Neural Ordinary Differential Equations

Karneedi Yasaswini Bala Aparna

21CE31008

Traditional Neural Networks:

- Traditional Neural Networks uses fixed number of layers (depth).
- They Transform the input data through discrete steps.
- Discrete-layer models cannot effectively represent continuous dynamics.
- Traditional Neural Networks Struggle with irregular time sampling and dynamic data.
- Architectures like Recurrent Neural Networks can handle sequences but are not continuous in nature.

Neural ODE:

- Neural Ordinary Differential Equations redefine deep learning architectures by representing hidden states as the solutions to ODEs, offering a continuous depth model

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

- Numerical ODE Solvers are Euler's Method which are Simple but unstable for stiff equations and Runge-Kutta Methods which are Popular and more accurate.

- Euler’s method is simple and always takes the same size step regardless of the system behaviour. If step size is too large leads to instability and if it's too small leads to inefficiency

- Neural ODEs uses Adaptive Step Sizes and dynamically adjust the step size. It takes Larger steps in smooth regions and smaller steps in regions with high curvature or rapid change

ResNets to ODEs:

- Residual Networks (ResNets) defines hidden states as:

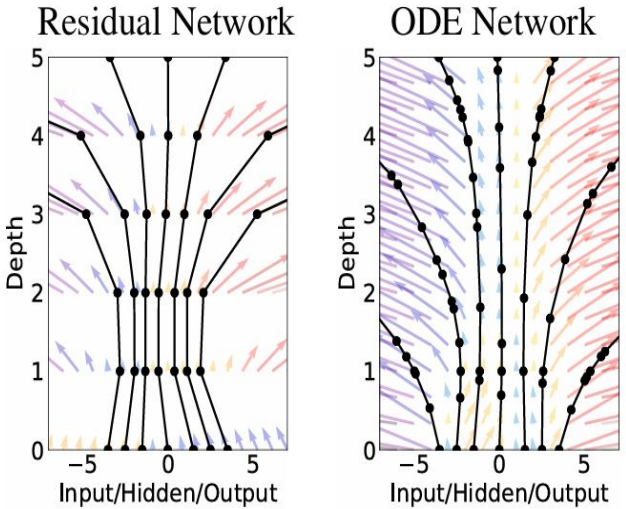
$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

- These are constrained by fixed-depth and fixed step size

- Neural ODEs defines hidden states as:

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

- These are the continuous-depth models



Adjoint Sensitivity Method:

- Adjoint method solves an augmented ODE backward in time

→ Let z(t) be the solution to the ODE: $\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = f(\mathbf{z}, t, \theta)$

→ And let L(z(t1)) be the loss at the final time. The adjoint state is defined as $\mathbf{a}(t) = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(t)}$

→The dynamics of the adjoint state are governed by another ODE:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}} \qquad (4)$$

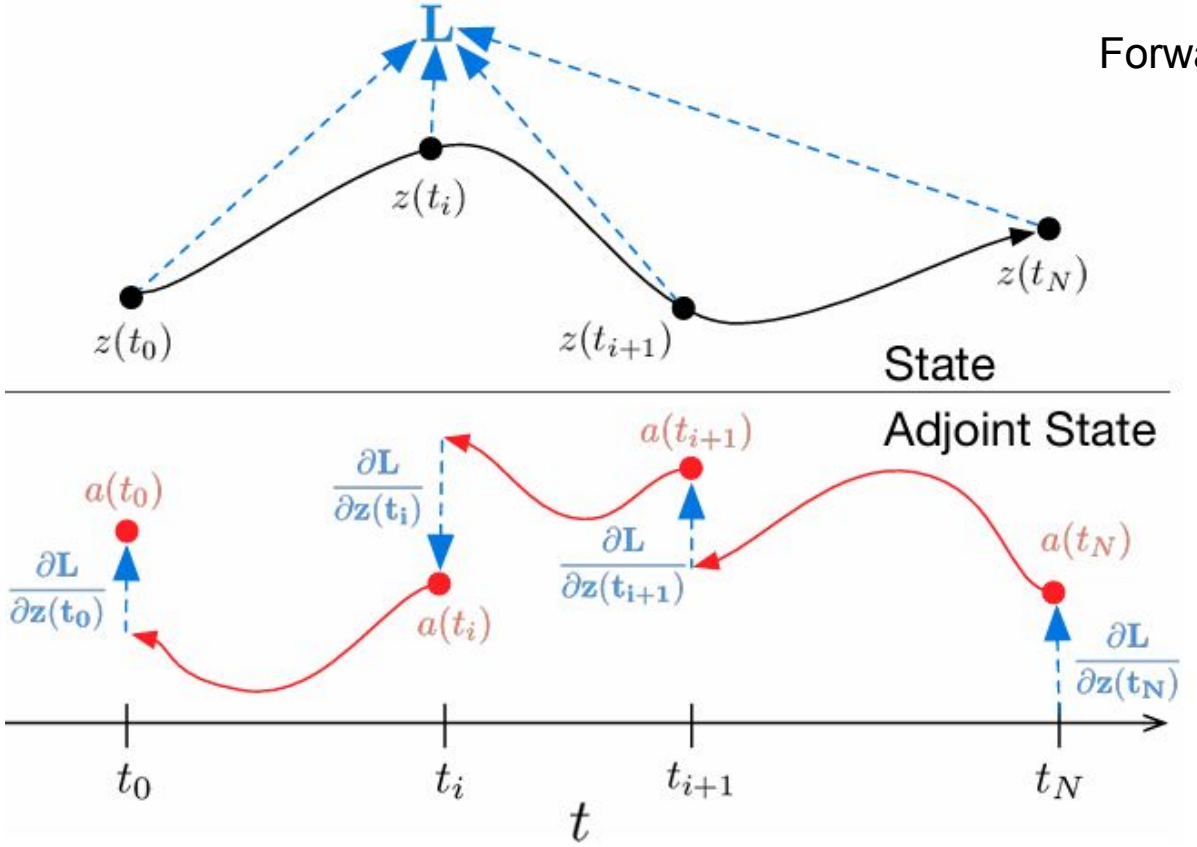
Then we solve this adjoint ODE backwards in time from $t1$ to $t0$

→To get gradients w.r.t. parameters θ, we compute:

$$\frac{dL}{d\theta} = -\int_{t_1}^{t_0} \mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt \qquad (5)$$

This final integral gives the required gradients.

Forward ODE solves hidden states z(t)



Backward ODE solves adjoint states a(t)

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient $\partial L / \partial \mathbf{z}(t_1)$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta$): ▷ Define dynamics on augmented state

return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

Architecture Comparison:

- In ResNet, we take fixed blocks
- In RK-Net,we use Runge-Kutta integration
- ODE-Net replaces discrete layers with continuous function learned via ODE

Table 1: Performance on MNIST. †From LeCun et al. (1998).

	Test Error	# Params	Memory	Time
1-Layer MLP†	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\boldsymbol{\mathcal{O}(1)}$	$\mathcal{O}(\tilde{L})$

Continuous Normalizing Flows (CNF):

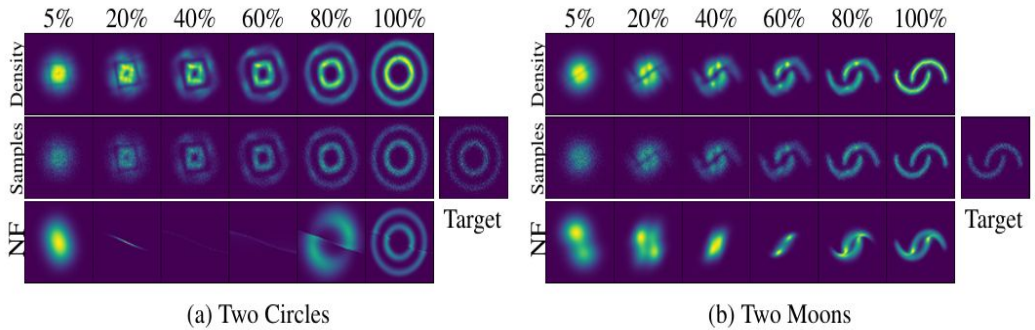
- CNFs solve ODEs in the latent space to transform distributions.

  Theorem 1 (Instantaneous Change of Variables):

- Log probability change follows:

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

- Trace Operation is cheaper than determinant that is used in the traditional normalizing flows.



(a) Two Circles (b) Two Moons

- CNF transformations are smooth and interpretable.
- NF transformations are less intuitive and they struggle with complex geometry.
- CNF adapts shape by rotating & spreading particles in a meaningful way.

Applications of Neural ODEs:

- Irregular Time Series Modeling
- Memory-Efficient Training
- Control Systems
- Generative Models

Advantages of Neural ODEs:

- Continuous Time Modeling
- Adaptive Computation
- Memory Efficiency
- Parameter Efficiency

Limitations of Neural ODEs:

- Slow Training
- Complex Gradient Computation
- Solver Instability
- Not Always Better Than Discrete Models

Summary of Neural ODEs:

- Neural ODEs represents the hidden states as continuous solutions to differential equations.
- Uses adaptive ODE solvers to control precision, error, and computation cost.
- Training involves reverse-mode differentiation using the Adjoint Sensitivity Method.
- Offers a trade-off between interpretability, memory efficiency, and computation time.

THANK YOU!!