A Crash Course On Neural Ordinary Differential Equations

Stéphane Gaudreault[1] Seth Taylor[2] Siqi Wei[3]

[1]Research Manager, Environment and Climate Change Canada
[2]Postdoctoral Fellow, University of Saskatchewan
[3]Postdoctoral Fellow, Environment and Climate Change Canada (ECCC)

June 20, 2025

Introduction

We want to solve an ODE of the form:

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = f(\mathbf{z}(t), t, \boldsymbol{\theta}), \tag{1}$$

where:

- f is a neural network
- $\boldsymbol{\theta}$ represents the parameters

Such ODEs arise naturally in many science and engineering problems.

Challenge: Find an appropriate form for the right-hand side function f.

Example 1: Classification with MNIST

MNIST Dataset:

- Database of handwritten digits (0-9)
- 60,000 training images, 10,000 test images
- Each image is 28×28 pixels

Neural ODE for Classification:

- Input: Initial state $\mathbf{z}(0) =$ image features
- Evolution: $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = f(\mathbf{z}(t), t, \boldsymbol{\theta})$
- Output: $\mathbf{z}(T)$ fed to classifier

The neural network f learns to transform features continuously through "time" to separate different digit classes.

Example 2: Discovering Lotka-Volterra Parameters

Lotka-Volterra Predator-Prey Model:

$$\frac{dx}{dt} = \alpha x - \beta xy \quad \text{(prey)} \tag{2}$$

$$\frac{dy}{dt} = \delta xy - \gamma y \quad \text{(predator)} \tag{3}$$

Applications:

- **Biology:** Population dynamics
- **Economy:** Market competition

Neural ODE Approach:

- Given: Time series data of populations
- Learn: Neural network f that captures dynamics
- Discover: Parameters $\alpha, \beta, \gamma, \delta$

Solving Neural ODEs

Given function f, integrate using standard schemes:

Explicit Methods:

- Euler: $\mathbf{z}_{n+1} = \mathbf{z}_n + hf(\mathbf{z}_n, t_n, \boldsymbol{\theta})$
- Runge-Kutta methods

Implicit Methods:

- Backward Euler: $\mathbf{z}_{n+1} = \mathbf{z}_n + hf(\mathbf{z}_{n+1}, t_{n+1}, \boldsymbol{\theta})$

Exponential Integrators:

- EPI2: $\mathbf{z}_1 = \mathbf{z}_0 + \varphi_1(\Delta t \mathbf{J}_0(\theta)) f(\mathbf{z}_0, t_0, \theta)\Delta t$
  where:
  - $\mathbf{J}_0(\theta) = \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}_0, t_0, \theta)$ is the Jacobian matrix,
  - $\varphi_1(\mathbf{A}) = \mathbf{A}^{-1}(e^{\mathbf{A}} - \mathbf{I})$ is the first φ-function,
  - θ represents the neural network parameters.

Popular option: Adaptive Runge-Kutta methods

- Automatically adjusts step size for accuracy
- E.g., **Dormand-Prince:** 5th order with 4th order error estimate

Training Neural ODEs (Option 1): Backpropagation Through Time

Training requires computing gradients through the ODE solver.

Backpropagation Through Time (BPTT):

- Unroll the ODE solver over time
- Track how parameter changes affect final output
- Compute gradients by backpropagating through all steps

Advantages:

- Accurate gradient computation
- Straightforward implementation

Disadvantages:

- **Memory intensive:** Must store all intermediate states
- Often intractable for long time horizons
- Suitable only for small problems

Training Neural ODEs (Option 2): The Adjoint Method

Alternative derivation using constrained optimization:

Optimization Problem:

$$\min_{\boldsymbol{\theta}} \quad L(\mathbf{z}(t_1), \boldsymbol{\theta}) \tag{4}$$

$$\text{s.t.} \quad \frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t, \boldsymbol{\theta}), \quad \mathbf{z}(t_0) = \mathbf{z}_0 \tag{5}$$

Lagrangian with adjoint variable $\mathbf{a}(t)$:

$$\mathcal{L}(\mathbf{z}(t), \boldsymbol{\theta}, \mathbf{a}) = L(\mathbf{z}(t_1), \boldsymbol{\theta}) - \int_{t_0}^{t_1} \mathbf{a}(t)^T \left(\frac{d\mathbf{z}}{dt} - f(\mathbf{z}(t), t, \boldsymbol{\theta}) \right) dt \tag{6}$$

Karush-Kuhn-Tucker (KKT) Optimality Conditions

$$\frac{\delta \mathcal{L}}{\delta \mathbf{a}(t)} = 0 \quad \Rightarrow \quad \frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t, \boldsymbol{\theta}) \tag{7}$$

$$\frac{\delta \mathcal{L}}{\delta \mathbf{z}(t)} = 0 \quad \Rightarrow \quad \begin{cases} \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \mathbf{z}} \\ \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{z}(t_1)} \end{cases} \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = 0 \quad \Rightarrow \quad \frac{\partial L}{\partial \boldsymbol{\theta}} = \int_{t_0}^{t_1} \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} dt \tag{9}$$

Deriving the Gradient ODE

Taking the derivative with respect to t on both sides of:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \int_{t_0}^{t_1} \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} dt \tag{10}$$

We obtain an ODE for the gradient evolution:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \boldsymbol{\theta}}\right) = \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{11}$$

This allows us to integrate the gradient alongside the adjoint dynamics.

Forward and Backward Passes of the Adjoint Method

Step 1 (Forward): Solve the forward ODE

$$\frac{d\mathbf{z}}{dt} = f(\mathbf{z}, t, \boldsymbol{\theta}), \quad \mathbf{z}(t_0) = \mathbf{z}_0, \tag{12}$$

using a time integration method from t_0 to t_1.

Step 2 (Backward): Solve the augmented adjoint system backward in time from t_1 to t_0:

$$\frac{d}{dt}\begin{bmatrix} \mathbf{z} \\ \mathbf{a} \\ \frac{\partial L}{\partial \boldsymbol{\theta}} \end{bmatrix} = -\begin{bmatrix} f(\mathbf{z}, t, \boldsymbol{\theta}) \\ \mathbf{a}^T \frac{\partial f}{\partial \mathbf{z}} \\ \mathbf{a}^T \frac{\partial f}{\partial \boldsymbol{\theta}} \end{bmatrix} \tag{13}$$

with initial conditions: $\mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{z}(t_1)}$ and $\left.\frac{\partial L}{\partial \boldsymbol{\theta}}\right|_{t_1} = 0.$

Stiffness, Non-Reversibility: A Critical Warning

- Stiffness occurs when the eigenvalues of the Jacobian matrix $\frac{\partial f}{\partial \mathbf{z}}$ have very different magnitudes, creating multiple time scales in the dynamics.
- The adjoint method requires integrating backward in time.

Simple Example - Exponential Decay:

$$\frac{dy}{dt} = -\lambda y, \quad t \in (0,1), \quad y(0) = 1 \tag{14}$$

The analytical solution is $y(t) = e^{-\lambda t}$. However, numerical behavior depends dramatically on λ:

- $\lambda = 100$: Forward solve gives $\sim$1% error, adjoint method works reasonably well
- $\lambda = 10{,}000$: **Adjoint method fails completely** - cannot be reversed numerically, even in double precision!

Practical Implementation with torchdiffeq

Installation:
pip install torchdiffeq

Resources:

- Documentation: `https://github.com/rtqichen/torchdiffeq`
- Examples: See repository's `examples/` folder
- Neural ODE tutorial notebooks available online

References and Further Reading

Original Neural ODE Paper:

- Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. NeurIPS. `https://arxiv.org/abs/1806.07366`
- Kim, S., Ji, W., Deng, S., Ma, Y., & Rackauckas, C. (2021). Stiff neural ordinary differential equations. Chaos: An Interdisciplinary Journal of Nonlinear Science, 31(9). `https://arxiv.org/abs/2103.15341`

Adjoint Method Derivation:

- Detailed derivation using Lagrange multipliers: `https://vaipatel.com/posts/` `deriving-the-adjoint-equation-for-neural-odes-using-la`