i}]p_2(\mathbf{x}, t)\} - \frac{g^2(t)}{2}\sum_i \frac{\partial^2}{\partial x_i^2}[p_2(\mathbf{x}, t)] \\
=&-\sum_i \frac{\partial}{\partial x_i}\{\left[f_i(\mathbf{x}, t) - g^2\frac{\partial \log p_2(\mathbf{x}, t)}{\partial x_i}\right]p_2(\mathbf{x}, t)\} - \frac{g^2(t)}{2}\sum_i \frac{\partial^2}{\partial x_i^2}[p_2(\mathbf{x}, t)].
\end{aligned}$$

Thus, given that $p_1(\mathbf{x}, T)$ and $p_2(\mathbf{x}, T)$ are identical, Equation (28) and Equation (30) are equivalent. Therefore, Equation (29) and Equation (27) share the same marginal distributions.

Figure 10: Illustration of impacts of time interval length T. First two rows show the trade-off between forward diffusion and backward reconstruction. The first row displays the $\mathbf{x}_N$ while the second row shows the reconstructed image from $\mathbf{x}_N$. We show generated samples from three models in the last row. With small T, it is easy to reconstruct origin data but difficult to diffuse the data. On the other hand, more noise makes it diffuse the data effectively but poses more challenges for reconstructions and as well as sampling high quality samples.

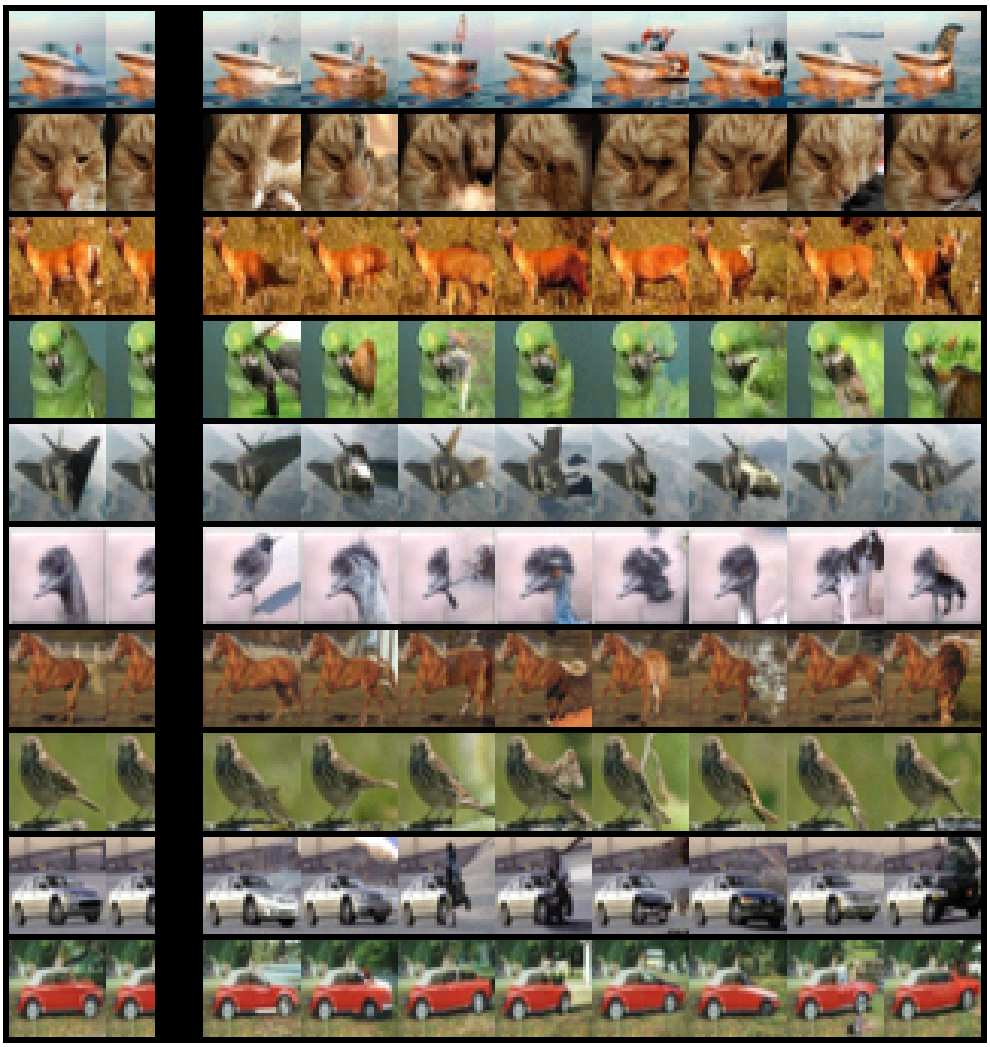


Figure 11: CIFAR10 image completion samples

Figure 12: CelebA image completion samples

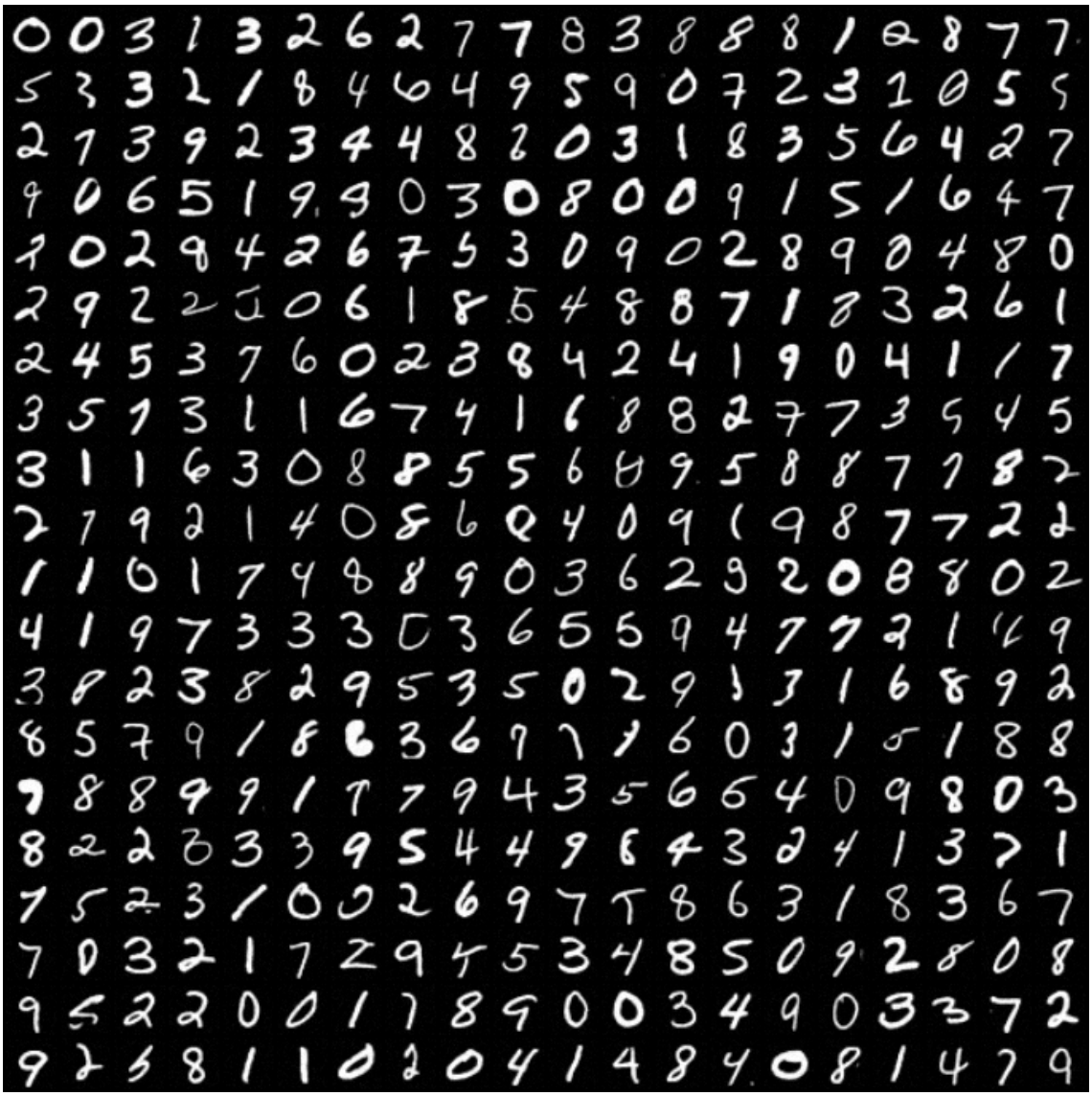


Figure 13: Uncurated MNIST Generated samples

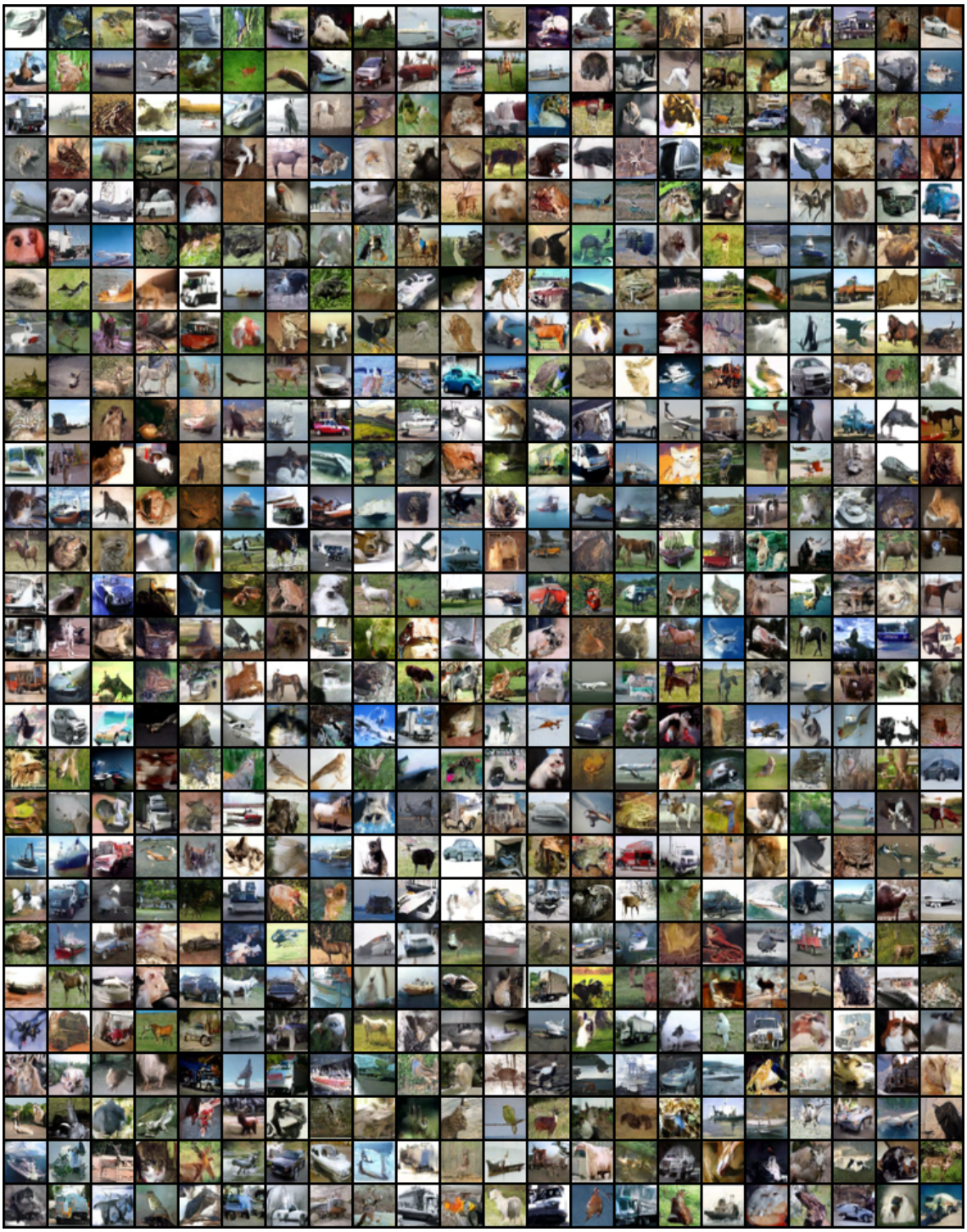


Figure 14: Uncurated CIFAR10 Generated samples

Figure 15: Uncurated CelebA Generated samples