Discretize-Optimize Methods for Neural ODEs in Continuous Normalizing Flows

SIAM MDS 2020

Derek Onken
Department of Computer Science
Emory University
donken@emory.edu
derekonken.com

Collaborators and Acknowledgments

Samy Wu Fung
UCLA

Xingjian Li
Emory

Lars Ruthotto
Emory

Emory Funding: DMS 1751636 US-Israel BSF 2018209

UCLA Funding: AFOSR MURI FA9550-18-1-0502 and FA9550-18-1-0167, ONR N00014-18-1-2527

Special thanks: Organizers and staff of IPAM Long Program MLP 2019.

Overview

- **Continuous ResNet and Neural ODEs**
  - Discrete neural networks viewed in continuous framework
- **Continuous Normalizing Flows (CNFs)**
  - Discrete normalizing flows similarly moved to the continuous framework
- **Discretize-Optimize vs Optimize-Discretize**
  - Comparing approaches in solving CNFs
- **OT-Flow**
  - Incorporating optimal transport with Discretize-Optimize for fast and accurate CNFs

Motivation: Existing CNFs approaches are prohibitively slow and expensive.

DO and L Ruthotto
Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and CNFs
arXiv:2005.13420, 2020.

DO, S Wu Fung, X Li, L Ruthotto
OT-Flow: Fast and Accurate CNFs via OT
arXiv:2006.00104, 2020.

Neural ODE

A *neural ODE* is an ordinary differential equation (ODE) with neural network components.

For input $\boldsymbol{x} \in \mathbb{R}^d$, neural network $f\colon \mathbb{R}^d \to \mathbb{R}^d$ models the solution to

$$\partial_t \boldsymbol{z}(\boldsymbol{x}, t) = \mathbf{v}\big(\boldsymbol{z}(\boldsymbol{x}, t), t; \boldsymbol{\theta}\big), \quad \boldsymbol{z}(\boldsymbol{x}, 0) = \boldsymbol{x} \tag{1}$$

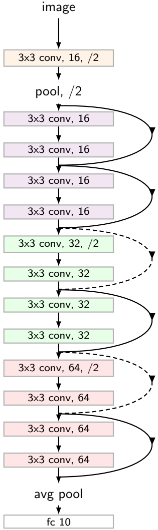
where

- time $t \in [0, T]$
- $\mathbf{v}\colon \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$ is a neural network layer with parameters $\boldsymbol{\theta}$
- $\boldsymbol{z}(\boldsymbol{x}, t)$ are the features for initial $\boldsymbol{x}$ at time t
- $f(\boldsymbol{x}) = \boldsymbol{z}(\boldsymbol{x}, T)$

Background

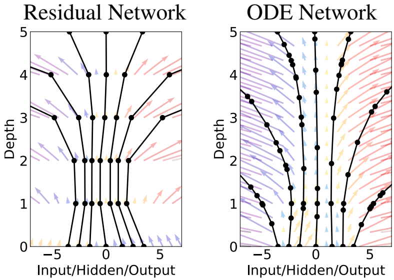
Historically

- Residual Neural Networks (ResNets)[1] perform well on image classification and more.
- ResNets are merely Forward Euler of some Continuous ResNet (1).[2,3]



[1] He et al. "Deep residual learning for image recognition". 2016.
[2] E. "A Proposal on Machine Learning via Dynamical Systems". 2017.
[3] Haber and Ruthotto. "Stable Architectures for Deep Neural Networks". 2017.

Background (cont.)



Neural ODE[4]

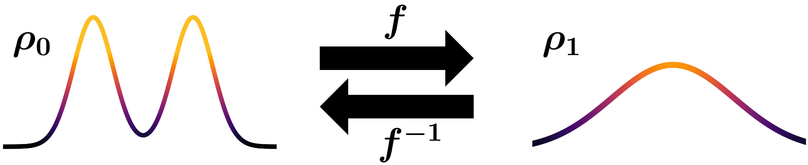
Popularized

- Incorporate a black-box solver and coin the term *neural ODE*.[4]
- Applied to normalizing flows.[5]

[4] Chen et al. "Neural Ordinary Differential Equations". 2018.
[5] Grathwohl et al. "FFJORD: Free-form continuous dynamics for scalable reversible generative models". 2019.

Discrete Normalizing Flows

A normalizing flow[6,7] 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.

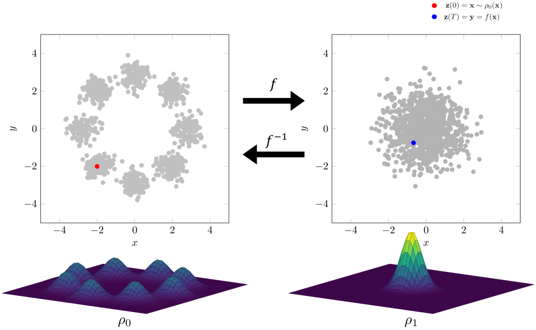


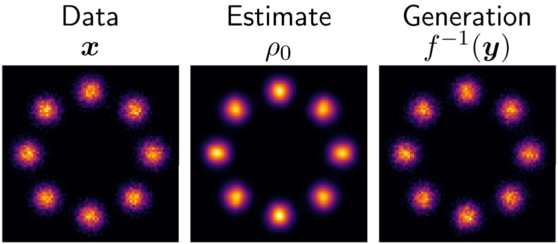
By the change of variables formula, the flow satisfies

$$\log \rho_0(\boldsymbol{x}) = \log \rho_1(f(\boldsymbol{x})) + \log |\det \nabla f(\boldsymbol{x})| \quad \text{for all} \quad \boldsymbol{x} \in \mathbb{R}^d. \tag{2}$$

[6]Rezende and Mohamed. "Variational Inference with Normalizing Flows". 2015.
[7]Papamakarios et al. "Normalizing Flows for Probabilistic Modeling and Inference". 2019.

Gaussian Mixture Toy Example





Continuous Normalizing Flows (CNFs)

Replace the log-det with a trace

Issue:

- log-determinants cost $\mathcal{O}(d^3)$ FLOPS in general.

Solutions:

- Use specific neural network architectures for $\mathbf{v}$ so the log-det computation is manageable.
- Replace the log-det with a trace computation.

Using the neural ODE f (1) and Jacobi's formula[8], we can rewrite (2) as

$$\ell(\boldsymbol{x}, T) := \log \rho_0(\boldsymbol{x}) - \log \rho_1(f(\boldsymbol{x})) = \int_0^T \mathrm{tr}\left(\nabla \mathbf{v}(\boldsymbol{z}(\boldsymbol{x}, t), t; \boldsymbol{\theta})\right) \mathrm{d}t. \quad (3)$$

[8] Chen et al. 2018.

CNF Optimization Problem

For expected negative log-likelihood [9,10]

$$C(\boldsymbol{x},T) := \frac{1}{2}\|\boldsymbol{z}(\boldsymbol{x},T)\|^2 - \ell(\boldsymbol{x},T) + \frac{d}{2}\log(2\pi),$$

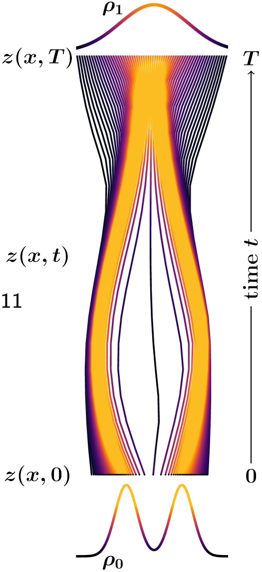
we optimize

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{\rho_0(\boldsymbol{x})}\, C(\boldsymbol{x},T)$$

where for a given $\boldsymbol{\theta}$, the trajectory $\boldsymbol{z}$ satisfies the CNF [11]

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

$$\begin{bmatrix} \boldsymbol{z}(\boldsymbol{x},0) \\ \ell(\boldsymbol{x},0) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix}.$$



In optimal control, $\boldsymbol{z}$ is the state and $\boldsymbol{\theta}$ is the control.

[9] Rezende and Mohamed. 2015.
[10] Papamakarios et al. 2019.
[11] Grathwohl et al. 2019.

Solving ODE-constrained Optimization Problems

Two Predominant Approaches:

- Discretize-Optimize (DO)
  - Discretize the ODE, then optimize on that discretization.
  - Typical machine learning approach: set up architecture with N layers, the optimize on that discretization (propagate forward, calculate loss, backpropagate)
  - ANODE[12]
- Optimize-Discretize (OD)
  - Optimize in the continuous space, then discretize.
  - Use the Karush-Kuhn-Tucker (KKT) conditions or the adjoint equations to optimize, then choose a discretization.
  - Neural ODEs paper[13] and FFJORD[14]

[12] Gholaminejad, Keutzer, and Biros. "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". 2019.

[13] Chen et al. 2018.

[14] Grathwohl et al. 2019.

Solving ODE-constrained Optimization Problems

Popular methods use Optimize-Discretize.

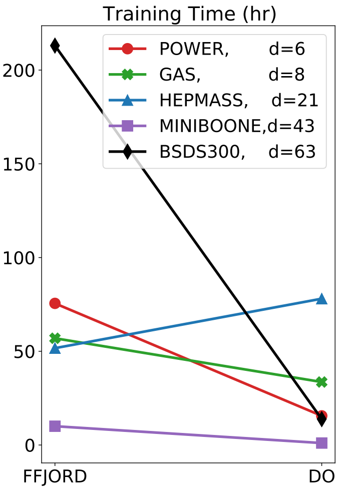
- Adaptive solver `dopri5` for the forward propagation
- Adjoint-based backpropagation recomputes the intermediate gradients
- **Drawback:** inaccurate gradients when the adjoint equation is not solved well enough.[15,16]

We choose Discretize-Optimize.

- Same discretization for the forward and backpropagation.
  - Use automatic differentiation (AD) for the backpropagation.
  - The gradients are accurate.
- We use Runge-Kutta 4 with a fixed step size.
- **Drawback:** have to tune a sufficiently small step size for the solver

[15] Li et al. "Maximum principle based algorithms for deep learning". 2017.
[16] Gholaminejad, Keutzer, and Biros. 2019.

Results



For all five data sets, DO and FFJORD[17] (OD) achieve similar results with different training time.

DO has an average speed-up of 6.4x even with slower training on HEPMASS.

Reasons

- Fewer function evaluations (RK4 instead of `dopri5`).
- Intermediate gradients are stored (with AD) rather than recomputed.
- DO has more accurate gradients.

DO and L Ruthotto
Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and CNFs
arXiv:2005.13420, 2020.

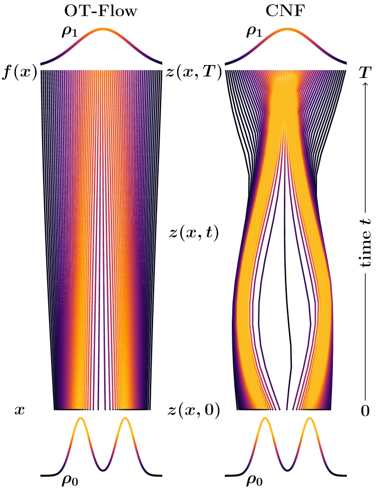
[17] Grathwohl et al. 2019.

Can we solve even faster?

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

What makes CNFs slow?

- Trajectories can be complicated leading to high number of function evaluations.
- Trace computation costs $\mathcal{O}(d^2)$ FLOPS in general.
  - FFJORD uses Hutchinson's estimator for $\mathcal{O}(d)$ FLOPS in training.



Straight Trajectories

Include some optimal transport (OT)

In OT, a unique mapping exists.

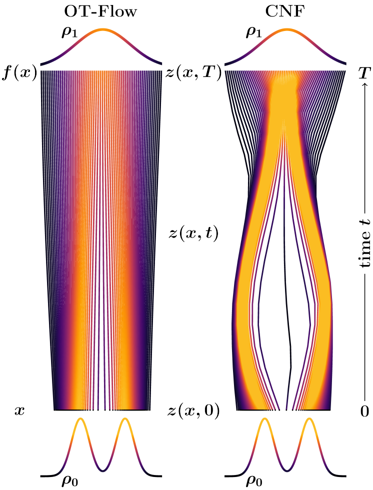
We regularize the optimization problem

$$\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) \right\} \quad (4)$$

subject to (1).

The L_2 transport costs are given by

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



More OT

Potential Function Φ

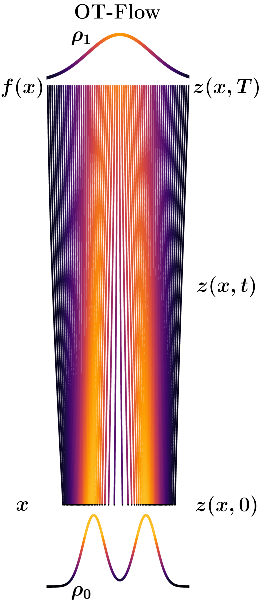
Apply the Pontryagin maximum principle[18] to (4)

There exists a scalar potential function $\Phi \colon \mathbb{R}^d \times [0, T] \to \mathbb{R}$ such that

$$\mathbf{v}(\boldsymbol{x}, t; \boldsymbol{\theta}) = -\nabla \Phi(\boldsymbol{x}, t; \boldsymbol{\theta}). \tag{5}$$

Analogous to classical physics, samples move in a manner to minimize their potential.

We parametrize potential Φ instead of $\mathbf{v}$.



[18] Evans. *An Introduction to Mathematical Optimal Control Theory Version 0.2*. 2013.

More OT

HJB Equation

The optimality conditions of (4) lead to another regularizer.

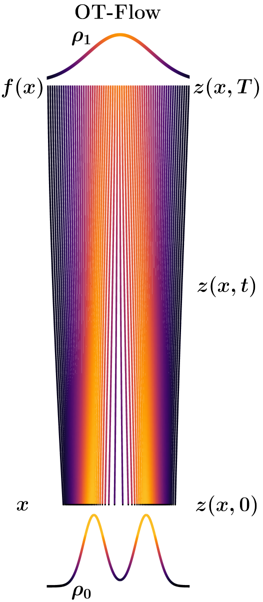
Potential Φ satisfies the Hamilton-Jacobi-Bellman (HJB) equation [19]

$$
\begin{aligned}
-\partial_t \Phi(\boldsymbol{x}, t) + \frac{1}{2}\|\nabla \Phi(\boldsymbol{z}(\boldsymbol{x}, t), t)\|^2 = 0, \\
\Phi(\boldsymbol{x}, T) = G(\boldsymbol{x})
\end{aligned}
\tag{6}
$$

where

$$
\begin{aligned}
& G(\boldsymbol{z}(\boldsymbol{x}, T)) = \\
& 1 + \log\big(\rho_0(\boldsymbol{x})\big) - \log\big(\rho_1(\boldsymbol{z}(\boldsymbol{x}, T))\big) - \ell(\boldsymbol{x}, T)
\end{aligned}
\tag{7}
$$

Terminal condition G derives from the variational derivative or the Kullback-Leibler (KL) divergence.



[19] Evans. "Partial differential equations and Monge-Kantorovich mass transfer". 1997.

More OT

HJB regularizer R

Penalize deviations from the HJB equation

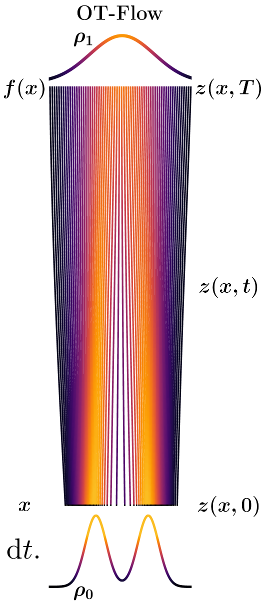
We add another regularizer, so the optimization problem is

$$\min_{\boldsymbol{\theta}} \ \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ C(\boldsymbol{x}, T) + L(\boldsymbol{x}, T) + R(\boldsymbol{x}, T) \right\}$$

subject to (1).

The HJB regularizer is computed as

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



OT-Flow Formulation

We incorporate the time integration in the ODE solver.

$$\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{\rho_0(\boldsymbol{x})} \left\{ C(\boldsymbol{x},T) + L(\boldsymbol{x},T) + R(\boldsymbol{x},T) \right\}$$

subject to

$$\partial_t \begin{pmatrix} \boldsymbol{z}(\boldsymbol{x},t) \\ \ell(\boldsymbol{x},t) \\ L(\boldsymbol{x},t) \\ R(\boldsymbol{x},t) \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta}) \\ -\operatorname{tr}(\nabla^2\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})) \\ \frac{1}{2}\|\nabla\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})\|^2 \\ \left|\partial_t\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta}) - \frac{1}{2}\|\nabla\Phi(\boldsymbol{z}(\boldsymbol{x},t),t;\boldsymbol{\theta})\|^2\right| \end{pmatrix}$$

with initial conditions

$$\boldsymbol{z}(\boldsymbol{x},0) = \boldsymbol{x} \quad \text{and} \quad \ell(\boldsymbol{x},0) = L(\boldsymbol{x},0) = R(\boldsymbol{x},0) = 0$$

Other OT approaches in CNFs

Table: Comparison of flow formulations.

Model	ODE (1)	Φ	L_2 cost	HJB reg.	$\|\nabla \mathbf{v}\|_F^2$
FFJORD[20]	✓	✗	✗	✗	✗
RNODE[21]	✓	✗	✓	✗	✓
Monge-Ampère Flows[22]	✓	✓	✗	✗	✗
Potential Flow Gen.[23]	✓	✓	✗	✓	✗
OT-Flow	✓	✓	✓	✓	✗

[20] Grathwohl et al. 2019.
[21] Finlay et al. "How to train your neural ODE". 2020.
[22] Zhang, E, and Wang. "Monge-Ampère Flow for Generative Modeling". 2018.
[23] Yang and Karniadakis. "Potential Flow Generator with L_2 Optimal Transport Regularity for Generative Models". 2019.

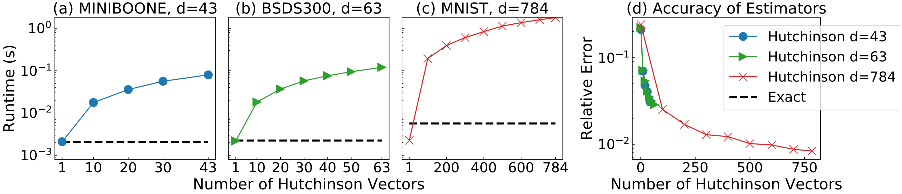
Improving the Trace Computation

General Trace Computation: $\mathcal{O}(d^2)$ FLOPS

Trace Estimators used in state-of-the-art: $\mathcal{O}(d)$ FLOPS

Our Exact Trace in OT-Flow $\mathcal{O}(d)$ FLOPS

In runtime, our exact trace is competitive with the estimators



Exact Trace Computation

Our model

Neural Network

$$\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \boldsymbol{w}^\top N(\boldsymbol{s};\boldsymbol{\theta}_N) + \frac{1}{2}\boldsymbol{s}^\top(\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b}^\top\boldsymbol{s} + c, \qquad \text{where} \quad \boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{\theta}_N, \boldsymbol{A}, \boldsymbol{b}, c) \tag{8}$$

Gradient

$$\nabla_{\boldsymbol{s}}\Phi(\boldsymbol{s};\boldsymbol{\theta}) = \nabla_{\boldsymbol{s}} N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w} + (\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{s} + \boldsymbol{b} \tag{9}$$

where

- space-time inputs $\boldsymbol{s} = (\boldsymbol{x}, t) \in \mathbb{R}^{d+1}$
- neural network $N(\boldsymbol{s};\boldsymbol{\theta}_N)\colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ (we choose ResNet)
- $\boldsymbol{\theta}$ consists of all the trainable weights:
  $\boldsymbol{w} \in \mathbb{R}^m$, $\boldsymbol{\theta}_N \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{r\times(d+1)}$, $\boldsymbol{b} \in \mathbb{R}^{d+1}$, $c \in \mathbb{R}$
  where $r = \min(10, d)$

Exact Trace Computation

Analytic Gradient Computation

N is an $(M+1)$-layer ResNet

Forward propagation

Compute $N(\boldsymbol{s};\boldsymbol{\theta}_N) = \boldsymbol{u}_M$.

$$
\begin{aligned}
\boldsymbol{u}_0 &= \sigma(\boldsymbol{K}_0 \boldsymbol{s} + \boldsymbol{b}_0) \\
\boldsymbol{u}_1 &= \boldsymbol{u}_0 + h\sigma(\boldsymbol{K}_1 \boldsymbol{u}_0 + \boldsymbol{b}_1) \\
\vdots \quad & \quad \quad \vdots \\
\boldsymbol{u}_M &= \boldsymbol{u}_{M-1} + h\sigma(\boldsymbol{K}_M \boldsymbol{u}_{M-1} + \boldsymbol{b}_M)
\end{aligned}
$$

where

- fixed step size $h > 0$
- ResNet weights $\boldsymbol{\theta}_N$ are
  - $\boldsymbol{K}_0 \in \mathbb{R}^{m \times (d+1)}$
  - $\boldsymbol{K}_1, \ldots, \boldsymbol{K}_M \in \mathbb{R}^{m \times m}$
  - $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_M \in \mathbb{R}^m$
- $\sigma(\boldsymbol{x}) = \log(\exp(\boldsymbol{x}) + \exp(-\boldsymbol{x}))$
  - the antiderivative of hyperbolic tangent
  - so, $\sigma'(\boldsymbol{x}) = \tanh(\boldsymbol{x})$

N is an $(M+1)$-layer ResNet

Forward propagation

Compute $N(\boldsymbol{s};\boldsymbol{\theta}_N) = \boldsymbol{u}_M$.

$$
\begin{aligned}
\boldsymbol{u}_0 &= \sigma(\boldsymbol{K}_0 \boldsymbol{s} + \boldsymbol{b}_0) \\
\boldsymbol{u}_1 &= \boldsymbol{u}_0 + h\sigma(\boldsymbol{K}_1 \boldsymbol{u}_0 + \boldsymbol{b}_1) \\
\vdots \quad & \quad \vdots \\
\boldsymbol{u}_M &= \boldsymbol{u}_{M-1} + h\sigma(\boldsymbol{K}_M \boldsymbol{u}_{M-1} + \boldsymbol{b}_M)
\end{aligned}
$$

Backpropagation

Compute $\nabla_{\boldsymbol{s}} N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w} = \boldsymbol{z}_0$

$$
\begin{aligned}
\boldsymbol{z}_{M+1} &= \boldsymbol{w} \\
\boldsymbol{z}_M &= \boldsymbol{z}_{M+1} \\
&\quad + h\boldsymbol{K}_M^\top \operatorname{diag}\big(\sigma'(\boldsymbol{K}_M \boldsymbol{u}_{M-1} + \boldsymbol{b}_M)\big)\boldsymbol{z}_{M+1} \\
\vdots \quad & \quad \vdots \\
\boldsymbol{z}_1 &= \boldsymbol{z}_2 + h\,\boldsymbol{K}_1^\top \operatorname{diag}\big(\sigma'(\boldsymbol{K}_1 \boldsymbol{u}_0 + \boldsymbol{b}_1)\big)\boldsymbol{z}_2 \\
\boldsymbol{z}_0 &= \boldsymbol{K}_0^\top \operatorname{diag}\big(\sigma'(\boldsymbol{K}_0 \boldsymbol{s} + \boldsymbol{b}_0)\big)\boldsymbol{z}_1
\end{aligned}
$$

Exact Trace Computation

Laplacian of the Potential

$$\mathrm{tr}\left(\nabla^2\Phi(\boldsymbol{s};\boldsymbol{\theta})\right) = \mathrm{tr}\left(\boldsymbol{E}^\top\left(\nabla^2_{\boldsymbol{s}}(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w}) + \boldsymbol{A}^\top\boldsymbol{A}\right)\boldsymbol{E}\right) \quad \text{for} \quad \boldsymbol{E} = \texttt{eye(d+1,d)}$$

Exact Trace Computation

Laplacian of the Potential

$$\mathrm{tr}\left(\nabla^2\Phi(\boldsymbol{s};\boldsymbol{\theta})\right) = \mathrm{tr}\left(\boldsymbol{E}^\top\left(\nabla^2_{\boldsymbol{s}}(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w}) + \boldsymbol{A}^\top\boldsymbol{A}\right)\boldsymbol{E}\right) \quad \text{for} \quad \boldsymbol{E} = \texttt{eye(d+1,d)}$$

Focus on the nontrivial part (the ResNet)

$$\mathrm{tr}\left(\boldsymbol{E}^\top\,\nabla^2_{\boldsymbol{s}}(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w})\,\boldsymbol{E}\right) = t_0 + h\sum_{i=1}^{M} t_i,$$

Exact Trace Computation

Laplacian of the Potential

$$\mathrm{tr}\left(\nabla^2\Phi(\boldsymbol{s};\boldsymbol{\theta})\right) = \mathrm{tr}\left(\boldsymbol{E}^\top\left(\nabla^2_{\boldsymbol{s}}(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w}) + \boldsymbol{A}^\top\boldsymbol{A}\right)\boldsymbol{E}\right) \quad \text{for} \quad \boldsymbol{E} = \texttt{eye(d+1,d)}$$

Focus on the nontrivial part (the ResNet)

$$\mathrm{tr}\left(\boldsymbol{E}^\top\,\nabla^2_{\boldsymbol{s}}(N(\boldsymbol{s};\boldsymbol{\theta}_N)\boldsymbol{w})\,\boldsymbol{E}\right) = t_0 + h\sum_{i=1}^{M} t_i,$$

With one pass, we calculate the trace of each layer

$$\begin{aligned}
t_0 &= \mathrm{tr}\left(J_{i-1}^\top\nabla_{\boldsymbol{s}}\left(\boldsymbol{K}_i^\top\mathrm{diag}(\sigma''(\boldsymbol{K}_i\boldsymbol{u}_{i-1}(\boldsymbol{s})+\boldsymbol{b}_i))\boldsymbol{z}_{i+1}\right)J_{i-1}\right)\\
&= \left(\sigma''(\boldsymbol{K}_0\boldsymbol{s}+\boldsymbol{b}_0)\odot\boldsymbol{z}_1\right)^\top\left((\boldsymbol{K}_0\boldsymbol{E})\odot(\boldsymbol{K}_0\boldsymbol{E})\right)\mathbf{1} \quad \text{and}\\
t_i &= \mathrm{tr}\left(J_{i-1}^\top\nabla_{\boldsymbol{s}}\left(\boldsymbol{K}_i^\top\mathrm{diag}(\sigma''(\boldsymbol{K}_i\boldsymbol{u}_{i-1}(\boldsymbol{s})+\boldsymbol{b}_i))\boldsymbol{z}_{i+1}\right)J_{i-1}\right)\\
&= \left(\sigma''(\boldsymbol{K}_i\boldsymbol{u}_{i-1}+\boldsymbol{b}_i)\odot\boldsymbol{z}_{i+1}\right)^\top\left((\boldsymbol{K}_iJ_{i-1})\odot(\boldsymbol{K}_iJ_{i-1})\right)\mathbf{1}.
\end{aligned}$$

where Jacobian $J_{i-1} = \nabla_{\boldsymbol{s}}\boldsymbol{u}_{i-1}^\top \in \mathbb{R}^{m\times d}$, $\odot$ denotes Hadamard product, and $\mathbf{1}$=`ones(d,1)`.

Exact Trace Computation

Efficiency

Update and overwrite $J_{i-1} = \nabla_{\boldsymbol{s}} \boldsymbol{u}_{i-1}^\top \in \mathbb{R}^{m \times d}$ in the forward pass as

$$\nabla_{\boldsymbol{s}} \boldsymbol{u}_i^\top = \nabla_{\boldsymbol{s}} \boldsymbol{u}_{i-1} + h\, \sigma'(\boldsymbol{K}_i \boldsymbol{u}_{i-1} + \boldsymbol{b}_i) \boldsymbol{K}_i^\top \nabla_{\boldsymbol{s}} \boldsymbol{u}_{i-1}$$

$$J \leftarrow J + h\, \sigma'(\boldsymbol{K}_i \boldsymbol{u}_{i-1} + \boldsymbol{b}_i)\, \boldsymbol{K}_i^\top J$$

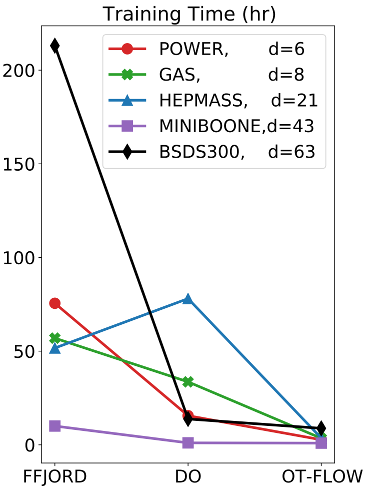
Overall Cost is $\mathcal{O}(m^2 \cdot d)$ FLOPS.

Recall: $\boldsymbol{K}_0 \in \mathbb{R}^{m \times (d+1)}$ and $\boldsymbol{K}_1, \ldots, \boldsymbol{K}_M \in \mathbb{R}^{m \times m}$

L Ruthotto, S Osher, W Li, L Nurbekyan, S Wu Fung
A ML Framework for Solving High-Dimensional MFG and MFC
PNAS 117 (17), 9183-9193, 2020.

Results

Fast Training



OT-Flow has 19x training speed-up on average

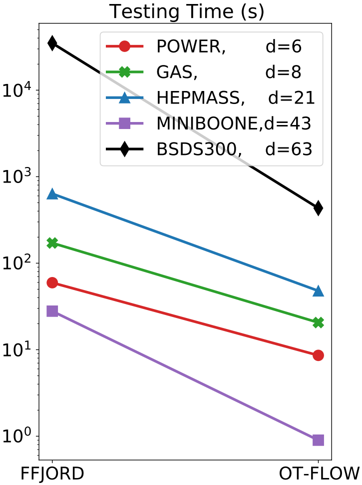
Reasons

- OT-inspired regularization leads to straight trajectories that are inexpensive to integrate.
- The trace computation is efficient and exact.
- The potential flows approach results in fewer weights and a smaller model.

DO, S Wu Fung, X Li, L Ruthotto
OT-Flow: Fast and Accurate CNFs via OT
arXiv:2006.00104, 2020.

Results

Fast Inference



OT-Flow has 28x testing speed-up on average

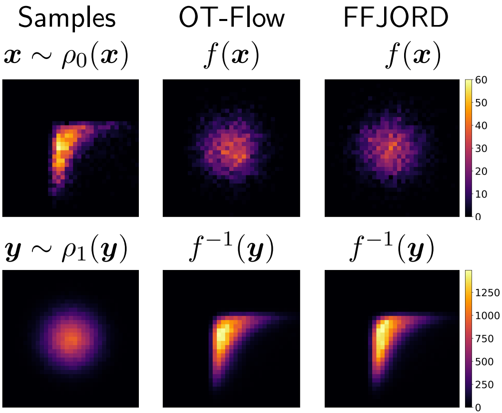
Inference-Specific Reasons

- Inference uses exact trace (no estimates)
  - State-of-the-art approaches use AD to obtain exact trace with $\mathcal{O}(d^2)$
  - Meanwhile, our exact trace is $\mathcal{O}(d)$

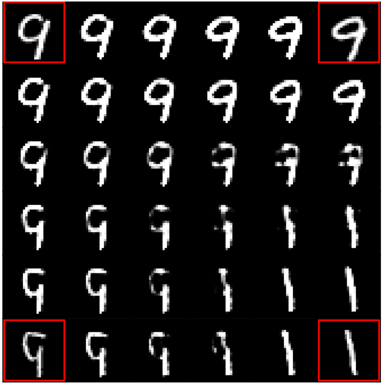
DO, S Wu Fung, X Li, L Ruthotto
OT-Flow: Fast and Accurate CNFs via OT
arXiv:2006.00104, 2020.

More Results

Details in paper



Two of the 43 dimensions in the MINIBOONE CNF.



MNIST synthetic generation. Original images boxed in red.

DO, S Wu Fung, X Li, L Ruthotto
OT-Flow: Fast and Accurate CNFs via OT
arXiv:2006.00104, 2020.

Conclusions

Discretize-Optimize

- DO often converges faster than OD when used in neural ODEs
- For CNFs, DO provides 6x training speedup

OT-Flow

- OT regularization $\Rightarrow$ well-posed and efficient time integration
- Potential flow $\Rightarrow$ smaller model
- OT-Flow achieves 19x training speedup and 28x inference speedup over same baseline

DO and L Ruthotto
DO vs. OD for Time-Series Regression and CNFs
arXiv:2005.13420, 2020.

DO, S Wu Fung, X Li, L Ruthotto
OT-Flow: Fast and Accurate CNFs via OT
arXiv:2006.00104, 2020.

Code: `github.com/EmoryMLIP/OT-Flow`

References I

Chen, Tian Qi et al. (2018). "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6571–6583.

E, Weinan (2017). "A Proposal on Machine Learning via Dynamical Systems". In: *Comm. Math. Statist.* 5.1, pp. 1–11.

Evans, Lawrence C (1997). "Partial differential equations and Monge-Kantorovich mass transfer". In: *Current developments in mathematics* 1997.1, pp. 65–126.

– (2013). *An Introduction to Mathematical Optimal Control Theory Version 0.2*.

Finlay, Chris et al. (2020). "How to train your neural ODE". In: *arXiv:2002.02798*.

Gholaminejad, Amir, Kurt Keutzer, and George Biros (2019). "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". In: *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 730–736.

References II

Grathwohl, Will et al. (2019). "FFJORD: Free-form continuous dynamics for scalable reversible generative models". In: *International Conference on Learning Representations (ICLR)*.

Haber, Eldad and Lars Ruthotto (2017). "Stable Architectures for Deep Neural Networks". In: *Inverse Probl.* 34 (1).

He, Kaiming et al. (2016). "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Li, Qianxiao et al. (2017). "Maximum principle based algorithms for deep learning". In: *The Journal of Machine Learning Research (JMLR)* 18.1, pp. 5998–6026.

Papamakarios, George et al. (2019). "Normalizing Flows for Probabilistic Modeling and Inference". In: *arXiv:1912.02762*.

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows". In: *International Conference on Machine Learning (ICML)*. Lille, France, pp. 1530–1538.

References III

Yang, Liu and George Em Karniadakis (2019). "Potential Flow Generator with L_2 Optimal Transport Regularity for Generative Models". In: *arXiv:1908.11462*.

Zhang, Linfeng, Weinan E, and Lei Wang (2018). "Monge-Ampère Flow for Generative Modeling". In: *arXiv:1809.10188*.