INVERTIBLE NORMALIZING FLOW NEURAL NETWORKS BY JKO SCHEME

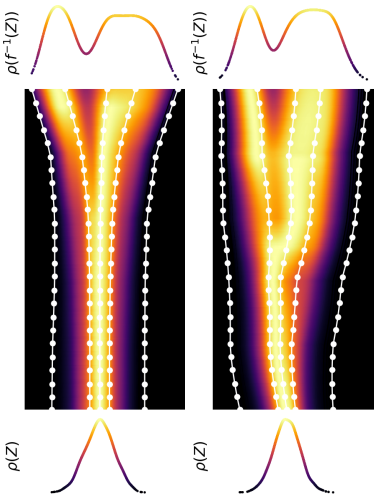
Anonymous authors
Paper under double-blind review

ABSTRACT

Normalizing flow is a class of deep generative models for efficient sampling and density estimation. In practice, the flow often appears as a chain of invertible neural network blocks. To facilitate training, past works have regularized flow trajectories and designed special network architectures. The current paper develops a neural ODE flow network inspired by the Jordan-Kinderleherer-Otto (JKO) scheme, which allows an efficient *block-wise* training procedure: as the JKO scheme unfolds the dynamic of gradient flow, the proposed model naturally stacks residual network blocks one-by-one and reduces the memory load as well as the difficulty of training deep networks. We also develop an adaptive time-reparametrization of the flow network with a progressive refinement of the trajectory in probability space, which improves the optimization efficiency and model accuracy in practice. On high-dimensional generative tasks for tabular data, JKO-Flow can process larger data batches and perform competitively as or better than continuous and discrete flow models, using 10X less number of iterations (e.g., batches) and significantly less time per iteration.

1 INTRODUCTION

Generative models have been widely studied in statistics and machine learning to infer data-generating distributions and sample from the estimated distributions (Ronquist et al., 2012; Goodfellow et al., 2014; Kingma & Welling, 2014). The normalizing flow has recently been a very popular generative framework. In short, a flow-based model learns the data distribution via an invertible mapping F between data density $\rho(X), X \in \mathbb{R}^d$ and the target standard multivariate Gaussian density $\rho(Z), Z \sim N(0, I_D)$ (Kobyzev et al., 2020). Benefits of the approach include efficient sampling and explicit likelihood computation. To make flow models practically useful, past works have made great efforts to develop flow models that facilitate training (e.g., in terms of loss objectives and computational techniques) and induce smooth trajectories (Dinh et al., 2017; Grathwohl et al., 2019; Onken et al., 2021).



(a) JKO-Flow (b) Generic Flow

Figure 1: Comparison of JKO-Flow (proposed) and other flow models. The JKO scheme approximates the transport of a diffusion process and the ResNet is trained block-wise.

Among flow models, continuous normalizing flow (CNF) transports the data density to that of the target through continuous dynamics (e.g, Neural ODE (Chen et al., 2018)). CNF models have shown promising performance on generative tasks Kobyzev et al. (2020). However, a known computational challenge of CNF models is model regularization, primarily due to the non-uniqueness of the flow transport. To regularize the flow model and guarantee invertibility, Behrmann et al. (2019) adopted spectral normalization of block weights that leads to additional computation. Finlay et al. (2020); Onken et al. (2021) utilizes optimal-transport costs, and (Xu et al., 2022) proposed Wasserstein-2 regularization. Regularization is important to maintain invertibility for general-form flow models and improves performance in practice, but merely using regularization does not resolve non-uniqueness of the flow and there remains variation in the trained flow depending on initialization.

Besides unresolved challenges in regularization, there remain several practical difficulties when training such models (1) *Training deep flow models:* in many settings, flows consist of stacked blocks, each of which can be arbitrarily complex. Training such deep models often places high demand on computational resources, numerical accuracy, and memory consumption. In addition, determining the flow depth (e.g., number of blocks) is also unclear. (2) *Reduce computation:* if more flow blocks than necessary are chosen, how to identify and remove them without affecting overall generative performance remains under-explored. Meanwhile, there lack principled ways to enhance the trajectory obtained by the rest blocks.

In this work, we propose JKO-Flow, a normalizing flow network which unfolds the Wasserstein gradient flow via a neural ODE invertible network, inspired by the JKO-scheme Jordan et al. (1998). The JKO scheme, cf. (5), can be viewed as a proximal step to unfold the Wasserstein gradient flow to minimize the KL divergence (relative entropy) between the current density and the equilibrium. Each block in the flow model implements one step in the JKO-scheme and can be trained by itself given the previous blocks. As the JKO scheme pushes forwards the density to approximate the solution of Fokker-Planck equation of a diffusion process with small step-size, the trained flow model induces a smooth trajectory of density evolution, as shown in Figure 1. Th theoretical assumption does not incur a restriction in practice when training, whereby one can use larger step sizes coupled with numerical integration techniques. The proposed JKO-Flow model can be viewed as trained to learn the unique transport map following the Fokker-Planck equation.

Unlike most CNF models where all the ResNet blocks are initialized together and trained end-to-end, the proposed model allows a block-wise training which reduces memory and computational load. We further introduce time reparametrization with progressive refinement in computing the flow network, where each block corresponds to a point on the density evolution trajectory in the space of probability. Algorithmically, one can thus determine the number of blocks adaptively and refine the trajectory determined by existing blocks. Empirically, such procedures yield competitive performance as other CNF models with significantly less computation.

The JKO Flow approach proposed in this work also suggests a potential constructive approximation analysis of deep flow model. Method-wise, the proposed model differs from other recent JKO deep models. We refer to Section 1.2 for more details. In summary, the contribution includes

- We propose a neural ODE model where each ResNet block computes a JKO step and the training objective can be computed from integrating the ODE on data samples. The network has general form and invertibility can be satisfied due to the regularity of the optimal pushforward map that minimizes the objective in each JKO step.

- We develop an block-wise procedure to train the invertible JKO-flow network, which determines the number of blocks adaptively. We also propose an enhancement technique to reparametrize and refine an existing JKO-Flow probability trajectory. Doing so removes unnecessary blocks and increases the overall accuracy.

- Experiment wise, JKO-Flow greatly reduces memory consumption and the amount of computation, with competitive/better performance as several existing continuous and discrete flow models.

1.1 Preliminaries

Normalizing flow. A normalizing flow can be mathematically expressed via a density evolution equation of $\rho(x,t)$ such that $\rho(x,0) = p_X$ and as t increases $\rho(x,t)$ approaches $p_Z \sim \mathcal{N}(0, I_d)$ Tabak & Vanden-Eijnden (2010). Given an initial distribtuion $\rho(x,0)$, such a flow typically is not unique. We consider when the flow is induced by an ODE of $x(t)$ in $\mathbb{R}^d$

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

where $x(0) \sim p_X$. The marginal density of $x(t)$ is denoted as $p(x,t)$, and it evolves according to the continuity equation (Liouville equation) of (1) written as

$$\partial_t p + \nabla \cdot (p\mathbf{f}) = 0, \quad p(x,0) = p_X(x). \tag{2}$$

Ornstein–Uhlenbeck (OU) process. Consider a Langevin dynamic denoted by the SDE $dX_t = -\nabla V(X_t)dt + \sqrt{2}dW_t$, where V is the potential of the equilibrium density. We focus on the case of normal equilibrium, that is, $V(x) = |x|^2/2$ and then $p_Z \propto e^{-V}$. In this case the process is known as

the (multivariate) OU process. Suppose $X_0 \sim p_X$, and let the density of X_t be $\rho(x,t)$ also denoted as $\rho_t(\cdot)$. The Fokker-Planck equation describes the evolution of ρ_t towards the equilibrium p_Z as

$$\partial_t \rho = \nabla \cdot (\rho \nabla V + \nabla \rho), \quad V(x) := |x|^2/2, \quad \rho(x,0) = p_X(x). \tag{3}$$

Under generic conditions, ρ_t converges to p_Z exponentially fast. For Wasserstein-2 distance and the standard normal p_Z, classical argument gives that (take $C = 1$ in Eqn (6) of Bolley et al. (2012))

$$W_2(\rho_t, p_Z) \leq e^{-t} W_2(\rho_0, p_Z), \quad t > 0. \tag{4}$$

JKO scheme. The seminal work Jordan et al. (1998) established a time discretization scheme of the solution to (3) by the gradient flow to minimize $\mathrm{KL}(\rho||p_Z)$ under the Wasserstein-2 metric in probability space. Denote by $\mathcal{P}$ the space of all probability densities on $\mathbb{R}^d$ with finite second moment. The JKO scheme at k-th step with step size $h > 0$, starting from $\rho^{(0)} = \rho_0 \in \mathcal{P}$, is written as

$$\rho^{(k+1)} = \arg\min_{\rho \in \mathcal{P}} F[\rho] + \frac{1}{2h} W_2^2(\rho^{(k)}, \rho), \quad F[\rho] := \mathrm{KL}(\rho||p_Z). \tag{5}$$

It was proved in Jordan et al. (1998) that as $h \to 0$, $\rho^{(k)}$ converges to the solution $\rho(\cdot, kh)$ of (3) for all k, and the convergence $\rho_h(\cdot,t) \to \rho(\cdot,t)$ is strongly in $L^1(\mathbb{R}^d, (0,T))$ for finite T, where ρ_h is piece-wise constant interpolated on $(0,T)$ from $\rho^{(k)}$.

1.2 Related works

For deep generative models, popular approaches include generative adversarial networks (GAN) (Goodfellow et al., 2014; Gulrajani et al., 2017; Isola et al., 2017) and variational auto-encoder (VAE)(Kingma & Welling, 2014; 2019). Apart from known training difficulties (e.g., mode collapse (Salimans et al., 2016) and posterior collapse (Lucas et al., 2019)), these models do not provide likelihood or inference of data density. The normalizing flow framework (Kobyzev et al., 2020) has been extensively developed, including continuous flow (Grathwohl et al., 2019), discrete flow (Chen et al., 2019), graph flow (Liu et al., 2019), etc. Efforts have been made to develop novel invertible mapping structures (Dinh et al., 2017; Papamakarios et al., 2017), regularize the flow trajectories (Finlay et al., 2020; Onken et al., 2021), and extend the use to non-Euclidean data (Mathieu & Nickel, 2020; Xu et al., 2022). Despite such efforts, the model and computational challenges of normalizing flow models include regularization and the large model size when using a large number of ResNet blocks, which cannot be determined a priori, and the associated memory and computational load.

In parallel to continuous normalizing flow which are neural ODE models, neural SDE models become an emerging tool for generative tasks. Diffusion process and Langevin dynamics in deep generative models have been studied in score-based generative models (Song & Ermon, 2019; Ho et al., 2020; Block et al., 2020; Song et al., 2021) under a different setting. Specifically, these models estimate the score function (i.e., gradient of the log probability density with respect to data) of data distribution via neural network parametrization, which may encounter challenges in learning and sampling of high dimensional data and call for special techniques (Song & Ermon, 2019). The recent work of Song et al. (2021) developed reverse-time SDE sampling for score-based generative models, and adopted the connection to neural ODE to compute the likelihood; using the same idea of backward SDE, Zhang & Chen (2021) proposed joint training of forward and backward neural SDEs. Theoretically, latent diffusion Tzen & Raginsky (2019b;a) was used to analyze neural SDE models. The current work focuses on neural ODE model where the deterministic vector field $\mathbf{f}(x,t)$ is to be learned following a JKO scheme of the Fokker-Planck equation. Rather than neural SDE, our approach involves no sampling of SDE trajectories nor learning of the score function.

JKO-inspired deep models have been studied in several recent works. Bunne et al. (2022) reformulates the JKO step for minimizing an arbitrary energy function as a problem solved over convex functions. However, adopting the framework for neural ODE under arbitrary block design can be challenging, especially when the method relies on input convex neural networks (Amos et al., 2017), which is a special type of network architecture. There have also been other models that use invertible convex neural networks to discretize the computation for Wasserstein gradient flow (Alvarez-Melis et al., 2021; Mokrov et al., 2021), which are prone to the same limitation due to model architecture. Therefore, as far as the authors know, none of the existing works has developed a block-wise training framework for neural ODE flow models based on this scheme. Neither have they proposed the enhancement techniques to be introduced in this work.

For the expressiveness of generating deep models, universal approximation properties of deep neural networks for representing probability distributions have been developed in several works. Lee et al. (2017) established approximation by composition of Barron functions (Barron, 1993); Bailey & Telgarsky (2018) developed space-filling approach, which was generalized in Perekrestenko et al. (2020; 2021); Lu & Lu (2020) constructed a deep ReLU network with guaranteed approximation under integral probability metrics, using techniques of empirical measures and optimal transport. These results show that deep neural networks can provably transport one source distribution to a target one with sufficient model capacity under certain regularity conditions of the pair of densities. In our proposed normalizing flow model, each ResNet block is trained to approximate the vector field $\mathbf{f}(x,t)$ that induces the Fokker-Planck equation, cf. (12) and Section 2.2. Our model potentially leads to a constructive approximation analysis of neural ODE flow model to generate data density p_X.

2 JKO SCHEME BY NEURAL ODE

Given i.i.d. observed data samples $X_i \in \mathbb{R}^d$, $i = 1, \ldots, N$, drawn from some unknown density p_X, the goal is to train an invertible neural network to transports the density p_X to an *a priori* specified density p_Z in $\mathbb{R}^d$, where each data sample X_i is mapped to a code Z_i. A prototypical choice of p_Z is the standard multivariate Gaussian $\mathcal{N}(0, I_d)$. By a slight abuse of notation, we denote by p_X and p_Z both the distributions and the density functions of data X and code Z respectively.

2.1 THE OBJECTIVE OF JKO STEP

We are to specify $\mathbf{f}(x,t)$ in the ODE (1), to be parametrized and learned by a neural ODE, such that the induced density evolution of $p(x,t)$ converges to p_Z as t increases. We start by dividing the time horizon $[0,T]$ into finite subintervals with step size h, let $t_k = kh$ and $I_{k+1} := [t_k, t_{k+1})$. Define $p_k(x) := p(x, kh)$, namely the density of $x(t)$ at $t = kh$.

The solution of (1) determined by the vector-field $\mathbf{f}(x,t)$ on $t \in I_{k+1}$ (assuming the ODE is well-posed) gives a one-to-one mapping T_{k+1} on $\mathbb{R}^d$, s.t. $T_{k+1}(x(t_k)) = x(t_{k+1})$ and T_{k+1} transports p_k into p_{k+1}, i.e., $(T_k)_{\#}p_{k-1} = p_k$, where we denote by $T_{\#}p$ the push-forward of distribution p by T, such that $(T_{\#}p)(\cdot) = p(T^{-1}(\cdot))$.

Suppose we can find $\mathbf{f}(\cdot,t)$ on I_{k+1} such that the corresponding T_{k+1} solves the JKO scheme (5), then with small h, p_k approximates the solution to the Fokker-Planck equation 3, which then flows towards p_Z. By the Monge formulation of the Wasserstein-2 distance between p and q as $W_2^2(p,q) = \min_{T:T_{\#}p=q} \mathbb{E}_{x\sim p}\|x - T(x)\|^2$, solving for the transported density p_k by (5) is equivalent to solving for the transport T_{k+1} by

$$T_{k+1} = \arg\min_{T:\mathbb{R}^d\to\mathbb{R}^d} F[T] + \frac{1}{2h}\mathbb{E}_{x\sim p_k}\|x - T(x)\|^2, \quad F[T] = \mathrm{KL}(T_{\#}p_k||p_Z). \tag{6}$$

The equivalence between (5) and (6) is proved in Lemma A.1. Furthermore, the following lemma gives that the value of $F[T]$ can be computed from $\mathbf{f}(x,t)$ on $t \in I_{k+1}$ only once p_k is determined by $\mathbf{f}(x,t)$ for $t \le t_k$, the proof is left to Appendix A.

Proposition 2.1. *Given p_k, up to a constant c independent from $\mathbf{f}(x,t)$ on $t \in I_{k+1}$,*

$$\mathrm{KL}(T_{\#}p_k||p_Z) = \mathbb{E}_{x(t_k)\sim p_k}\left(V(x(t_{k+1})) - \int_{t_k}^{t_{k+1}} \nabla\cdot\mathbf{f}(x(s),s)ds\right) + c. \tag{7}$$

By Proposition 2.1, the minimization (6) is equivalent to

$$\min_{\{\mathbf{f}(x,t)\}_{t\in I_{k+1}}} \mathbb{E}_{x(t_k)\sim p_k}\left(V(x(t_{k+1})) - \int_{t_k}^{t_{k+1}} \nabla\cdot\mathbf{f}(x(s),s)ds + \frac{1}{2h}\|x(t_{k+1}) - x(t_k)\|^2\right), \tag{8}$$

where $x(t_{k+1}) = x(t_k) + \int_{t_k}^{t_{k+1}} \mathbf{f}(x(s),s)ds$. Taking a neural ODE approach, we parametrize $\{\mathbf{f}(x,t)\}_{t\in I_{k+1}}$ as a ResNet block with parameter θ_{k+1}, and then (8) is reduced to minimizing over θ_{k+1}. This leads to block-wise learning algorithm to be introduced in Section 3.

2.2 INFINITESIMAL OPTIMAL $\mathbf{f}(x,t)$

In each JKO step of (8), let $p = p_k$ denote the current density, $q = p_Z$ be the target equilibrium density. In this subsection, we consider general equilibrium distribution q with a differentiable potential V. To analyze the optimal pushforward mapping in the small h limit, we shift the time interval $[kh, (k+1)h]$ to be $[0, h]$ to simplify notation. Then (8) is reduced to

$$\min_{\{\mathbf{f}(x,t)\}_{t\in[0,h)}} \mathbb{E}_{x(0)\sim p}\left(V(x(h)) - \int_0^h \nabla\cdot\mathbf{f}(x(s),s)ds + \frac{1}{2h}\|x(h)-x(0)\|^2\right), \tag{9}$$

where $x(h) = x(0) + \int_0^h \mathbf{f}(x(s),s)ds$. In the limit of $h \to 0+$, formally, $x(h) - x(0) = h\mathbf{f}(x(0),0) + O(h^2)$, and suppose V of q is C^2, $V(x(h)) = V(x(0)) + h\nabla V(x(0))\cdot\mathbf{f}(x(0),0) + O(h^2)$. For any differentiable density ρ, the (Stein) score function is defined as $\mathbf{s}_\rho = \nabla\log\rho$, and we have $\nabla V = -\mathbf{s}_q$. Taking the formal expansion of orders of h, the objective in (9) is written as

$$\mathbb{E}_{x\sim p}\left(V(x) + h\left(-\mathbf{s}_q(x)\cdot\mathbf{f}(x,0) - \nabla\cdot\mathbf{f}(x,0) + \frac{1}{2}\|\mathbf{f}(x,0)\|^2\right) + O(h^2)\right). \tag{10}$$

Note that $\mathbb{E}_{x\sim p}V(x)$ is independent of $\mathbf{f}(x,t)$, and the $O(h)$ order term in (10) is over $\mathbf{f}(x,0)$ only, thus the minimization of the leading term is equivalent to

$$\min_{\mathbf{f}(\cdot)=\mathbf{f}(\cdot,0)} \mathbb{E}_{x\sim p}\left(-T_q\mathbf{f} + \frac{1}{2}\|\mathbf{f}\|^2\right), \quad T_q\mathbf{f} := \mathbf{s}_q\cdot\mathbf{f} + \nabla\cdot\mathbf{f}, \tag{11}$$

where T_q is known as the Stein operator (Stein, 1972). Let $L^2(p)$ be the L^2 space on $(\mathbb{R}^d, p(x)dx)$, and for vector field $\mathbf{v}$ on $\mathbb{R}^d$, $\mathbf{v} \in L^2(p)$ if $\int |\mathbf{v}(x)|^2 p(x)dx < \infty$. One can verify that, when both $\mathbf{s}_p$ and $\mathbf{s}_q$ are in $L^2(p)$, the minimizer of (11) is

$$\mathbf{f}^*(\cdot,0) = \mathbf{s}_q - \mathbf{s}_p.$$

This shows that the infinitesimal optimal $\mathbf{f}(x,t)$ equals the difference of the score functions of the equilibrium and the current density.

2.3 INVERTIBILITY OF FLOW MODEL AND EXPRESSIVENESS

At time t the current density of $x(t)$ is ρ_t, the analysis in Section 2.2 implies that the optimal vector field $\mathbf{f}(x,t)$ has the expression as

$$\mathbf{f}(x,t) = \mathbf{s}_q - \mathbf{s}_{\rho_t} = -\nabla V - \nabla\log\rho_t. \tag{12}$$

With this $\mathbf{f}(x,t)$, the Liouville equation (2) coincides with the Fokker-Planck equation (3). This is consistent with that JKO scheme with small h recovers the solution to the Fokker-Planck equation. Under proper regularity condition of the initial density ρ_0, the $\mathbf{f}(x,t)$ as in (12) is also regular over space and time. This leads to two consequences, in approximation and in learning: Approximation-wise, the regularity of $\mathbf{f}(x,t)$ allows to construct a k-th ResNet block in the flow network to approximate $\{\mathbf{f}(x,t)\}_{t\in I_k}$ when there is sufficient model capacity, by classical universal approximation theory of shallow networks (Barron, 1993; Yarotsky, 2017). For learning, when properly trained with sufficient data, the neural ODE vector field $\mathbf{f}(x,t;\theta_k)$ will learn to approximate (12) which can be viewed as inferring the score function of ρ_t.

Suppose the trained $\mathbf{f}(x,t;\theta_k)$ is close enough to (12), it will also has bounded Lipschitz constant. Then the ResNet block is invertible as long as the step size h is sufficiently small, e.g. less than $1/L$ where L is the Lipschitz bound of $\mathbf{f}(x,t;\theta_k)$. In practice, we typically use smaller h than needed merely by invertibility (allowed by model budget) so that the flow network can more closely track the Fokker-Planck equation of the diffusion process. The invertibility of the proposed model is numerically verified in experiments (see Table 1). As for model expressiveness, a block-wise approximation guarantee of $\mathbf{f}(x,t)$ as in (12) can lead to approximation of the Fokker-Planck flow (3), which pushes forward the density to be ϵ-close to normality in $T = \log(1/\epsilon)$ time, cf. (4). Reversing the time of the ODE then leads to an approximation of the initial density $\rho_0 = p_X$ by flowing backward in time from T to zero. The JKO-flow network proposed in this work suggests a constructive proof of the expressiveness of the invertible neural ODE model to generate any sufficiently regular density p_X. The analysis is postponed here.

3 TRAINING OF JKO-FLOW NET

3.1 BLOCK-WISE TRAINING

Note that the training of $(k+1)$-th block in (8) can be conducted once the previous k blocks are trained. Specifically, with finite training data $\{X_i = x_i(0)\}_{i=1}^n$, the expectation $\mathbb{E}_{x(t)\sim p_k}$ in (8) is replaced by the sample average over $\{x_i(kh)\}_{i=1}^n$ which can be computed from the previous k blocks. Note that for each given $x(t) = x(t_k)$, both $x(t_{k+1})$ and the integral of $\nabla \cdot \mathbf{f}$ in (8) can be computed by any numerical ODE integrator, where one can use the Hutchinson trace estimator (Hutchinson, 1989; Grathwohl et al., 2019) to estimate the quantity $\nabla \cdot \mathbf{f}$ in high dimensions.

This leads to a block-wise training of the normalizing flow network, as summarized in Algorithm 1. In particular, it significantly reduces memory and computational complexity: only one block is trained when optimizing (8), regardless of flow depth. Therefore, one can use larger data batches and more refined numerical integrator without memory explosion. In practice, we found the generative performance JKO-Flow may vary depending on starting choices of t_k, but a simple choice such as $t_k = k$ often yields reasonably good performance. Meanwhile, one can optimize (8) per block for a fixed number of epochs using either back-propagation or the NeuralODE integrator (Grathwohl et al., 2019, adjoint method). We found direct back-propagation enables faster training but may also lead to greater numerical errors and memory consumption. Despite greater inaccuracies, we observed similar empirical performances across both methods for JKO-Flow, possibly due to the block-wise training that accumulates fewer errors than a generic flow model composed of multiple blocks. Meanwhile, one can use any suitable termination criterion (e.g., KL($T_\# p_k || p_Z$) and/or $W_2^2(T_\# p_k, p_k)$ in (6)), or simply pre-specify a fixed number of blocks to train.

Algorithm 1 Block-wise JKO-Flow training

Require: Time steps $\{t_k\}$, Training data $\{X_i = x_i(0)\}_{i=1}^n$, network architecture $\boldsymbol{f}$ for each block, termination criterion $T : \mathbb{Z} \to \mathbb{R}^+$ for each block, tolerance level ϵ

$\boldsymbol{f}_0 = I, k = 0$
while $T(k) > \epsilon$ **do**
  Set $k = k + 1$. Initialize $\boldsymbol{f} := \boldsymbol{f}_k \circ \ldots \circ \boldsymbol{f}_0$ using the k-th block.
  Optimize $\boldsymbol{f}_k$ upon minimizing (8), during which parameters for $\{\boldsymbol{f}_i\}_{i=0}^{k-1}$ are fixed.
end while

3.2 ENHANCED COMPUTATION OF TRAJECTORIES IN PROBABILITY SPACE

We adopt two additional computational techniques to facilitate learning of the trajectories in the probability space, represented by the sequence of densities p_k, $k = 1, \cdots, K$, associated with the K ResNet blocks of the proposed normalizing flow network. The two techniques are illustrated in Figure 2. Additional details can be found in Appendix B.

• *Trajectory Reparametrization*. We empirically observe fast decay of the movements $W_2^2(T_\# p_k, p_k)$; in other words, initial blocks transport the densities much further than the later ones. This is especially unwanted because in order to train the current block, the flow model needs to transport data through all previous blocks. If the current block barely transports the density, such training can be unnecessary. Hence, we develop a procedure to *remove* blocks with small W_2 movements. In addition, instead of having $t_k := kh$ with fixed increments per block, we *reparametrize* the values of t_k through an adaptive procedure, which is entirely based on the W_2 distance at each block and averaged over all blocks.

• *Progressive refinement*. After block removal and reparametrization, it likely appears that $t_{k+1} - t_k$ for the later remaining blocks is much larger than those of the initial blocks, which appears due to larger W_2 movement by the initial ones. Numerically, this can lead to greater inaccuracies as the loss (8) is computed over a longer time horizon. To address this, we introduce a *refinement* technique: when transporting p_k to p_{k+1} through the $k-$th block $\boldsymbol{f}_k$, we compute C "intermediate" densities $\{p_{k+c/(C+1)}\}_{c=1}^C$. In practice, this is similar to increasing the number of steps for the numerical integrator at the $k-$th block.

Table 1: Absolute inversion error $\mathbb{E}_x[\|\boldsymbol{f}^{-1}(\boldsymbol{f}(x)) - x\|_2]$ of JKO-Flow on test data in Section 4.

POWER	GAS	MINIBOONE	BSD300	Rose	Fractal tree	Olympic rings	Checkerboard
1.48e-05	1.58e-06	1.09e-06	1.53e-05	3.30e-06	3.58e-05	2.24e-06	3.07e-05

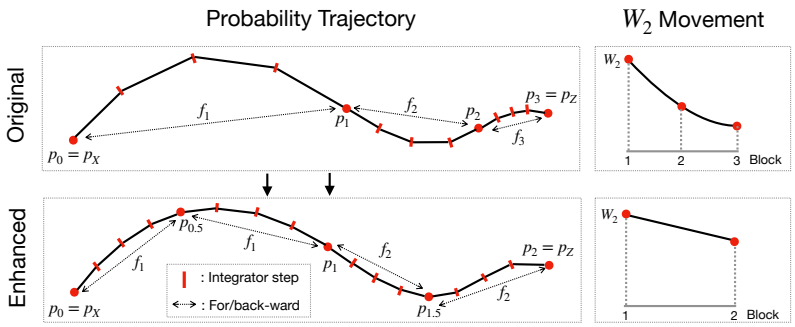
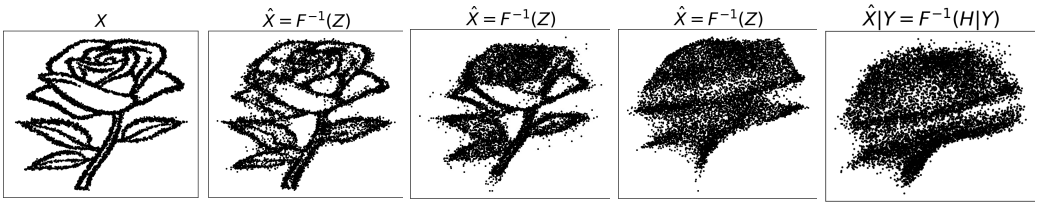
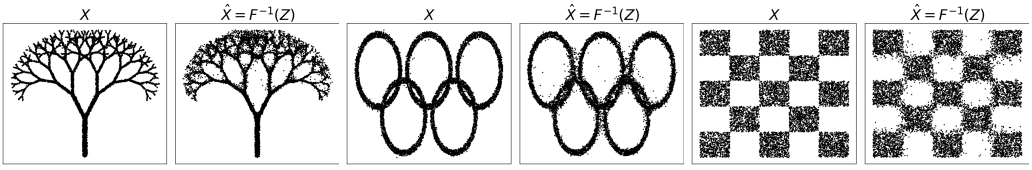


Figure 2: Diagram illustrating trajectory reparametrization and refinement. The original trajectory via Algorithm 1 can be less smooth and yield fast decaying W_2 movement, which causes unnecessary computation when training later blocks. The enhanced trajectory is smoother and can be obtained with fewer blocks under more integration steps. It also shows more uniform W_2 movement.



(a) True data JKO-Flow (b) FFJORD (c) OT-Flow (d) W_2 IResNet
τ: **2.06e-04**, MMD[m]: 3.70e-04 4.10e-04 7.25e-04 3.34e-03
τ: **2.96e-04**, MMD[c]: 9.94e-04 8.57e-04 1.32e-03 3.39e-03



(e) Fractal tree (f) Olympic rings (g) Checkerboard

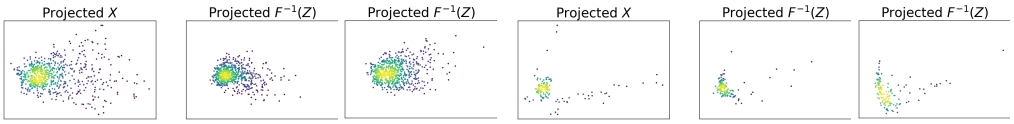
Figure 3: Two-dimensional simulated datasets. The generated samples $\hat{X}$ by JKO-Flow in Figure 3a are closer to the true data X than competitors in Figure 3b—3d. Under the more carefully selected bandwidth via sample median, MMD[m] in (20) by JKO-Flow is also closer to the threshold τ in (21) than others.

4 Experiment

We first generate based on two-dimensional simulated samples. We then perform unconditional and conditional generation on high-dimensional real tabular data. We also show JKO-Flow's generative performance on MNIST. Due to space limitation, we describe additional details in Appendix C.

4.1 Setup

Competing Methods We compare JKO-Flow with 4 other flow-based models. The first two continuous flow models are FFJORD (Grathwohl et al., 2019) and OT-Flow (Onken et al., 2021). The next two discrete flow models are IResNet (Behrmann et al., 2019) and IGNN (Xu et al., 2022), which replaces the expensive spectral normalization in IResNet with Wasserstein-2 regularization to promote smoothness. We are primarily interested in two types of criteria. The first is the *computational efficiency* in terms of the number of iterations (e.g., batches that the model uses in training) and training time per iteration. Due to the block-wise training scheme, we measure the number of iterations for JKO-Flow as the sum of iterations over all blocks. Using this metric allows us to examine performance easily across models, under a fixed-budget framework in terms of batches available to the model. The second is the *maximum mean discrepancy* (MMD) comparison (Gretton et al., 2012; Onken et al., 2021), which is a way of measuring the difference between two distributions based on samples. Additional details for MMD appear in Appendix C.



(a) Top1 Y: JKO-Flow
τ: **2.78e-03**, MMD[m]: 2.29e-02
τ: **3.90e-03**, MMD[c]: 3.65e-02

(b) IGNN
2.60e-02
3.11e-02

(c) Top2 Y: JKO-Flow
τ: **1.30e-02,** MMD[m]: 3.65e-02
τ: **1.69e-02,** MMD[c]: 7.50e-02

(d) IGNN
1.09e-01
1.53e-01

Figure 4: Conditional graph node feature generation. Both JKO-Flow and IGNN use residuals blocks with one Chebnet layer composed with two fully-connected layers. "Top j Y" indicates the distribution $X|Y$ with the j-th most number of observations.

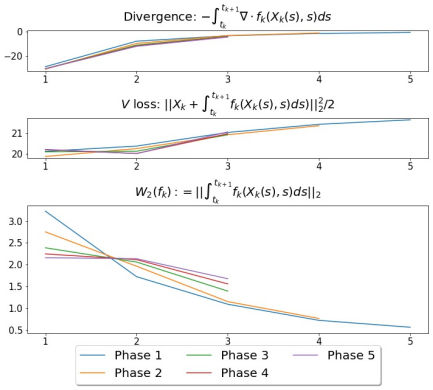
Table 2: Comparison on high-dimensional real datasets. All competitors are trained after 10 times more passes over the training data (roughly 10 times more iteration), because their performance under the same number of iterations (i.e., batches) as JKO-Flow is not comparable. FFJORD results on BSDS300 are omitted due to incomparably longer training time.

Data Set	Model	# Param	Training				Testing	
			Time (h)	# Iter	Time/Iter (s)	Batch size	MMD[m]	MMD[c]
POWER $d = 6$							τ: **1.75e-04**	τ: **2.84e-04**
	JKO-Flow	76K	0.12	0.76K	0.57	10000	8.21e-04	1.26e-03
	OT-Flow	76K	1.21	7.58K	0.57	10000	5.69e-04	9.62e-04
	FFJORD	76K	3.40	7.58K	1.61	10000	1.38e-03	1.98e-03
	W_2 IResNet	304K	0.49	7.58K	0.23	10000	2.76e-03	2.72e-03
	IResNet	304K	0.76	7.58K	0.36	10000	4.50e-03	2.49e-02
GAS $d = 8$							τ: **1.93e-04**	τ: **2.83e-04**
	JKO-Flow	76K	0.08	0.76K	0.38	5000	5.96e-04	1.79e-03
	OT-Flow	76K	0.72	7.60K	0.34	5000	1.51e-03	3.64e-03
	FFJORD	76K	3.49	7.60K	1.65	5000	3.62e-03	6.09e-03
	W_2 IResNet	304K	0.57	7.60K	0.27	5000	7.14e-03	1.50e-02
	IResNet	304K	0.86	7.60K	0.41	5000	3.26e-03	2.72e-02
MINIBOONE $d = 43$							τ: **4.59e-04**	τ: **6.87e-04**
	JKO-Flow	112K	0.03	0.34K	0.30	2000	7.97e-04	1.01e-03
	OT-Flow	112K	0.75	3.39K	0.80	2000	1.23e-03	1.01e-03
	FFJORD	112K	1.74	3.39K	1.85	2000	5.47e-03	1.04e-03
	W_2 IResNet	448K	0.80	3.25K	0.89	2000	1.27e-02	1.03e-03
	IResNet	448K	1.32	3.25K	1.46	2000	2.58e-03	1.04e-03
BSDS300 $d = 63$							τ: **1.35e-04**	τ: **9.63e-05**
	JKO-Flow	396K	0.16	0.88K	0.66	5000	4.83e-03	3.03e-03
	OT-Flow	396K	3.50	10.29K	1.22	1000	8.55e-02	8.44e-02
	W_2 IResNet	990K	2.01	10.29K	0.70	1000	5.52e-01	6.88e-01
	IResNet	990K	3.47	10.29K	1.21	1000	5.42e-01	5.95e-01

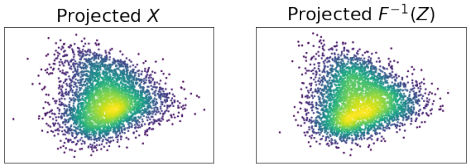
Conditional Generation Due to the increasing need for conditional generation, we also apply JKO-Flow for conditional generation: generate samples based on the conditional distribution $X|Y$. Most existing conditional generative methods treat Y as an additional input of the generator, which increases training difficulty due to dimensionality increase. Instead, we follow the IGNN approach (Xu et al., 2022), which also incurs minimal changes to our training. The only modification of the training objective (8) is that, instead of evaluating V for a standard multivariate Gaussian, we evaluate V for a Gaussian mixture $H|Y$.

4.2 Results

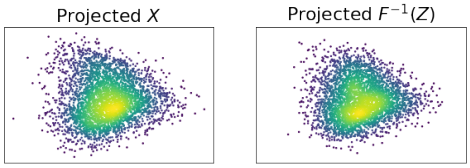
Two-dimensional toy data Figures 3a—3d compare JKO-Flow with the competitors on non-conditional generation, where the subcaption indicates MMD values (20) under both bandwidths and the corresponding thresholds τ in (21). We omit showing IResNet with spectral normalization as it yields similar results as W_2 IResNet. The generative quality by JKO-Flow is the closest to that of the ground truth, and when the MMD bandwidth is more carefully selected via the median trick, JKO-Flow also yields smaller MMD than others. Meanwhile, Figures 3e–3g the satisfactory generative performance by JKO-Flow on other examples. In Appendix, Figure A.2 shows additional unconditional and conditional generation and Figure A.3 shows even better generative results by JKO-Flow under the enhancement technique described in Section 3.2.



(a) Components of loss (8) in enhancement phases.



(b) Results at phase 1 (pre-enhancement).



(c) Results at phase 5 (after 4 enhancement).

Figure 5: Miniboone, enhancement of JKO-Flow over 5 phases. Here, phase 1 is the pre-enhancement phase during which we specify $t_k = k$ for training the residual blocks. Phase 5 results are obtained by using Algorithm 2 (modified for probability spaces) 4 times. We see more uniform W_2 losses (left column, third row) with fewer blocks after enhancement.

High-dimensional tabular data In terms of conditional graph node feature generation, Figure 4 compares JKO-Flow with IGNN on the solar dataset introduced in that work. The results show that JKO-Flow yields competitive or clearly better MMD values on the conditional distribution $X|Y$ with the most or second most observations, respectively. Next, Table 2 assesses the performance of JKO-Flow and competitors on four high-dimensional real datasets. Dataset details are described in Appendix C. To ensure a fair comparison, we keep the number of parameters for continuous flow models the same and properly increase model sizes for discrete flow models. We also use the same batch size during training; one exception is on BSDS300, where all except JKO-Flow cannot handle large batches during training due to GPU memory limit—JKO-Flow can do so as it trains one block at a time. When training generative models, we comment that using larger batches when other settings are fixed can yield better performance, as larger batches yields more information about the underlying distribution. In terms of results, we first remark that JKO-Flow takes much less iterations to reach a small enough MMD value. In fact, except on POWER, JKO-Flow yields smaller MMD under both bandwidths than all other methods, which are trained after 10 times more passes on the training data. In addition, the average time per iteration by JKO-Flow is much less sensitive to the increase in data dimension: on BSDS300, JKO-Flow takes 50% less average time than OT-Flow even when the former is trained on 5 times more samples per iteration. Furthermore, we illustrate the enhancement technique on MINIBOONE in Figure 5, where the benefit appears in reducing the number of blocks for a competitive generative performance. Lastly, we visually compare the generative quality upon projecting the generated samples onto the first two principal components defined by the test data. Results are shown in Appendix C, Figure A.4.

MNIST We illustrate the generative quality of JKO-Flow using an AutoEncoder. Consider a pre-trained encoder $E : \mathbb{R}^{784} \to \mathbb{R}^d$ and decoder $D : \mathbb{R}^d \to \mathbb{R}^{784}$ such that $D(E(X)) \approx x$ for a flattened image X. We choose $d = 16$. The encoder (resp. decoder) uses one fully-connected layer followed by the ReLU (resp. Sigmoid) activation. Then, JKO-Flow is trained on N encoded images $\{E(x)\}_{i=1}^N$. The new images are then generated upon sampling noises Z following a standard multivariate normal distribution in $\mathbb{R}^d$, flow backward through F^{-1} defined by the JKO-Flow, and mapped through the decoder $D(F^{-1}(Z))$.

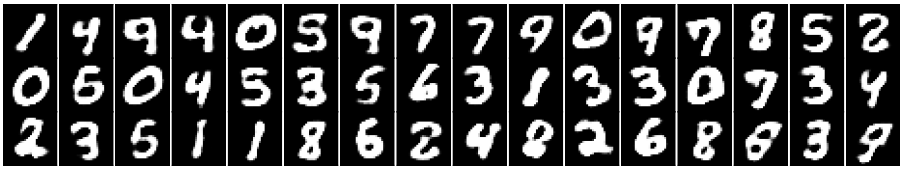


Figure 6: MNIST generation by JKO-Flow coupled with a pre-trained auto-encoder.

REFERENCES

David Alvarez-Melis, Yair Schiff, and Youssef Mroueh. Optimizing functionals on the space of probabilities with input convex neural networks. *arXiv preprint arXiv:2106.00774*, 2021.

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Bolton Bailey and Matus J Telgarsky. Size-noise tradeoffs in generative networks. *Advances in Neural Information Processing Systems*, 31, 2018.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pp. 573–582. PMLR, 2019.

Adam Block, Youssef Mroueh, and Alexander Rakhlin. Generative modeling with denoising auto-encoders and langevin sampling. *arXiv preprint arXiv:2002.00107*, 2020.

François Bolley, Ivan Gentil, and Arnaud Guillin. Convergence to equilibrium in wasserstein distance for fokker–planck equations. *Journal of Functional Analysis*, 263(8):2430–2457, 2012.

Charlotte Bunne, Laetitia Papaxanthos, Andreas Krause, and Marco Cuturi. Proximal optimal transport modeling of population dynamics. In *International Conference on Artificial Intelligence and Statistics*, pp. 6511–6528. PMLR, 2022.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. *Advances in Neural Information Processing Systems*, 32, 2019.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *ArXiv*, abs/1605.08803, 2017.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International conference on machine learning*, pp. 3154–3164. PMLR, 2020.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Kristjanson Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *ArXiv*, abs/1810.01367, 2019.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alex Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, 2012.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. In *NIPS*, 2017.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Michael F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 18:1059–1076, 1989.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5967–5976, 2017.

Richard Jordan, David Kinderlehrer, and Felix Otto. The variational formulation of the fokker–planck equation. *SIAM journal on mathematical analysis*, 29(1):1–17, 1998.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. doi: 10.1561/2200000056.

Ivan Kobyzev, Simon JD Prince, and Marcus A Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43 (11):3964–3979, 2020.

F Shampine Lawrence. Some practical runge-kutta formulas. *Mathematics of Computation*, 46: 135–150, 1986.

Holden Lee, Rong Ge, Tengyu Ma, Andrej Risteski, and Sanjeev Arora. On the ability of neural nets to express distributions. In *Conference on Learning Theory*, pp. 1271–1296. PMLR, 2017.

Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. *Advances in Neural Information Processing Systems*, 32, 2019.

Yulong Lu and Jianfeng Lu. A universal approximation theorem of deep neural networks for expressing probability distributions. *Advances in neural information processing systems*, 33: 3094–3105, 2020.

James Lucas, G. Tucker, Roger B. Grosse, and Mohammad Norouzi. Understanding posterior collapse in generative latent variable models. In *DGS@ICLR*, 2019.

Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows. *Advances in Neural Information Processing Systems*, 33:2503–2515, 2020.

Petr Mokrov, Alexander Korotin, Lingxiao Li, Aude Genevay, Justin M Solomon, and Evgeny Burnaev. Large-scale wasserstein gradient flows. *Advances in Neural Information Processing Systems*, 34:15243–15256, 2021.

Derek Onken, S Wu Fung, Xingjian Li, and Lars Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Dmytro Perekrestenko, Stephan Müller, and Helmut Bölcskei. Constructive universal high-dimensional distribution generation through deep relu networks. In *International Conference on Machine Learning*, pp. 7610–7619. PMLR, 2020.

Dmytro Perekrestenko, Léandre Eberhard, and Helmut Bölcskei. High-dimensional distribution generation through deep neural networks. *Partial Differential Equations and Applications*, 2(5): 1–44, 2021.

Fredrik Ronquist, Maxim Teslenko, Paul van der Mark, Daniel L. Ayres, Aaron E. Darling, Sebastian Höhna, Bret R. Larget, Liang Liu, Marc A. Suchard, and John P. Huelsenbeck. Mrbayes 3.2: Efficient bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology*, 61:539 – 542, 2012.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *ArXiv*, abs/1606.03498, 2016.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Charles Stein. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the sixth Berkeley symposium on mathematical statistics and probability, volume 2: Probability theory*, volume 6, pp. 583–603. University of California Press, 1972.

Esteban G Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019a.

Belinda Tzen and Maxim Raginsky. Theoretical guarantees for sampling and inference in generative models with latent diffusions. In *Conference on Learning Theory*, pp. 3084–3114. PMLR, 2019b.

Chen Xu, Xiuyuan Cheng, and Yao Xie. Invertible neural networks for graph prediction. *arXiv preprint arXiv:2206.01163*, 2022.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94: 103–114, 2017.

Qinsheng Zhang and Yongxin Chen. Diffusion normalizing flow. *Advances in Neural Information Processing Systems*, 34:16280–16291, 2021.

A PROOFS

A.1 PROOFS IN SECTION

Lemma A.1. *Suppose p and q are two densities on $\mathbb{R}^d$ in $\mathcal{P}$, the following two problems*

$$\min_{\rho\in\mathcal{P}} L_\rho[\rho] = \mathrm{KL}(\rho||q) + \frac{1}{2h}W_2^2(p,\rho), \tag{13}$$

$$\min_{T:\mathbb{R}^d\to\mathbb{R}^d} L_T[T] = \mathrm{KL}(T_\# p||q) + \frac{1}{2h}\mathbb{E}_{x\sim p}\|x - T(x)\|^2, \tag{14}$$

have the same minimum, and

(a) If $T^*:\mathbb{R}^d\to\mathbb{R}^d$ is a minimizer of (14)*, then $\rho^* = (T^*)_\# p$ is a minimizer of* (13)*.*

(b) If ρ^* is a minimizer of (13)*, then the optimal transport from p to ρ^* minimizes* (14)*.*

Proof of Lemma A.1. Let the minimum of (14) be L_T^*, and that of (13) be L_ρ^*.

Proof of (a): Suppose L_T achieves minimum at T^*, then T^* is the optimal transport from p to $\rho^* = (T^*)_\# p$ because otherwise L_T can be further improved. By definition of L_ρ, we have $L_T^* = L_T[T^*] = L_\rho[\rho^*] \geq L_\rho^*$. We claim that $L_T^* = L_\rho^*$. Otherwise, there is another ρ' such that $L_\rho[\rho'] < L_T^*$. Let T' be the optimal transport from p to ρ', and then $L_T[T'] = L_\rho[\rho'] < L_T^*$, contradicting with that L_T^* is the minimum of L_T. This also shows that $L_\rho[\rho^*] = L_T^* = L_\rho^*$, that is, ρ^* is a minimizer of L_ρ.

Proof of (b): Suppose L_ρ achieves minimum at ρ^*. Let T^* be the OT from p to ρ^*, then $\mathbb{E}_{x\sim p}|x - T^*(x)|^2 = W_2(p,\rho^*)^2$, and then $L_T[T^*] = L_\rho[\rho^*] = L_\rho^*$ which equals L_T^* as proved in (a). This shows that T^* is a minimizer of L_T. □

Proof of Proposition 2.1, Given p_k being the density of $x(t)$ at $t = kh$, recall that T is the solution map from $x(t)$ to $x(t+h)$. We denote $\rho_t := p_k$, and $\rho_{t+h} := T_\# p_k$. By definition,

$$\mathrm{KL}(T_\# p_k || p_Z) = \mathbb{E}_{x\sim\rho_{t+h}}(\log \rho_{t+h}(x) - \log p_Z(x)). \tag{15}$$

Because $p_Z \propto e^{-V}$, $V(x) = |x|^2/2$, we have $\log p_Z(x) = -V(x) + c_1$ for some constant c_1. Thus

$$\mathbb{E}_{x\sim\rho_{t+h}} \log p_Z(x) = \mathbb{E}_{x(t)\sim\rho_t} \log p_Z(x(t+h)) = c_1 - \mathbb{E}_{x(t)\sim\rho_t} V(x(t+h)). \tag{16}$$

To compute the first term in (15), note that

$$\mathbb{E}_{x\sim\rho_{t+h}} \log \rho_{t+h}(x) = \mathbb{E}_{x(t)\sim\rho_t} \log \rho_{t+h}(x(t+h)), \tag{17}$$

and by the expression

$$\partial_t \log \rho(x(t),t) = -\nabla\cdot \mathbf{f}(x(t),t),$$

we have that for each value of $x(t)$,

$$\log \rho_{t+h}(x(t+h)) = \log \rho(x(t+h), t+h) = \log \rho(x(t),t) - \int_t^{t+h} \nabla\cdot \mathbf{f}(x(s),s)ds.$$

Inserting back to (17), we have

$$\mathbb{E}_{x\sim\rho_{t+h}} \log \rho_{t+h}(x) = \mathbb{E}_{x(t)\sim\rho_t} \log \rho_t(x(t)) - \mathbb{E}_{x(t)\sim\rho_t} \int_t^{t+h} \nabla\cdot \mathbf{f}(x(s),s)ds).$$

The first term is determined by $\rho_t = p_k$, and thus is a constant c_2 independent from $\mathbf{f}(x,t)$ on $t \in [kh, (k+1)h]$. Together with (16), we have shown that

$$\text{r.h.s. of (15)} = c_2 - \mathbb{E}_{x(t)\sim\rho_t} \int_t^{t+h} \nabla\cdot \mathbf{f}(x(s),s)ds) - c_1 + \mathbb{E}_{x(t)\sim\rho_t} V(x(t+h)),$$

which proves (7). □

B TECHNICAL DETAILS OF SECTION 3.2

Although the layer-wise training formulation in Section 3.1 enjoys several aforementioned benefits, there exists undesirable movement patterns along the trajectory. Empirically, the movement by initial blocks $\boldsymbol{f}_k$ is much larger than later ones. The blue curve labeled "Phase 1" in Figure A.3a visualizes one typical pattern of the movement measured by W_2 distances.

Such unwanted behavior causes practical concerns for computation. Note that to train block $\boldsymbol{f}_k$, we must flow through pre-trained $\boldsymbol{f}_0, \ldots, \boldsymbol{f}_{k-1}$. As k increases, this computational efforts increase. However, if f_k barely contributes to the overall movement, such training is unnecessary.

In fact, this issue is not specific to training flow networks by the JKO scheme. It essentially arises due to smaller gradient magnitude at later estimates, which gradually approach a local minimum. In particular, such irregular movement also appears in gradient descent in vector space. We thus propose a "prune-reparametrize-refine" enhancement technique based on previous irregular movements.

1. Vector-space case

We first motivate our method with optimization in vector space; the probability trajectory enhancement adopts the same idea. Suppose our goal is to find a local minimum x^* of $F(x)$ for a nonlinear differentiable function $F : \mathbb{R}^d \to \mathbb{R}$. Starting at $x^{(0)}$, consider the following sequential optimization problem, where $x^{(t)}$ denotes the estimate at the t-th iteration and h_t is a pre-specified regularization parameter:

$$x^{(t+1)} = \arg\min_x F(x) + \frac{1}{2h_t}\|x - x^{(t)}\|_2^2. \tag{18}$$

Using the first order Taylor expansion $F(x) \approx F(x^{(t)}) + \nabla F(x^{(t)})^T(x - x^{(t)})$ at $x^{(t)}$, we get

$$x^{(t+1)} = x^{(t)} - h_t g_t,\ g_t := \nabla_x F(x^{(t)}).$$

Hence, h_t is also the learning rate. Define the *arc length* of iterates $S_t := \|x^{(t+1)} - x^{(t)}\|_2 = h_t \|g_t\|_2$, whereby it appears in practice that the magnitude of S_t is near zero as $x^{(t)} \to x^*$. This issue is typical as a result of small gradient as estimates approach the local minimum. We thus propose Algorithm 2 to resolve this uneven arc length issue, which takes in iterates $x^{(t,\text{old})}$ and step sizes h_t^{old} from the previous trajectory.

We first explain the pruning step in lines 4. Numerically, S_t^{old} is near zero for later iterates of $x^{(t,\text{old})}$. Thus, such small values leads to singularity in the update $1/S_t^{\text{old}}$, leading to undefined estimates $x^{(t,\text{new})}$. By computing $\tilde{S}_t^{\text{old}}$ in line 2, which measures the distance from iterate $x^{(t,\text{old})}$ to the local minimum x^*, we can prune all sufficiently small movements using the threshold δ in line 4. If the threshold is small, the re-optimized final estimate will still be close to the true minimum. In practice, we choose it to be $\bar{S}/2$ for adaptiveness.

We then motivate and explain the reparametrization step in line 7. Mathematically, we want arc lengths defined using re-optimized values $x^{(t,\text{new})}$ to satisfy $S_t^{\text{new}} \approx \bar{S}$. This is equivalent to requiring $h_t^{\text{new}} \|g_t^{\text{new}}\|_2 \approx \bar{S}$, where $g_t^{\text{new}} := \nabla F(x_t^{\text{new}})$. Note that the quantity $\|g_t^{\text{new}}\|_2$ is unknown at the before re-optimization takes place, so that we approximate it using $\|g_t^{\text{old}}\|_2 = S_t^{\text{old}}/x^{(t,\text{old})}$, which is exactly the case when $\eta = 1$. Heuristically, this approximation is reasonably close if the total number of iterates T is large enough and the old step size h_t^{old} is small enough. In practice, however, even if there is no singularity, using the quantity $\bar{S}h_t^{\text{old}}/S_t^{\text{old}}$ alone as h_t^{new} can be undesirable. This occurs as larger h_t^{new} tend to cause non-smooth trajectories and importantly, inaccurate final estimates. We thus introduce inertia controlled by parameter η to allow more flexibility.

We lastly explain the refinment step in line 8. We interpolate $C \geq 0$ intermediate points between each pair of $(x^{(t,\text{new})}, x^{(t+1,\text{new})})$. For instance, if $C = 1$, we optimize for the "mid-point" $x^{(t+1/2,\text{new})}$ with $h_t^{\text{new}}/2$ in (18) before reaching $x^{(t+1,\text{new})}$. Using this approach ensures smoother new trajectories $\{x^{(t,\text{new})}\}_{t\geq 1}$ and an accurate final estimate. Figure A.1 illustrates the behavior and our solution on minimizing the Muller-Brown energy potential in $\mathbb{R}^2$.

Algorithm 2 Trajectory enhancement (vector-space case)

Require: Old step sizes h_t^{old} and optimized iterates $x^{(t,\text{old})}$ for $t = 1, \ldots, T$.
1: Compute $S_t^{\text{old}} := \|x^{(t+1,\text{old})} - x^{(t,\text{old})}\|_2$ and $\bar{S} := \sum_{t=1}^{N} S_t^{\text{old}}/T$.
2: Compute $\tilde{S}_t^{\text{old}} := \|x^{(t,\text{old})} - x^*\|_2$.
3: **for** $T' = 1, \ldots, T$ **do** {▷ Prune current trajectory}
4: If $\tilde{S}_{T'}^{\text{old}} \leq \delta$ for some $\delta > 0$, store T' and exit the loop.
5: **end for**
6: **for** $t = 1, \ldots, T'$ **do**
7: Compute $h_t^{\text{new}} := h_t^{\text{old}} + \eta(\bar{S}h_t^{\text{old}}/S_t^{\text{old}} - h_t^{\text{old}})$ for some $\eta \in (0, 1]$. {▷ Reparametrize}
8: For $C \geq 1$, store $\hat{h}_t^{\text{new}} := [h_t^{\text{new}}/(C+1), \ldots, h_t^{\text{new}}/(C+1)]$ for $C+1$ times. {▷ Refine}
9: **end for**
10: Re-optimize (18) for $x^{(t,\text{new})}$ with $\{\hat{h}_t^{\text{new}}\}$.
11: Repeat all steps above if new arc lengths S_t^{new} still greatly differ in magnitude.

2. JKO Flownet reparametrization

Although Algorithm 2 is developed for re-parametrizing trajectories in vector space $\mathbb{R}^d$, it can be directly used to reparametrize h for JKO-Flow under the W_2 metric for "arc-length" under minor changes.

In particular, let K be the total number of trained blocks via Algorithm 1 and denote $h_k := t_{k+1} - t_k$ as the "step-size" for block $\boldsymbol{f}_k$. Meanwhile, we replace the vector-space iterates $x^{(t)}$ with $x(t_k)$, which is the mapping through previous $k-1$ blocks. As a result, the arc length S_t becomes the W_2 distance, which can be easily computed using N samples $\{x_i(t_k)\}_{i=1}^N$ along each step of the trajectory.

We comment on the refinement procedure in line 8 of Algorithm 2 under this new setting. In theory, these C "interpolated points" requires training C *more* residuals block along *each* flow from p_k to p_{k+1}, which amounts to re-optimizing the JKO loss objective (6) C times more. In practice, doing so is infeasible and against our motivation for computational efficiency. Instead, we use the *same* block $\boldsymbol{f}_k$ to compute $C \geq 1$ "intermediate points":

$$x(t_k(c+1)) := x(t_k(c)) + \int_{t_k(c)}^{t_k(c+1)} \boldsymbol{f}_k(x(s), s; \theta_{k+1}) ds. \tag{19}$$

$$t_k(c) := \sum_{i=0}^{k-1} h_i + \frac{c}{C+1} h_k, c = 0, \ldots, C-1.$$

Then, we still optimize objective (6) *only once* using $x(t_k(C+1))$, which is the result of $C+1$ steps starting at $x(t_k(0))$. In addition, we warm start weights of $\boldsymbol{f}_k^{(\text{new})}$ with $\boldsymbol{f}_k^{(\text{old})}$.

We lastly explain the benefits of such "prune-reparametrize-refine" enhancement for training the residual blocks. First, the pruning procedure helps reduce overall computation, as it removes later blocks that barely contribute to the overall flow movement but require even more computational efforts than earlier blocks (due to having to flow x_i through previous blocks). Second, the refinement procedure yields more accurate transported densities p_{k+1} through the intermediate steps $k + \frac{c}{C+1}$, which also induce smoother overall transport trajectories. In addition, as we do not optimize the block at each of the C intermediate steps (19), the increase in overall computation is marginal. The blue (Phase 1) and brown (Phase 6) curves in Figure A.3a show that such refinement after six enhancement phases evidently ensures more uniform movement. As shown in Figure A.3c, the generated $\hat{X}$ through inverting the network with inputs Z are also closer to the true X.

C Experimental details

Other setup details. All experiments are conducted using PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey & Lenssen, 2019) and trained with one Tesla P100-PCIE-16GB GPU. Except for the graph node feature generative task, all residual blocks use fully-connected layers of the form $d \to H \to H \to d$, where d (resp. H) is the feature (resp. hidden nodes') dimension. The activation function is chosen as Tanh. In addition, we use Rugge-Kutta 4 (Lawrence, 1986) to numerically estimate the integrals in the continuous flow model, where the upper limites $T_b = 1$ for all blocks when reparametrization is not used. The Adam optimizer is used throughout training.

Datasets. For the 2D rose example, we generate 1000 random draws of 10000 training points representing the rose, and each flow model uses the same batch size of 1000 training observations. The four high-dimensional real datasets (POWER, GAS, HEP- MASS, MINIBOONE) come from the University of California Irvine (UCI) machine learning data repository. These datasets are commonly used to compare flow models (Grathwohl et al., 2019; Finlay et al., 2020; Onken et al., 2021). The solar dataset is retrieved from the National Solar Radiation Database (NSRDB).

MMD Metrics Besides visual comparison, the maximum mean discrepancy (MMD) (Gretton et al., 2012; Onken et al., 2021) provides a quantitative way to evaluate the performance of generative models. Given samples $\boldsymbol{X} := \{x_i\}_{i=1}^N$ and $\boldsymbol{Y} := \{y_j\}_{j=1}^M$ and a kernel function $k(x, y)$, we compute

$$\text{MMD}(\boldsymbol{X}, \boldsymbol{Y}) := \frac{1}{N^2} \sum_{i=1}^N \sum_{j=1}^N k(x_i, x_j) + \frac{1}{M^2} \sum_{i=1}^M \sum_{j=1}^M k(y_i, y_j) - \frac{2}{NM} \sum_{i=1}^N \sum_{j=1}^M k(x_i, y_j). \tag{20}$$

For our purpose, we use the Gaussian kernel $k(x, y) := \exp\big(-\|x - y\|^2/h\big)$ with bandwidth h. We select the bandwidth both as a constant value $h_c = 2$ and via the "median trick" (Gretton et al., 2012) $h_m := 2\text{median}(\{\|x_i - x_j\|^2\}_{i,j})$, where x_i are test samples. We thus denote MMD[c] (resp. MMD[m]) as evaluating (20) using the Gaussian kernel with constant (resp. median) bandwidth, where $\boldsymbol{X}$ (resp. $\boldsymbol{Y}$) denotes true (resp. generated) test sample. Note that a low MMD value indicates two samples $\boldsymbol{X}$ and $\boldsymbol{Y}$ are likely drawn from the same distribution (Gretton et al., 2012). In this

setting, MMD is an impartial evaluation metric as it is not used to train JKO-Flow or any competing methods.

We can also determine the statistical significance of a MMD value. First, compute the threshold

$$\tau := Q_{1-\alpha}(\{\text{MMD}(\boldsymbol{X}[\mathcal{I}_1^b], \boldsymbol{X}[\mathcal{I}_2^b]\}_{b=1}^B), \tag{21}$$

where $Q_{1-\alpha}$ denotes the upper $1-\alpha$ quantile of a set of scalars and $\mathcal{I}_j^b \subset \{1, \ldots, N\}$ denotes the j-th index set at the b-th bootstrapping. Then, under the null hypothesis that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are drawn from the same distribution, this hypothesis is rejected if MMD exceeds the threshold τ. The Type-I error is controlled at level α. Thus, if the MMD values by two models both exceed τ, we prefer the model with the smaller MMD. If both values are under τ, then they generate equally well.

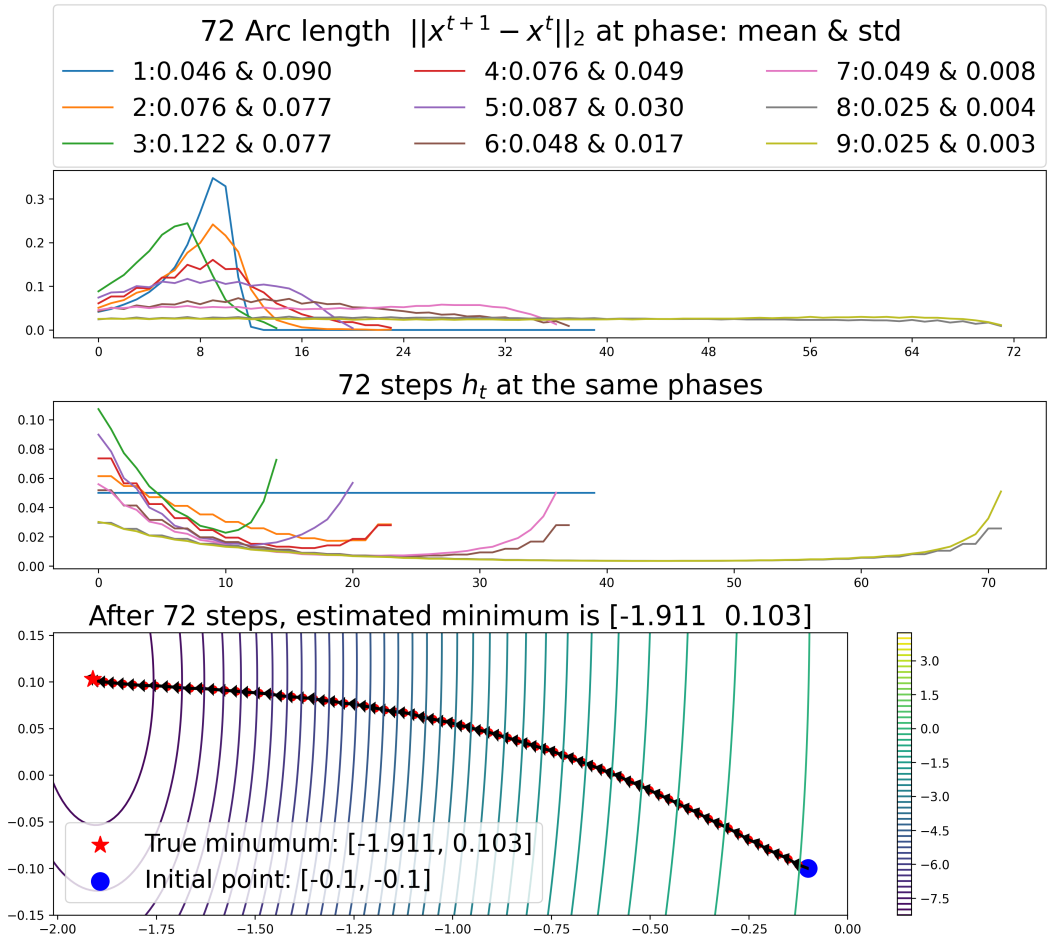
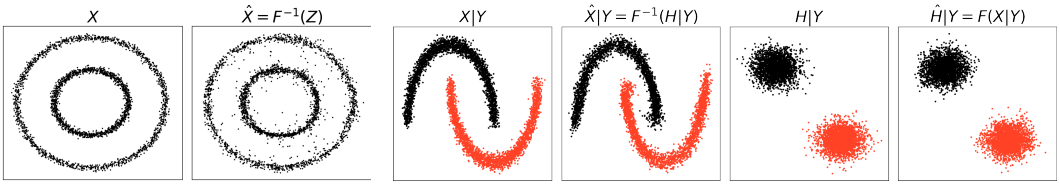


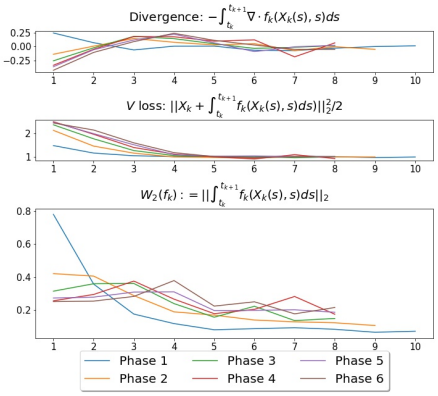
Figure A.1: Reparametrization of Muller potential in vector space. After 8 enhancement phases (e.g., at phase 9), all arc lengths are approximately uniform in width. Quantitative, there is an 17X reduction in the ratio $\frac{\text{standard deviation (std.)}}{\text{mean}}$ of arc lengths from phase 1 to phase 9.



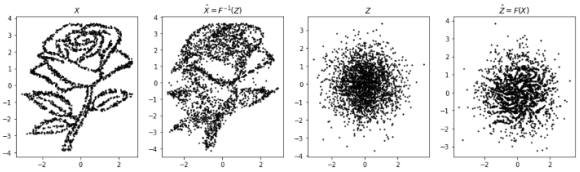
(a) Two-circles

(b) Conditional generation illustration by JKO-Flow.

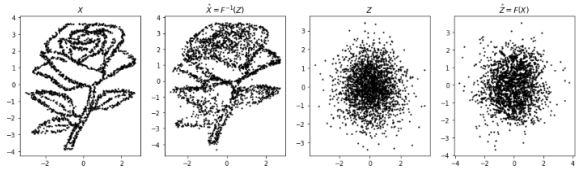
Figure A.2: Additional unconditional and conditional generation on simulated datasets.



(a) Losses in enhancement phases.

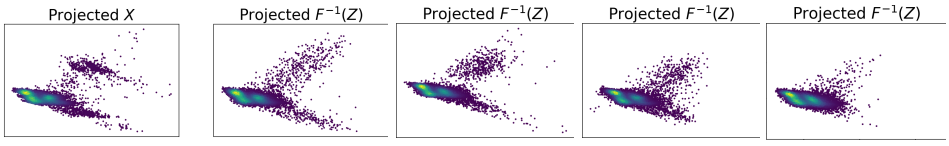


(b) Results at phase 1 (no enhancement yet).

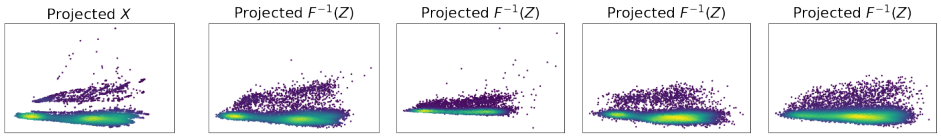


(c) Results at phase 6 (after 5 enhancement).

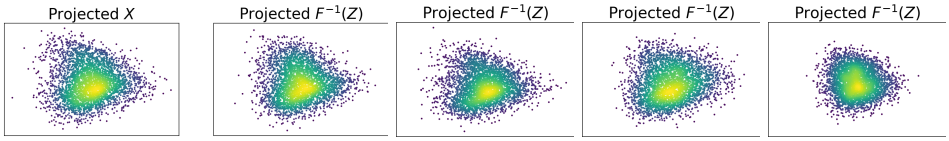
Figure A.3: Rose, enhancement of JKO-Flow over 6 phases. Here, phase 1 is the pre-enhancement phase during which we specify $t_k = k$ for training the residual blocks. Phase 6 results are obtained from using Algorithm 2 (modified for probability spaces) 5 times.



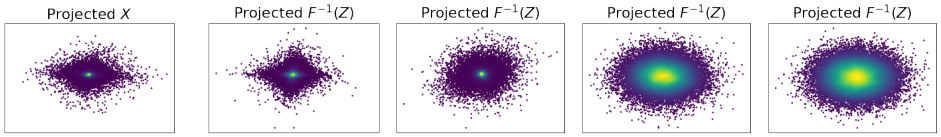
(a) Power: JKO-Flow (b) OT-Flow (c) FFJORD (d) W_2 IResNet



(e) Gas: JKO-Flow (f) OT-Flow (g) FFJORD (h) W_2 IResNet



(i) Miniboone: JKO-Flow (j) OT-Flow (k) FFJORD (l) IResNet



(m) BSDS300: JKO-Flow (n) OT-Flow (o) W_2 IResNet (p) IResNet

Figure A.4: Generative quality on high-dimensional datasets via PCA projection. The generative quality in general aligns with the MMD values shown in Table 2.