Continuous-Time Deep Learning for Generative AI and High-Dim PDEs

54th John H. Barrett Memorial Lectures

University of Tennessee, Knoxville

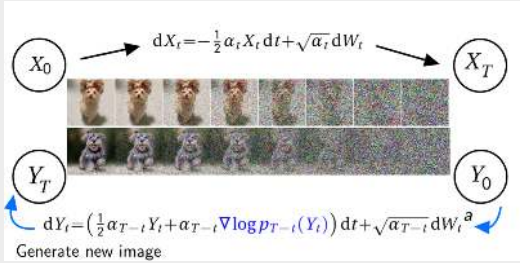
Lars Ruthotto

Departments of Mathematics and Computer Science
Emory University

lruthotto@emory.edu
larsruthotto

Differential Equations $\rightleftharpoons$ Modern AI

Ex 1: Score-Based Diffusion



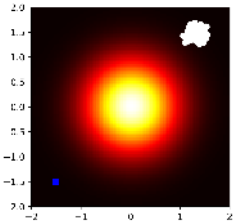
Given samples $x_1, x_2, \ldots \sim \rho_{\text{data}}$; schedule $\beta(t)$ (e.g., VP-SDE)

Learn score $s_\theta(x,t) \approx \nabla \log p_t(x)$

Sample via reverse-time SDE

$$dx = \left[-\tfrac{1}{2}\beta(t)\,x - \beta(t)\,s_\theta(x,t)\right]dt + \sqrt{\beta(t)}\,d\bar{W}$$

Ex 2: Stochastic Optimal Control



Given dynamics $dz = f(z,u)\,ds + \sigma\,dW$; cost L, g

Learn value function $\Phi_\theta(t,x)$ via HJB residual + terminal match

Sample via PMP-driven SDE

$$dz = -\nabla\Phi_\theta(t,z)\,ds + \sigma\,dW$$

Continuous-Time Deep Learning

ResNets as Discretized ODEs

Residual networks He et al. 2016: forward propagation

$$z_{j+1} = z_j + h\,\sigma(K_j z_j + b_j), \quad j = 0, \ldots, N-1$$

Continuous limit E 2017; Haber and Ruthotto 2017 as $h \to 0$, $N \to \infty$:

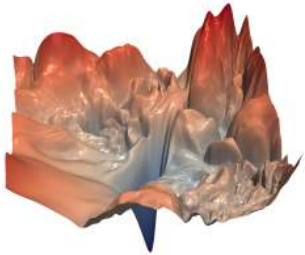
$$\frac{d}{dt} z(t) = f(z(t), \theta(t)), \quad t \in (0, T], \quad z(0) = x$$

ResNet = forward-Euler discretization of an ODE.

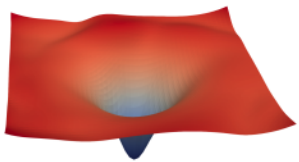
Why this perspective helps

- stability analysis via ODE theory
- structured architectures (Hamiltonian, antisymmetric)
- adaptive time integrators $\to$ adaptive depth

loss landscape Li et al. 2018



plain 56-layer



56-layer ResNet

depth becomes time – and the ODE toolbox becomes available

Neural Ordinary Differential Equations

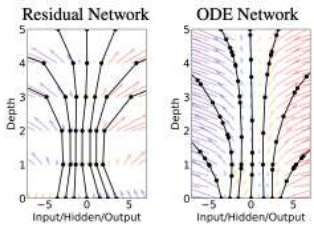
Main idea Chen et al. 2018b

$$\frac{d}{dt}z(t) = f(z(t), \theta(t)), \quad z(0) = x$$

forward propagation via an adaptive ODE solver; differentiation via reverse-mode AD through the chosen time stepper (**DTO**, revisited later).

What it brought

- depth-adaptive ODE solvers
- memory savings via checkpointing / reverse-time recovery
- bridge to scientific computing



from Chen et al. 2018b


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

MIT Tech. Rev., 2018

popularized continuous-time models in the ML community

Training as Optimal Control

Deep learning as a constrained optimization problem

$$\begin{aligned} \text{minimize}_{\theta,W,\mu} \quad & \mathbb{E}\big[\text{loss}(g(Wz(T)+\mu),c)\big] + \text{reg}[\theta,W,\mu] \\ \text{subject to} \quad & \tfrac{d}{dt}z(t) = f(z(t),\theta(t)), \quad z(0) = x \end{aligned}$$

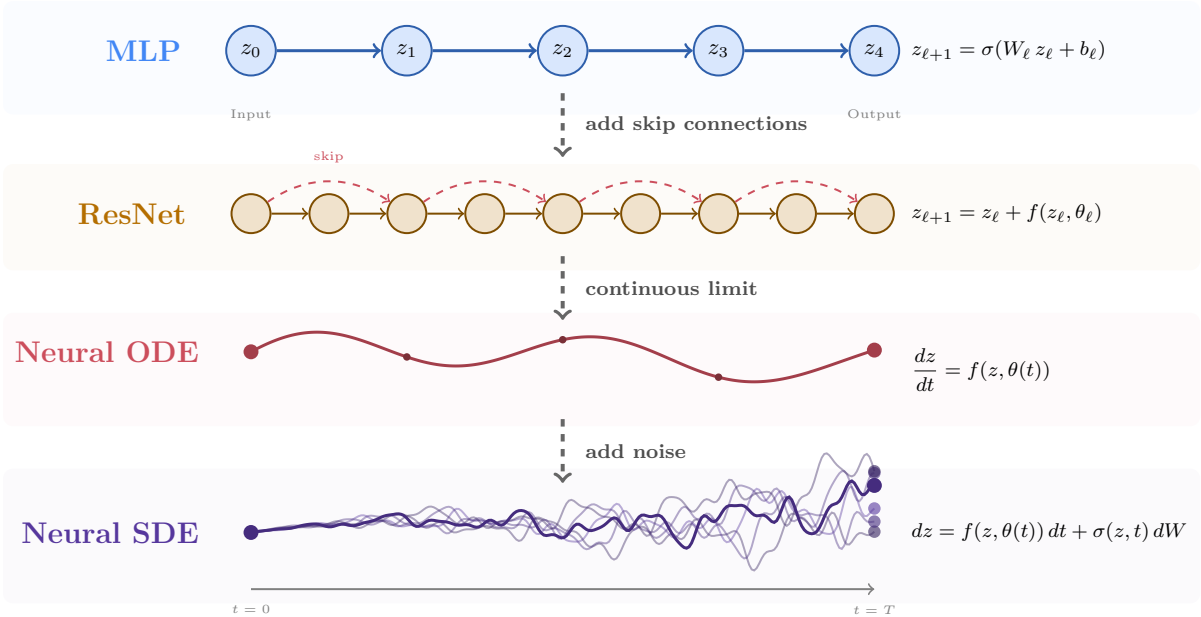
Two roads to the gradient

- **Discretize-then-Optimize (DTO)** – choose a time integrator, then differentiate via reverse-mode AD
- **Optimize-then-Discretize (OTD)** – derive the continuous adjoint, then discretize

DTO is what backpropagation does; computational cost and Stratonovich/Itô subtleties returned to later.

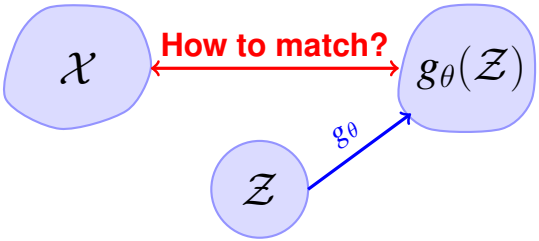
training = state-constrained optimal control

Continuous-Time Models for Modern AI



Generative AI: Learning Distributions

Generative Modeling as Distribution Matching



Mathematical framework (L.R. & E. Haber, 2021)

- Learn generator $g_\theta : \mathbb{R}^q \to \mathbb{R}^n$ that transforms latent $\mathcal{Z}$ to match data $\mathcal{X}$
- **Challenges**:
  - n typically large
  - $\mathcal{X}$ multimodal, disjoint support
- **Core problem**: match distributions

$$p_{g_\theta(\mathcal{Z})}(x) \approx p_{\mathcal{X}}(x)$$

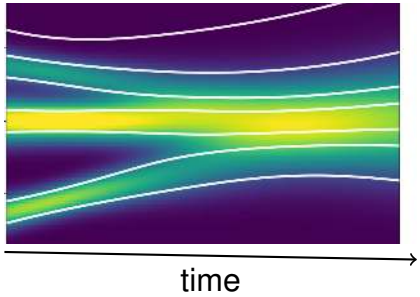
- **Today**: distribution matching with PDEs

generative modeling = matching high-dimensional distributions

The Master Equation for Generative Modeling

$$\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t v_t) = 0, \quad \rho_0 = p_{\mathcal{X}}, \quad \rho_1 = p_{\mathcal{Z}}$$

Given velocity v_t, determines the density path ρ_t through probability space



Strategies to construct feasible (ρ_t, v_t) pairs

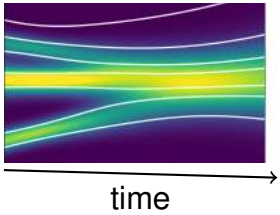
1. **Continuous Normalizing Flows**: optimize v_t so that $\rho_1 \approx p_{\mathcal{Z}}$
2. **Optimal Transport**: regularize v_t by kinetic energy
3. **Flow Matching**: construct (ρ_t, v_t) from conditional flows (superposition)
4. Score-based Diffusion: Fokker–Planck $\rightarrow$ log transform $\rightarrow$ another (ρ_t, v_t) pair

From Continuity Equation to Neural ODE

$$\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t v_t) = 0, \quad t \in (0,1], \quad \rho_0 = p_{\mathcal{X}}$$

- Define **characteristic curves** (particle trajectories):

$$\frac{d}{dt} z(x,t) = v_t(z(x,t),t), \quad z(x,0) = x \sim p_{\mathcal{X}}$$



- Jacobian log-determinant along the trajectory:

$$\frac{d}{dt} \log\det \nabla_x z(x,t) = \nabla \cdot v_t(z(x,t),t)$$

Key Insight

- **PDE** (continuity equation) $\iff$ **System of ODEs** (particle trajectories)
- If particles follow ODEs, density automatically satisfies PDE!

CNF Idea Chen et al. 2018a: Parameterize velocity field v_t as neural network $v_\theta(z,t)$

PDE transport $\Rightarrow$ Method of characteristics $\Rightarrow$ Neural ODE

Continuous Normalizing Flows (CNF)

Likelihood Maximization

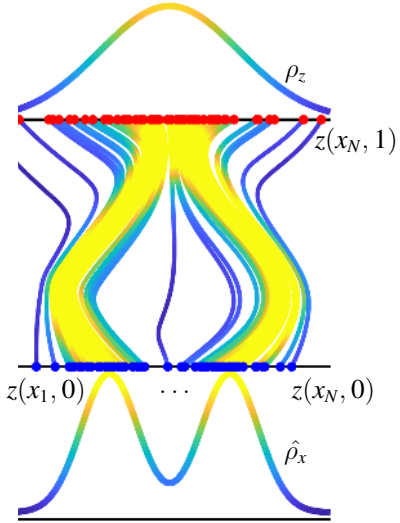
Given samples $x_1, x_2, \ldots, x_N \in \mathbb{R}^d$, find a velocity v that maximizes the likelihood of the samples w.r.t. the push-forward of the standard normal distribution ρ_z, i.e.,

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

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

with $z(x_k,0) = x_k$ and $l(x_k,0) = 0$ for $k = 1,2,\ldots,N$.

Here: $l(x_k,1) = \log\det(\nabla z(x_k,1))$ – see Grathwohl et al. 2018.



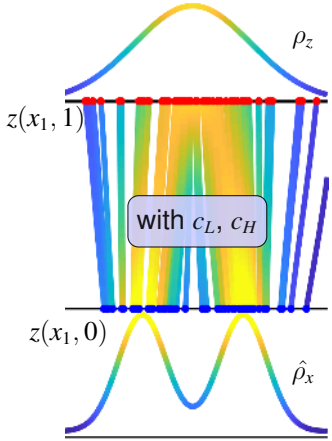
OT-Flow: Regularized CNF

Given samples $x_1, x_2, \ldots, x_N \sim \rho_x$, 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.,

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

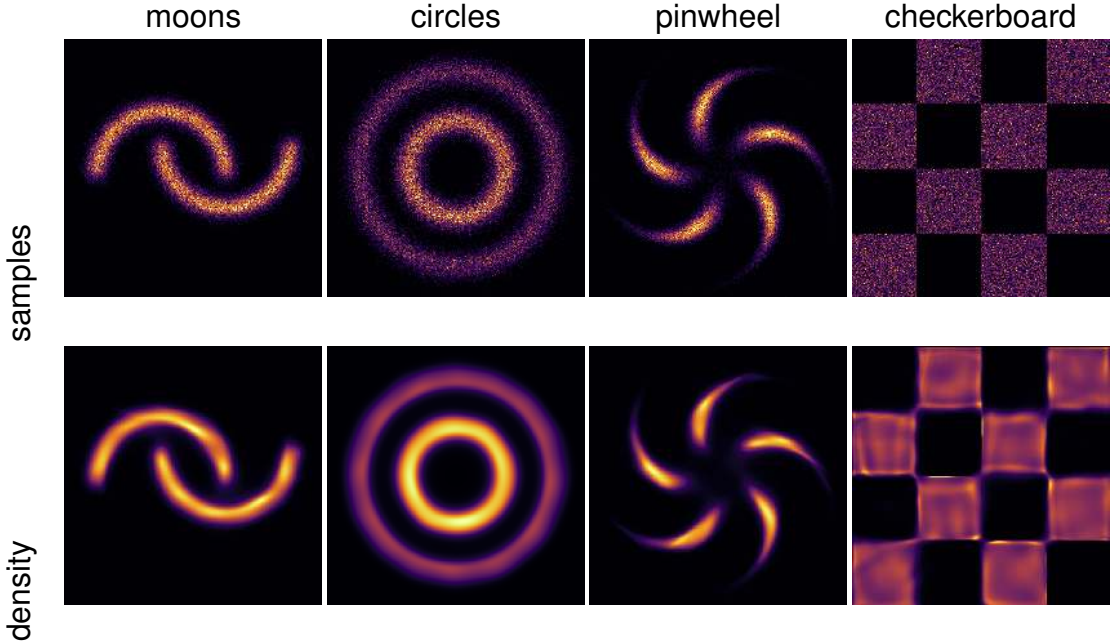
$$\text{subject to} \quad \frac{d}{dt}\begin{pmatrix} z(x,t) \\ l(x,t) \\ c_{\mathrm{L}}(x,t) \\ c_{\mathrm{H}}(x,t) \end{pmatrix} = \begin{pmatrix} -\nabla\Phi(z(x,t),t) \\ -\Delta\Phi(z(x,t),t) \\ \frac{1}{2}\|\nabla\Phi(z(x,t),t)\|^2 \\ \left|\frac{d}{dt}\Phi(z(x,t),t) + \frac{1}{2}\|\nabla\Phi(z(x,t),t)\|^2\right|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

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



OT $\rightsquigarrow$ uniqueness, more efficient time integration Onken et al. 2020

OT-Flow: Two-Dimensional Examples



Mean Field Games: Beyond OT-Flow

Setting Ruthotto et al. 2020 many rational agents; cost depends on population density ρ_t.

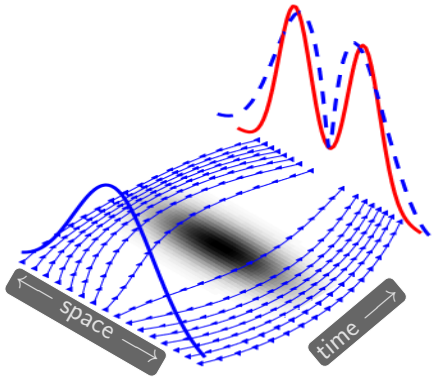
Cost functional

$$J[u] = \mathbb{E}\left[\int_0^T \big(L(z,u) + F(\rho_t)\big)\, dt + g(z(T)) + G(\rho_T)\right]$$

subject to deterministic dynamics $\frac{d}{dt}z = f(z,u)$, $z(0) \sim \rho_0$.

Population-density terms

- $F(\rho_t)$: congestion / collision cost
- $G(\rho_T)$: terminal interaction
- extends OT-Flow with density-dependent costs



crowd motion: agents reach target while avoiding obstacles & congestion

OT-Flow $\rightarrow$ high-dim MFG via Lagrangian transport

Computational Algorithms

Optimize-Discretize vs. Discretize-Optimize

Compare optimal control approaches for learning problems

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

where z_N in F approximates $z(T)$ given by

$$\frac{d}{dt}z(t) = f(z(t),\theta^{\text{ODE}}(t)), \quad t \in (0,T], \quad z(0) = x$$

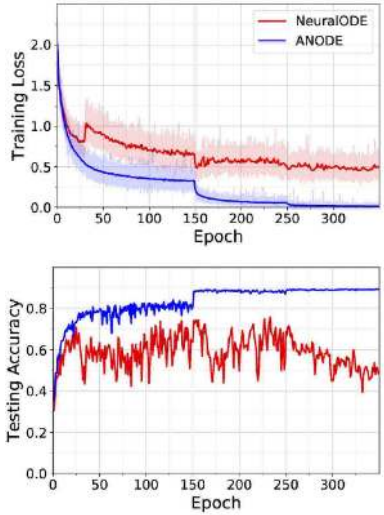
$O \to D$: Optimize-Discretize (Neural ODE)

1. keep θ^{ODE}, z continuous in time
2. Euler–Lagrange equations $\rightsquigarrow$ adjoint equation
3. use adaptive time integrators in optimization

$D \to O$: Discretize-Optimize (ANODE)

1. discretize θ^{ODE}, z in time (could use different grids)
2. differentiate discrete problem $\rightsquigarrow$ backpropagation
3. keep discretization fixed during optimization

example (image classification)



more examples in Gholami et al. 2019; Onken and Ruthotto 2020

advice: use $D \to O$ – accurate gradients, fixed costs, robust convergence

Mixed Precision for Neural ODEs: Setup and Cost

Continuous-time model: for $t \in (0, T]$,

$$\frac{d}{dt} z(t) = f(z(t), \theta(t))$$

$F(x, \theta) := z(T)$. Training:

$$\min_{\theta} \ \mathbb{E}_{x \sim \pi_{\text{data}}} [\ell(F(x, \theta))] + \frac{\alpha}{2} \|\theta\|^2$$

Examples:

- residual networks $\cong$ explicit Euler
- continuous normalizing flows
- NN surrogates for control Ruthotto 2024

Why MPT is attractive:

- f = a neural net evaluated *many* times
- activations dominate GPU memory
- training-time bottleneck on large images

Why MPT is risky:

- N explicit time steps $\Rightarrow$ N rounding events
- adjoint backprop traverses them in reverse
- $\|a(t)\|$ can span the full FP16 range

Naive `autocast` on an ODE solver $\Rightarrow$ NaNs after a few steps

Forward Solve: Quantize, Accumulate High

Mixed-precision time step:

$$y_{\mathrm{H}}^{(i+1)} = y_{\mathrm{H}}^{(i)} \oplus_{\mathrm{H}} h_i \, Q_{\mathrm{H}}\big(\Phi\big(f,\, Q_{\mathrm{L}}(y_{\mathrm{H}}^{(i)}),\, t_i,\, h_i,\, Q_{\mathrm{L}}(\theta_{\mathrm{H}})\big)\big)$$

Φ = any explicit RK step (Euler, RK4, ...).

Why it works:

- the network evaluation $\Phi(f, \ldots)$ runs in low precision (tensor cores)
- the state y_{H} is accumulated in high precision
- only the *stored* trajectory $\{Q_{\mathrm{L}}(y_{\mathrm{H}}^{(i)})\}$ is kept in 16-bit
  $\Rightarrow \approx 2\times$ memory savings

Forward Algorithm

1. $y_{\mathrm{H}} \leftarrow Q_{\mathrm{H}}(x)$
2. **for** $i = 0, \ldots, N-1$:
3. $\quad h_i \leftarrow t_{i+1} - t_i$
4. $\quad dy \leftarrow \Phi(f, Q_{\mathrm{L}}(y_{\mathrm{H}}), t_i, h_i, Q_{\mathrm{L}}(\theta))$
5. $\quad y_{\mathrm{H}} \leftarrow y_{\mathrm{H}} \oplus_{\mathrm{H}} h_i \, Q_{\mathrm{H}}(dy)$
6. $\quad$ store $Q_{\mathrm{L}}(y_{\mathrm{H}})$
7. **end for**
8. **return** $\{Q_{\mathrm{L}}(y_i)\}$

Storage in 16-bit, accumulation in 32-bit — decoupled from `autocast`

Backward Solve: Dynamic Adjoint Scaling

Adjoint recursion:

$$[da^\top, dt, dh, d\theta^\top] = Q_{\mathrm{L}}\big(S_i\, a_{\mathrm{H}}\big)^\top J_\Phi(\cdot)$$

$$a_{\mathrm{H}}{}^{(i-1)} = a_{\mathrm{H}}{}^{(i)} \oplus_{\mathrm{H}} (h_i/S_i)\, Q_{\mathrm{H}}(da)$$

Dynamic per-step scaling S_i:

- initialize: $S_N = 2^{\lfloor -\log_2(u_{\mathrm{L}}\|a_{\mathrm{H}}\|)\rfloor}$
- on overflow $\Rightarrow$ halve S_i and retry
- on safe margin $\Rightarrow$ double S_{i+1}

$\Rightarrow$ adjoint always near the top of the FP16 range, never under/overflows.

Backward Algorithm

$a_{\mathrm{H}} \leftarrow Q_{\mathrm{H}}(\partial_{y_N}\ell)$, $g \leftarrow 0$
$S_N \leftarrow 2^{\lfloor -\log_2(u_{\mathrm{L}}\|a_{\mathrm{H}}\|)\rfloor}$
for $i = N-1, \dots, 0$:
  $dy \leftarrow \Phi(f, y_i, t_i, h_i, \theta)$
  for $k = 1, \dots, k_{\max}$:
    VJP: $[da^\top, \dots] \leftarrow Q_{\mathrm{L}}(S_i a_{\mathrm{H}})^\top J_\Phi$
    if finite **then break**
    $S_i \leftarrow S_i/2$
  $a_{\mathrm{H}} \leftarrow a_{\mathrm{H}} \oplus_{\mathrm{H}} (h_i/S_i)\, Q_{\mathrm{H}}(da)$
  $g \leftarrow g \oplus_{\mathrm{H}} (h_i/S_i)\, Q_{\mathrm{H}}(d\theta)$
  if safe **then** $S_{i-1} \leftarrow 2S_i$
end for; **return** a_{H}, g

No global loss-scale hyperparameter, no manual per-model tuning

Roundoff Analysis: Errors Do Not Pile Up with N

Theorem Celledoni et al. 2025

Under standard regularity assumptions on f and Φ, the mixed-precision forward solve and custom backward pass produce, for every $i \leq N$,

$$\begin{aligned} \|\delta_{y_i}\|_\infty &= \mathcal{O}(u_{\mathrm{L}}) + \mathcal{O}(u_{\mathrm{H}}/h), \\ \|\delta_{a_i}\|_\infty &= \mathcal{O}(u_{\mathrm{L}}) + \mathcal{O}(u_{\mathrm{H}}/h) + \kappa \|\delta_y\|_\infty, \\ \|\delta_{\nabla_\theta \mathcal{L}}\|_\infty &= \mathcal{O}(u_{\mathrm{L}}) + \mathcal{O}(u_{\mathrm{H}}/h). \end{aligned}$$

- dominant term $\mathcal{O}(u_{\mathrm{L}})$ is *independent* of step count N
- the $\mathcal{O}(u_{\mathrm{H}}/h)$ term hurts only for absurdly small h
- **safe regime:** $h \gg \sqrt{u_{\mathrm{H}}} \approx 10^{-3.5}$
- **danger zone:** $h \ll 10^{-4} \Rightarrow$ accumulate in FP64

No error blowup with N — the key property that makes MPT viable for ODEs

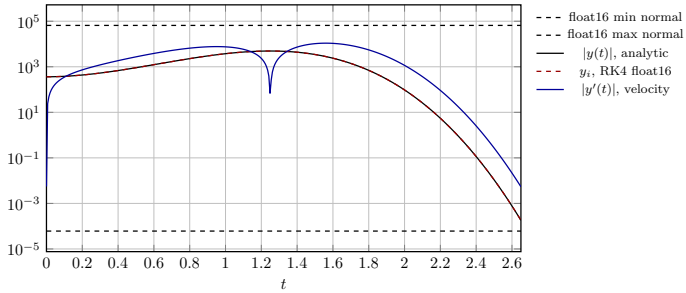
Validation: Adjoint Scaling on Stiff ODE

RK4 with $N = 400$ steps on

$$y'(t) = -\lambda(t)\, y(t), \quad \lambda(t) = \theta_1 t^2 + \theta_2 t + \theta_3$$

$T = 2.65$, $\theta = (8, -11, 2^{-16})$,
$\ell(y_N) = \frac{1}{2}\|y_N\|^2$.

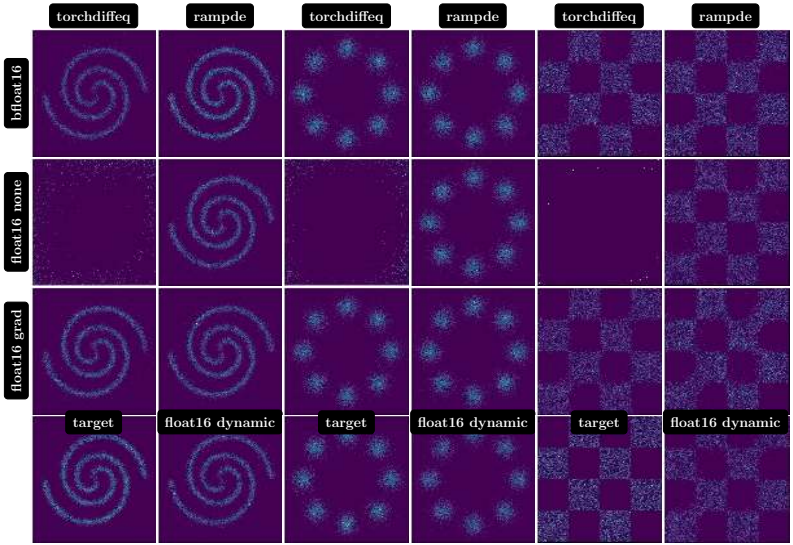
Why hard: $|y(t)|$ and $|y'(t)|$ span the *full* FP16 dynamic range.



dtype	scaling	RE $y(T)$	RE ∂_{y_0}	RE ∂_{θ_1}
`float32`	none	7.0e-5	1.4e-4	1.3e-4
`float32`	dynamic	7.0e-5	1.6e-4	1.3e-4
`float16`	none	3.7e-3	2.0e+3	1.0e+0
`float16`	dynamic	3.7e-3	5.9e-3	6.0e-3

Dynamic scaling recovers ∼6 orders of magnitude on the FP16 gradient

CNF in 2D: Same Quality at fp16 with `rampde`



3 datasets $\times$ {`torchdiffeq`, `rampde`}, multiple precisions. fp16 without scaling: breaks `torchdiffeq`; `rampde` dynamic scaling matches fp32.

BSDS300 (d=63): Memory Cuts & Stability

OT-Flow density estimation on natural-image patches RK4, 16 steps, batch 512

precision	solver	scaling	val NLL	HJB	max mem (GB)	train (s)
tfloat32	`torchdiffeq`	–	−163.7	1.08	**18.9**	6016
tfloat32	`rampde`	–	−163.5	1.04	**1.6**	6230
bfloat16	`rampde`	–	−163.4	1.00	**1.5**	6569
float16	`torchdiffeq`	none	fails	–	–	–
float16	`torchdiffeq`	grad	fails	–	–	–
float16	`rampde`	dynamic	−164.0	1.09	**1.5**	9141

The story in three numbers

- **12**× peak-memory cut: 18.9 GB → 1.5 GB
- `torchdiffeq` at fp16: NaN at init, both scalings
- `rampde` + dynamic scaling: best NLL at lowest memory

high-dim CNF fits in 1.5 GB – without losing accuracy

STL-10: MP Matches FP32 Accuracy, Faster

- STL-10: 13k images, 96×96, 10 classes
- Parabolic-CNN, 12.6M params, 3 neural-ODE blocks (RK4)
- SGD, 160 epochs, batch 16; RTX A6000

Test accuracy (`rampde`):

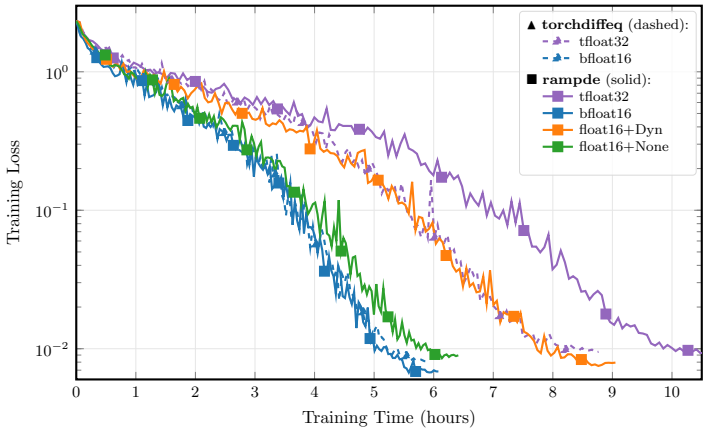
- FP32: 0.786 TF32: 0.792
- BF16: 0.795 FP16 (dyn): 0.778

Memory & speed (BF16 vs. TF32):

- peak memory: $4.3\text{GB}\rightarrow2.2\text{GB}$
- forward: $0.27\text{s}\rightarrow0.15\text{s}$ $(\approx\mathbf{1.8}\times)$

vs. `torchdiffeq`:

- FP32 mem: $21.5\text{GB}\rightarrow4.5\text{GB}$ $(\mathbf{5}\times)$
- FP16 fails (NaN even w/ grad-scaling)



Training loss vs. wall-clock; `rampde` (solid) vs. `torchdiffeq` (dashed).

Differentiating Through SDEs

Setup Leburu et al. 2026 gradient of $J(x_0) = \mathbb{E}[\Phi(X_T)]$ for Itô SDE.

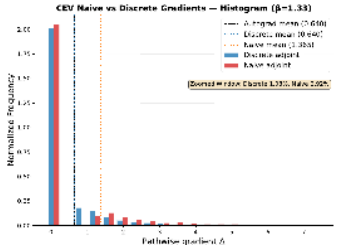
Two tales

- **DTO** – Euler–Maruyama (Itô) or Heun (Stratonovich), AD through discrete steps
- **OTD** – continuous adjoint SDE, then discretize

Leburu–Nurbekyan–R. 2026

- DTO is *exact* for the discrete objective in both Itô and Stratonovich
- naive OTD on Itô is biased when $\partial_x g$ depends on x
- fix: Itô $\to$ Stratonovich correction (Øksendal)

CEV model, $\beta = 1.33$

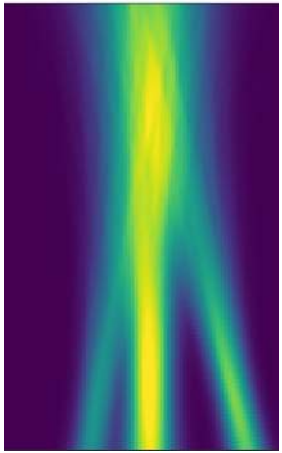


discrete adjoint $= 0.640$ (correct); naive $= 1.365$ (biased)

Flow Matching

Flow Matching: Feasible Paths via Superposition

$$\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t v_t) = 0, \quad \rho_0 = p_{\mathcal{X}}, \quad \rho_1 = p_{\mathcal{Z}}$$



Special Case: Two Dirac Deltas

For point pair $p_{\mathcal{X}} = \delta(x_0)$ and $p_{\mathcal{Z}} = \delta(x_1)$, OT map is

- Conditional path: $\psi_t(x_0, x_1) = (1-t)x_0 + tx_1$
- Conditional density: $\rho_t(\cdot|x_0, x_1) = \delta(x - \psi_t(x_0, x_1))$
- Conditional velocity: $u_t(x|x_0, x_1) = \frac{d\psi_t}{dt} = x_1 - x_0$

Superposition via Linearity

Sample $x_0 \sim p_{\mathcal{X}}$ and $x_1 \sim p_{\mathcal{Z}}$ independently.
By linearity of the PDE, the marginal density is:

$$\rho_t(x) = \int \delta(x - \psi_t(x_0, x_1)) p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1)\, dx_0\, dx_1 = \mathbb{E}_{x_0, x_1}[\rho_t(x|x_0, x_1)]$$

This gives a **feasible** probability path!

Question: How do we get the marginal velocity v_t?

From Conditional to Marginal Velocity

For each (x_0, x_1) the conditional density and conditional velocity satisfy

$$\frac{\partial \rho_t(x|x_0,x_1)}{\partial t} + \nabla \cdot (\rho_t(x|x_0,x_1) u_t(x|x_0,x_1)) = 0$$

Step 1: Take expectation over $(x_0, x_1) \sim p_{\mathcal{X}} \times p_{\mathcal{Z}}$

$$\int \frac{\partial \rho_t(x|x_0,x_1)}{\partial t} p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1) dx_0 dx_1 + \int \nabla \cdot (\rho_t(x|x_0,x_1) u_t(x|x_0,x_1)) p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1) dx_0 dx_1 = 0$$

Step 2: Interchange differentiation and integration

$$\frac{\partial}{\partial t}\left[\int \rho_t(x|x_0,x_1) p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1) dx_0 dx_1\right] + \nabla \cdot \left[\int \rho_t(x|x_0,x_1) u_t(x|x_0,x_1) p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1) dx_0 dx_1\right] = 0$$

Step 3: Identify Coefficients. This is $\frac{\partial \rho_t}{\partial t} + \nabla \cdot (\rho_t v_t) = 0$ with:

$$v_t(x) = \frac{\int u_t(x|x_0,x_1) \rho_t(x|x_0,x_1) p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1) dx_0 dx_1}{\int \rho_t(x|x_0,x_1) p_{\mathcal{X}}(x_0) p_{\mathcal{Z}}(x_1) dx_0 dx_1}$$

marginal velocity is weighted average of conditional velocities

From Expectation to Function Approximation

Idea: Regression to Compute Expectation Use a neural network $v_\theta(x,t)$ and minimize:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,x_0,x_1}[\|v_\theta(\psi_t(x_0,x_1),t) - u_t(x|x_0,x_1)\|^2]$$

where $u_t(x|x_0,x_1) = x_1 - x_0$ is known analytically!

Why This Works: A Simple Example

Two data points with the same x but different y values: (x,y_1) and (x,y_2)

Minimize: $L(v) = |v(x)-y_1|^2 + |v(x)-y_2|^2$

Optimality: $\frac{\partial L}{\partial v(x)} = 2(v(x)-y_1) + 2(v(x)-y_2) = 0 \quad \Rightarrow \quad v^*(x) = \frac{y_1+y_2}{2}$

Key insight: $v^*(x)$ is the **average** of y-values at x = conditional expectation!

Conditional Flow Matching Training

Training Objective Lipman et al. 2023

$$\mathcal{J}_{\text{CFM}}(\theta) = \mathbb{E}_{t,x_0,x_1}\left[\|v_\theta(\psi_t, t) - (x_1 - x_0)\|^2\right]$$

where the target velocity $u_t = x_1 - x_0$ is known analytically!

Training Procedure

1. Sample $x_0 \sim p_{\mathcal{X}}$, $x_1 \sim p_{\mathcal{Z}}$, $t \sim U[0,1]$
2. Compute path location: $\psi_t = (1-t)x_0 + tx_1$
3. Compute target velocity: $u_t = x_1 - x_0$
4. Compute prediction: $v_\theta(\psi_t, t)$
5. Minimize squared error with stochastic gradient descent

Advantages

- No ODE solve during training
- No trace computation
- Simple supervised learning (not adversarial or variational)

Sampling: Solve ODE with learned v_θ from $t = 1$ to $t = 0$ (same as in CNF)

supervised learning on analytically known conditional velocities

Discussion – Flow Matching

The Flow Matching Recipe

1. Construct conditional flows from point pairs (Dirac deltas)
2. Use superposition via linearity $\rightarrow$ marginal densities
3. Fit neural network to match conditional velocities (supervised learning)

Computational Advantages

- **Training**: No ODE solves, no trace estimation $\rightarrow$ significantly faster than CNF
- **Sampling**: Linear interpolation $\rightarrow$ fewer function evaluations
- **Simplicity**: Convexity in v_t, supervised learning

Trade-offs

- Produces **feasible** pairs (satisfies continuity equation) ✓
- Does NOT minimize Benamou-Brenier kinetic energy (not optimal)
- Construction beats optimization in high dimensions!

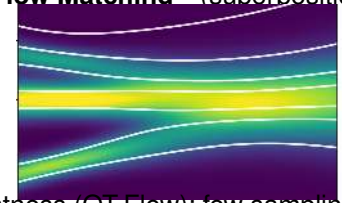
State-of-the-Art: Stable Diffusion 3, Sora, AlphaFold 3

the alternative path: skip time integration, regress on conditional velocities

Trajectories: Straight vs Curvy — Holy Grail

OT-Flow (min kinetic energy)

Flow Matching (superposition)



The tension terminal-density match (FM) vs. trajectory straightness (OT-Flow); few sampling steps need both.

Recent attempts – and what each still loses

- **Rectified flow** (Liu et al. 2023): iterate FM on straightened pairs – repeated training rounds
- **Consistency models** (Song et al. 2023): distill a multi-step sampler – drifts from marginal as step shrinks
- **OT-coupled FM** (Tong et al. 2024): mini-batch OT pairing – scales with pairing cost

straight + feasible in one shot: the open problem

Stochastic Optimal Control

Why SOC: Three Application Families

$$\min_u \mathbb{E}\left[G(z_T) + \int_t^T L(s, z_s, u_s)\, ds \right] \quad \text{s.t.} \quad dz_t = f(t, z_t, u_t)\, dt + \sigma(t, z_t)\, dW_t$$

Rare Event Analysis

Probe tails, quantify rare-event probabilities

components: f system dynamics, L trajectory penalty, G event indicator

examples: chaotic Lorenz, extreme weather, Burgers–Huxley

Bayesian Inverse Problems

Characterize posteriors from observations

components: f score / flow prior, L control penalty, G neg. log-likelihood

examples: inverse scattering, structural health, inverse design

Stochastic PDE Control

Control infinite-dim stochastic systems

components: f discretized SPDE, L tracking + control, G terminal error

examples: stochastic reaction-diffusion, trajectory opt, FitzHugh–Nagumo

common challenge: high-dim, complex dynamics, reliable training

HJB: The Master Equation for SOC

Consider the value function of the stochastic optimal control problem

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

Quick facts from optimal control theory

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

$$u^*(s) \in \text{argmax}_u \mathcal{H}(s, z(s), p(s), u)$$

with Hamiltonian $\mathcal{H}(s,z,p,u) = p^\top f(s,z,u) - L(s,z,u)$ and $p(s) = \nabla_z \Phi(s, z(s))$

2. value function Φ satisfies HJB

$$-\partial_s \Phi(s,x) - \tfrac{\sigma^2}{2}\Delta\Phi(s,x) + \sup_u \mathcal{H}(s,x,-\nabla_x\Phi,u) = 0$$
$$\Phi(T,x) = g(x)$$

HJB: the master equation of SOC

Solvers for High-Dim SOC: Four Families

Value function $\Phi(t,x) = \inf_u \mathbb{E}[J^u]$ satisfies HJB

$$-\partial_t \Phi + \sup_u \mathcal{H}(t,x,u,\nabla\Phi,\nabla^2\Phi) = 0$$

PINN-based HJB solvers

- learn Φ with NNs
- fixed sampling distribution
- loss: HJB residual

Sirignano and Spiliopoulos 2018; Raissi 2018

Forward–Backward SDE

- learn Φ or u with NNs
- fixed sampling distribution
- loss: Feynman–Kac

Han et al. 2018

Diffusion-Optimization

- learn u directly
- sample under current policy
- loss: various (matching, log-ratio)

Domingo-Enrich et al. 2024/25; Berner et al. 2024

Neural ODE/SDE Onken et al. 2022; Li et al. 2024

- learn Φ with NNs
- sample via PMP feedback
- loss: HJB residual + terminal

← today's focus

the sampling distribution is everything

NeuralSOC: SOC as a Learning Problem

Value Function Learning

Given dynamics $dz = f(z,u)\,ds + \sigma\,dW$ and costs L, g, find a value function $\Phi_\theta(t,x)$ that minimizes terminal cost + HJB residual + terminal match along the sampled forward roll-out, i.e.,

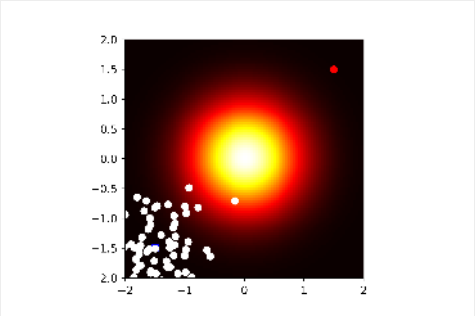
$$\begin{aligned}
&\text{minimize}_\theta \quad \mathbb{E}_{x\sim\rho_0}\left[\,\alpha\big(\ell(T)+g(z(T))\big)+\beta_1\,c_{\mathrm{HJ}}(T)+\beta_2\,|\Phi_\theta(T,z(T))-g(z(T))|\,\right]\\
&\text{subject to} \quad \frac{d}{ds}\begin{pmatrix} z(s)\\ \ell(s)\\ c_{\mathrm{HJ}}(s)\end{pmatrix} = \begin{pmatrix} -\nabla\Phi_\theta(s,z(s))\\ L(s,z,-\nabla\Phi_\theta)\\ \left|\partial_s\Phi_\theta+\mathcal{H}(s,z,-\nabla\Phi_\theta)+\frac{\sigma^2}{2}\Delta\Phi_\theta\right|\end{pmatrix} ds + \begin{pmatrix}\sigma\,dW\\ 0\\ 0\end{pmatrix}\\
&\qquad\qquad z(0)=x, \quad \ell(0)=0, \quad c_{\mathrm{HJ}}(0)=0
\end{aligned}$$

DTO via reverse-mode AD through the Euler–Maruyama discretization Li et al. 2024.

NeuralSOC = HJB residual + PMP-driven forward roll-out

Where Do We Sample?

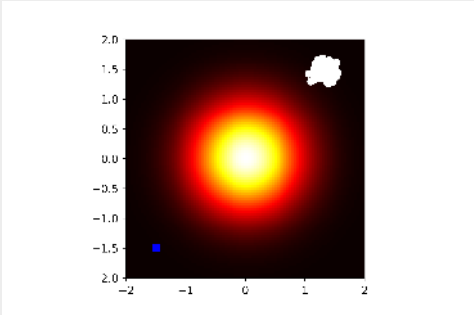
Pure random walk



Forward SDE $dz(s) = \sigma\, dW$

samples wander off the relevant region

PMP-driven drift



Forward SDE $dz(s) = -\nabla\Phi_\theta\, ds + \sigma\, dW$

drifts via current value-function estimate

sample along (approximately) optimal trajectories Li et al. 2024

100D Stress Test: Shifted Target

Setup $d = 100$, $\sigma = 2\sqrt{2}/5$

- dynamics $dz = u\,ds + \sigma\,dW$, $z(0) = 0$
- running cost $L(z, u) = \frac{1}{2}\|u\|^2$
- terminal cost

$$g(z) = \ln\left(\tfrac{1}{2}(1 + \|z - x_{\text{ref}}\|^2)\right)$$

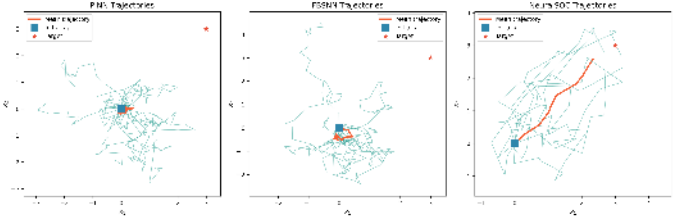
- target $x_{\text{ref}} = (3, 3, \ldots, 3)^\top$

Why this is hard

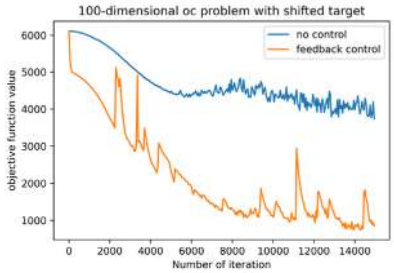
- pure Brownian sampling never reaches target region
- BSDE-style baselines Raissi 2018; Han et al. 2018 stall

Outcome PMP drift carries trajectories to target → semi-global

trajectories (2D projection): start 0, target $\star$ at $(3, 3)$

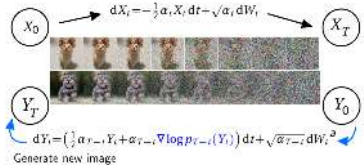


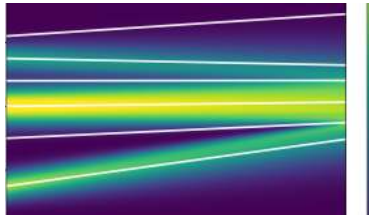
control objective vs. iteration

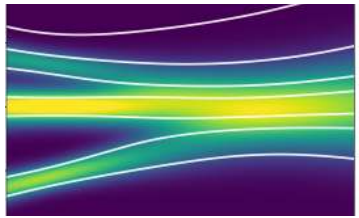


Summary

Σ: Continuous-Time DL for GenAI and High-Dim PDEs







- **Generative AI: CNF, OT-Flow**
  neural ODEs for density transport; OT-Flow potential Φ: closed-form trace, straight paths
- **Mean Field Games (extension)**
  same OT-Flow architecture solves coupled HJB/FP semi-globally, $d \gg 4$
- **Stochastic Optimal Control**
  HJB master equation, neural value function Φ_θ, PMP-driven sampling
- **Computational efficiency**
  DTO via AD + mixed precision (`torchmpnode`); Itô + Stratonovich (Leburu et al. 2026)
- **Matching for OT + SOC**
  extending FM ideas to OT-Flow and NeuralSOC: the open frontier

References

- Celledoni, E. et al. (2025). *Mixed-Precision Training of Neural ODEs*. in preparation.
- Chen, R. T. Q. et al. (2018a). “Neural Ordinary Differential Equations”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31.
- Chen, T. Q. et al. (June 2018b). “Neural Ordinary Differential Equations”. In: *NeurIPS*.
- E, W. (Mar. 2017). “A Proposal on Machine Learning via Dynamical Systems”. In: *Communications in Mathematics and Statistics* 5.1, pp. 1–11.
- Gholami, A. et al. (Feb. 2019). “ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs”. In: *arXiv.org*. arXiv: 1902.10298v1 [cs.LG].
- Grathwohl, W. et al. (2018). “Ffjord: Free-form continuous dynamics for scalable reversible generative models”. In: *arXiv preprint arXiv:1810.01367*.
- Haber, E. and L. Ruthotto (2017). “Stable architectures for deep neural networks”. In: *Inverse Problems* 34.1, pp. 1–22.
- Han, J. et al. (Aug. 2018). “Solving high-dimensional partial differential equations using deep learning”. In: *Proceedings of the National Academy of Sciences*, pp. 1 –6. DOI: 10.1073/pnas.1718942115.
- He, K. et al. (2016). “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
- Leburu, R. et al. (2026). “Differentiating through Stochastic Differential Equations: A Primer”. arXiv:2601.08594.
- Li, H et al. (2018). “Visualizing the loss landscape of neural nets”. In: *Advances in Neural Information Processing Systems*.
- Li, X. et al. (2024). “A Neural Network Approach for Stochastic Optimal Control”. In: *SIAM SISC*, arXiv:2209.13104.

References (cont.)

- Lipman, Y. et al. (2023). “Flow Matching for Generative Modeling”. In: *International Conference on Learning Representations (ICLR)*.
- Onken, D. and L. Ruthotto (May 2020). “Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows”. In: *arXiv.org*. arXiv: 2005.13420v1 [cs.LG].
- Onken, D. et al. (May 2020). “OT-Flow: Fast and Accurate Continuous Normalizing Flows via Optimal Transport”. In: *arXiv.org*. arXiv: 2006.00104v1 [cs.LG].
- Onken, D. et al. (2022). “A neural network approach for high-dimensional optimal control applied to multiagent path finding”. In: *IEEE TCST* 31.1, pp. 235–251.
- Raissi, M. (2018). “Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations”. In: *arXiv preprint arXiv:1804.07010*.
- Ruthotto, L. (2024). “Differential Equations for Continuous-Time Deep Learning”. In: *AMS Notices* 71.05.
- Ruthotto, L. et al. (2020). “A machine learning framework for solving high-dimensional mean field game and mean field control problems”. In: *Proceedings of the National Academy of Sciences* 117.17, pp. 9183–9193.
- Sirignano, J. and K. Spiliopoulos (Dec. 2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of Computational Physics* 375, pp. 1339–1364.