I&R Final Project - The Transport Maps of Flows and SVGD: Tradeoffs and a Middle Ground

Michael Albergo msa616, Guy Davidson gd1279, Matthew Drnevich mdd424, Camille Taltas ct2522

November 5, 2020

1 Introduction

Recent progress in generative modeling has introduced a number of methods which rely on discerning a transport map between a simple, tractable distribution, and a complicated (perhaps unknown) target distribution. Among those are a class of models known as normalizing flows, which do this explicitly by parameterizing this transport map with discrete, simple transformations that make exact inference and efficient sampling possible using a change-of-variables formula. The maps that flows create are largely defined by learned parameters, but much structure is imposed on the functional form of the flow in order make the change of variables calculation possible. The goal of this research project is to detail the structure behind the maps flows define, and in the process, gain insight into the benefits and limitations of such choices.

To do this, we will describe an alternative iterative transport algorithm known as Stein Variational Gradient Descent (SVGD), which imposes stricter, non-parametric dynamics to sample from a target distribution. In comparing the two methods, we hope to shed light on the efficacy of both, including their advantages and disadvantages, to get a sense of where the development of these approaches might head in the future. In the process, we will discuss recent continuous-time flows that adopt some of the characteristics of both methods, and describe them in this context.

2 Normalizing Flows: Theory and Motivation

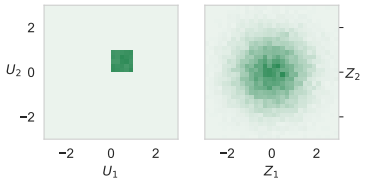
A primary goal of modern unsupervised machine learning is to find efficient ways of performing inference on high dimensional data. Given some data $\{x\}_i^n \in \mathbb{R}^d$ from some unknown distribution P with density p, one might seek to maximize the likelihood of this data under some model distribution with density q. Alternatively, one might want to sample or compute expectations under some known P without any means to do so, and instead seek to find the optimal approximate density q in some variational family. This is, of course, a challenging task in general, and a variety of methods have emerged in recent years that rely on some inductive bias to perform this inference exactly, with approximate bounds, or implicitly. In the following section, we will detail how a class of models, called normalizing flows, has exploited a well known relation connecting two distributions together via a *change of variables* formula. We will explain how it is possible to define a probabilistic model for which sampling and exact likelihood calculation are both feasible.

2.1 A look at the change of variables formula

Imagine that we have samples $\{z\}$ from some distribution with a known tractable density $p_z(z)$. If we were to transform our samples $\{z\}$ by passing them through an arbitrary function $T(z)$, then unless T is the identity transformation, the new samples $x = T(z)$ no longer have density $p_z(z)$.

Let's take a closer look at what we mean here with a well known example – the Box-Muller transform. Given two random variables U_1, U_2 each with $U_i \sim \text{unif}(0, 1)$, we say that they have joint density $p_u(U_1, U_2)$. It turns out we can construct a transformation given by

$$Z_1 = \sqrt{-2 \ln U_1} \cos(2\pi U_2) \quad \text{and} \quad Z_2 = \sqrt{-2 \ln U_1} \sin(2\pi U_2). \tag{1}$$



What we've done here is transformed our uniform variables to Z_1, Z_2, which are jointly distributed as a bivariate Gaussian with identity covariance. We *changed variables* from U_i to Z_i, and in doing so, we can calculate the new joint density for the transformed variables $p_z(Z_1, Z_2)$ by keeping track of all the partial derivatives of each Z_i with respect to U_j. The determinant of the matrix of these pairwise partials tells us how the volume element changes under the transformation:

$$\begin{aligned} p_z(Z_1, Z_2) &= p_u(U_1, U_2) \left| \det_{ij} \frac{\partial Z_i(U_1, U_2)}{\partial U_j} \right|^{-1} \\ &= 1 \times \left| \det \begin{pmatrix} \frac{-1}{U_1\sqrt{-2\ln U_1}} \cos(2\pi U_2) & -2\pi\sqrt{-2\ln U_1} \sin(2\pi U_2) \\ \frac{-1}{U_1\sqrt{-2\ln U_1}} \sin(2\pi U_2) & 2\pi\sqrt{-2\ln U_1} \cos(2\pi U_2) \end{pmatrix} \right|^{-1} = \left| \frac{2\pi}{U_1} \right|^{-1}. \end{aligned} \tag{2}$$

where we call the barred term the determinant of the Jacobian. A quick check given that $U_1 = e^{-(Z_1^2+Z_2^2)/2}$ verifies $p_z(Z_1, Z_2) = \frac{1}{2\pi} e^{-(Z_1^2+Z_2^2)/2}$ is indeed the density for the 2D Gaussian. What we've done here is made use of the *change of variables* formula, which tells us how to relate the probability densities for two variables that are connected by a differentiable 1-to-1 function. More generally, for $x \in \mathbb{R}^d$ subject to transformation $T^{-1}(x) = z$ we can write the distribution $p_x(x)$ as

$$p_x(x) = p_z(z) \left| \det J_{T^{-1}}(x) \right| \tag{3}$$

where $J_T(z)$ is the Jacobian for $T(z)$. That is, we can define $p_x(x)$ with respect to some base distribution $p_z(z)$. We will see in the next section how we can use this idea to learn highly expressive maps.

2.2 Parameterizing the change of variables formula: Affine flows

We can consider making use of this mapping in machine learning if we can come up with a way to parameterize the transformation in a calculable way [1, 2]. In order to be able to use the change of variables and compute the density of x, we need to define a transport map $T : Z \to X$ that is invertible (although the inverse does not have to be tractable depending on its use) and has a tractable Jacobian. However, computing the target density requires taking the determinant which costs up to $\mathcal{O}(d^3)$ and easily becomes intractable for high-dimensional datasets. Hence, flows can have computational and/or expressiveness limitations depending on the choice of transformation.

One example of normalizing flows with tractable Jacobians are affine flows, as popularized by the RealNVP model [3]. The latter have the following transformation, for data $z \in \mathbb{R}^d$ and $k < d$, typically $k = \frac{d}{2}$:

$$z_{1:k}^{t+1} = z_{1:k}^{t}$$
$$z_{k+1:d}^{t+1} = T(z) = e^{a(z_{1:k}^{t})} z_{k+1:d}^{t} + b(z_{1:k}^{t})$$

At each transformation, we pass half of the variables along to the next layer as-is, and use this frozen half to as inputs to parametric transformations $a, b : \mathbb{R}^k \to \mathbb{R}^{d-k}$, which we use to shift and scale the subset latent variables changed in this transformation. To make sure all variables change through the model, we reverse the vector after each layer, to use the half that previously changed as the frozen half, and vice versa. Note that this transformation is invertible since $e^a \neq 0$. From here, it is easy to see that the derivative with respect to each z_i will be equal to e^{a_i} and the Jacobian for this transform is upper triangular with its determinant being the product of the terms of its diagonal. This gives us

$$\det |J_T(z)| = \prod_i e^{a_i}$$

which is both easy to compute and invert. However, the downfall of these affine flows is that they have little expressivity. As mentioned above, they can only shift or scale our sampled data which limits the space of target distributions greatly. One way of remedying to such drawbacks is by composing transformations to reach multimodal distributions.

Note that composing invertible and differentiable transformations gives an invertible and differentiable composition. Thus, we can use this technique in order to reach more complex transformations while still having a well-defined Jacobian. The use of the term flow in this context makes more sense as we are gradually transforming our samples by running it through this chain of transformations. Thus we have,

$$T = T^K \circ \ldots \circ T^1$$

where

$$z_k = T^k(z_{k-1})$$

and

$$\det J_T(z) = \prod_{k=1}^{K} \det J_{T^k}(z_{k-1})$$

for $k = 1, \ldots, K$. Here, our starting samples, $z = z_0$ and our final samples, $x = z_K$.

Note that composing K flows only increases the computational complexity of the Jacobian determinant by $\mathcal{O}(K)$ which is a low cost for achieving more expressive target distributions.

Implementing each layer T^k requires making a choice for the function class of the parametric transformations a and b, most often parameterizing each by a neural network. Architectures as simple as multi-layer perceptions can work, although incorporating data-appropriate inductive biases may enable learning better transformations. While the neural network layers are not necessarily invertible, due to the nonlinearities used, the entire transformation remains invertible due to the structure of the flow. In the k'th transformation T^k, we can denote the parameterizations as $a^k(z; \theta_k)$ and $b^k(z; \phi_k)$, such that $T^k(z; \theta_k, \phi_k)$ is the entire trasnformation, and denote by $\Theta = \{\theta, \phi_1, \cdots, \theta_K, \phi_K\}$ the entire parameterization of the model.

Now, in order to optimize the parameters of these flow-based models so that our final distribution $q_x(x; \Theta)$ fits our target distribution $p_x^*(x)$, we minimize a divergence measure between the two probability distribution, and use stochastic gradient descent to optimize with respect to the transformation parameters Θ.

One such measure is the reverse KL divergence which, using the change of variables formula, is defined as

$$\begin{aligned} D_{KL}[q_x(x;\Theta)||p_x^*(x)] &= \mathbb{E}_{q_x(x;\Theta)}[\log q_x(x;\Theta) - \log p_x^*(x)] \\ &= \mathbb{E}_{p_z(z)}[\log p_z(z) - \log|\det J_T(z;\Theta)| - \log p_x^*(T(z;\Theta))] \end{aligned}$$

Thus, in order to minimize the divergence between our flow-based model and our target distribution, we only need samples from our prior distribution $p_z(z)$ and a tractable Jacobian determinant for T. We then approximate the true KL divergence by taking these samples, transforming them using our model, evaluating their probability under the target distribution, and backpropagating through to update the model parameters.

Even though this parametric variational optimization is computationally feasible and leads to more expressive solutions when composing simple flows, our space of target distributions may be limited by the parameterization, and our map from our base distribution to our target distribution may not be direct. Moreover, there is recent work suggesting that the functions made of affine compositions may not be universal approximators [4].

3 Flows in practice

To examine the differences between the transport maps learned by the methods we compare here, we will evaluate them on a fairly simple toy example. The code for this example, and producing the plots in this section, is available here at the link in the footnote[1]. We will attempt to learn to map a unimodal Gaussian in $\mathbb{R}^2$ with zero mean and diagonal covariance to a mixture of two non-zero-mean Gaussians, whose means are at $\binom{4}{0}$ and $\binom{-4}{0}$, both with covariance $0.5\mathcal{I}_2$, the identity scaled by 0.5. A minor side benefit of the RealNVP model updating half of its latent variables in a two-dimensional example is that the sequence of transformations will update between alternative the x and y coordinates of each sample, which we will be able to nicely visualize.

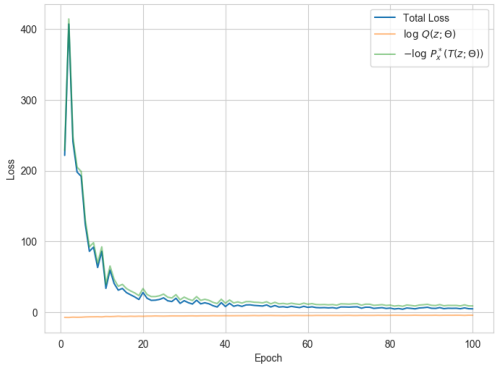


Figure 1: **Affine Flow Loss.** Behavior of the affine flow through the first 100 epochs of training, plotting the total loss as well as the two contributions independently.

In practice, our implementation makes the following decisions: each transformation uses multi-layer perceptrons for $a^k(z;\theta_k)$ and $b^k(z;\phi_k)$, both taking an input in $\mathbb{R}^2$, using two hidden layers with 8 units each and LeakyReLU nonlinearities, and a final output layer with 2 output units and a tanh nonlinearity. Note that the transformations as defined above only act on half of the data at a time; in practice it is easier to define the transformations to act on the entire inputs and mask them to achieve the same effect. We use a model with sixteen chained transformations, eight updating each of the two coordinates of each sample. We optimize the model with the Adam optimizer [5] using a learning rate of 1e-3 and a batch size of 128 for a total of 2500 epochs. Figure 1 depicts the result of the loss over the first 100 epochs of training – we see that the primary contribution is the loss under the target distribution, rather than the prior or determinant of the Jacobian, and that 100 epochs already allow it to get quite low.

[1] https://colab.research.google.com/drive/1d6F1VMxzyKSZ7Akqqq-ZXD5z4qTGrCqn?usp=sharing

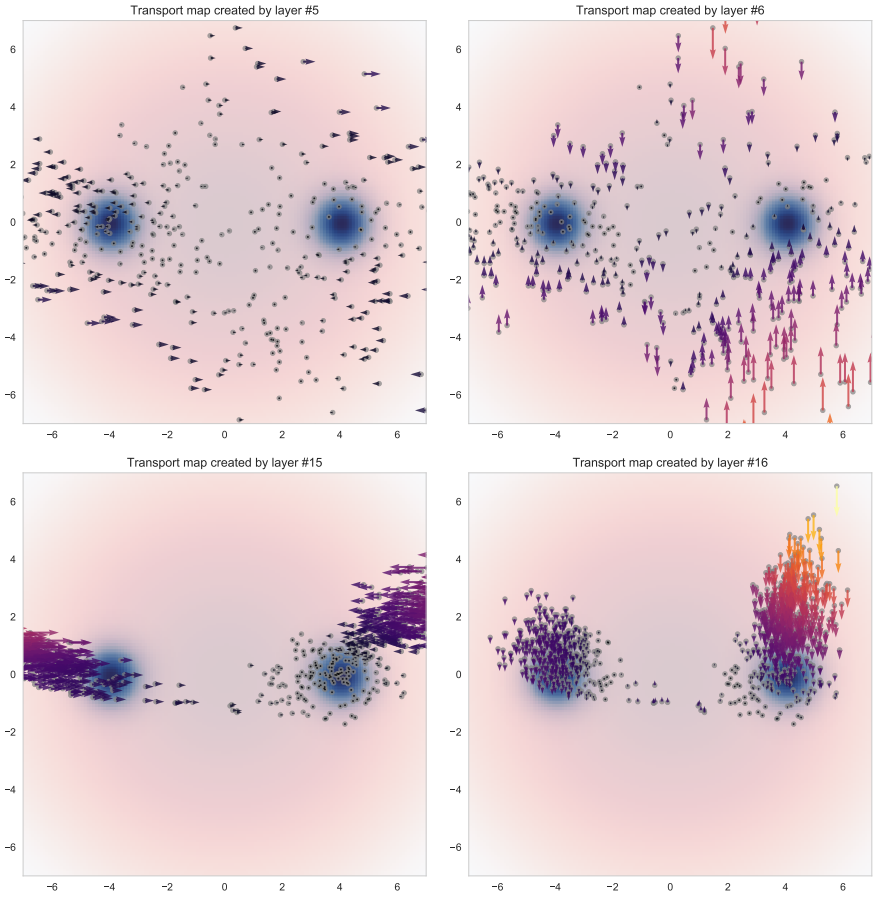


Figure 2: **Transformations learned by different layers.** Each panel depicts the transformation learned in a single transformation of affine normalizing flow.

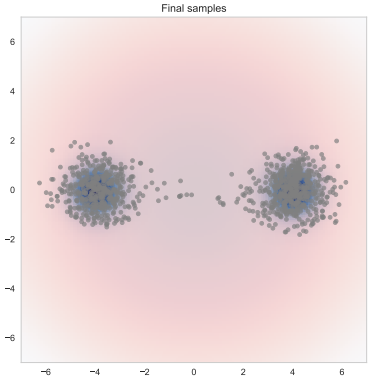


Figure 3: **Final samples from the affine flow.**

To further examine the behavior of this learned flow, in Figure 2 we plot the transport maps learned by individual transformation layers in the final trained version of the flow. The diffuse red density in the background represents the prior, while the two smaller blue densities represent the posterior. In each image, each point represents a single sample, and the arrow represents the effect of a that layer's transformation on the flow, with brighter colors corresponding to greater magnitudes. For an animation of the transformation of every step, see the link in the footnote[2]. As mentioned before, we can observe that odd layers (starting from 1) transform the x coordinate, while the even layers manipulate the y coordinate. We can also see that while the early layers still offer a rather diffuse flow, by the penultimate and ultimate layers the flow is focused on making adjustments very close to the target regions of high density, leaving alone points that are already reasonably close and pushing those that are further away. Finally, in Figure 3 we plot the final samples transformed by the trained affine normalizing flow model, which appear to match the target bimodal distribution quite well.

[2] https://drive.google.com/file/d/1TxHql63Dj-Y798GiZeT6Mx3VVmPqs9r2/view?usp=sharing

Note that the affine transport maps learned by RealNVP do manage to learn this (simple) distribution, but the restricted form of the map they use forces the flow to learn a non-trivial path from the base distribution to the target. Other recent approaches investigated other parameterizations of coupling layers, improving the flexibility of the types of coupling layers the models can learn, at limited additional computational cost or complexity. See, for example, Masked Autoregressive Flows [6], Neural Autoregressive Flows [7], or Neural Spline Flows [8]. Yet, all of these methods still enforce restrictive architectures on the transport maps that we define. This begs the question of whether or not we can construct less restrictive maps which allow us to follow more natural paths from our base distribution to our target.

4 Comparing transport maps: Stein Variational Gradient Descent

How does the affine flow transport map compare to those which are based on *non-parametric* variational optimization? We'd like to compare the discrete transformations to a different form of iterative data transport, but one that has strict molecular dynamics with some convergence guarantees.[3] To do this we will describe a recently proposed discrepancy measure based off of Stein's method, which has a form that allows us to devise a transport procedure that steps in the direction of minimizing the KL-divergence between the empirical sample distribution and the target.

4.1 Stein's Method

Stein's method is an analytic tool conventionally used for proving convergence of distributions, and was initially used for proving central-limit theorems [11]. We start with the following problem. We are given some target distribution P with density p that has support on $\mathbb{R}^d$ and is known up to a normalizing constant. However, we do not know how to integrate under P, i.e. we can't compute the expected value of quantities of interest under P.

As such, we'd like to approximate P with some empirical distribution Q made from samples $\{x_i\}$. This begs the question: Can we quantify how well the sample average expectation $\mathbb{E}_Q$ approximates $\mathbb{E}_P$ in a computationally tractable way? Moreover, can we detect if subsequent instantiations of Q are or are not converging to P? One thing we could turn to in order to do so are Integral Probability Metrics (IPMs) [12, 13], which measure the maximum discrepancy between expectations under P and Q for a class of test functions G:

$$d_G(Q, P) = \sup_{g \in G} \left| \int g \, dQ - \int g \, dP \right| \tag{4}$$

$$= \sup_{g \in G} \left| \mathbb{E}_Q[g(x)] - \mathbb{E}_P[g(x)] \right| . \tag{5}$$

Here $g : \mathcal{X} \to \mathbb{R}^d$ from a measureable space to the set of reals. If one chooses a sufficiently large class of functions, then one can get some convergence guarantees, such that if the IPM goes to 0, it implies Q weakly converges to P. Unfortunately, for function classes that are useful for proving this convergence, this discrepancy is generally intractable.

Computing this discrepancy would be much easier if we could choose our class of test functions such that they are *all mean 0 under* P. In this case, the IPM would only involve integration under Q. Enter Stein's method. Stein's method introduces a Stein operator $\mathcal{T}$ that acts on the test functions $g \in G$ to generate

[3] The convergence detection properties of the Stein Discrepancy depend on the kernel which defines the distance. Some kernels fail to detect non-convergence. See [9]. In the large-particle limit, SVGD can be described under differential equataions with invariant solutions in infinite time [10]

mean 0 functions under P. We say that a distribution P is characterized by the Stein operator $\mathcal{T}$ and Stein class G iff

$$\text{for } x \sim P \qquad \mathbb{E}_P\left[\mathcal{T}g(x)\right] = 0 \quad \forall g \in G \tag{6}$$

The operator $\mathcal{T}$ takes a vector-valued function g and outputs a function which meets the mean zero criteria.

4.2 Stein Discrepancy

Equipped with the notion of the Stein Operator, we can now define a discrepancy under its application, such that the resulting IPM no longer requires integration under P. Give the stein operator $\mathcal{T}$ and the Stein set G, we can define the Stein discrepancy [14]

$$S(Q, \mathcal{T}_p, G) = \sup_{g \in G} \left[\mathbb{E}_Q\left[\text{trace}\big((\mathcal{T}_p g)(x)\big)\right]\right]^2 \tag{7}$$

where we have squared the IPM description to remove the absolute value.[4] In this sense Stein's method has been rerouted to derive a measure of sample quality. Intuitively, the term $\mathbb{E}_Q[(T_p g)(x)]$ is not zero, as the requirement in (6) assumes expectation under P. The size of the expectation under Q, then, tells us about how different Q is from P. A remaining question is how to choose the Stein operator $\mathcal{T}$ and Stein set G such that they characterize P and $S(Q, \mathcal{T}, G)$ is tractable.

To choose the Stein operator, Gorham and Mackey [14] use the method of generators [15, 16], which involves specifying the target distribution P as the stationary distribution of a Markov process whose infinitesimal generator satisfies the Stein condition under P. By choosing Overdamped Langevin Diffusion [17] as the process, one attains the Stein operator [14, 18]:

$$(\mathcal{T}_p g)(x) = \nabla_x \log p(x) g(x)^\top + \nabla_x g(x). \tag{8}$$

This operator[5] is very appealing because it only depends on the gradient of the logarithm of the target density p, which means we do not need to compute any normalization.

Simultaneously proposed approaches in [19, 20] introduced the idea of using kernel functions as the Stein set, building on concepts from [18, 21] where kernels and Stein's identity were used for variance reduction in Monte Carlo. These results were extended in [9] for better convergence guarantees on the choice of kernel.

A kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function $k(x, y) = k(y, x)$ such that the matrix of all pairwise computations of points x_i form a positive semi-definite matrix.[6] These kernels induce an associated function space $\mathcal{H}$ called the Reproducing Kernel Hilbert Space (RKHS) [22], which implies it has a well-defined norm. If we choose the Stein set G_k to be reproducing kernels with regularity conditions imposed by the norm

$$G_k = \left\{g = (g_1, \ldots, g_d) \,\middle|\, ||v|| \leq 1, v_j = ||g_j||_{\mathcal{H}}\right\} \tag{9}$$

then the Kernelized Stein Discrepancy (KSD) takes on a computable closed form:

$$S(Q, \mathcal{T}_p, G_k) = ||g^*||^2 \tag{10}$$

where

$$g^*(\cdot) = \mathbb{E}_Q\left[k(x, \cdot)\nabla_x \log p(x) + \nabla_x k(x, \cdot)\right] = \underset{g \in G_k, ||g|| \leq 1}{\text{argmax}} \left[\mathbb{E}_Q\left[\text{trace}\big((\mathcal{T}_p g)(x)\big)\right]\right]^2 \tag{11}$$

[4] This will also be useful when deriving SVGD, as the update rule is clearer then.

[5] Goram and Mackey [14] write the operator formula as an inner product, such that the Stein Discrepancy can be written without the need for a trace function. We will follow [19] and maintain outer product notation so that the exposition of SVGD is clearer.

[6] Also referred to as a Gram matrix.

is the optimal test function for the KSD (subject to normalization), and where we take $||\cdot||$ to refer to the L^2 norm.

4.3 Variational Descent using the optimal Stein function

With a computationally tractable Stein discrepancy and an explicit form of the optimal test function, it is now possible to explore a variational optimization procedure with no parameters in sight. Returning to the principles behind normalizing flows regarding the change of variables formula, we can consider an iterative transformation $T : \mathcal{X} \to \mathcal{X}$ that takes a set x^0 sampled from some base distribution Q_0 with density q_0 and transforms it to $x^1 = T(x^0)$ such that

$$q_{[1]}(x^1) = q_0(T^{-1}(x^1))\big|\det(J_{T^{-1}}(x^1)\big| \tag{12}$$

This set of transformations defines a family of distributions $\{\mathcal{Q}\}$. Using this family, we can variationally optimize the set of samples so that they approach the optimal approximate density q^* to some target density p. That is, we want to update our empirical sample distribution such that we find:

$$q^* = \underset{q\in\mathcal{Q}}{\operatorname{argmin}}\, \mathrm{D}_{KL}(q||p) \tag{13}$$

To find this minimum, [23] proposed a method to descend the sample set toward the target distribution by taking optimal steps of the form $T(x) = x^{i+1} = x^i + \epsilon g(x^i)$ that are informed by the Stein Discrepancy.[7] After applying the transformation T, Liu et. al. recognized that the gradient of the D_{KL} with respect to the ϵ shift in this transformation is given by

$$\nabla\epsilon\, \mathrm{D}_{KL}(q_{i+1}||p) = -\mathbb{E}_{Q^i}\left[\mathrm{trace}\big((\mathcal{T}_p g)(x^i)\big)\right] \tag{14}$$

By taking g to be g^*, the above gradient will be maximized, such that g^* will give the direction of steepest descent in bringing the empirical distribution q^{i+1} closer to p:

$$\begin{aligned}\nabla\epsilon\, \mathrm{D}_{KL}(q_{i+1}||p) &= -\mathbb{E}_{Q^i}\left[\mathrm{trace}\big((\mathcal{T}_p g*)(x^i)\big)\right] &(15)\\ &= -\left[\mathbb{E}_Q\left[\mathrm{trace}\big((\mathcal{T}_p g)(x^i)\big)\right]\right]^2 &(16)\\ &= -S(Q_i, T_p, G_k) &(17)\end{aligned}$$

This tell us that the ϵ-magnitude of descent on the KL-divergence corresponds to the discrepancy, with, as stated before, the direction given by g^*.

Rewriting the transformation generically as $T(x^{i+1}) = x^i + g(x^i)$, we can treat the KL-divergence as a functional of the function g. Then, the above result can be reinterpreted as a form of functional gradient descent, with the functional gradient given by g^*

$$\nabla g\, \mathrm{D}_{KL}(q_{i+1}||p) = -g^* \tag{18}$$

which decreases D_{KL} until T reduces to the identity. This tells us about the dynamics each of the n samples in the sample set $\{x^i\}_{m=1}^n$ experiences at time-step i:

$$\dot{x}_m = \frac{1}{n}\sum_{j=1}^{n}\left[k(x_j, x_m)\nabla_{x_j}\log p(x_j) + \nabla_{x_j}k(x_j, x_m)\right] = g^*(x_m) \tag{19}$$

This procedure is known as Stein Variational Gradient Descent (SVGD). So in the end we have arrived at a non-parametric *flow*, instead of discrete learnable coupling layers. The incremental update in the infinite limit behaves as a differential equation. Indeed, in the $n \to \infty$ limit and treating $g*$ as a continuous velocity field, SVGD can be seen as following a Vlasov process, and its mean-field PDE converges to a unique solution as $t \to \infty$ [24, 10].

[7] This incremental step looks a lot like the affine flow, but now all the dimensions are updated at once.

4.4 Stein's in practice

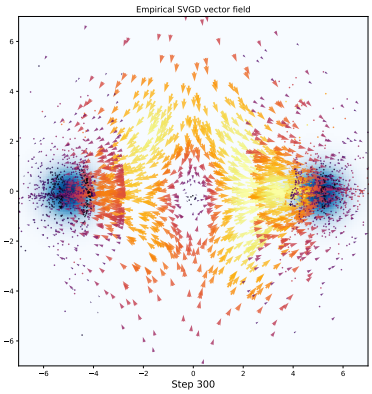


Figure 4: Empirical vector field of the SVGD.

We now compare the transport dynamics of SVGD to the affine flows presented in Section 3. Again we pose the same task as we did earlier: map samples from an approximate distribution (initialized as a diffuse 2D Gaussian) under a target distribution, a bimodal Gaussian mixture. As a reminder, in SVGD, instead of coupling layers, we take infinitesimal steps moving the sample set from the base distribution according to the optimal Stein function, updating all dimensions of the data at once. As such, the Jacobian is dense and the transport of the samples follows a smooth vector field toward the bimodal target, approximating it well. Empirically, they also monotonically approach the density, and applying more SVGD steps will bring the sample set under even better approximation

This procedure comes with a number of costs compared to the flows we've seen before, and it is worth addressing them to explain why we have considered SVGD in this report discussion. Because we have put no restrictions on the Jacobian other than the dynamics imposed by the optimal Stein function, density estimation is intractable once more, as the determinant in the change-of-variables has returned to being $O(d^3)$ (even though the Jacobian is just perturbations of the identity). Moreover, there is no amortized cost in SVGD, so the mapping must be recomputed at every step whenever one wants to map samples from the base density under the target of interest. No parameters store this information for later use. This is costly in the case of SVGD, where it takes $O(n^2)$ computations to compute the next step trajectory for all the data. Finally, SVGD (and Kernelized Stein Discrepancies in general) suffer from the curse of dimensionality, as the kernel can become less informative in higher dimensions [25].

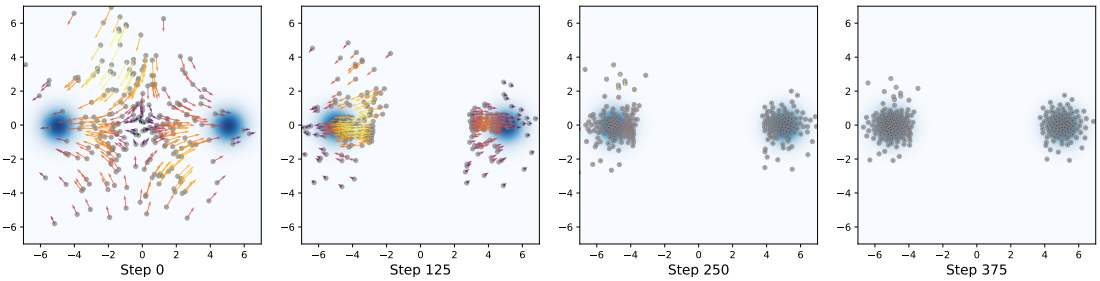


Figure 5: SVGD learning two separated Gaussians from the intraparticle dynamics.

Indeed, we mainly introduce SVGD as a non-parametric analogue to parametric flows to inspire the consideration of continuous time flows. Given that the dynamics of SVGD tend toward those given by a differential equation, and they admit a smoother, more direct mapping than coupling layer flows, it begs the question of whether or not it is possible to parameterize this sort of differential equation, and whether a flow can be constructed from such dynamics. In this case, is density estimation feasible again, and are the costs of optimization amortized?

5 Can Flows learn "better" paths?

In this section, we'll make the transition from SVGD back to flows, now under a continuous-time parameterization. We will describe recent work that allows flows to learn smoother/more direct maps from base distribution to target via ODEs, and compare them on our toy problem.

5.1 Continuous time flows (ODEs)

Traditional flows are constructed as a sequence of compositions (sometimes called "layers") of the form

$$\mathbf{z}_{n+1} = f_n(\mathbf{z}_n; \theta_n) \tag{20}$$

such that each f_n is invertible and is parameterized by θ_n. The overall flow, or transport map, would then be

$$T = f_N \circ f_{N-1} \circ \cdots \circ f_1$$

for a flow with N layers such that $T : Z \to X$ where Z is the base distribution space and X is the target distribution space.

If we take the limit of infinitesimal time steps, we can view this sequence of transformations as discrete states of a continuous transformation from the base to target space. We can re-write our flow in a suggestive way to make this analogy more clear. It's equivalent to equation (20) to say

$$\mathbf{z}_{t+1} = \mathbf{z}_t + g(\mathbf{z}_t; \theta_t) \tag{21}$$

and in the limit of infinitesimal time steps, we extract that

$$\frac{d\mathbf{z}(t)}{dt} = g(\mathbf{z}(t), t; \theta)$$

Now we can see that $\mathbf{z}(t)$ is a curve, i.e. $\mathbf{z} : \mathbb{R} \to \mathbb{R}^d$ where d is the dimension of Z and X, with its derivative modelled by g. To recover our target variable, $\mathbf{x} \in X$, we can globally fix some times t_0 and t_1 and define

$$\mathbf{x} = T(\mathbf{z}) \equiv \mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} g(\mathbf{z}(t), t; \theta) dt \tag{22}$$

In practice, this would be computed using an ODE solver such as the Runge-Kutta method which computationally only requires evaluating g at specific points. Often, we might also want to evaluate some scalar-valued loss function from the output of this transport map, which would have the form

$$L(\mathbf{z}(t_1) = L(\mathbf{z}(t_0) + \int_{t_0}^{t_1} g(\mathbf{z}(t), t; \theta) dt) \tag{23}$$

and our optimization algorithm would require how the parameters, θ, vary with respect to this loss. At first glance this may seem like either a very difficult problem or a very costly one. Fortunately, it is only slightly complicated and comes at nearly no additional cost! The derivative of interest satisfies

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^{\mathrm{T}} \frac{\partial f}{\partial \theta} dt \tag{24}$$

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathrm{T}} \frac{\partial f}{\partial \mathbf{z}} \tag{25}$$

which allows us to compute $\mathbf{z}(t)$, $\mathbf{a}(t)$, and $dL/d\theta$ all at the same time through one (backwards) pass of the ODE solver, noting that we can use automatic differentiation for $\partial f / \partial \mathbf{z}$.

So far, we have neglected the relation to one of the main advantages of normalizing flows - being able to compute the likelihood. Initially, it might seem that making the flow continuous would complicate our ability to track the change in volume. However, Chen et al. [26] proved an astonishingly simple result (adapted from [26])

Theorem 1 (Instantaneous Change of Variables) *Let $\mathbf{z}(t)$ be a finite continuous random variable with probability $p(\mathbf{z}(t))$ dependent on time. Let $\frac{d\mathbf{z}}{dt} = g(\mathbf{z}(t), t)$ be a differential equation describing a continuous-in-time transformation of $\mathbf{z}(t)$. Assuming that g is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, then the change in log probability also follows a differential equation*

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -Tr\left(\frac{dg}{d\mathbf{z}(t)}\right) \tag{26}$$

which means that we can explicitly write out the likelihood as

$$\log p(\mathbf{x}) = \log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{dg}{d\mathbf{z}(t)}\right) dt \tag{27}$$

Now it's good to pause and reflect on the main changes that have resulted from our new view on normalizing flows. We have the following key changes so far:

Computational Complexity In general, computing the target likelihood of a normalizing flow has a time cost of $\mathcal{O}(d^3)$ for computing the determinant, which can limit its usefulness in high dimensions. By forming a continuous analogue to flows, the computation cost has been reduced to the cost of computing the trace, which is $\mathcal{O}(d^2)$, and using an ODE solver[8].

Architecture Restrictions Normalizing flows require that your family of functions be bijective and differentiable in order to use the change of variables formula. Continuous normalizing flows instead require that your family of functions is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, which will imply that the corresponding transport map (the solution of the ODE) is differentiable and bijective. In particular, one can use any neural network with finite weights and Lipschitz activation functions, such as *tanh* and *ReLU*. The resulting transport map $T : Z \to X$ will then be defined by equation (22) which allows for a more diverse family of transport maps in practice.

Optimization Continuous normalizing flows can be trained in the same way as traditional normalizing flows, with one additional caveat. In order to use gradient descent methods, we must also incorporate an ODE solver to compute the gradients as in equation (24). This introduces an explicit trade-off between speed and precision determined by the tolerance of the ODE solver.

5.1.1 FFJORD

Even with these computational improvements, one might wonder if we can improve things further. It just so happens that by introducing an estimator we can reduce the time cost of computing the trace to $\mathcal{O}(d)$. The idea is that for any d-dimensional distribution $p(\boldsymbol{\epsilon})$ where $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $\mathrm{Cov}(\boldsymbol{\epsilon}) = I$ we have

$$\mathrm{Tr}(A) = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[\boldsymbol{\epsilon}^T A \boldsymbol{\epsilon}\right] \tag{28}$$

for any matrix $A \in \mathbb{R}^{d\times d}$ [27, 28]. The corresponding Monte-Carlo estimator of equation (28) is called Hutchinson's estimator [27]

$$\mathrm{Tr}(A) \approx \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{\epsilon}_i^T A \boldsymbol{\epsilon}_i \tag{29}$$

where each $\boldsymbol{\epsilon}_i \sim p(\boldsymbol{\epsilon})$. It's important to note that this is an unbiased estimator that has time cost $\mathcal{O}(d)$, which is a significant improvement over the general $\mathcal{O}(d^3)$ cost of normalizing flows!

[8] Fourth order Runge-Kutta is a common choice. It has error $\mathcal{O}(h^4)$ with h being the step size in time and requires four evaluations of the dynamics per step.

It would now be natural to ask about the variance of this estimator. Hutchinson showed that the variance of this estimator grows asymptotic to $\|A\|_F^2$. Therefore, a trivial way to decrease the variance of A is to reduce its dimensions. In the case of continuous normalizing flows, we're interested in the trace of the Jacobian of a function g. Assume g is a neural network such that it is a composition of a series of functions. Call these functions g_i such that $g = g_H \circ g_{H-1} \circ \cdots \circ g_1$ and each g_i corresponds to a "layer" of the neural network. Now, define h to be the index of the layer with the smallest dimension (say dimension m), so we can define two more functions

$$\psi \equiv g_H \circ g_{H-1} \circ \cdots \circ g_{h+1}, \quad \phi \equiv g_h \circ g_{h-1} \circ \cdots \circ g_1$$

Finally, we can deduce using the cyclic property of the trace that

$$\operatorname{Tr}\left(\frac{dg}{d\mathbf{z}(t)}\right) = \operatorname{Tr}\left(\frac{\partial\psi}{\partial\phi}\frac{\partial\phi}{\partial\mathbf{z}(t)}\right) = \operatorname{Tr}\left(\frac{\partial\phi}{\partial\mathbf{z}(t)}\frac{\partial\psi}{\partial\phi}\right) \tag{30}$$

where we have turned a $d \times d$ dimensional matrix into an $m \times m$ dimensional matrix to reduce the variance of the estimator. This is called the "bottleneck trick" [28]. Ultimately, combining this trick with the continuous normalizing flows approach allow us to compute the likelihood of our target distribution with time cost $\mathcal{O}(d)$ by introducing an ODE solver of arbitrary tolerance and some variance on the trace.

5.2 OT-Flow

Continuous normalizing flows have two major computational challenges. The first is that the likelihood computation depends on an ODE solver. This ODE solver will have to evaluate the function g at a number of points which are dependent upon the dynamics of the flow and has the potential to grow quite large, particularly towards the end of the training [26]. The second is that computing the trace of the Jacobian exactly costs $\mathcal{O}(d^2)$ or $\mathcal{O}(d)$ with an unbiased estimator. Improving these will drastically improve the efficiency of CNFs with higher dimensional data. One exciting approach to solving these challenges is through integrating optimal transport regularization with continuous normalizing flows [29].

To help motivate why it might be a good idea to introduce regularization, we should think about the setup of the problem again. We have one distribution that lives in X and another in Z and we want to find a transport map $T : Z \to X$ with dynamics determined by an arbitrary neural network $g : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$. This problem has infinitely many solutions! Our approach so far has been just to find *a solution* to this problem, but any perturbation of $\mathbf{z}(t)$ at $t \notin \{t_0, t_1\}$ will also be a solution of this problem, and potentially one that is more difficult to compute. So the new question is how do we encourage solutions that are easier to compute?

Trying to "simplify" curves is a very natural question in mathematics and has been rigorously studied in geometry. The idea is that you choose some metric for your space which defines a "length" for any curve, and then you want to find the curves that minimize these lengths. Such curves are called "geodesics". Geodesics tell us the fastest and simplest way to get from point A to point B given the constraints of the world around us. This approach has found particular success in physics with the Principle of Least Action, which simply says that everything around us moves along geodesics determined by the governing dynamics of nature. This approach also has a strong connection to optimal transport.

Optimal transport is concerned with minimizing the cost of moving probability mass from one distribution to another, and calling the map with the minimal cost the optimal transport map. This optimal transport map is equivalent to a geodesics on the space of probability distributions. To simplify computations for the continuous normalizing flows case, it makes sense to want "straight" lines. Since any perturbation between the endpoints maintains the solution, we can just "squeeze" the curves until there are no perturbations. The metric that induces straight-line geodesics is precisely the usual L^2 norm, which induces the "action" of the curve to be

$$\mathrm{L}(\mathbf{z}, t_1, t_0) \equiv \frac{1}{2}\int_{t_0}^{t_1} \left\|\frac{d\mathbf{z}(t)}{dt}\right\|^2 dt = \frac{1}{2}\int_{t_0}^{t_1} \|g(\mathbf{z}(t), t)\|^2 \, dt \tag{31}$$

Another way to say this is that this "loss" functional is minimized by curves $\mathbf{z} : \mathbb{R} \to \mathbb{R}^d$ which are straight lines. Onken et al. [29] came up with the idea of applying this loss to continuous normalizing flows as a regularization term. This also comes at almost no additional cost since it can be computed in the ODE solver.

Additionally, Onken et al. took a famous result from optimal control theory, which states that any optimal flow between two fixed points must satisfy conditions of a Hamiltonian system defined by this initial value problem. This, again, is closely related to the Principle of Least Action described earlier. The analog in this case requires that the flow satisfies the Hamilton-Jacobi-Bellman equation if it is in fact an optimal path. This approach requires that we consider the function governing the dynamics to be a "force" resulting from some physical potential function $\Phi(\mathbf{z}(t), t, \theta) : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ such that

$$g(\mathbf{z}(t), t; \theta) = -\nabla_{\mathbf{z}} \Phi(\mathbf{z}(t), t; \theta)$$

The Hamilton-Jacobi-Bellman equation then reads

$$-\partial_t \Phi(\mathbf{z}(t), t; \theta) + \frac{1}{2} \left\| \nabla_{\mathbf{z}} \Phi(\mathbf{z}(t), t; \theta) \right\|^2 = 0 \tag{32}$$

which leads to a regularization term of [29]

$$\mathrm{R}(\mathbf{z}, t_0, t_1) \equiv \int_{t_0}^{t_1} \left| \partial_t \Phi(\mathbf{z}(t), t; \theta) - \frac{1}{2} \left\| \nabla_{\mathbf{z}} \Phi(\mathbf{z}(t), t; \theta) \right\|^2 \right| dt \tag{33}$$

Adding both of these terms to our optimization objective for continuous normalizing flows encourages the flows to learn straight, optimal paths while on average needing one-fourth the number of weights and training 19x faster and performing inference 28x faster at the same level of performance [29]. However, the implementation of these flow must be implemented differently to be able to use the Hamilton-Jacobi-Bellman equation.

Instead of modelling the dynamics, g, with a neural network, we now use a neural network to model its potential, Φ. However, we can be even more explicit and make the potential less "black box" by including specific physical terms related to various types of dynamical systems. Onken et al. use the following architecture

$$\Phi(\mathbf{s}; \boldsymbol{\theta}) = \mathbf{w}^T N(s; \boldsymbol{\theta}_N) + \frac{1}{2} \mathbf{s}^T \left(\mathbf{A}^T \mathbf{A} \right) \mathbf{s} + \mathbf{b}^T \mathbf{s} + c \tag{34}$$

where $\boldsymbol{\theta} = (\mathbf{w}, \boldsymbol{\theta}_N, \mathbf{A}, \mathbf{b}, c)$ are parameters, $\mathbf{s} = (\mathbf{z}, t)$ are the inputs, and $N(s; \boldsymbol{\theta}_N) : \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a neural network. The dimensions of the other parameters are $\mathbf{w} \in \mathbb{R}^m$, $\boldsymbol{\theta}_N \in \mathbb{R}^p$, $\mathbf{A} \in \mathbb{R}^{r \times (d+1)}$, $\mathbf{b} \in \mathbb{R}^{d+1}$, and $c \in \mathbb{R}$. Consequently, the potential is written as a sum of linear and nonlinear dynamics. Finally, one large benefit to this architecture design is that the time cost of the exact trace computation is now reduced to $\mathcal{O}(d)$. Please see the original paper (Onken et al. [29]) for how this simplifies the trace calculation.

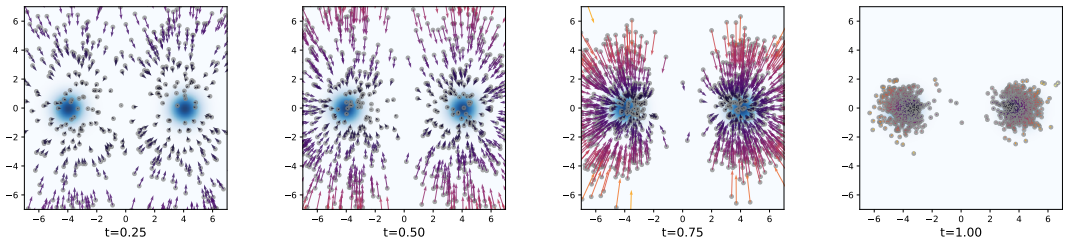


Figure 6: Continuous normalizing flow with optimal transport and control theory regularizations learning two separated Gaussians.

6 Discussion

Through comparing these three approaches to learning transport maps, we can see a clear tradeoff between the expressiveness of the model, its efficiency, and what we can calcualte along the way. Traditional normalizing flows, such as affine flows, have a very simple structure which makes them highly efficient with exact density estimation, but the transport maps they learn are often far from optimal due to the indirect path they are restricted to follow. We then compared this discrete, parametized mapping with a more incremental, non-parametric analogue in Stein Variational Gradient Descent, which, although more costly than affine flows, inspires a transport map where each step more directly brings the sample set under the target. To blend this more direct mapping with the benefits of parameterization and gradient descent, we considered recent work on continuous versions of flows based on ODEs. This allows the dynamics of the transport map to be completely unrestricted, hence allowing it to learn more optimal and intuitive paths between the distributions with unbiased density estimation. This freedom comes at a numerical cost, but more recent approaches have significantly improved upon this through using optimal transport-based loss functions.

It is worth asking whether or not there is a real advantage to having the less restricted Jacobian compared to the affine flows. Indeed, this comes at the cost of only having an unbiased estimate of the log-likelihood, instead of the exact form seen in the coupling layer method. The answer to this might be that it depends on the task. While extensions of affine flows such as the recently proposed Glow [30] show promise on image generation tasks, they are quickly being outperformed by continuous time models [31]. However, if having precise density estimation is essential to your task (say, in scientific domains), one might shy away from relying on methods which rely on estimations of the likelihood. Ultimately, the best method to use depends on the situation and, as always, there's no free lunch, but research in these areas is clearly flowing in many exciting directions.

References

[1] J. P. Agnelli, M. Cadeiras, E. Tabak, C. Turner, and E. Vanden-Eijnden. Clustering and classification through normalizing flows in feature space. *Multiscale Model. Simul.*, 8:1784–1802, 2010.

[2] Esteban G. Tabak and Eric Vanden-Eijnden. Density estimation by dual ascent of the log-likelihood. *Commun. Math. Sci.*, 8(1):217–233, 03 2010.

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[4] Antoine Wehenkel and Gilles Louppe. You say normalizing flows i see bayesian networks, 2020.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, 2015.

[6] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation, 2018.

[7] Chin-Weia Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. 2018.

[8] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 7511–7522. Curran Associates, Inc., 2019.

[9] Jackson Gorham and Lester Mackey. Measuring sample quality with kernels. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1292–1301, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[10] Jianfeng Lu, Yulong Lu, and James Nolen. Scaling limit of the stein variational gradient descent: the mean field regime, 2018.

[11] 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*, pages 583–602, Berkeley, Calif., 1972. University of California Press.

[12] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

[13] Bharath K. Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, and Gert R. G. Lanckriet. On the empirical estimation of integral probability metrics. *Electron. J. Statist.*, 6:1550–1599, 2012.

[14] Jackson Gorham and Lester Mackey. Measuring sample quality with stein's method. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 226–234. Curran Associates, Inc., 2015.

[15] A. D. Barbour. Stein's method and poisson process convergence. *Journal of Applied Probability*, 25(A):175–184, 1988.

[16] A. D. Barbour. Stein's method for diffusion approximations. *Probability Theory and Related Fields*, 84(3):297–322, 1990.

[17] G.A. Pavliotis. *Stochastic Processes and Applications: Diffusion Processes, the Fokker-Planck and Langevin Equations*. Texts in Applied Mathematics. Springer New York, 2014.

[18] Chris J. Oates, Mark Girolami, and Nicolas Chopin. Control functionals for monte carlo integration. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(3):695–718, 2017.

[19] Qiang Liu, Jason D. Lee, and Michael I. Jordan. A kernelized stein discrepancy for goodness-of-fit tests and model evaluation, 2016.

[20] Kacper Chwialkowski, Heiko Strathmann, and Arthur Gretton. A kernel test of goodness of fit, 2016.

[21] Chris J. Oates, Jon Cockayne, François-Xavier Briol, and Mark Girolami. Convergence rates for a class of estimators based on stein's method, 2017.

[22] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

[23] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 2378–2386. Curran Associates, Inc., 2016.

[24] Qiang Liu. Stein variational gradient descent as gradient flow. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 3115–3123. Curran Associates, Inc., 2017.

[25] Wenbo Gong, Yingzhen Li, and José Miguel Hernández-Lobato. Sliced kernelized stein discrepancy, 2020.

[26] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

[27] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communication in Statistics- Simulation and Computation*, 18:1059–1076, 01 1989.

[28] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: free-form continuous dynamics for scalable reversible generative models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[29] Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport, 2020.

[30] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018.

[31] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2020.