PRELIMINARY VERSION: DO NOT CITE
The AAAI Digital Library will contain the published
version some time after the conference

OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport

Derek Onken,[1] **Samy Wu Fung,**[2] **Xingjian Li,**[3] **Lars Ruthotto,**[3,1]
[1] Department of Computer Science, Emory University
[2] Department of Mathematics, University of California, Los Angeles
[3] Department of Mathematics, Emory University
donken@emory.edu, swufung@math.ucla.edu, xingjian.li@emory.edu, lruthotto@emory.edu

Abstract

A normalizing flow is an invertible mapping between an arbitrary probability distribution and a standard normal distribution; it can be used for density estimation and statistical inference. Computing the flow follows the change of variables formula and thus requires invertibility of the mapping and an efficient way to compute the determinant of its Jacobian. To satisfy these requirements, normalizing flows typically consist of carefully chosen components. Continuous normalizing flows (CNFs) are mappings obtained by solving a neural ordinary differential equation (ODE). The neural ODE's dynamics can be chosen almost arbitrarily while ensuring invertibility. Moreover, the log-determinant of the flow's Jacobian can be obtained by integrating the trace of the dynamics' Jacobian along the flow. Our proposed OT-Flow approach tackles two critical computational challenges that limit a more widespread use of CNFs. First, OT-Flow leverages optimal transport (OT) theory to regularize the CNF and enforce straight trajectories that are easier to integrate. Second, OT-Flow features exact trace computation with time complexity equal to trace estimators used in existing CNFs. On five high-dimensional density estimation and generative modeling tasks, OT-Flow performs competitively to state-of-the-art CNFs while on average requiring one-fourth of the number of weights with an 8x speedup in training time and 24x speedup in inference.

1 Introduction

A normalizing flow (Rezende and Mohamed 2015) is an invertible mapping $f\colon \mathbb{R}^d \to \mathbb{R}^d$ between an arbitrary probability distribution and a standard normal distribution whose densities we denote by ρ_0 and ρ_1, respectively. By the change of variables formula, for all $\boldsymbol{x} \in \mathbb{R}^d$, the flow must satisfy (Rezende and Mohamed 2015; Papamakarios et al. 2019)

$$\log \rho_0(\boldsymbol{x}) = \log \rho_1(f(\boldsymbol{x})) + \log |\det \nabla f(\boldsymbol{x})| \,. \quad (1)$$

Given ρ_0, a normalizing flow is constructed by composing invertible layers to form a neural network and training their weights. Since computing the log-determinant in general requires $\mathcal{O}(d^3)$ floating point operations (FLOPS), effective normalizing flows consist of layers whose Jacobians have exploitable structure (e.g., diagonal, triangular, low-rank).

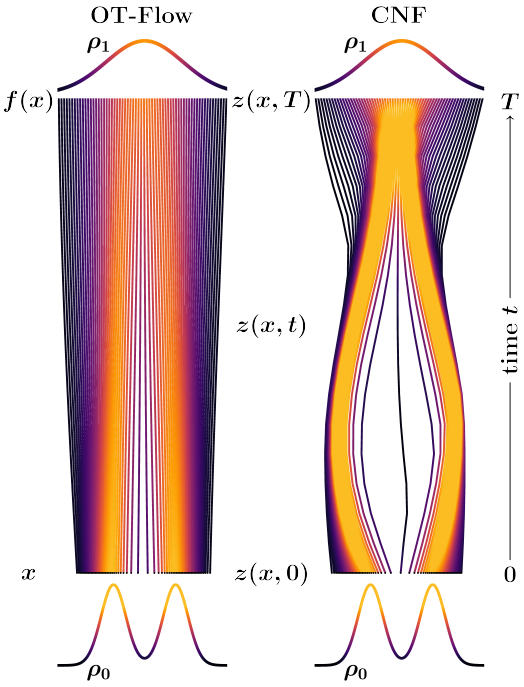


Figure 1: Two flows with approximately equal loss (modification of Fig. 1 in Grathwohl et al. 2019; Finlay et al. 2020). While OT-Flow enforces straight trajectories, a generic CNF can have curved trajectories.

Alternatively, in continuous normalizing flows (CNFs), f is obtained by solving the neural ordinary differential equation (ODE) (Chen et al. 2018b; Grathwohl et al. 2019)

$$\partial_t \begin{bmatrix} \boldsymbol{z}(\boldsymbol{x},t) \\ \ell(\boldsymbol{x},t) \end{bmatrix} = \begin{bmatrix} \mathbf{v}\big(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta}\big) \\ \operatorname{tr}\big(\nabla \mathbf{v}(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})\big) \end{bmatrix}, \quad \begin{bmatrix} \boldsymbol{z}(\boldsymbol{x},0) \\ \ell(\boldsymbol{x},0) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix}, \quad (2)$$

for artificial time $t \in [0,T]$ and $\boldsymbol{x} \in \mathbb{R}^d$. The first component maps a point $\boldsymbol{x}$ to $f(\boldsymbol{x}) = \boldsymbol{z}(\boldsymbol{x},T)$ by following the trajectory $\boldsymbol{z}\colon \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$ (Fig. 1). This mapping is invertible and orientation-preserving under mild assumptions on the dynamics $\mathbf{v}\colon \mathbb{R}^d \times [0,T] \to \mathbb{R}^d$. The final state of the second

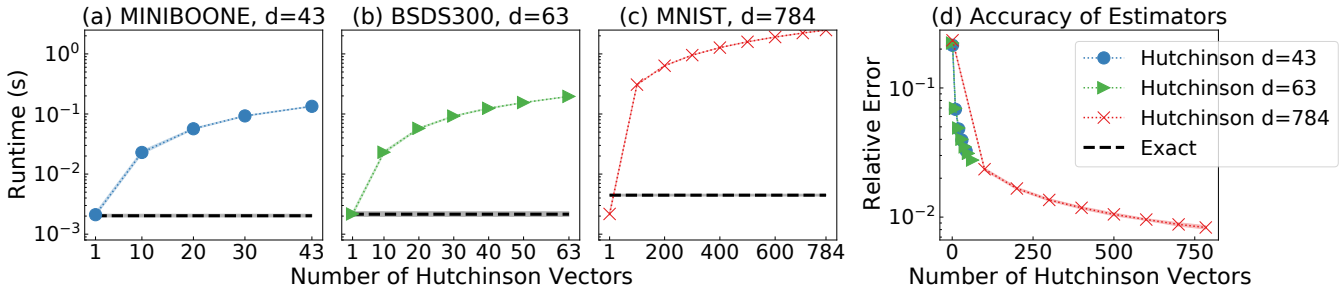


Figure 2: Performance comparison of trace computation using exact approach presented in Sec. 3 and Hutchinson's trace estimator using automatic differentiation. (a-c): runtimes (in seconds) over dimensions 43, 63, and 784, corresponding to the MINIBOONE, BSDS300, and MNIST data sets, respectively. (d): relative errors vs. number of Hutchinson vectors for different dimensions. We present means with shaded 99% error bounds computed from twenty runs via bootstrapping (App. C).

component satisfies $\ell(\boldsymbol{x}, T) = \log\det \nabla f(\boldsymbol{x})$, which can be derived from the instantaneous change of variables formula as in Chen et al. (2018b). Replacing the log determinant with a trace reduces the FLOPS to $\mathcal{O}(d^2)$ for exact computation or $\mathcal{O}(d)$ for an unbiased but noisy estimate (Zhang, E, and Wang 2018; Grathwohl et al. 2019; Finlay et al. 2020).

To train the dynamics, CNFs minimize the expected negative log-likelihood given by the right-hand-side in (1) (Rezende and Mohamed 2015; Papamakarios, Pavlakou, and Murray 2017; Papamakarios et al. 2019; Grathwohl et al. 2019) via

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{\rho_0(\boldsymbol{x})} \ \{C(\boldsymbol{x}, T)\}, \quad \text{for}$$
$$C(\boldsymbol{x}, T) \coloneqq \frac{1}{2}\|\boldsymbol{z}(\boldsymbol{x}, T)\|^2 - \ell(\boldsymbol{x}, T) + \frac{d}{2}\log(2\pi), \tag{3}$$

where for a given $\boldsymbol{\theta}$, the trajectory $\boldsymbol{z}$ satisfies the neural ODE (2). We note that the optimization problem (3) is equivalent to minimizing the Kullback-Leibler (KL) divergence between ρ_1 and the transformation of ρ_0 given by f (derivation in App. A or Papamakarios et al. 2019).

CNFs are promising but come at considerably high costs. They perform well in density estimation (Chen et al. 2018a; Grathwohl et al. 2019; Papamakarios et al. 2019) and inference (Ingraham et al. 2019; Papamakarios et al. 2019), especially in physics and computational chemistry (Noé et al. 2019; Brehmer et al. 2020). CNFs are computationally expensive for two predominant reasons. First, even using state-of-the-art ODE solvers, the computation of (2) can require a substantial number of evaluations of $\mathbf{v}$; this occurs, e.g., when the neural network parameters lead to a stiff ODE or dynamics that change quickly in time (Ascher 2008). Second, computing the trace term in (2) without building the Jacobian matrix is challenging. Using automatic differentiation (AD) to build the Jacobian requires separate vector-Jacobian products for all d standard basis vectors, which amounts to $\mathcal{O}(d^2)$ FLOPS. Trace estimates, used in many CNFs (Zhang, E, and Wang 2018; Grathwohl et al. 2019; Finlay et al. 2020), reduce these costs but introduce additional noise (Fig. 2). Our approach, OT-Flow, addresses these two challenges.

Modeling Contribution Since many flows exactly match two densities while achieving equal loss C (Fig. 1), we can choose a flow that reduces the number of time steps required to solve (2). To this end, we phrase the CNF as an optimal transport (OT) problem by adding a transport cost to (3). From this reformulation, we exploit the existence of a potential function whose derivative defines the dynamics $\mathbf{v}$. This potential satisfies the Hamilton-Jacobi-Bellman (HJB) equation, which arises from the optimality conditions of the OT problem. By including an additional cost, which penalizes deviations from the HJB equations, we further reduce the number of necessary time steps to solve (2) (Sec. 2). Ultimately, encoding the underlying regularity of OT into the network absolves it from learning unwanted dynamics, substantially reducing the number of parameters required to train the CNF.

Numerical Contribution To train the flow with reduced time steps, we opt for the discretize-then-optimize approach and use AD for the backpropagation (Sec. 3). Moreover, we analytically derive formulas to efficiently compute the exact trace of the Jacobian in (2). We compute the *exact* Jacobian trace with $O(d)$ FLOPS, matching the time complexity of *estimating* the trace with one Hutchinson vector as used in state-of-the-art CNFs (Grathwohl et al. 2019; Finlay et al. 2020). We demonstrate the competitive runtimes of the trace computation on several high-dimensional examples (Fig. 2). Ultimately, our PyTorch implementation[1] of OT-Flow produces results of similar quality to state-of-the-art CNFs at 8x training and 24x inference speedups on average (Sec. 5).

2 Mathematical Formulation of OT-Flow

Motivated by the similarities between training CNFs and solving OT problems (Benamou and Brenier 2000; Peyré and Cuturi 2019), we regularize the minimization problem (3) as follows. First, we formulate the CNF problem as an OT problem by adding a transport cost. Second, from OT theory, we leverage the fact that the optimal dynamics $\mathbf{v}$ are the negative gradient of a potential function Φ, which satisfies the HJB equations. Finally, we add an extra term to the learning problem that penalizes violations of the HJB equations. This reformulation encourages straight trajectories (Fig. 1).

[1] Code is available at https://github.com/EmoryMLIP/OT-Flow .

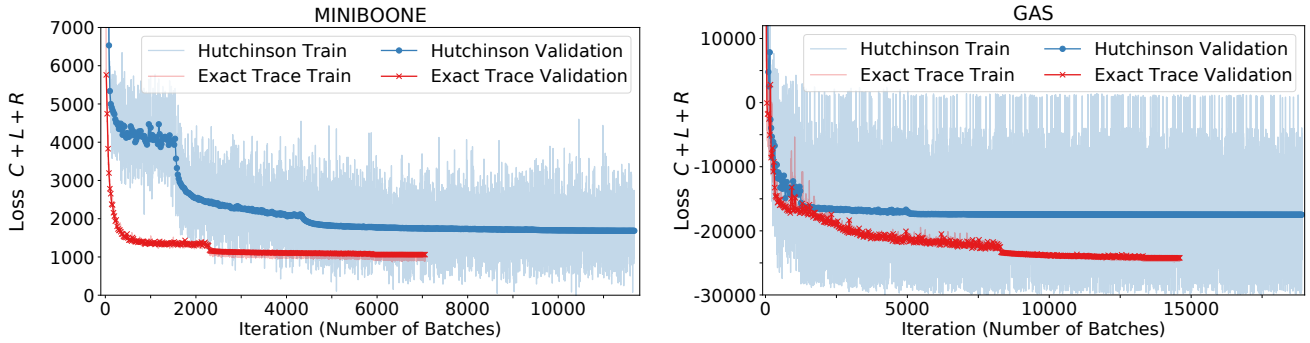


Figure 3: The exact trace computation in OT-Flow leads to faster decay of validation loss and lower training loss variance compared to an identical model using a randomized trace estimator (also used in FFJORD and RNODE) for two data sets.

Transport Cost We add the L_2 transport cost

$$L(\boldsymbol{x}, T) = \int_0^T \frac{1}{2} \|\mathbf{v}\big(\boldsymbol{z}(\boldsymbol{x}, t), t\big)\|^2 \, \mathrm{d}t, \tag{4}$$

to the objective in (3), which results in the regularized problem

$$\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{\rho_0(\boldsymbol{x})} \Big\{ C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) \Big\} \quad \text{s.t. (2).} \tag{5}$$

This transport cost penalizes the squared arc-length of the trajectories. In practice, this integral can be computed in the ODE solver, similar to the trace accumulation in (2). The OT problem (5) is the relaxed Benamou-Brenier formulation, i.e., the final time constraint is given here as the soft constraint $C(\boldsymbol{x}, T)$. This formulation has mathematical properties that we exploit to reduce computational costs (Evans 1997; Villani 2008; Lin, Lensink, and Haber 2019; Finlay et al. 2020). In particular, (5) is now equivalent to a *convex* optimization problem (prior to the neural network parameterization), and the trajectories matching the two densities ρ_0 and ρ_1 are straight and non-intersecting (Gangbo and McCann 1996). This reduces the number of time steps required to solve (2). The OT formulation also guarantees a solution flow that is smooth, invertible, and orientation-preserving (Ambrosio, Gigli, and Savaré 2008).

Potential Model We further capitalize on OT theory by incorporating additional structure to guide our modeling. In particular, from the Pontryagin Maximum Principle (Evans 2013, 2010), there exists a potential function $\Phi\colon \mathbb{R}^d \times [0, T] \to \mathbb{R}$ such that

$$\mathbf{v}(\boldsymbol{x}, t; \boldsymbol{\theta}) = -\nabla \Phi(\boldsymbol{x}, t; \boldsymbol{\theta}). \tag{6}$$

Analogous to classical physics, samples move in a manner to minimize their potential. In practice, we parameterize Φ with a neural network instead of $\mathbf{v}$. Moreover, optimal control theory states that Φ satisfies the HJB equations (Evans 2013)

$$\begin{aligned} -\partial_t \Phi(\boldsymbol{x}, t) + \frac{1}{2} \|\nabla \Phi(\boldsymbol{z}(\boldsymbol{x}, t), t)\|^2 &= 0 \\ \Phi(\boldsymbol{x}, T) = 1 + \log\big(\rho_0(\boldsymbol{x})\big) - \log\big(\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\big) & \\ - \ell(\boldsymbol{z}(\boldsymbol{x}, T), T). & \end{aligned} \tag{7}$$

We derive the terminal condition in App. B. The existence of this potential allows us to reformulate the CNF in terms of Φ instead of $\mathbf{v}$ and add an additional regularization term which penalizes the violations of (7) along the trajectories by

$$R(\boldsymbol{x}, T) = \int_0^T \left| \partial_t \Phi\big(\boldsymbol{z}(\boldsymbol{x}, t), t\big) - \frac{1}{2} \|\nabla \Phi\big(\boldsymbol{z}(\boldsymbol{x}, t), t\big)\|^2 \right| \mathrm{d}t. \tag{8}$$

This HJB regularizer $R(\boldsymbol{x}, T)$ favors plausible Φ without affecting the solution of the optimization problem (5).

With implementation similar to $L(\boldsymbol{x}, T)$, the HJB regularizer R requires little computation, but drastically simplifies the cost of solving (2) in practice. We assess the effect of training a toy Gaussian mixture problem with and without the HJB regularizer (Fig. A1 in Appendix). For this demonstration, we train a few models using varied number of time steps and regularizations. For unregularized models with few time steps, we find that the L_2 cost is not penalized at enough points. Therefore, without an HJB regularizer, the model achieves poor performance and unstraight characteristics (Fig. A1). This issue can be remedied by adding more time steps or the HJB regularizer (see examples in Yang and Karniadakis 2020; Ruthotto et al. 2020; Lin et al. 2020). Whereas additional time steps add significant computational cost and memory, the HJB regularizer is inexpensive as we already compute $\nabla \Phi$ for the flow.

OT-Flow Problem In summary, the regularized problem solved in OT-Flow is

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \mathbb{E}_{\rho_0(\boldsymbol{x})} \Big\{ C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) + R(\boldsymbol{x}, T) \Big\}, \\ & \text{subject to} \quad (2), \end{aligned} \tag{9}$$

combining aspects from Zhang, E, and Wang (2018), Grathwohl et al. (2019), Yang and Karniadakis (2020), and Finlay et al. (2020) (Tab. 1). The L_2 and HJB terms add regularity and are accumulated along the trajectories. As such, they make use of the ODE solver and computed $\nabla \Phi$ (App. D).

3 Implementation of OT-Flow

We define our model, derive analytic formulas for fast and exact trace computation, and describe our efficient ODE solver.

Model	Formulation					Training Implementation			Inference
	ODEs (2)	Φ	L	R	$\|\nabla \mathbf{v}\|_F^2$	ODE Solver	Approach	Trace	Trace
FFJORD	✓	✗	✗	✗	✗	Runge-Kutta (4)5	OTD	Hutch w/ Rad	exact w/ AD loop
RNODE	✓	✗	✓	✗	✓	Runge-Kutta 4	OTD	Hutch w/ Rad	exact w/ AD loop
Monge-Ampère	✓	✓	✗	✗	✗	Runge-Kutta 4	DTO	Hutch w/ Gauss	
Potential Flow	✓	✓	✗	✓	✗	Runge-Kutta 1	DTO	exact w/ AD loop	
OT-Flow	✓	✓	✓	✓	✗	Runge-Kutta 4	DTO	efficient exact (Sec. 3)	

Table 1: All methods share the underlying neural ODEs but differ in use of a potential Φ, regularizers (L, R, $\|\nabla \mathbf{v}\|_F^2$), ODE solver, approach (discretize-then-optimize DTO or optimize-then-discretize OTD), and trace computation (exact using automatic differentiation AD, Hutchinson's estimator with a single vector sampled from a Rademacher or Gaussian distribution).

Network We parameterize the potential as

$$\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \boldsymbol{w}^\top N(\boldsymbol{s};\boldsymbol{\theta}_N) + \frac{1}{2}\boldsymbol{s}^\top(\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}^\top\boldsymbol{s} + c, \quad \text{where} \quad \boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{\theta}_N, \boldsymbol{A}, \boldsymbol{b}, c). \tag{10}$$

Here, $\boldsymbol{s} = (\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$ are the input features corresponding to space-time, $N(\boldsymbol{s};\boldsymbol{\theta}_N)\colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a neural network chosen to be a residual neural network (ResNet) (He et al. 2016) in our experiments, and $\boldsymbol{\theta}$ consists of all the trainable weights: $\boldsymbol{w} \in \mathbb{R}^m$, $\boldsymbol{\theta}_N \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{r\times(d+1)}$, $\boldsymbol{b} \in \mathbb{R}^{d+1}$, $c \in \mathbb{R}$. We set a rank $r = \min(10, d)$ to limit the number of parameters of the symmetric matrix $\boldsymbol{A}^\top\boldsymbol{A}$. Here, $\boldsymbol{A}$, $\boldsymbol{b}$, and c model quadratic potentials, i.e., linear dynamics; N models the nonlinear dynamics. This formulation was found to be effective in Ruthotto et al. (2020).

ResNet Our experiments use a simple two-layer ResNet. When tuning the number of layers as a hyperparameter, we found that wide networks promoted expressibility but deep networks offered no noticeable improvement. For simplicity, we present the two-layer derivation (for the derivation of a ResNet of any depth, see App. E or Ruthotto et al. 2020). The two-layer ResNet uses an opening layer to convert the $\mathbb{R}^{d+1}$ inputs to the $\mathbb{R}^m$ space, then one layer operating on the features in hidden space $\mathbb{R}^m$

$$\begin{aligned} \boldsymbol{u}_0 &= \sigma(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0) \\ N(\boldsymbol{s};\boldsymbol{\theta}_N) = \boldsymbol{u}_1 &= \boldsymbol{u}_0 + h\,\sigma(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1). \end{aligned} \tag{11}$$

We use step-size h=1, dense matrices $\boldsymbol{K}_0 \in \mathbb{R}^{m\times(d+1)}$ and $\boldsymbol{K}_1 \in \mathbb{R}^{m\times m}$, and biases $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^m$. We select the element-wise activation function $\sigma(\boldsymbol{x}) = \log(\exp(\boldsymbol{x}) + \exp(-\boldsymbol{x}))$, which is the antiderivative of the hyperbolic tangent, i.e., $\sigma'(\boldsymbol{x}) = \tanh(\boldsymbol{x})$. Therefore, hyperbolic tangent is the activation function of the flow $\nabla\Phi$.

Gradient Computation The gradient of the potential is

$$\nabla_{\boldsymbol{s}}\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \nabla_{\boldsymbol{s}}N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w} + (\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}, \tag{12}$$

where we simply take the first d components of $\nabla_{\boldsymbol{s}}\Phi$ to obtain the space derivative $\nabla\Phi$. The first term is computed using chain rule (backpropagation)

$$\begin{aligned} \boldsymbol{z}_1 &= \boldsymbol{w} + h\boldsymbol{K}_1^\top \mathrm{diag}(\sigma'(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1))\boldsymbol{w}, \\ \boldsymbol{z}_0 &= \boldsymbol{K}_0^\top \mathrm{diag}(\sigma'(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0))\boldsymbol{z}_1, \quad \text{where} \\ \nabla_{\boldsymbol{s}}N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w} &= \boldsymbol{z}_0. \end{aligned} \tag{13}$$

Here, $\mathrm{diag}(\boldsymbol{q}) \in \mathbb{R}^{m\times m}$ denotes a diagonal matrix with diagonal elements given by $\boldsymbol{q} \in \mathbb{R}^m$. Multiplication by diagonal matrix is implemented as an element-wise product.

Trace Computation We compute the trace of the Hessian of the potential model. We first note that

$$\begin{aligned} \mathrm{tr}\left(\nabla^2\Phi(\boldsymbol{s};\boldsymbol{\theta})\right) = \mathrm{tr}\left(\boldsymbol{E}^\top\,\nabla_{\boldsymbol{s}}^2\big(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w}\big)\,\boldsymbol{E}\right) \\ + \mathrm{tr}\left(\boldsymbol{E}^\top(\boldsymbol{A}^\top\boldsymbol{A})\,\boldsymbol{E}\right), \end{aligned} \tag{14}$$

where the columns of $\boldsymbol{E} \in \mathbb{R}^{(d+1)\times d}$ are the first d standard basis vectors in $\mathbb{R}^{d+1}$. All matrix multiplications with $\boldsymbol{E}$ can be implemented as constant-time indexing operations. The trace of the $\boldsymbol{A}^\top\boldsymbol{A}$ term is trivial. We compute the ResNet term via

$$\begin{aligned} &\mathrm{tr}\left(\boldsymbol{E}^\top\,\nabla_{\boldsymbol{s}}^2(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w})\,\boldsymbol{E}\right) = t_0 + h\,t_1, \quad \text{where} \\ &t_0 = \big(\sigma''(\boldsymbol{K}_0\boldsymbol{s} + \boldsymbol{b}_0)\odot\boldsymbol{z}_1\big)^\top\big((\boldsymbol{K}_0\boldsymbol{E})\odot(\boldsymbol{K}_0\boldsymbol{E})\big)\mathbf{1}, \\ &t_1 = \big(\sigma''(\boldsymbol{K}_1\boldsymbol{u}_0 + \boldsymbol{b}_1)\odot\boldsymbol{w}\big)^\top\big((\boldsymbol{K}_1 J)\odot(\boldsymbol{K}_1 J)\big)\mathbf{1}, \end{aligned} \tag{15}$$

where $\odot$ is the element-wise product of equally sized vectors or matrices, $\mathbf{1} \in \mathbb{R}^d$ is a vector of all ones, and $J = \nabla_{\boldsymbol{s}}\boldsymbol{u}_0^\top = \boldsymbol{K}_0^\top\sigma'(\boldsymbol{K}_0\,\boldsymbol{s} + \boldsymbol{b}_0)$. For deeper ResNets, the Jacobian term $J = \nabla_{\boldsymbol{s}}\boldsymbol{u}_{i-1}^\top \in \mathbb{R}^{m\times(d+1)}$ can be updated and over-written at a computational cost of $\mathcal{O}(m^2\cdot d)$ FLOPS (App. E).

The trace computation of the first layer uses $\mathcal{O}(m\cdot d)$ FLOPS, and each additional layer uses $\mathcal{O}(m^2\cdot d)$ FLOPS (App. E). Thus, our exact trace computation has similar computational complexity as FFJORD's and RNODE's trace estimation. In clocktime, the analytic exact trace computation is competitive with the Hutchinson's estimator using AD, while introducing no estimation error (Fig. 2). Our efficiency in trace computation (15) stems from exploiting the identity structure of matrix E and not building the full Hessian.

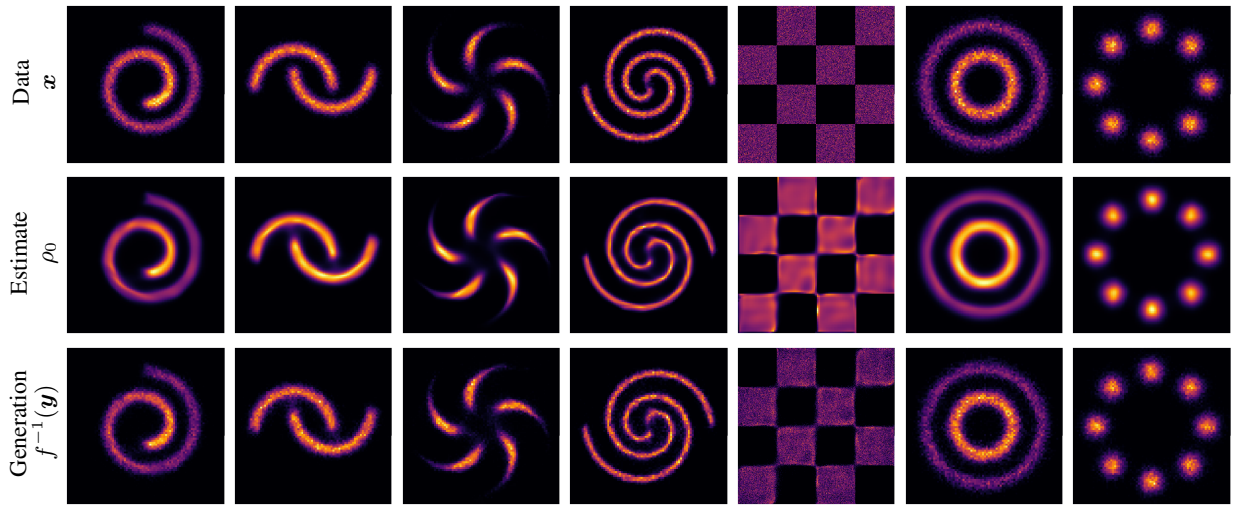


Figure 4: Density estimation on 2-D toy problems. **Top:** samples from the unknown distribution. **Middle:** density estimate for unknown ρ_0 computed by inverse flowing from ρ_1 via (2). **Bottom:** samples generated by inverse flow where $\boldsymbol{y} \sim \rho_1(\boldsymbol{y})$.

We find that using the exact trace instead of a trace estimator improves convergence (Fig. 3). Specifically, we train an OT-Flow model and a replicate model in which we only change the trace computation, i.e., we replace the exact trace computation with Hutchinson's estimator using a single random vector. The model using the exact trace (OT-Flow) converges more quickly and to a lower validation loss, while its training loss has less variance (Fig. 3).

Using Hutchinson's estimator without sufficiently many time steps fails to converge (Onken and Ruthotto 2020) because such an approach poorly approximates the time integration *and* the trace in the second component of (2). Whereas FFJORD and RNODE estimate the trace but solve the time integral well, OT-Flow trains with the exact trace and notably fewer time steps (Tab. 2). At inference, all three solve the trace and integration well.

ODE Solver For the forward propagation, we use Runge-Kutta 4 with equidistant time steps to solve (2) as well as the time integrals (4) and (8). The number of time steps is a hyperparameter. For validation and testing, we use more time steps than for training, which allows for higher precision and a check that our discrete OT-Flow still approximates the continuous object. A large number of training time steps prevents overfitting to a particular discretization of the continuous solution and lowers inverse error; too few time steps results in high inverse error but low computational cost. We tune the number of training time steps so that validation and training loss are similar with low computational cost.

For the backpropagation, we use AD. This technique corresponds to the discretize-then-optimize (DTO) approach, an effective method for ODE-constrained optimization problems (Collis and Heinkenschloss 2002; Abraham, Behr, and Heinkenschloss 2004; Becker and Vexler 2007; Leugering et al. 2014). In particular, DTO is efficient for solving neural ODEs (Li et al. 2017; Gholaminejad, Keutzer, and Biros 2019; Onken and Ruthotto 2020). Our implementation exploits the benefits of our proposed exact trace computation combined with the efficiency of DTO.

4 Related Works

Finite Flows Normalizing flows (Tabak and Turner 2013; Rezende and Mohamed 2015; Papamakarios et al. 2019; Kobyzev, Prince, and Brubaker 2020) use a composition of discrete transformations, where specific architectures are chosen to allow for efficient inverse and Jacobian determinant computations. NICE (Dinh, Krueger, and Bengio 2015), RealNVP (Dinh, Sohl-Dickstein, and Bengio 2017), IAF (Kingma et al. 2016), and MAF (Papamakarios, Pavlakou, and Murray 2017) use either autoregressive or coupling flows where the Jacobian is triangular, so the Jacobian determinant can be tractably computed. GLOW (Kingma and Dhariwal 2018) expands upon RealNVP by introducing an additional invertible convolution step. These flows are based on either coupling layers or autoregressive transformations, whose tractable invertibility allows for density evaluation and generative sampling. Neural Spline Flows (Durkan et al. 2019) use splines instead of the coupling layers used in GLOW and RealNVP. Using monotonic neural networks, NAF (Huang et al. 2018) require positivity of the weights. UMNN (Wehenkel and Louppe 2019) circumvent this requirement by parameterizing the Jacobian and then integrating numerically.

Infinitesimal Flows Modeling flows with differential equations is a natural and common concept (Suykens, Verrelst, and Vandewalle 1998; Welling and Teh 2011; Neal 2011; Salimans, Kingma, and Welling 2015). In particular, CNFs (Chen et al. 2018a,b; Grathwohl et al. 2019) model their flow via (2).

To alleviate the expensive training costs of CNFs, FFJORD (Grathwohl et al. 2019) sacrifices the exact but slow trace computation in (2) for a Hutchinson's trace estimator with complexity $\mathcal{O}(d)$ (Hutchinson 1990). This es-

Data Set	Model	# Param	Training				Testing			
			Time (h)	# Iter	$\frac{\text{Time}}{\text{Iter}}$(s)	NFE	Time (s)	Inv Err	MMD	C
POWER $d = 6$	OT-Flow	18K	3.1	22K	0.56	40	10.6	4.10e-6	4.68e-5	-0.30
	RNODE	43K	25.0	32K	2.78	200	88.2	5.95e-6	5.64e-5	-0.39
	FFJORD	43K	68.9	29K	8.63	583	72.4	7.60e-6	4.34e-5	-0.37
GAS $d = 8$	OT-Flow	127K	6.1	52K	0.42	40	30.9	1.79e-4	2.47e-4	-9.20
	RNODE	279K	36.3	59K	2.23	200	763.7	2.53e-5	8.03e-5	-11.10
	FFJORD	279K	75.4	49K	5.54	475	892.4	1.78e-5	1.02e-4	-10.69
HEPMASS $d = 21$	OT-Flow	72K	5.2	35K	0.53	48	47.9	2.98e-6	1.58e-5	17.32
	RNODE	547K	46.5	40K	4.16	400	446.7	1.91e-5	1.58e-5	16.37
	FFJORD	547K	99.4	47K	7.56	770	450.4	2.98e-5	1.58e-5	16.13
MINIBOONE $d = 43$	OT-Flow	78K	0.8	7K	0.44	24	0.8	5.65e-6	2.84e-4	10.55
	RNODE	821K	1.4	15K	0.33	16	33.0	4.42e-6	2.84e-4	10.65
	FFJORD	821K	9.0	16K	2.01	115	31.5	4.80e-6	2.84e-4	10.57
BSDS300 $d = 63$	OT-Flow	297K	7.1	37K	0.70	56	432.7	5.54e-5	4.24e-4	-154.20
	RNODE	6.7M	106.6	16K	23.4	200	15253.3	2.66e-6	1.64e-2	-129.75
	FFJORD	6.7M	166.1	18K	33.6	345	20061.2	3.41e-6	6.52e-3	-133.94

Table 2: Density estimation on real data sets. We present the number of training iterations, the number of function evaluations for the forward ODE solve (NFE), and the time per iteration. For BSDS300 training, FFJORD and RNODE were terminated when validation loss C hit -140. All values are the average across three runs on a single NVIDIA TITAN X GPU with 12GB RAM. We present the standard deviations computed from the three runs in Tab. A3 located in the Appendix.

timator helps FFJORD achieve training tractability by reducing the trace cost from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ per time step. However, during inference, FFJORD has $\mathcal{O}(d^2)$ trace computation cost since accurate CNF inference requires the exact trace (Sec. 1,Tab. 1). FFJORD also uses the optimize-then-discretize (OTD) approach and an adjoint-based backpropagation where the intermediate gradients are recomputed. In contrast, our exact trace computation is competitive with FFJORD's trace approach during training and faster during inference (Fig. 2). OT-Flow's use of DTO has been shown to converge quicker when training neural ODEs due to accurate gradient computation, storing intermediate gradients, and fewer time steps (Li et al. 2017; Gholaminejad, Keutzer, and Biros 2019; Onken and Ruthotto 2020) (Sec 3).

Flows Influenced by Optimal Transport To encourage straight trajectories, RNODE (Finlay et al. 2020) regularizes FFJORD with a transport cost $L(\boldsymbol{x}, T)$. RNODE also includes the Frobenius norm of the Jacobian $\|\nabla \mathbf{v}\|_F^2$ to stabilize training. They estimate the trace and the Frobenius norm using a stochastic estimator and report 2.8x speedup. Numerically, RNODE, FFJORD, and OT-Flow differ. Specifically, OT-Flow's exact trace allows for stable training without $\|\nabla \mathbf{v}\|_F^2$ (Fig. 3). In formulation, OT-Flow shares the L_2 cost with RNODE but follows a potential flow approach (Tab. 1).

Monge-Ampère Flows (Zhang, E, and Wang 2018) and Potential Flow Generators (Yang and Karniadakis 2020) similarly draw from OT theory but parameterize a potential function (Tab. 1). However, OT-Flow's numerics differ substantially due to our scalable exact trace computation (Tab. 1). OT is also used in other generative models (Sanjabi et al. 2018; Salimans et al. 2018; Lei et al. 2019; Lin, Lensink, and Haber 2019; Avraham, Zuo, and Drummond 2019; Tanaka 2019).

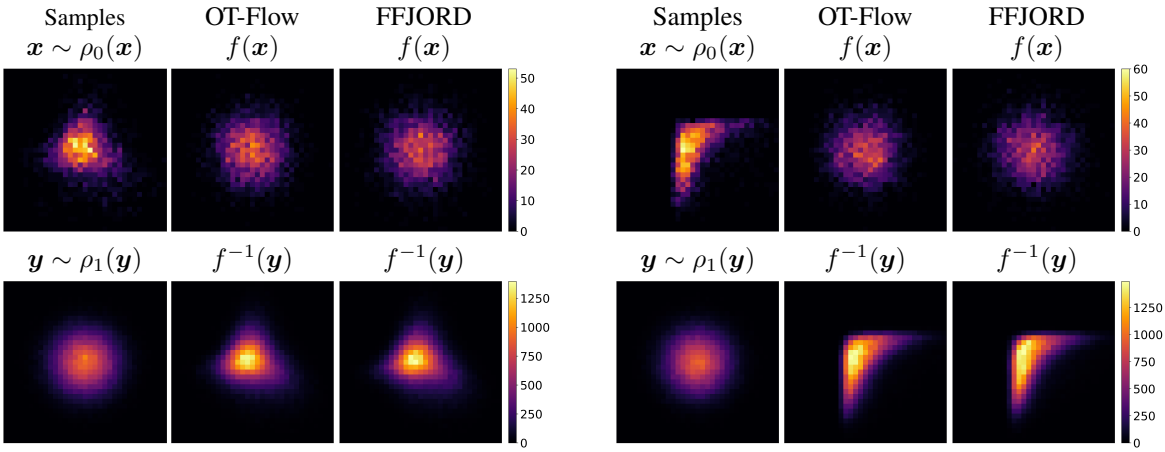
5 Numerical Experiments

We perform density estimation on seven two-dimensional toy problems and five high-dimensional problems from real data sets. We also show OT-Flow's generative abilities on MNIST.

Metrics In density estimation, the goal is to approximate ρ_0 using observed samples $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$, where $\boldsymbol{x}_i$ are drawn from the distribution ρ_0. In real applications, we lack a ground-truth ρ_0, rendering proper evaluation of the density itself untenable. However, we can follow evaluation techniques applied to generative models. Drawing random points $\{\boldsymbol{y}_i\}_{i=1}^M$ from ρ_1, we invert the flow to generate synthetic samples $\boldsymbol{Q} = \{\boldsymbol{q}_i\}_{i=1}^M$, where $\boldsymbol{q}_i = f^{-1}(\boldsymbol{y}_i)$. We compare the known samples to the generated samples via maximum mean discrepancy (MMD) (Gretton et al. 2012; Li, Swersky, and Zemel 2015; Theis, van den Oord, and Bethge 2016; Peyré and Cuturi 2019)

$$\begin{aligned} \text{MMD}(\boldsymbol{X}, \boldsymbol{Q}) &= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ +\frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} k(\boldsymbol{q}_i, \boldsymbol{q}_j) &- \frac{2}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} k(\boldsymbol{x}_i, \boldsymbol{q}_j), \end{aligned} \tag{16}$$

for Gaussian kernel $k(\boldsymbol{x}_i, \boldsymbol{q}_j) = \exp(-\frac{1}{2}\|\boldsymbol{x}_i - \boldsymbol{q}_j\|^2)$. MMD tests the difference between two distributions (ρ_0 and our estimate of ρ_0) on the basis of samples drawn from each ($\boldsymbol{X}$ and $\boldsymbol{Q}$). A low MMD value means that the two sets



(a) MINIBOONE dimension 16 vs 17

(b) MINIBOONE dimension 28 vs 29

Figure 5: MINIBOONE density estimation. Two-dimensional slices using the 3,648 43-dimensional testing samples $\boldsymbol{x} \sim \rho_0(\boldsymbol{x})$ and 10^5 samples $\boldsymbol{y}$ from distribution ρ_1 (more visuals in Fig. A7).

of samples are likely to have been drawn from the same distribution (Gretton et al. 2012). Since MMD is not used in the training, it provides an external, impartial metric to evaluate our model on the hold-out test set (Tab. 2).

Many normalizing flows use C for evaluation. The loss C is used to train the forward flow to match ρ_1. Testing loss, i.e., C evaluated on the testing set, should provide the same quantification on a hold-out set. However, in some cases, the testing loss can be low even when $f(\boldsymbol{x})$ is poor and differs substantially from ρ_1 (Fig. A2, A3). Furthermore, because the model's inverse contains error, accurately mapping to ρ_1 with the forward flow does not necessarily mean the inverse flow accurately maps to ρ_0.

Testing loss varies drastically with the integration computation (Theis, van den Oord, and Bethge 2016; Wehenkel and Louppe 2019; Onken and Ruthotto 2020). It depends on ℓ, which is computed along the characteristics via time integration of the trace (App. G). Too few discretization points leads to an inaccurate integration computation and greater inverse error. Thus, a low inverse error implies an accurate integration computation because the flow closely models the ODE. An adaptive ODE solver alleviates this concern when provided a sufficiently small tolerance (Grathwohl et al. 2019). Similarly, we check that the flow models the continuous solution of the ODE by computing the inverse error

$$\mathbb{E}_{\rho_0(\boldsymbol{x})} \| f^{-1}\left(f(\boldsymbol{x})\right) - \boldsymbol{x} \|_2 \tag{17}$$

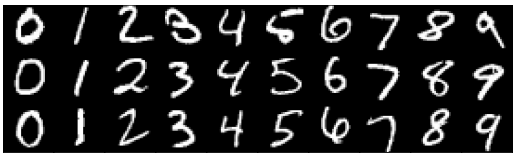
on the testing set using a finer time discretization than used in training. We evaluate the expectation values in (9) and (17) using the discrete samples $\boldsymbol{X}$, which we assume are randomly drawn from and representative of the initial distribution ρ_0.

Toy Problems We train OT-Flow on several toy distributions that serve as standard benchmarks (Grathwohl et al. 2019; Wehenkel and Louppe 2019). Given random samples, we train OT-Flow then use it to estimate the density ρ_0 and generate samples (Fig. 4). We present a thorough comparison with a state-of-the-art CNF on one of these (Fig. A2) .

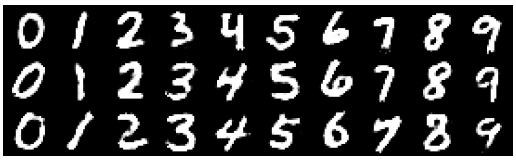
Density Estimation on Real Data Sets We compare our model's performance on real data sets (POWER, GAS, HEPMASS, MINIBOONE) from the University of California Irvine (UCI) machine learning data repository and the BSDS300 data set containing natural image patches. The UCI data sets describe observations from Fermilab neutrino experiments, household power consumption, chemical sensors of ethylene and carbon monoxide gas mixtures, and particle collisions in high energy physics. Prepared by Papamakarios, Pavlakou, and Murray (2017), the data sets are commonly used in normalizing flows (Dinh, Sohl-Dickstein, and Bengio 2017; Grathwohl et al. 2019; Huang et al. 2018; Wehenkel and Louppe 2019). The data sets vary in dimensionality (Tab. 2).

For each data set, we compare OT-Flow with FFJORD (Grathwohl et al. 2019) and RNODE (Finlay et al. 2020) (current state-of-the-art) in speed and performance. We compare speed both in training the models and when running the model on the testing set. To compare performance, we compute the MMD between the data set and $M{=}10^5$ generated samples $f^{-1}(\boldsymbol{y})$ for each model; for a fair comparison, we use the same $\boldsymbol{y}$ for FFJORD and OT-Flow (Tab. 2). We show visuals of the samples $\boldsymbol{x} \sim \rho_0(\boldsymbol{x})$, $\boldsymbol{y} \sim \rho_1(\boldsymbol{y})$, $f(\boldsymbol{x})$, and $f^{-1}(\boldsymbol{y})$ generated by OT-Flow and FFJORD (Fig. 5, App. H). We report the loss C values (Tab. 2) to be comparable to other literature but reiterate the inherent flaws in using C to compare models.

The results demonstrate the computational efficiency of OT-Flow relative to the state-of-the-art (Tab. 2). With the exception of the GAS data set, OT-Flow achieves comparable MMD to the state-of-the-art with drastically reduced training time. OT-Flow learns a slightly smoothed representation of the GAS data set (Fig. A5). We attribute most of the training speedup to the efficiency from using our exact trace instead of the Hutchinson's trace estimation (Fig. 2, Fig. 3). On the testing set, our exact trace leads to faster testing time than the state-of-the-art's exact trace computation via AD (Tab. 1,Tab. 2). To evaluate the testing data, we use more time



(a) Originals



(b) Generations

Figure 6: MNIST generation conditioned by class.

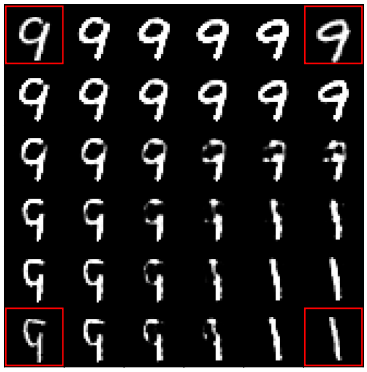


Figure 7: MNIST interpolation in the latent space. Original images are boxed in red.

steps than for training, effectively re-discretizing the ODE at different points. The inverse error shows that OT-Flow is numerically invertible and suggests that it approximates the true solution of the ODE. Ultimately, OT-Flow's combination of OT-influenced regularization, reduced parameterization, DTO approach, and efficient exact trace computation results in fast and accurate training and testing.

MNIST We demonstrate the generation quality of OT-Flow using an encoder-decoder structure. Consider encoder $B\colon \mathbb{R}^{784} \to \mathbb{R}^d$ and decoder $D\colon \mathbb{R}^d \to \mathbb{R}^{784}$ such that $D(B(\boldsymbol{x})) \approx \boldsymbol{x}$. We train d-dimensional flows that map distribution $\rho_0(B(\boldsymbol{x}))$ to ρ_1. The encoder and decoder each use a single dense layer and activation function (ReLU for B and sigmoid for D). We train the encoder-decoder separate from and prior to training the flows. The trained encoder-decoder, due to its simplicity, renders digits $D(B(\boldsymbol{x}))$ that are a couple pixels thicker than the supplied digit $\boldsymbol{x}$.

We generate new images via two methods. First, using d=64 and a flow conditioned on class, we sample a point $\boldsymbol{y} \sim \rho_1(\boldsymbol{y})$ and map it back to the pixel space to create image $D(f^{-1}(\boldsymbol{y}))$ (Fig. 6b). Second, using d=128 and an unconditioned flow, we interpolate between the latent representations $f(B(\boldsymbol{x}_1)), f(B(\boldsymbol{x}_2))$ of original images $\boldsymbol{x}_1, \boldsymbol{x}_2$. For interpolated latent vector $\boldsymbol{y} \in \mathbb{R}^d$, we invert the flow and decode back to the pixel space to create image $D(f^{-1}(\boldsymbol{y}))$ (Fig. 7).

6 Discussion

We present OT-Flow, a fast and accurate approach for training and performing inference with CNFs. Our approach tackles two critical computational challenges in CNFs.

First, solving the neural ODEs in CNFs can require many time steps resulting in high computational cost. Leveraging OT theory, we include a transport cost and add an HJB regularizer. These additions help carry properties from the continuous problem to the discrete problem and allow OT-Flow to use few time steps without sacrificing performance. Second, computing the trace term in (2) is computationally expensive. OT-Flow features exact trace computation at time complexity and cost comparable to trace estimators used in existing state-of-the-art CNFs (Fig. 2). The exact trace provides better convergence than the estimator (Fig. 3). Our analytic gradient and trace approach is not limited to the ResNet architectures, but expanding to other architectures requires further derivation.

Acknowledgments

This research was supported by the NSF award DMS 1751636, Binational Science Foundation Grant 2018209, AFOSR Grants 20RT0237 and FA9550-18-1-0167, AFOSR MURI FA9550-18-1-0502, ONR Grant No. N00014-18-1-2527, a gift from UnitedHealth Group R&D, and a GPU donation by NVIDIA Corporation. Important parts of this research were performed while LR was visiting the Institute for Pure and Applied Mathematics (IPAM), which is supported by the NSF Grant No. DMS 1440415.

References

Abraham, F.; Behr, M.; and Heinkenschloss, M. 2004. The Effect of Stabilization in Finite Element Methods for the Optimal Boundary Control of the Oseen Equations. *Finite Elements in Analysis and Design* 41(3): 229–251.

Ambrosio, L.; Gigli, N.; and Savaré, G. 2008. *Gradient Flows: In Metric Spaces and in the Space of Probability Measures*. Springer Science & Business Media.

Ascher, U. M. 2008. *Numerical Methods for Evolutionary Differential Equations*, volume 5. SIAM.

Avraham, G.; Zuo, Y.; and Drummond, T. 2019. Parallel Optimal Transport GAN. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4406–4415.

Avron, H.; and Toledo, S. 2011. Randomized Algorithms for Estimating the Trace of an Implicit Symmetric Positive Semi-Definite Matrix. *Journal of the ACM* 58(2): 1–34.

Becker, R.; and Vexler, B. 2007. Optimal Control of the Convection-Diffusion Equation using Stabilized Finite Element Methods. *Numerische Mathematik* 106(3): 349–367.

Benamou, J.-D.; and Brenier, Y. 2000. A Computational Fluid Mechanics Solution to the Monge-Kantorovich Mass Transfer Problem. *Numerische Mathematik* 84(3): 375–393.

Benamou, J.-D.; Carlier, G.; and Santambrogio, F. 2017. Variational Mean Field Games. In *Active Particles. Vol. 1. Advances in Theory, Models, and Applications*, Model. Simul. Sci. Eng. Technol., 141–171. Birkhäuser/Springer, Cham.

Brehmer, J.; Kling, F.; Espejo, I.; and Cranmer, K. 2020. MadMiner: Machine Learning-Based Inference for Particle Physics. *Computing and Software for Big Science* 4(1): 1–25.

Chen, C.; Li, C.; Chen, L.; Wang, W.; Pu, Y.; and Duke, L. C. 2018a. Continuous-Time Flows for Efficient Inference and Density Estimation. In *International Conference on Machine Learning (ICML)*, 824–833.

Chen, T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018b. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 6571–6583.

Collis, S. S.; and Heinkenschloss, M. 2002. Analysis of the Streamline Upwind/Petrov Galerkin Method Applied to the Solution of Optimal Control Problems. *CAAM TR02-01* 108.

Dinh, L.; Krueger, D.; and Bengio, Y. 2015. NICE: Nonlinear Independent Components Estimation. In Bengio, Y.; and LeCun, Y., eds., *International Conference on Learning Representations (ICLR)*.

Dinh, L.; Sohl-Dickstein, J.; and Bengio, S. 2017. Density Estimation using Real NVP. In *International Conference on Learning Representations (ICLR)*.

Durkan, C.; Bekasov, A.; Murray, I.; and Papamakarios, G. 2019. Neural Spline Flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 7509–7520.

Evans, L. C. 1997. Partial Differential Equations and Monge-Kantorovich Mass Transfer. *Current developments in mathematics* 1997(1): 65–126.

Evans, L. C. 2010. *Partial Differential Equations*, volume 19. American Mathematical Society.

Evans, L. C. 2013. An Introduction to Mathematical Optimal Control Theory Version 0.2.

Finlay, C.; Jacobsen, J.-H.; Nurbekyan, L.; and Oberman, A. M. 2020. How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization. In *International Conference on Machine Learning (ICML)*, 3154–3164.

Gangbo, W.; and McCann, R. J. 1996. The Geometry of Optimal Transportation. *Acta Mathematica* 177(2): 113–161.

Germain, M.; Gregor, K.; Murray, I.; and Larochelle, H. 2015. MADE: Masked Autoencoder for Distribution Estimation. In *International Conference on Machine Learning (ICML)*, 881–889.

Gholaminejad, A.; Keutzer, K.; and Biros, G. 2019. ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 730–736.

Grathwohl, W.; Chen, R. T.; Betterncourt, J.; Sutskever, I.; and Duvenaud, D. 2019. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models.

Gretton, A.; Borgwardt, K. M.; Rasch, M. J.; Schölkopf, B.; and Smola, A. 2012. A Kernel Two-Sample Test. *Journal of Machine Learning Research (JMLR)* 13(25): 723–773.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Huang, C.-W.; Krueger, D.; Lacoste, A.; and Courville, A. 2018. Neural Autoregressive Flows. In *International Conference on Machine Learning (ICML)*, 2078–2087.

Hutchinson, M. F. 1990. A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines. *Communications in Statistics-Simulation and Computation* 19(2): 433–450.

Ingraham, J.; Riesselman, A.; Sander, C.; and Marks, D. 2019. Learning Protein Structure with a Differentiable Simulator. In *International Conference on Learning Representations (ICLR)*.

Kingma, D. P.; and Dhariwal, P. 2018. Glow: Generative Flow with Invertible 1x1 Convolutions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 10215–10224.

Kingma, D. P.; Salimans, T.; Jozefowicz, R.; Chen, X.; Sutskever, I.; and Welling, M. 2016. Improved Variational Inference with Inverse Autoregressive Flow. In *Advances in Neural Information Processing Systems (NeurIPS)*, 4743–4751.

Kobyzev, I.; Prince, S.; and Brubaker, M. 2020. Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* .

Lei, N.; Su, K.; Cui, L.; Yau, S.-T.; and Gu, X. D. 2019. A Geometric View of Optimal Transportation and Generative Model. *Computer Aided Geometric Design* 68: 1–21.

Leugering, G.; Benner, P.; Engell, S.; Griewank, A.; Harbrecht, H.; Hinze, M.; Rannacher, R.; and Ulbrich, S. 2014. *Trends in PDE Constrained Optimization*, volume 165. Springer.

Li, Q.; Chen, L.; Tai, C.; and Weinan, E. 2017. Maximum Principle Based Algorithms for Deep Learning. *The Journal of Machine Learning Research (JMLR)* 18(1): 5998–6026.

Li, Y.; Swersky, K.; and Zemel, R. 2015. Generative Moment Matching Networks. In *International Conference on Machine Learning (ICML)*, 1718–1727.

Lin, A. T.; Fung, S. W.; Li, W.; Nurbekyan, L.; and Osher, S. J. 2020. APAC-Net: Alternating the Population and Agent Control via Two Neural Networks to Solve High-Dimensional Stochastic Mean Field Games. *arXiv:2002.10113* .

Lin, J.; Lensink, K.; and Haber, E. 2019. Fluid Flow Mass Transport for Generative Networks. *arXiv:1910.01694* .

Neal, R. M. 2011. MCMC using Hamiltonian Dynamics. *Handbook of Markov Chain Monte Carlo* 2(11): 2.

Noé, F.; Olsson, S.; Köhler, J.; and Wu, H. 2019. Boltzmann Generators: Sampling Equilibrium States of Many-Body Systems with Deep Learning. *Science* 365(6457).

Onken, D.; and Ruthotto, L. 2020. Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows. *arXiv:2005.13420* .

Papamakarios, G.; Nalisnick, E.; Rezende, D. J.; Mohamed, S.; and Lakshminarayanan, B. 2019. Normalizing Flows for Probabilistic Modeling and Inference. *arXiv:1912.02762* .

Papamakarios, G.; Pavlakou, T.; and Murray, I. 2017. Masked Autoregressive Flow for Density Estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2338–2347.

Peyré, G.; and Cuturi, M. 2019. Computational Optimal Transport. *Foundations and Trends in Machine Learning* 11(5-6): 355–607.

Rezende, D. J.; and Mohamed, S. 2015. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning (ICML)*, 1530–1538.

Ruthotto, L.; Osher, S. J.; Li, W.; Nurbekyan, L.; and Fung, S. W. 2020. A Machine Learning Framework for Solving High-Dimensional Mean Field Game and Mean Field Control Problems. *Proceedings of the National Academy of Sciences (PNAS)* 117(17): 9183–9193.

Salimans, T.; Kingma, D.; and Welling, M. 2015. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. In *International Conference on Machine Learning (ICML)*, 1218–1226.

Salimans, T.; Zhang, H.; Radford, A.; and Metaxas, D. N. 2018. Improving GANs Using Optimal Transport. In *International Conference on Learning Representations (ICLR)*.

Sanjabi, M.; Ba, J.; Razaviyayn, M.; and Lee, J. D. 2018. On the Convergence and Robustness of Training GANs with Regularized Optimal Transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 7091–7101.

Suykens, J.; Verrelst, H.; and Vandewalle, J. 1998. On-Line Learning Fokker-Planck Machine. *Neural Processing Letters* 7: 81–89.

Tabak, E. G.; and Turner, C. V. 2013. A Family of Nonparametric Density Estimation Algorithms. *Communications on Pure and Applied Mathematics* 66(2): 145–164.

Tanaka, A. 2019. Discriminator Optimal Transport. In *Advances in Neural Information Processing Systems (NeurIPS)*, 6816–6826.

Theis, L.; van den Oord, A.; and Bethge, M. 2016. A Note on the Evaluation of Generative Models. In *International Conference on Learning Representations (ICLR)*.

Ubaru, S.; Chen, J.; and Saad, Y. 2017. Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature. *SIAM Journal on Matrix Analysis and Applications* 38(4): 1075–1099.

Villani, C. 2008. *Optimal Transport: Old and New*, volume 338. Springer Science & Business Media.

Wehenkel, A.; and Louppe, G. 2019. Unconstrained Monotonic Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1543–1553.

Welling, M.; and Teh, Y. W. 2011. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *International Conference on Machine Learning (ICML)*, 681–688.

Yang, L.; and Karniadakis, G. E. 2020. Potential Flow Generator With L_2 Optimal Transport Regularity for Generative Models. *IEEE Transactions on Neural Networks and Learning Systems* .

Zhang, L.; E, W.; and Wang, L. 2018. Monge-Ampère Flow for Generative Modeling. *arXiv:1809.10188* .