Differential Equations for Continuous-Time Deep Learning

UCI International Zoom Inverse Problems Seminar

Lars Ruthotto

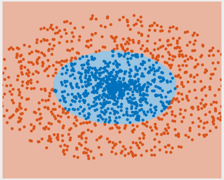
Departments of Mathematics and Computer Science
Emory University

`lruthotto@emory.edu`
@lruthotto

Slides Funding

Examples: Continuous Time Deep Learning

Ex 1: Supervised Learning



Given:

- features $\mathbf{Y}_0$
- labels $\mathbf{C}$

Find $F(\cdot, \boldsymbol{\theta})$ that minimizes

$$\text{loss}[F(\mathbf{Y}_0, \boldsymbol{\theta}), \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}]$$

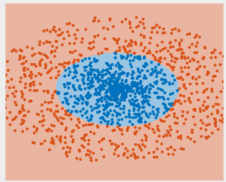
Options for F:

- Multilayer Perceptron
- ResNet, Neural ODE

LR
DiffEq for Continuous-Time Deep Learning
AMS Notices, arXiv:2401.03965

Examples: Continuous Time Deep Learning

Ex 1: Supervised Learning



Given:

- features $\mathbf{Y}_0$
- labels $\mathbf{C}$

Find $F(\cdot, \boldsymbol{\theta})$ that minimizes

$$\text{loss}[F(\mathbf{Y}_0, \boldsymbol{\theta}), \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}]$$

Options for F:

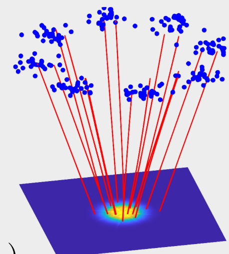
- Multilayer Perceptron
- ResNet, Neural ODE

LR
DiffEq for Continuous-Time Deep Learning
AMS Notices, arXiv:2401.03965

Ex 2: Generative Modeling

Given:

- samples $\mathbf{x}_1, \mathbf{x}_2, \ldots \sim \rho_x$
- target density ρ_z



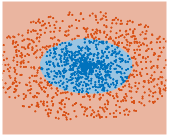
Find $F : \mathbb{R}^d \to \mathbb{R}^d$ that maximizes

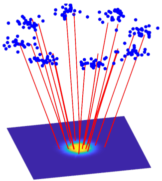
$$\rho_x(\mathbf{x}_j) = \rho_z(F(\mathbf{x}_j)) \det \nabla F(\mathbf{x}_j)$$

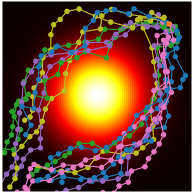
conditional generative modeling: learn $\rho_x(\mathbf{x}|\mathbf{y})$ using samples from joint distribution

LR, S Osher, W Li, L Nurbekyan, S Wu Fung
An ML Framework for Solving High-Dimensional MFG/C
PNAS 117 (17), 9183-9193, 2020

Agenda: Diff Eq for Continuous-Time Deep Learning







- **Intro to Continuous-Time Deep Learning**
  - ResNet and Neural ODEs
  - Training via Inverse Problems / Optimal Control
  - A PDE Perspective for Supervised Learning
- **Generative Modeling**
  - (Conditional) Continuous Normalizing Flows
  - Application in Simulation Based Inference
  - Score-Based Diffusion Models
- **Optimal Control**
  - High-dimensional HJB Equations
  - Amortized PDE control

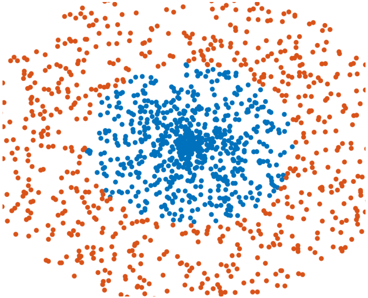
LR
DiffEq for Continuous-Time Deep Learning
AMS Notices, arXiv:2401.03965

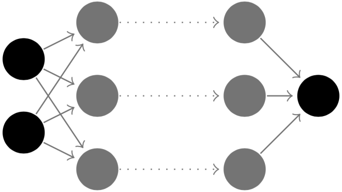
Z. Wang, D. Verma, R. Baptista, Y. Marzouk, LR
NNs for COT and Bayesian Inference
arXiv preprint 2310.16975, 2023

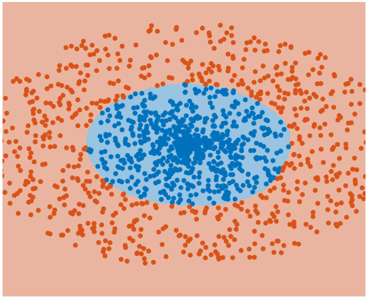
T. Yang, P. Hagemann, S. Mildenberger, LR, G. Steidl
ML Diffusion: ∞-dim SBDM
arXiv: 2303:04772, 2023

Continuous-Time Deep Learning

Example: Supervised Classification with a DNN

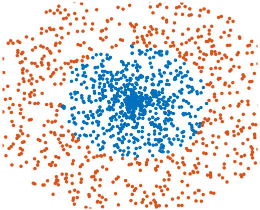

training data


DNN sketch


classification result

ResNet: Residual Neural Networks (He et al. 2016)

Training data $\{(\mathbf{y}^{(1)}, c^{(1)}), (\mathbf{y}^{(2)}, c^{(2)}), \ldots\} \subset \mathbb{R}^2 \times \{0,1\}$.

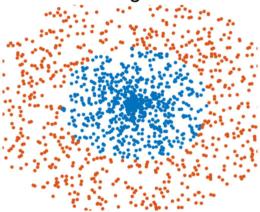


ResNet: Residual Neural Networks (He et al. 2016)

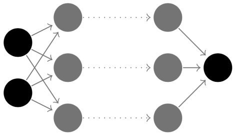
Training data $\{(\mathbf{y}^{(1)}, c^{(1)}), (\mathbf{y}^{(2)}, c^{(2)}), \ldots\} \subset \mathbb{R}^2 \times \{0, 1\}$.
Forward propagation of input $\mathbf{y}$ through simple ResNet

$$\mathbf{u}_0 = \mathbf{K}_{\text{in}}\mathbf{y}$$

training data:



DNN sketch:



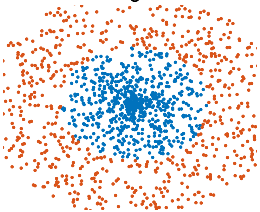
ResNet: Residual Neural Networks (He et al. 2016)

Training data $\{(\mathbf{y}^{(1)}, c^{(1)}), (\mathbf{y}^{(2)}, c^{(2)}), \ldots\} \subset \mathbb{R}^2 \times \{0, 1\}$.
Forward propagation of input $\mathbf{y}$ through simple ResNet

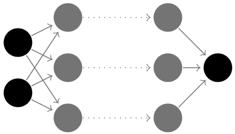
$$
\begin{aligned}
\mathbf{u}_0 &= \mathbf{K}_{\text{in}}\mathbf{y} \\
\mathbf{u}_1 &= \mathbf{u}_0 + h\,\sigma(\mathbf{K}_0\mathbf{u}_0 + \mathbf{b}_0) \\
\vdots &= \quad \vdots \\
\mathbf{u}_N &= \mathbf{u}_{N-1} + h\,\sigma(\mathbf{K}_{N-1}\mathbf{u}_{N-1} + \mathbf{b}_{N-1}))
\end{aligned}
$$

with $h > 0$,

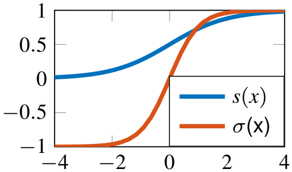
training data:



DNN sketch:



activation σ and sigmoid s:



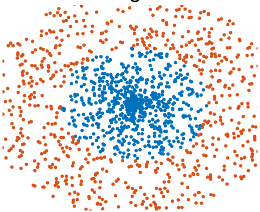
ResNet: Residual Neural Networks (He et al. 2016)

Training data $\{(\mathbf{y}^{(1)}, c^{(1)}), (\mathbf{y}^{(2)}, c^{(2)}), \ldots\} \subset \mathbb{R}^2 \times \{0,1\}$.
Forward propagation of input $\mathbf{y}$ through simple ResNet

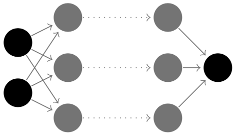
$$\begin{aligned} \mathbf{u}_0 &= \mathbf{K}_{\text{in}}\mathbf{y} \\ \mathbf{u}_1 &= \mathbf{u}_0 + h\,\sigma(\mathbf{K}_0\mathbf{u}_0 + \mathbf{b}_0) \\ \vdots &= \quad \vdots \\ \mathbf{u}_N &= \mathbf{u}_{N-1} + h\,\sigma(\mathbf{K}_{N-1}\mathbf{u}_{N-1} + \mathbf{b}_{N-1})) \end{aligned}$$

with $h > 0$,

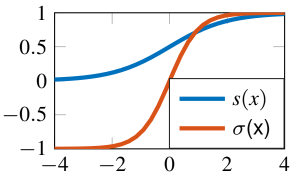
training data:



DNN sketch:



activation σ and sigmoid s:



ResNet: Residual Neural Networks (He et al. 2016)

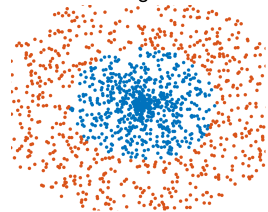
Training data $\{(\mathbf{y}^{(1)}, c^{(1)}), (\mathbf{y}^{(2)}, c^{(2)}), \dots\} \subset \mathbb{R}^2 \times \{0,1\}$.
Forward propagation of input $\mathbf{y}$ through simple ResNet

$$\begin{aligned}
\mathbf{u}_0 &= \mathbf{K}_{\text{in}}\mathbf{y} \\
\mathbf{u}_1 &= \mathbf{u}_0 + h\,\sigma(\mathbf{K}_0\mathbf{u}_0 + \mathbf{b}_0) \\
\vdots &= \quad \vdots \\
\mathbf{u}_N &= \mathbf{u}_{N-1} + h\,\sigma(\mathbf{K}_{N-1}\mathbf{u}_{N-1} + \mathbf{b}_{N-1}))
\end{aligned}$$

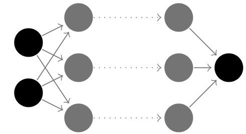
with $h > 0$, $\quad \boldsymbol{\theta}_i^{\text{Res}} := (\mathbf{K}_i, \mathbf{b}_i)$.

Let $F(\mathbf{y}, \boldsymbol{\theta}) := s(\mathbf{K}_{\text{out}}\mathbf{u}_N + \mathbf{b}_{\text{out}})$, weights $\boldsymbol{\theta} := (\boldsymbol{\theta}_0^{\text{Res}}, \dots, \boldsymbol{\theta}_{N-1}^{\text{Res}}, \mathbf{K}_{\text{out}}, \mathbf{b}_{\text{out}})$.

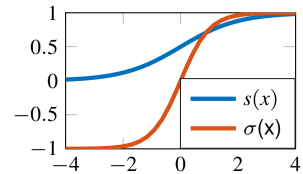
training data:



DNN sketch:



activation σ and sigmoid s:



ResNet: Residual Neural Networks (He et al. 2016)

Training data $\{(\mathbf{y}^{(1)}, c^{(1)}), (\mathbf{y}^{(2)}, c^{(2)}), \ldots\} \subset \mathbb{R}^2 \times \{0, 1\}$.
Forward propagation of input $\mathbf{y}$ through simple ResNet

$$
\begin{aligned}
\mathbf{u}_0 &= \mathbf{K}_{\text{in}}\mathbf{y} \\
\mathbf{u}_1 &= \mathbf{u}_0 + h\,\sigma(\mathbf{K}_0\mathbf{u}_0 + \mathbf{b}_0) \\
\vdots &= \quad \vdots \\
\mathbf{u}_N &= \mathbf{u}_{N-1} + h\,\sigma(\mathbf{K}_{N-1}\mathbf{u}_{N-1} + \mathbf{b}_{N-1}))
\end{aligned}
$$

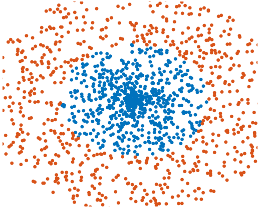
with $h > 0, \qquad \boldsymbol{\theta}_i^{\text{Res}} := (\mathbf{K}_i, \mathbf{b}_i)$.

Let $F(\mathbf{y}, \boldsymbol{\theta}) := s(\mathbf{K}_{\text{out}}\mathbf{u}_N + \mathbf{b}_{\text{out}})$, weights
$\boldsymbol{\theta} := (\boldsymbol{\theta}_0^{\text{Res}}, \ldots, \boldsymbol{\theta}_{N-1}^{\text{Res}}, \mathbf{K}_{\text{out}}, \mathbf{b}_{\text{out}})$.
Train weights by solving (Bottou et al. 2018)

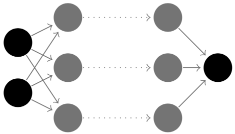
$$\min_{\boldsymbol{\theta}} \mathbb{E}\left[\ell(F(\mathbf{y}, \boldsymbol{\theta}), c)\right] + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2,$$

with cross entropy loss $\ell(z, c) = -c\log(z) - (1-c)\log(1-z)$.

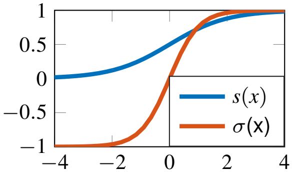
training data:



DNN sketch:



activation σ and sigmoid s:



ResNet: Discussion

In ResNet, $\mathbf{u}_N$ is forward Euler approximation of $\mathbf{u}(T)$,

$$\partial_t \mathbf{u}(t) = f(\mathbf{u}(t), \boldsymbol{\theta}^{\mathrm{ODE}}(t)), \quad t \in (0, T], \quad \mathbf{u}(0) = \mathbf{u}_0;$$

see (E 2017; Haber and Ruthotto 2017).

Remarks

1. $f(\mathbf{u}, \boldsymbol{\theta}^{\mathrm{ODE}}(t)) = \sigma(\mathbf{K}(t)\mathbf{u} + \mathbf{b}(t))$ gives ResNet

ResNet: Discussion

In ResNet, $\mathbf{u}_N$ is forward Euler approximation of $\mathbf{u}(T)$,

$$\partial_t \mathbf{u}(t) = f(\mathbf{u}(t), \boldsymbol{\theta}^{\mathrm{ODE}}(t)), \quad t \in (0, T], \quad \mathbf{u}(0) = \mathbf{u}_0;$$

see (E 2017; Haber and Ruthotto 2017).

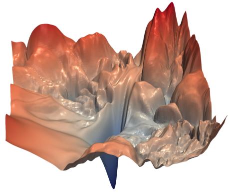
Advantages over other Architectures

1. ResNets often improve with depth
2. state-of-the-art results for many tasks
3. easy to train and easy to add depth

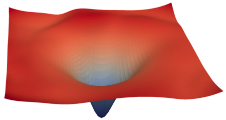
Remarks

1. $f(\mathbf{u}, \boldsymbol{\theta}^{\mathrm{ODE}}(t)) = \sigma(\mathbf{K}(t)\mathbf{u} + \mathbf{b}(t))$ gives ResNet

impact on loss function (Li et al. 2018)



56-layer network (no ResNet)



56-layer ResNet

ResNet: Discussion

In ResNet, $\mathbf{u}_N$ is forward Euler approximation of $\mathbf{u}(T)$,

$$\partial_t \mathbf{u}(t) = f(\mathbf{u}(t), \boldsymbol{\theta}^{\mathrm{ODE}}(t)), \quad t \in (0, T], \quad \mathbf{u}(0) = \mathbf{u}_0;$$

see (E 2017; Haber and Ruthotto 2017).

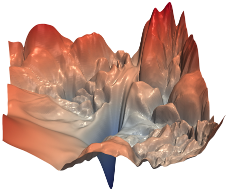
Advantages over other Architectures

1. ResNets often improve with depth
2. state-of-the-art results for many tasks
3. easy to train and easy to add depth

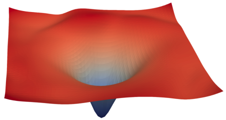
Remarks

1. $f(\mathbf{u}, \boldsymbol{\theta}^{\mathrm{ODE}}(t)) = \sigma(\mathbf{K}(t)\mathbf{u} + \mathbf{b}(t))$ gives ResNet
2. in practice: more complicated layer f, concatenate ResNets to change width or image resolution
3. similar continuous networks, extensions to PDEs, and implicit time integrators in (Rico-Martínez et al. 1992; González-García et al. 1998).

impact on loss function (Li et al. 2018)



56-layer network (no ResNet)



56-layer ResNet

Stable Architectures for DNNs (Haber and Ruthotto 2017)

When is forward propagation stable? That is, when $\exists M > 0$ such that

$$\|F(\mathbf{y} + \boldsymbol{\epsilon}, \boldsymbol{\theta}) - F(\mathbf{y}, \boldsymbol{\theta})\| \leq M \|\boldsymbol{\epsilon}\| \quad (\boldsymbol{\epsilon} \text{ input perturbation})$$

Motivation: well-posed training problem, adversarial attacks, efficient optimization,...

Stable Architectures for DNNs (Haber and Ruthotto 2017)

When is forward propagation stable? That is, when $\exists M > 0$ such that

$$\|F(\mathbf{y}+\boldsymbol{\epsilon},\boldsymbol{\theta}) - F(\mathbf{y},\boldsymbol{\theta})\| \leq M\|\boldsymbol{\epsilon}\| \quad (\boldsymbol{\epsilon} \text{ input perturbation})$$

Motivation: well-posed training problem, adversarial attacks, efficient optimization,. . .

Main Findings and Contributions

1. $\partial_t \mathbf{u}(t) = \sigma(\mathbf{K}(t)\mathbf{u}(t) + \mathbf{b}(t))$ not stable for all $\mathbf{K}(\cdot), \mathbf{b}(\cdot)$
2. alternative f: antisymmetric $\mathbf{K}$, Hamiltonian-inspired networks
3. symplectic integrators to obtain stable architecture ($\neq$ ResNet)
4. stable DNNs perform competitively (on simple tasks)

Stable Architectures for DNNs (Haber and Ruthotto 2017)

When is forward propagation stable? That is, when $\exists M > 0$ such that

$$\|F(\mathbf{y}+\boldsymbol{\epsilon},\boldsymbol{\theta}) - F(\mathbf{y},\boldsymbol{\theta})\| \leq M\|\boldsymbol{\epsilon}\| \quad (\boldsymbol{\epsilon} \text{ input perturbation})$$

Motivation: well-posed training problem, adversarial attacks, efficient optimization,. . .

Main Findings and Contributions

1. $\partial_t \mathbf{u}(t) = \sigma(\mathbf{K}(t)\mathbf{u}(t) + \mathbf{b}(t))$ not stable for all $\mathbf{K}(\cdot), \mathbf{b}(\cdot)$
2. alternative f: antisymmetric $\mathbf{K}$, Hamiltonian-inspired networks
3. symplectic integrators to obtain stable architecture ($\neq$ ResNet)
4. stable DNNs perform competitively (on simple tasks)

Improvements and Related Works

1. more expressive architectures (Chang et al. 2018), multilevel training (Chang et al. 2017)
2. improved stability results (Ruthotto and Haber 2020; Celledoni et al. 2020)
3. analysis: convergence (Thorpe and Gennip 2018), opt. conditions (Benning et al. 2019)
4. multi-step and other time integrators (Lu et al. 2017)
5. discrete weights (Li and Hao 2018), maximum principles (Li et al. 2017)

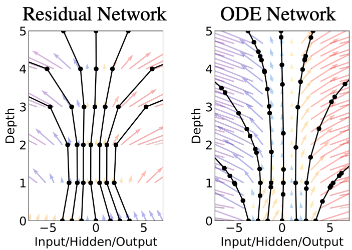
Neural Ordinary Differential Equations (Chen et al. 2018)

Main Novelties and Contributions

1. apply adaptive time integrator to continuous ResNet
2. compute gradients of loss function using adjoint equation

$$-\partial_t \mathbf{p}(t) = \nabla f(\mathbf{u}(t), \boldsymbol{\theta}(t))^\top \mathbf{p}(t), \quad \mathbf{p}(T) = \nabla_{\mathbf{u}_N} \ell(F(\mathbf{y}), c)$$

3. save memory by re-computing $\mathbf{u}(t)$ backward in time.
4. popularized continuous models in ML community



from (Chen et al. 2018)

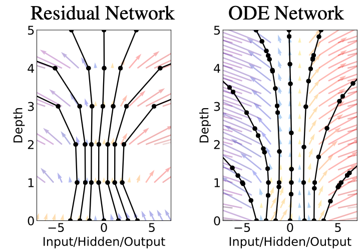
Neural Ordinary Differential Equations (Chen et al. 2018)

Main Novelties and Contributions

1. apply adaptive time integrator to continuous ResNet
2. compute gradients of loss function using adjoint equation

$$-\partial_t \mathbf{p}(t) = \nabla f(\mathbf{u}(t), \boldsymbol{\theta}(t))^\top \mathbf{p}(t), \quad \mathbf{p}(T) = \nabla_{\mathbf{u}_N} \ell(F(\mathbf{y}), c)$$

3. save memory by re-computing $\mathbf{u}(t)$ backward in time.
4. **popularized continuous models in ML community**



from (Chen et al. 2018)

Artificial intelligence / Machine learning

A radical new neural network design could overcome big challenges in AI

Researchers borrowed equations from calculus to redesign the core machinery of deep learning so it can model continuous processes like changes in health.

by **Karen Hao** December 12, 2018

MIT Tech Review, 2018

Neural Ordinary Differential Equations (Chen et al. 2018)

Main Novelties and Contributions

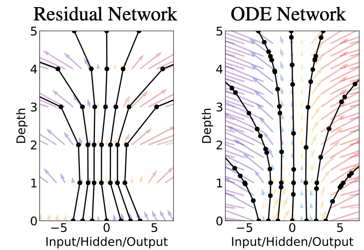
1. apply adaptive time integrator to continuous ResNet
2. compute gradients of loss function using adjoint equation

$$-\partial_t \mathbf{p}(t) = \nabla f(\mathbf{u}(t), \boldsymbol{\theta}(t))^\top \mathbf{p}(t), \quad \mathbf{p}(T) = \nabla_{\mathbf{u}_N} \ell(F(\mathbf{y}), c)$$

3. save memory by re-computing $\mathbf{u}(t)$ backward in time.
4. popularized continuous models in ML community

Improvements and Related Works

1. item 3 above can be unstable $\rightsquigarrow$ checkpointing (Gholami et al. 2019)
2. invertible ResNet (Behrmann et al. 2019), generative modeling (Grathwohl et al. 2018; Chen et al. 2019)
3. augmentation needed for expressiveness (Dupont et al. 2019)



from (Chen et al. 2018)

Artificial intelligence / Machine learning

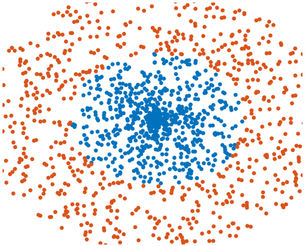
A radical new neural network design could overcome big challenges in AI

Researchers borrowed equations from calculus to redesign the core machinery of deep learning so it can model continuous processes like changes in health.

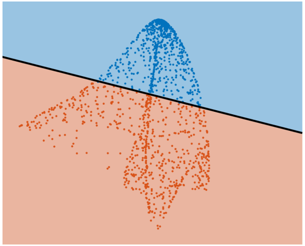
by **Karen Hao** December 12, 2018

MIT Tech Review, 2018

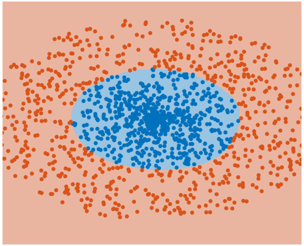
Optimal Control Framework for Deep Learning



training data, $\mathbf{Y}_0, \mathbf{C}$



prop. features, $\mathbf{Y}(T), \mathbf{C}$



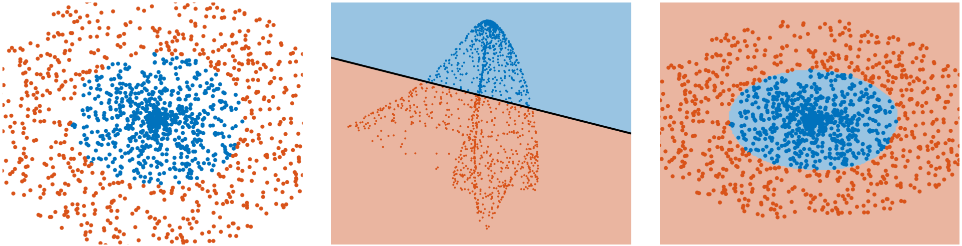
classification result

Supervised Deep Learning Problem

Given training data, $\mathbf{Y}_0$, and labels, $\mathbf{C}$, find **network parameters** $\boldsymbol{\theta}$ and **classification weights** $\mathbf{W}$**,** $\boldsymbol{\mu}$ such that the DNN predicts the data-label relationship (and generalizes to new data), i.e., solve

$$\begin{aligned} \text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad & \text{loss}[\mathbf{W}\mathbf{Y}(T) + \boldsymbol{\mu}, \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}] \\ \text{subject to} \quad & \partial_t \mathbf{Y}(t) = f\left(\mathbf{Y}(t), \boldsymbol{\theta}(t)\right), \ \mathbf{Y}(0) = \mathbf{Y}_0. \end{aligned}$$

Optimal Control Framework for Deep Learning

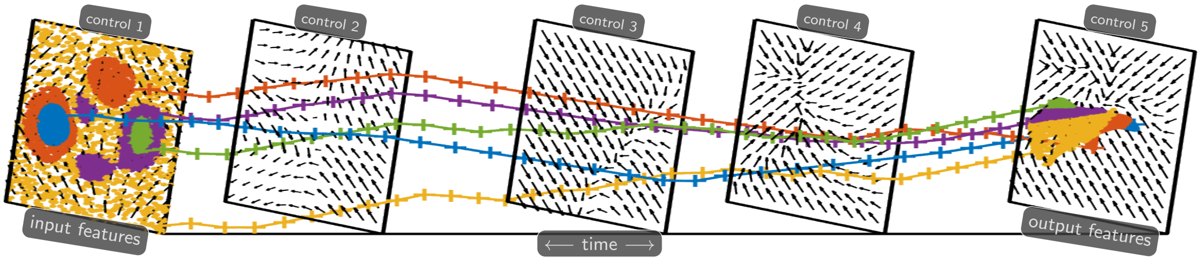


training data, $\mathbf{Y}_0, \mathbf{C}$ prop. features, $\mathbf{Y}(T), \mathbf{C}$ classification result

Supervised Deep Learning Problem

Given training data, $\mathbf{Y}_0$, and labels, $\mathbf{C}$, find **network parameters θ** and **classification weights $\mathbf{W}$, μ** such that the DNN predicts the data-label relationship (and generalizes to new data), i.e., solve

$$\text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad \text{loss}[\mathbf{W}\mathbf{Y}(T,\boldsymbol{\theta}) + \boldsymbol{\mu}, \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}]$$

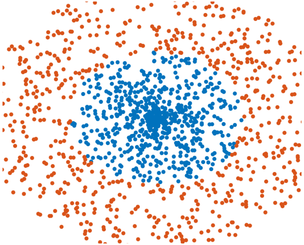


Supervised Classification with Continuous ResNet

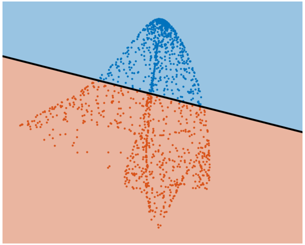
Given $(\mathbf{y}_1, \mathbf{c}_1), (\mathbf{y}_2, \mathbf{c}_2), \ldots$ find **network weights (θ)** and **classification weights ($\mathbf{W}$, μ)** such that the DNN predicts the data-label relationship (and generalizes to new data), by solving

$$\begin{aligned} \text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad & \mathbb{E}\left(\text{loss}[g(\mathbf{W}\mathbf{u}(T) + \boldsymbol{\mu}), \mathbf{c}]\right) + \text{regularizer}[\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}] \\ \text{subject to} \quad & \partial_t \mathbf{u}(t) = f(\mathbf{u}(t), \boldsymbol{\theta}(t)), \quad \forall t \in [0, T], \quad \mathbf{u}(0) = \mathbf{y}. \end{aligned}$$

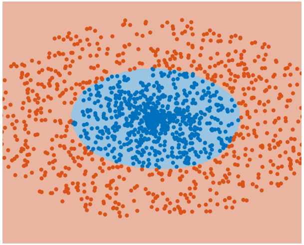
A PDE Perspective of Continuous-Time Learning



training data, $\mathbf{Y}_0, \mathbf{C}$



prop. features, $u(\mathbf{X}, 0)$



classification result $u(\mathbf{X}, 1)$

Supervised Deep Learning Problem

Given training data, $\mathbf{Y}_0$, and labels, $\mathbf{C}$, find **network parameters** $\boldsymbol{\theta}$ and **classification weights** $\mathbf{W}, \boldsymbol{\mu}$ such that the DNN predicts the data-label relationship (and generalizes to new data), i.e., solve

$$\begin{aligned} \text{minimize}_{\boldsymbol{\theta},\mathbf{W},\boldsymbol{\mu}} \quad & \text{loss}[u(\mathbf{Y}_0, 1), \mathbf{C}] + \text{regularizer}[\boldsymbol{\theta}, \mathbf{W}, \boldsymbol{\mu}] \\ \text{subject to} \quad & \partial_t u(\mathbf{X}, t) + f(\mathbf{X}, \boldsymbol{\theta}(t))^\top \nabla u(\mathbf{X}, t) = 0 \\ & u(\mathbf{X}, 0) = \mathbf{W}\mathbf{X} + \boldsymbol{\mu}. \end{aligned}$$

Continuous Normalizing Flows

Continuous Normalizing Flows (CNF)

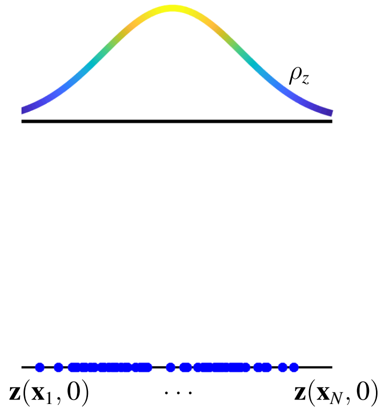
Likelihood Maximization

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, find a velocity v that maximizes the likelihood of the samples w.r.t. the push-forward of the standard normal distribution ρ_z, i.e.,

$$\text{maximize}_{v,\mathbf{z}} \quad \frac{1}{N}\sum_{k=1}^{N} \rho_z(\mathbf{z}(\mathbf{x}_k, 1)) \cdot \det \nabla(\mathbf{z}(\mathbf{x}_k, 1))$$

$$\text{subject to} \quad \frac{d}{dt}\mathbf{z}(\mathbf{x}_k, t) = v(\mathbf{z}(\mathbf{x}_k, t), t),$$

with $\mathbf{z}(\mathbf{x}_k, 0) = \mathbf{x}_k$ for $k = 1, 2, \ldots, N$.



W Grathwohl et al.
FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv*, 2018.

Continuous Normalizing Flows (CNF)

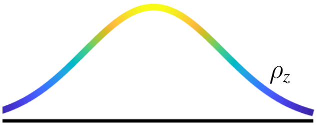
Likelihood Maximization

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, find a velocity v that maximizes the likelihood of the samples w.r.t. the push-forward of the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_{v,\mathbf{z}} \quad G_{CNF}(v, \mathbf{z}) := \frac{1}{N} \sum_{k=1}^{N} \left(\frac{1}{2} \|\mathbf{z}(\mathbf{x}_k, 1)\|^2 - l(\mathbf{x}_k, 1) \right)$$

$$\text{subject to} \quad \frac{d}{dt} \begin{pmatrix} \mathbf{z}(\mathbf{x}_k, s) \\ l(\mathbf{x}_k, s) \end{pmatrix} = \begin{pmatrix} v(\mathbf{z}(\mathbf{x}_k, s), s) \\ \operatorname{trace}(\nabla v(\mathbf{z}(\mathbf{x}_k, s), s)) \end{pmatrix}$$

with $\mathbf{z}(\mathbf{x}_k, 0) = \mathbf{x}_k$ and $l(\mathbf{x}_k, 0) = 0$ for $k = 1, 2, \ldots, N$.



Here: $l(\mathbf{x}_k, 1) = \log\det(\nabla \mathbf{z}(\mathbf{x}_k, 1))$

W Grathwohl et al.
FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv*, 2018.

Continuous Normalizing Flows (CNF)

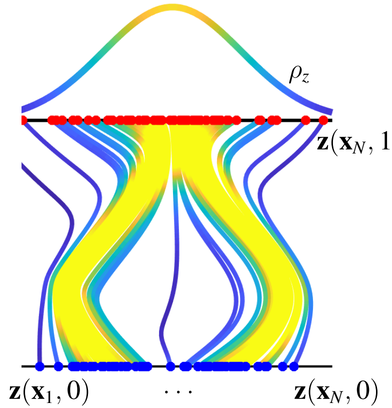
Likelihood Maximization

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, find a velocity v that maximizes the likelihood of the samples w.r.t. the push-forward of the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_{v,\mathbf{z}} \quad G_{CNF}(v, \mathbf{z}) := \frac{1}{N} \sum_{k=1}^{N} \left(\frac{1}{2} \|\mathbf{z}(\mathbf{x}_k, 1)\|^2 - l(\mathbf{x}_k, 1) \right)$$

$$\text{subject to} \quad \frac{d}{dt} \begin{pmatrix} \mathbf{z}(\mathbf{x}_k, s) \\ l(\mathbf{x}_k, s) \end{pmatrix} = \begin{pmatrix} v(\mathbf{z}(\mathbf{x}_k, s), s) \\ \operatorname{trace}(\nabla v(\mathbf{z}(\mathbf{x}_k, s), s)) \end{pmatrix}$$

with $\mathbf{z}(\mathbf{x}_k, 0) = \mathbf{x}_k$ and $l(\mathbf{x}_k, 0) = 0$ for $k = 1, 2, \ldots, N$.



Here: $l(\mathbf{x}_k, 1) = \log\det(\nabla \mathbf{z}(\mathbf{x}_k, 1))$

W Grathwohl et al.
FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv*, 2018.

Continuous Normalizing Flows (CNF)

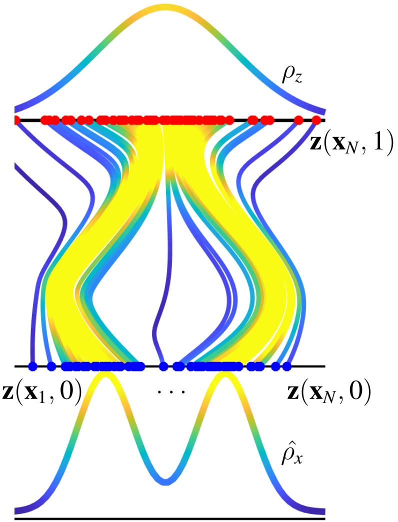
Likelihood Maximization

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, find a velocity v that maximizes the likelihood of the samples w.r.t. the push-forward of the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_{v,\mathbf{z}} \quad G_{CNF}(v, \mathbf{z}) := \frac{1}{N} \sum_{k=1}^{N} \left(\frac{1}{2} \|\mathbf{z}(\mathbf{x}_k, 1)\|^2 - l(\mathbf{x}_k, 1) \right)$$

$$\text{subject to} \quad \frac{d}{dt} \begin{pmatrix} \mathbf{z}(\mathbf{x}_k, s) \\ l(\mathbf{x}_k, s) \end{pmatrix} = \begin{pmatrix} v(\mathbf{z}(\mathbf{x}_k, s), s) \\ \operatorname{trace}(\nabla v(\mathbf{z}(\mathbf{x}_k, s), s)) \end{pmatrix}$$

with $\mathbf{z}(\mathbf{x}_k, 0) = \mathbf{x}_k$ and $l(\mathbf{x}_k, 0) = 0$ for $k = 1, 2, \ldots, N$.



Here: $l(\mathbf{x}_k, 1) = \log\det(\nabla \mathbf{z}(\mathbf{x}_k, 1))$

W Grathwohl et al.
FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. *arXiv*, 2018.

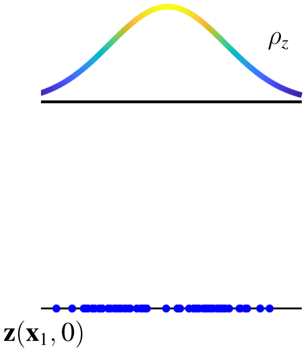
OT-Flow: Regularized Continuous Normalizing Flow

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \sim \rho_x$, find the value function Φ such that the flow given by $v = -\nabla\Phi$ maximizes the likelihood of the samples w.r.t. the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_\Phi \quad \mathbb{E}_{\mathbf{x}\sim\rho_x}\left[\frac{1}{2}\|z(\mathbf{x},1)\|^2 - l(\mathbf{x},1) + \qquad\qquad\qquad\right]$$

$$\text{subject to} \quad \frac{d}{dt}\begin{pmatrix} \mathbf{z}(\mathbf{x},t) \\ l(x,t) \\ \\ \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(\mathbf{z}(\mathbf{x},t),t) \\ -\Delta\Phi(\mathbf{z}(\mathbf{x},t),t) \\ \\ \end{pmatrix}$$

$$\mathbf{z}(\mathbf{x},0) = \mathbf{x}, \qquad\qquad = l(\mathbf{x},0) = 0$$



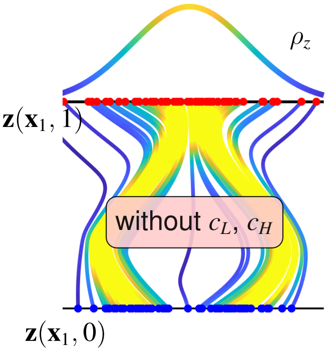
OT-Flow: Regularized Continuous Normalizing Flow

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \sim \rho_x$, find the value function Φ such that the flow given by $v = -\nabla\Phi$ maximizes the likelihood of the samples w.r.t. the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_{\Phi} \quad \mathbb{E}_{\mathbf{x}\sim\rho_x}\left[\frac{1}{2}\|z(\mathbf{x},1)\|^2 - l(\mathbf{x},1) + \qquad \right]$$

$$\text{subject to} \quad \frac{d}{dt}\begin{pmatrix} \mathbf{z}(\mathbf{x},t) \\ l(x,t) \\ \\ \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(\mathbf{z}(\mathbf{x},t),t) \\ -\Delta\Phi(\mathbf{z}(\mathbf{x},t),t) \\ \\ \end{pmatrix}$$

$$\mathbf{z}(\mathbf{x},0) = \mathbf{x}, \qquad = l(\mathbf{x},0) = 0$$



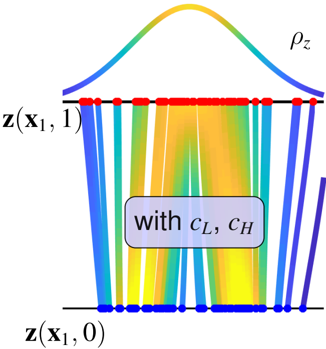
OT-Flow: Regularized Continuous Normalizing Flow

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \sim \rho_x$, find the value function Φ such that the flow given by $v = -\nabla\Phi$ maximizes the likelihood of the samples w.r.t. the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_\Phi \quad \mathbb{E}_{\mathbf{x}\sim\rho_x}\left[\frac{1}{2}\|z(\mathbf{x},1)\|^2 - l(\mathbf{x},1) + c_{\mathrm{L}}(\mathbf{x},1) + \alpha_1 c_{\mathrm{H}}(\mathbf{x},1)\right]$$

$$\text{subject to} \quad \frac{d}{dt}\begin{pmatrix} \mathbf{z}(\mathbf{x},t) \\ l(x,t) \\ c_{\mathrm{L}}(\mathbf{x},t) \\ c_{\mathrm{H}}(\mathbf{x},t) \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(\mathbf{z}(\mathbf{x},t),t) \\ -\Delta\Phi(\mathbf{z}(\mathbf{x},t),t) \\ \frac{1}{2}\|\nabla\Phi(\mathbf{z}(\mathbf{x},t),t)\|^2 \\ \left|\frac{d}{dt}\Phi(\mathbf{z}(\mathbf{x},t),t) + \frac{1}{2}\|\nabla\Phi(\mathbf{z}(\mathbf{x},t),t)\|^2\right| \end{pmatrix}$$

$$\mathbf{z}(\mathbf{x},0) = \mathbf{x}, \quad c_{\mathrm{L}}(\mathbf{x},0) = c_{\mathrm{H}}(\mathbf{x},0) = l(\mathbf{x},0) = 0$$



OT $\rightsquigarrow$ ☼ uniqueness, more efficient time integration ☼

Benefits of OT also observed in Zhang et al. 2018; Yang and Karniadakis 2020; Finlay et al. 2020

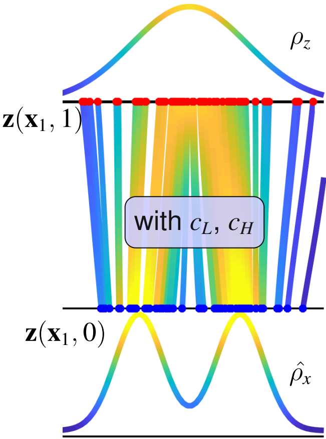
OT-Flow: Regularized Continuous Normalizing Flow

Given samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \sim \rho_x$, find the value function Φ such that the flow given by $v = -\nabla\Phi$ maximizes the likelihood of the samples w.r.t. the standard normal distribution ρ_z, i.e.,

$$\text{minimize}_{\Phi} \quad \mathbb{E}_{\mathbf{x}\sim\rho_x}\left[\frac{1}{2}\|z(\mathbf{x},1)\|^2 - l(\mathbf{x},1) + c_{\mathrm{L}}(\mathbf{x},1) + \alpha_1 c_{\mathrm{H}}(\mathbf{x},1)\right]$$

$$\text{subject to} \quad \frac{d}{dt}\begin{pmatrix} \mathbf{z}(\mathbf{x},t) \\ l(x,t) \\ c_{\mathrm{L}}(\mathbf{x},t) \\ c_{\mathrm{H}}(\mathbf{x},t) \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(\mathbf{z}(\mathbf{x},t),t) \\ -\Delta\Phi(\mathbf{z}(\mathbf{x},t),t) \\ \frac{1}{2}\|\nabla\Phi(\mathbf{z}(\mathbf{x},t),t)\|^2 \\ \left|\frac{d}{dt}\Phi(\mathbf{z}(\mathbf{x},t),t) + \frac{1}{2}\|\nabla\Phi(\mathbf{z}(\mathbf{x},t),t)\|^2\right| \end{pmatrix}$$

$$\mathbf{z}(\mathbf{x},0) = \mathbf{x}, \quad c_{\mathrm{L}}(\mathbf{x},0) = c_{\mathrm{H}}(\mathbf{x},0) = l(\mathbf{x},0) = 0$$



OT $\rightsquigarrow$ ☀ uniqueness, more efficient time integration ☀

Benefits of OT also observed in Zhang et al. 2018; Yang and Karniadakis 2020; Finlay et al. 2020

Background: OT-Flow as Mean Field Game

$$\text{minimize}_{v,\rho} \int -\log(\rho(\mathbf{x},1))\rho_x(\mathbf{x})d\mathbf{x} + \int_0^1 \int \frac{1}{2}\|v(\mathbf{x},t)\|^2 \rho(\mathbf{x},t)d\mathbf{x}dt$$

$$\text{subject to} \quad \partial_t \rho(\mathbf{x},t) + \nabla \cdot (\rho(\mathbf{x},t)v(\mathbf{x},t)) = 0, \quad \rho(\cdot,0) = \rho_z$$

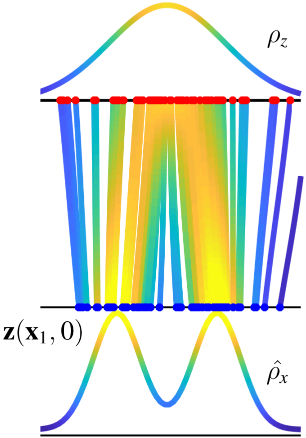
From Pontryagin Maximum Principle, we get

$$v(\mathbf{x},\cdot) = -\nabla\Phi(\mathbf{x},\cdot)$$

and Φ satisfies the Hamilton-Jacobi-Bellman equation

$$\partial_t \Phi(\mathbf{x},t) - \frac{1}{2}\|\nabla\Phi(\mathbf{x},t)\|^2 = 0, \quad \Phi(\mathbf{x},1) = -\frac{\rho_x(\mathbf{x})}{\rho(\mathbf{x},1)}$$

Challenges: fwd/bwd structure, high-dim, density ρ_x unknown.



Neural Network Model for Value Function

Let $s = (\mathbf{x}, t) \in \mathbb{R}^{d+1}$ and use (NN + quadratic) model for value function

$$\Phi(s, \theta) = w^\top N(s, \theta_N) + \frac{1}{2} s^\top A s + c^\top s + b, \quad \theta = (w, \theta_N, \mathrm{vec}(A), c, b)$$

$N(s, \theta_N)$ is an M-layer ResNet with weights $\theta_N = (\mathrm{vec}(K_0), \ldots, \mathrm{vec}(K_M), b_0, \ldots, b_M)$.

Neural Network Model for Value Function

Let $s = (\mathbf{x}, t) \in \mathbb{R}^{d+1}$ and use (NN + quadratic) model for value function

$$\Phi(s, \theta) = w^\top N(s, \theta_N) + \frac{1}{2} s^\top A s + c^\top s + b, \quad \theta = (w, \theta_N, \mathrm{vec}(A), c, b)$$

$N(s, \theta_N)$ is an M-layer ResNet with weights $\theta_N = (\mathrm{vec}(K_0), \ldots, \mathrm{vec}(K_M), b_0, \ldots, b_M)$.

forward propagation:

$$\begin{aligned}
u_0 &= \sigma(K_0 s + b_0) \\
u_1 &= u_0 + h\sigma(K_1 u_0 + b_1) \\
\vdots & \qquad \vdots \\
u_M &= u_{M-1} + h\sigma(K_M u_{M-1} + b_M),
\end{aligned}$$

Output: $w^\top u_M = w^\top N(s, \theta_N)$

Neural Network Model for Value Function

Let $s = (\mathbf{x}, t) \in \mathbb{R}^{d+1}$ and use (NN + quadratic) model for value function

$$\Phi(s,\theta) = w^\top N(s,\theta_N) + \frac{1}{2} s^\top A s + c^\top s + b, \quad \theta = (w, \theta_N, \mathrm{vec}(A), c, b)$$

$N(s,\theta_N)$ is an M-layer ResNet with weights $\theta_N = (\mathrm{vec}(K_0), \ldots, \mathrm{vec}(K_M), b_0, \ldots, b_M)$.

forward propagation:

$$\begin{aligned} u_0 &= \sigma(K_0 s + b_0) \\ u_1 &= u_0 + h\sigma(K_1 u_0 + b_1) \\ \vdots \quad & \quad \vdots \\ u_M &= u_{M-1} + h\sigma(K_M u_{M-1} + b_M), \end{aligned}$$

Output: $w^\top u_M = w^\top N(s,\theta_N)$

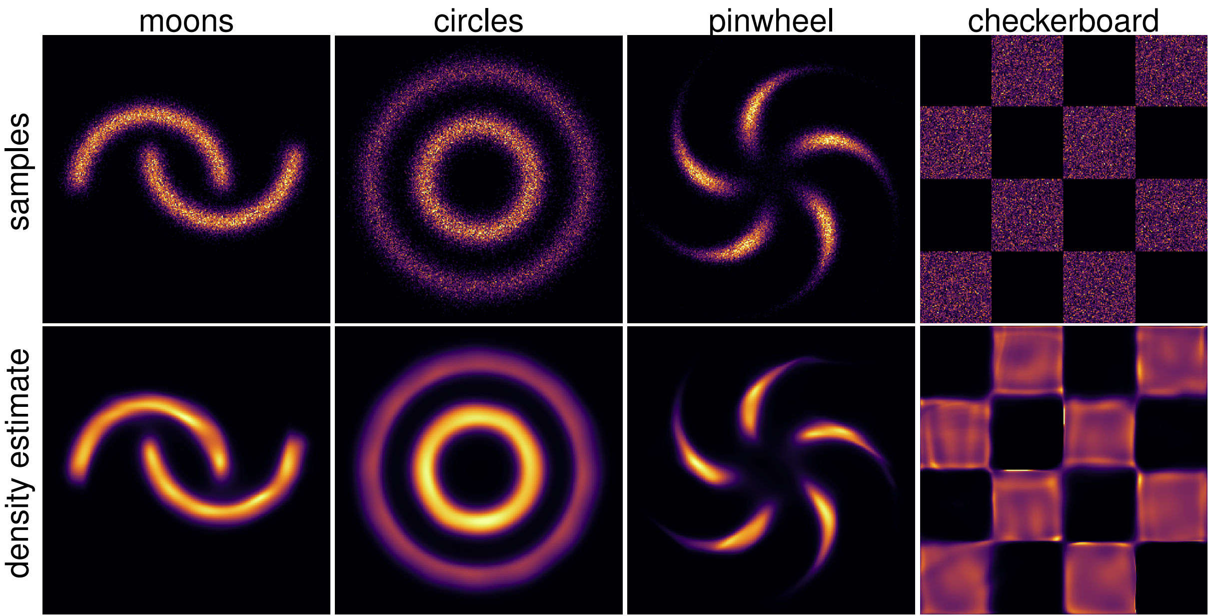
Remark: need also $\nabla_s \Phi$ and $\Delta_x \Phi$

1. automatic differentiation, limited to matrix-vector products

$$\Delta_x \Phi(s,\theta) = \sum_{k=1}^{d} e_k^\top \nabla_x^2 \Phi(s,\theta) e_k$$

2. trace estimators add inaccuracy
3. better compute derivatives manually
4. efficient algorithm $\rightsquigarrow \mathcal{O}(m^2 \cdot d)$ flops
5. implementation easily parallelizes

OT-Flow: Two-Dimensional Examples

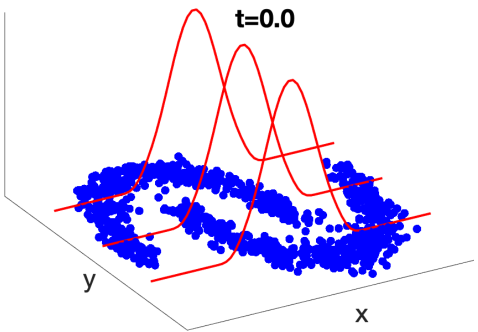


Generative Modeling for Simulation-Based Inference

Motivation: Simulation-Based Inference

Goal: Learn posterior $\pi(\mathbf{x}|\mathbf{y})$ from samples $(\mathbf{x}, \mathbf{y}) \sim \pi(\mathbf{x}, \mathbf{y})$

- $\mathbf{x} \in \mathbb{R}^n$ - parameter of interest
- $\mathbf{y} \in \mathbb{R}^m$ - indirect, noisy measurements



Continuous normalizing flow approach:

1. pick simple reference distribution $\rho_Z \sim \mathcal{N}(0, I_n)$
2. train invertible generator $g_\theta : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ such that

$$\pi(\mathbf{x}|\mathbf{y}) \approx \rho_Z \left(g_\theta^{-1}(\mathbf{x}, \mathbf{y})\right) \cdot |\det \nabla_{\mathbf{x}} g_\theta^{-1}(\mathbf{x}, \mathbf{y})|.$$

3. penalize transport costs $\rightsquigarrow$ conditional optimal transport
4. define g_θ as neural ODE $\rightsquigarrow$ parameterized mean field game

Advantages for SBI:

- non-intrusive
- widely applicable
- computationally efficient

Parameterized Mean Field Game

$$
\begin{aligned}
\min_{\rho,v} \quad & \int_{\mathbb{R}^m}\int_{\mathbb{R}^n} -\log\rho(1,\mathbf{x})\pi(\mathbf{x},\mathbf{y}) + \alpha\int_0^1 \frac{1}{2}\|v(t,\mathbf{x},\mathbf{y})\|^2\rho(t,\mathbf{x},\mathbf{y})dtd\mathbf{x}d\mathbf{y} \\
\text{subject to} \quad & \partial_t\rho(t,\mathbf{x},\mathbf{y}) + \nabla_x\cdot(\rho(t,\mathbf{x},\mathbf{y})v(t,\mathbf{x},\mathbf{y})) = 0, \quad t\in(0,1] \\
& \rho(0,\mathbf{x},\mathbf{y}) = \rho_Z(\mathbf{x}).
\end{aligned}
$$

Derivation: Consider OT-Penalized Maximum Likelihood problem

$$
\min_\theta \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\pi}\left[\frac{1}{2}\left\|g_\theta^{-1}(\mathbf{x},\mathbf{y})\right\|^2 - \log\det\nabla_{\mathbf{x}}g_\theta^{-1}(\mathbf{x},\mathbf{y}) + \alpha\int_0^1\frac{1}{2}\|v_\theta(t,\mathbf{p},\mathbf{y})\|^2dt\right]
$$

with generator given by neural ODE; that is, $g_\theta(\mathbf{z},\mathbf{y}) = \mathbf{u}(1)$ where

$$
\frac{d}{dt}\mathbf{u} = v_\theta(t,\mathbf{u},\mathbf{y}), \quad t\in(0,T], \quad \mathbf{u}(0) = \mathbf{z}
$$

Insights from Optimal Control Theory

Parameterized Hamilton Jacobi Bellman Equations

$$\partial_t \Phi(t, \mathbf{x}, \mathbf{y}) - \frac{1}{2\alpha} \|\nabla_x \Phi(t, \mathbf{x}, \mathbf{y})\|^2 = 0, \quad t \in [0, 1)$$

$$\Phi(1, \mathbf{x}, \mathbf{y}) = -\frac{\pi(\mathbf{x}, \mathbf{y})}{\rho(1, \mathbf{x}, \mathbf{y})},$$

Similar Ansatz to prior work:

1. approximate $\Phi \approx \Phi_\theta$ with scalar-valued NN
2. use feedback form: $v_\theta(t, \mathbf{u}, \mathbf{y}) = -\frac{1}{\alpha} \nabla_x \Phi_\theta(t, \mathbf{u}, \mathbf{y})$
3. Jacobi identity: $\log\det \nabla_x g_\theta^{-1}(\mathbf{x}, \mathbf{y}) = \int_0^1 \Delta \Phi_\theta(t, \mathbf{p}, \mathbf{y}) dt$
4. penalize HJB violation in training

Experiment: Stochastic Lotka-Volterra

Inference for predator-prey

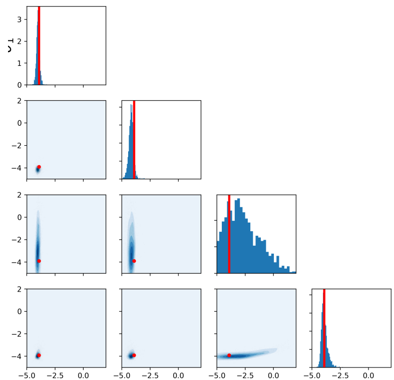
- $\mathbf{x} \in \mathbb{R}^4$ - rate of change for populations
- $\mathbf{y} \in \mathbb{R}^9$ - summary stats
- comparison: sequential MC
- metric: quality / # samples

Training

- 100 pilot runs, 1 epoch
- 1 final training

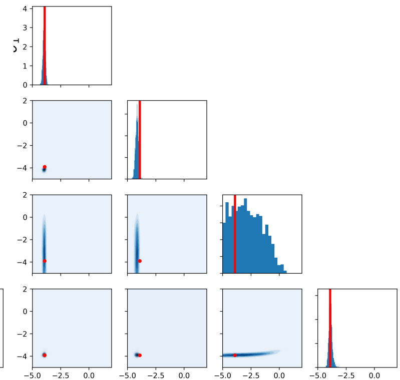
Z. Wang, D. Verma, R. Baptista, Y. Marzouk, LR
NNs for COT and Bayesian Inference
arXiv preprint 2310.16975, 2023

COT-Flow



50k samples

sequential MC



≈5.8M samples

Samples closely match SMC @ much lower computational costs

Experiment: Stochastic Lotka-Volterra

Inference for predator-prey

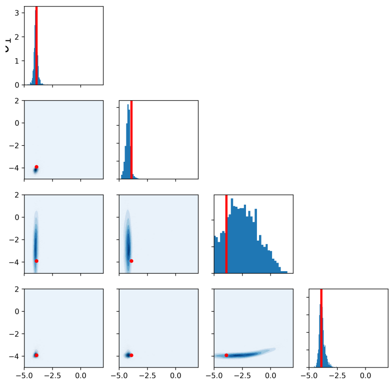
- $\mathbf{x} \in \mathbb{R}^4$ - rate of change for populations
- $\mathbf{y} \in \mathbb{R}^9$ - summary stats
- comparison: sequential MC
- metric: quality / # samples

Training

- 100 pilot runs, 1 epoch
- 1 final training

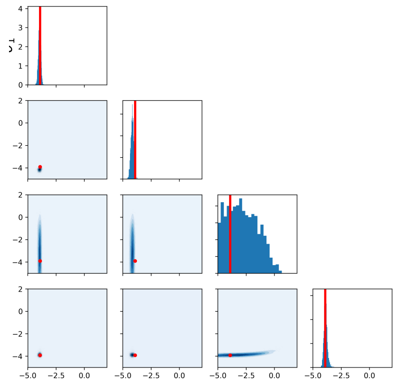
Z. Wang, D. Verma, R. Baptista, Y. Marzouk, LR
NNs for COT and Bayesian Inference
arXiv preprint 2310.16975, 2023

COT-Flow



500k samples

sequential MC



≈5.8M samples

Samples closely match SMC @ much lower computational costs

Experiment: Stochastic Lotka-Volterra

Inference for predator-prey

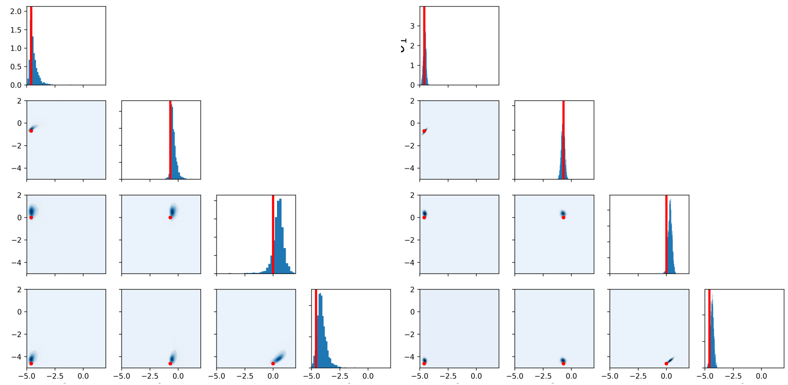
- $\mathbf{x} \in \mathbb{R}^4$ - rate of change for populations
- $\mathbf{y} \in \mathbb{R}^9$ - summary stats
- comparison: sequential MC
- metric: quality / # samples

Training

- 100 pilot runs, 1 epoch
- 1 final training

Z. Wang, D. Verma, R. Baptista, Y. Marzouk, LR
NNs for COT and Bayesian Inference
arXiv preprint 2310.16975, 2023

COT-Flow | sequential MC

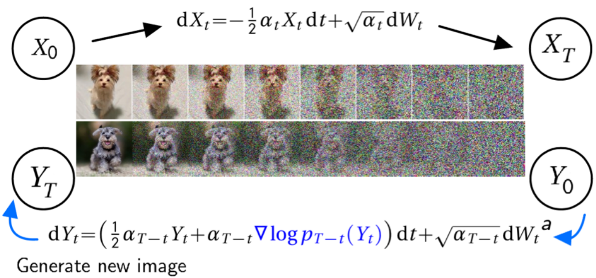


samples | ≈17.9M samples

Samples closely match SMC @ much lower computational costs

Infinite-Dimensional Score-Based Diffusion

Multilevel Diffusion: ∞-Dimensional Score-Based Diffusion



	X_t	s_θ	Q
	$\in \mathbb{R}^d$	U-Net	I_d
	$\in L^2([0,1]^2)$	FNO	trace class

SBDM in a nutshell:

1. approach: $s_\theta(t, X) \approx \nabla \log p_t(X)$
2. set $dW_t \sim \mathcal{N}(0, Q)$ and train via

$$\mathbb{E}_{X_0,t}\mathbb{E}_{X_t \sim \mathbb{P}_{X_t|X_0}} \left\| s_\theta(t, X_t) - Q\nabla \log p_t(X_t|X_0) \right\|^2$$

Developments

- well-posed formulation of ∞-dim SBDMs
- convergence guarantee as $d \to \infty$
- towards multilevel training

Related findings: Kovachki, Marzouk, ...

T. Yang, P. Hagemann, S. Mildenberger, LR, G. Steidl
ML Diffusion: ∞-dim SBDM
arXiv: 2303:04772, 2023

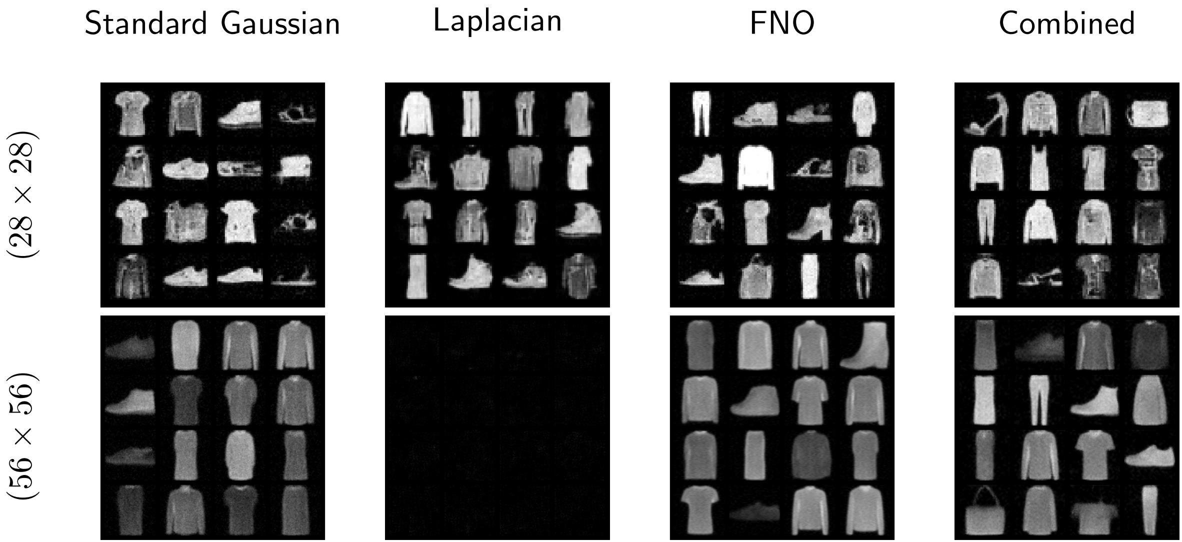
Comparisons of Networks and Priors

$s_\theta(t, X_t)$ modeled as UNET, trained on 28^2 (top row). Columns are different priors.

architecture + prior important to generalize to higher resolution

Comparisons of Networks and Priors

$s_\theta(t, X_t)$ modeled as FNO, trained on 28^2 (top row). Columns are different priors.



architecture + prior important to generalize to higher resolution

Optimal Control

Hamilton Jacobi Bellman and Pontryagin Max Principle

Consider value function of (stochastic) optimal control problem

$$\Phi(t,\mathbf{x}) = \min_{\mathbf{u}} \{J_{t,\mathbf{x}}[\mathbf{u}], \quad \text{subject to} \quad d\mathbf{z}(s) = f(\mathbf{z},\mathbf{u})ds + \sigma dW, \quad \mathbf{z}(t) = \mathbf{x}\}$$

Quick facts from optimal control theory:

1. feedback form relates optimal control and value function (PMP)

$$\mathbf{u}^*(s) \in \operatorname{argmax}_{\mathbf{u}} \mathcal{H}(s, \mathbf{z}(s), \mathbf{p}(s), \mathbf{M}(s), \mathbf{u})$$

   - Hamiltonian $\mathcal{H}(s,\mathbf{z},\mathbf{p},\mathbf{M},\mathbf{u}) = \frac{1}{2}\mathrm{tr}(\sigma M) + \mathbf{p}^\top f(s,\mathbf{z},\mathbf{u}) - L(s,\mathbf{z},\mathbf{u})$
   - $\mathbf{p}(s) = \nabla\Phi(s,\mathbf{z}(s))$ and $\mathbf{M}(s) = -\sigma\nabla^2\Phi(s,\mathbf{z}(s))$

2. value function satisfies HJB

$$-\partial_s\Phi(s,\mathbf{x}) + \sup_{\mathbf{u}} \mathcal{H}(s,\mathbf{x},-\nabla\Phi(s,\mathbf{x}),-\sigma\nabla^2\Phi(s,\mathbf{z}),\mathbf{u}) = 0$$

$$\Phi(T,\mathbf{x}) = g(\mathbf{x})$$

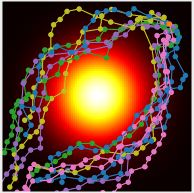
Challenges: fwd/bwd structure, nonlinearity, regularity, high-dimensionality,. . .

Neural Network Algorithms for Stochastic Optimal Control

Stochastic Optimal Control

SDE dynamics

$$d\mathbf{z}(s) = f(\mathbf{z}, \mathbf{u})ds + \sigma dW,$$



where $s \in (t, T)$ and

- $\mathbf{z}(s) \in \mathbb{R}^d$, $\mathbf{u}(s) \in \mathbb{R}^n$
- $\mathbf{z}(t) = \mathbf{x} \sim \mu$

Idea: Learn policy that minimizes

$$J_{t,\mathbf{x}}[\mathbf{u}] = \mathbb{E}\left[\int_t^T L(s, \mathbf{z}(s), \mathbf{u}(s))ds + g(\mathbf{z}(T))\right]$$

through value function Φ (feedback form).

X. Li, D. Verma, LR
A Neural Network Approach for SOC
arXiv:2209:13104, accepted at SIAM SISC, 2024

Contributions:

- inform sampling by PMP
- consistent with method of characteristics when $\sigma = 0$
- min objective function and HJB loss

Numerical Evaluations:

- comparison with FEM for $d = 2$
  - $\approx 2\%$ rel. error for Φ at $t = 0$
- comparison with neural solvers for semilinear elliptic PDEs in $d = 100$
  - faster convergence for benchmark problem
  - 2x smaller error for modified problem
- robust control of quadcopters
  - outperforms deterministic solver

Amortizing PDE Controls with HJB, PMP

Source Mitigation Problem

$$\min_{u,a} \int_0^T L(t,u,a)dt + G(u(T))$$

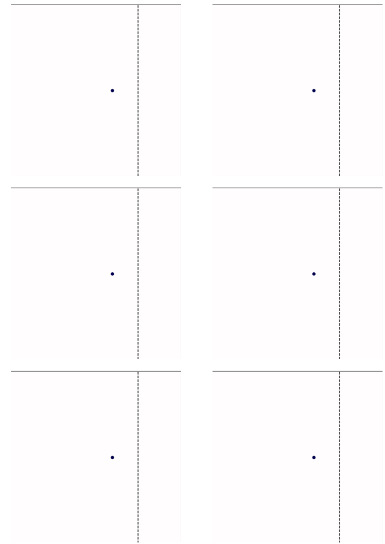
subject to

$$\partial_t u = \Delta u - v^\top \nabla u + f - g(a)$$

- u - concentration of pollutant
- v - velocity
- f - source ●
- g - sink ● parameterized by a

Goal: Learn policy

$$a^*(t) = p(t,u,v,f,g)$$



Comparison with Reinforcement Learning

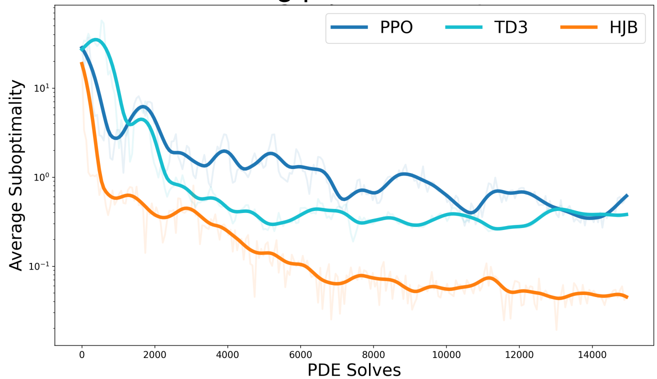
HJB Approach:

- new CNN architecture
- FEniCS to solve PDE
- similar training as before

RL Approach:

- actor/critic approach
- critic architecture similar to HJB
- two training approaches
  - Proximal Policy Optimization
  - Temporal Difference
- difficult hyperparameter tuning

Training performance:



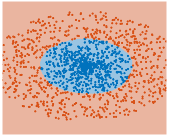
HJB: ☀ accuracy ☀ fewer PDE solves
☁ intrusive

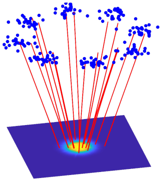
D. Verma, N. Winovich, LR, B v Bloemen Waanders
NNs for Parameterized Optimal Control
arXiv:2402.10033

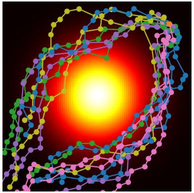
Summary

Σ: Diff Eq for Continuous-Time Deep Learning

- **Intro to Continuous-Time Deep Learning**
  - ResNet and Neural ODEs
  - Training via Inverse Problems / Optimal Control
  - A PDE Perspective for Supervised Learning
- **Generative Modeling**
  - (Conditional) Continuous Normalizing Flows
  - Application in Simulation Based Inference
  - Score-Based Diffusion Models
- **Optimal Control**
  - High-dimensional HJB Equations
  - Amortized PDE control







LR
DiffEq for Continuous-Time Deep Learning
AMS Notices, arXiv:2401.03965

Z. Wang, D. Verma, R. Baptista, Y. Marzouk, LR
NNs for COT and Bayesian Inference
arXiv preprint 2310.16975, 2023

T. Yang, P. Hagemann, S. Mildenberger, LR, G. Steidl
ML Diffusion: ∞-dim SBDM
arXiv: 2303:04772, 2023

References

- Behrmann, Jens et al. (2019). "Invertible residual networks". In: *International Conference on Machine Learning*, pp. 573–582.
- Benning, Martin et al. (2019). "Deep learning as optimal control problems: models and numerical methods". In: *arXiv preprint arXiv:1904.05657*.
- Bottou, L et al. (2018). "Optimization methods for large-scale machine learning". In: *SIAM Journal on Mathematics of Data Science* 60.2, pp. 223–311.
- Celledoni, Elena et al. (2020). *Structure preserving deep learning*. arXiv: 2006.03364 [cs.LG].
- Chang, Bo et al. (Oct. 2017). "Multi-level Residual Networks from Dynamical Systems View". In: *arXiv.org*. arXiv: 1710.10348v2 [stat.ML].
- Chang, Bo et al. (2018). "Reversible architectures for arbitrarily deep residual neural networks". In: *Thirty-Second AAAI Conference on Artificial Intelligence*, pp. 1–8.
- Chen, Ricky TQ et al. (2019). "Residual flows for invertible generative modeling". In: *Advances in Neural Information Processing Systems*, pp. 9916–9926.
- Chen, Tian Qi et al. (June 2018). "Neural Ordinary Differential Equations". In: *NeurIPS*.
- Dupont, Emilien et al. (2019). "Augmented Neural ODEs". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 3140–3150. URL: http://papers.nips.cc/paper/8577-augmented-neural-odes.pdf.
- E, Weinan (Mar. 2017). "A Proposal on Machine Learning via Dynamical Systems". In: *Communications in Mathematics and Statistics* 5.1, pp. 1–11.
- Finlay, Chris et al. (2020). "How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization". In: *International Conference on Machine Learning (ICML)*, pp. 3154–3164.

References (cont.)

- Gholami, Amir et al. (Feb. 2019). "ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs". In: *arXiv.org*. arXiv: `1902.10298v1 [cs.LG]`.
- González-García, R et al. (Mar. 1998). "Identification of distributed parameter systems: A neural net based approach". In: *Computers Chem Engn.* 22, S965–S968.
- Grathwohl, Will et al. (2018). "Ffjord: Free-form continuous dynamics for scalable reversible generative models". In: *arXiv preprint arXiv:1810.01367*.
- Haber, Eldad and Lars Ruthotto (2017). "Stable architectures for deep neural networks". In: *Inverse Problems* 34.1, pp. 1–22.
- He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
- Li, H et al. (2018). "Visualizing the loss landscape of neural nets". In: *Advances in Neural Information Processing Systems*.
- Li, Qianxiao and Shuji Hao (2018). "An optimal control approach to deep learning and applications to discrete-weight neural networks". In: *arXiv preprint arXiv:1803.01299*.
- Li, Qianxiao et al. (2017). "Maximum principle based algorithms for deep learning". In: *The Journal of Machine Learning Research* 18.1, pp. 5998–6026.
- Lu, Yiping et al. (Oct. 2017). "Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations". In: *arXiv.org*. arXiv: `1710.10121v2 [cs.CV]`.
- Rico-Martínez, R et al. (1992). "Discrete- vs. Continuous-time Nonlinear Signal Processing of Cu Electrodissolution Data". In: *Chemical Engineering Communications* 118.1, pp. 25–48.
- Ruthotto, Lars and Eldad Haber (2020). "Deep neural networks motivated by partial differential equations". In: *Journal of Mathematical Imaging and Vision* 62.3, pp. 352–364.

References (cont.)

- Thorpe, Matthew and Yves van Gennip (2018). "Deep limits of residual neural networks". In: *arXiv preprint arXiv:1810.11741*.
- Yang, L. and G. E. Karniadakis (2020). "Potential Flow Generator With L_2 Optimal Transport Regularity for Generative Models". In: *IEEE Transactions on Neural Networks and Learning Systems*.
- Zhang, Linfeng et al. (2018). "Monge-Ampère Flow for Generative Modeling". In: *arXiv:1809.10188*.