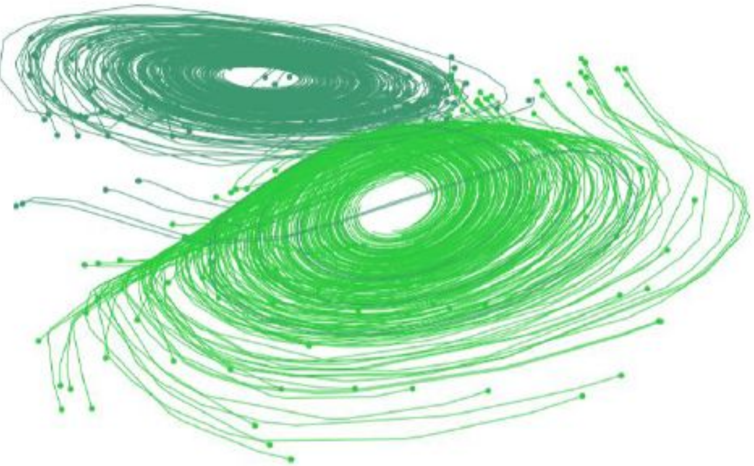
Neural Ordinary Differential Equations



Mohd Adnan
MASc Candidate
https://adnan1306.github.io/

Background: Ordinary Differential Equations (ODEs)

- Model the instantaneous change of a state.

$$\frac{dz(t)}{dt} = f(z(t), t) \qquad \text{(explicit form)}$$

- Solving an **initial value problem** (IVP) corresponds to integration.

$$z(t) = z(t_0) + \int_{t_0}^{t} f(z(t), t)dt \qquad \text{(solution is a trajectory)}$$

- Euler method approximates with small steps:

$$z(t+h) = z(t) + hf(z(t), t)$$

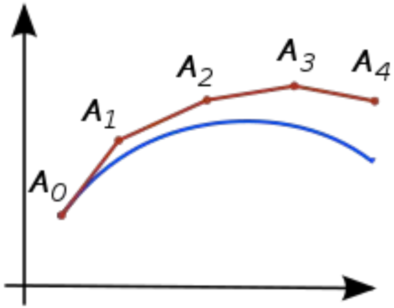
Residual Networks interpreted as an ODE Solver

- Hidden units look like: $z_{l+1} = F_l(z_l) = z_l + f_l(z_l)$
- Final output is the composition: $z_L = F_{L-1} \circ F_{L-2} \cdots \circ F_0(z_0)$

Haber & Ruthotto (2017). E (2017).

Residual Networks interpreted as an ODE Solver

- Hidden units look like: $z_{l+1} = F_l(z_l) = z_l + f_l(z_l)$
- Final output is the composition: $z_L = F_{L-1} \circ F_{L-2} \cdots \circ F_0(z_0)$
- This can be interpreted as an **Euler discretization** of an ODE.



- In the limit of smaller steps: $\frac{dz(t)}{dt} = \lim_{h \to 0} \frac{z_{t+h} - z_t}{h} = f(z_t)$

Haber & Ruthotto (2017). E (2017).

Deep Learning as Discretized Differential Equations

Many deep learning networks can be interpreted as ODE solvers.

Network	**Fixed-step Numerical Scheme**
ResNet, RevNet, ResNeXt, etc.	Forward Euler
PolyNet	Approximation to Backward Euler
FractalNet	Runge-Kutta
DenseNet	Runge-Kutta

Lu et al. (2017)
Chang et al. (2018)
Zhu et al. (2018)

Deep Learning as Discretized Differential Equations

Many deep learning networks can be interpreted as ODE solvers.

Network	**Fixed-step Numerical Scheme**
ResNet, RevNet, ResNeXt, etc.	Forward Euler
PolyNet	Approximation to Backward Euler
FractalNet	Runge-Kutta
DenseNet	Runge-Kutta

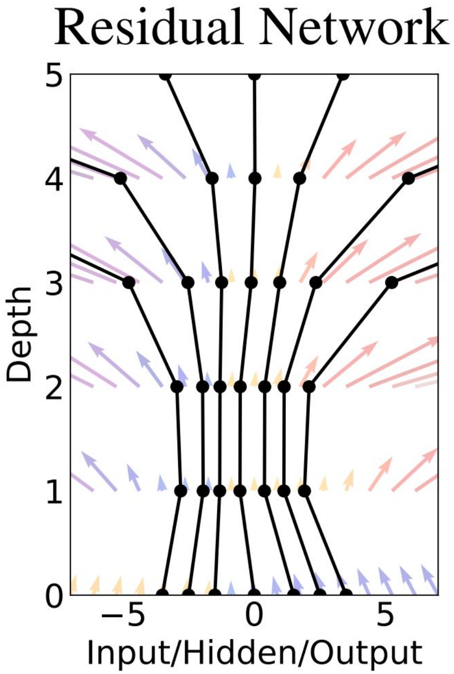
Lu et al. (2017)
Chang et al. (2018)
Zhu et al. (2018)

But:

(1) What is the underlying dynamics?
(2) Adaptive-step size solvers provide better error handling.

“Neural” Ordinary Differential Equations

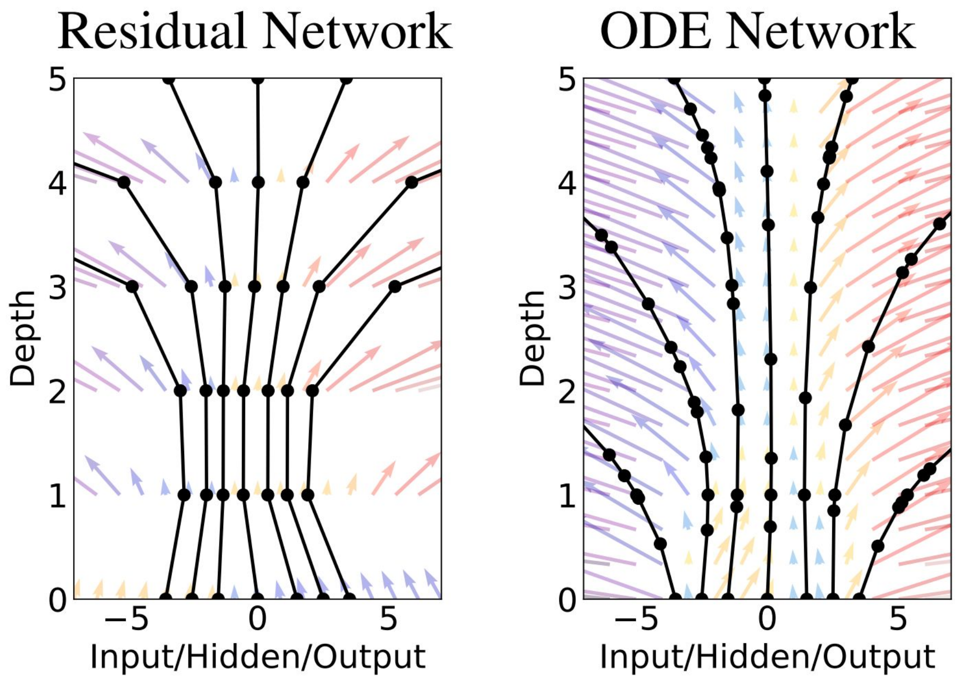
Instead of **y = F(x)**,



"Neural" Ordinary Differential Equations

Instead of **y = F(x)**, solve **y = z(T)** given the initial condition **z(0) = x**.

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



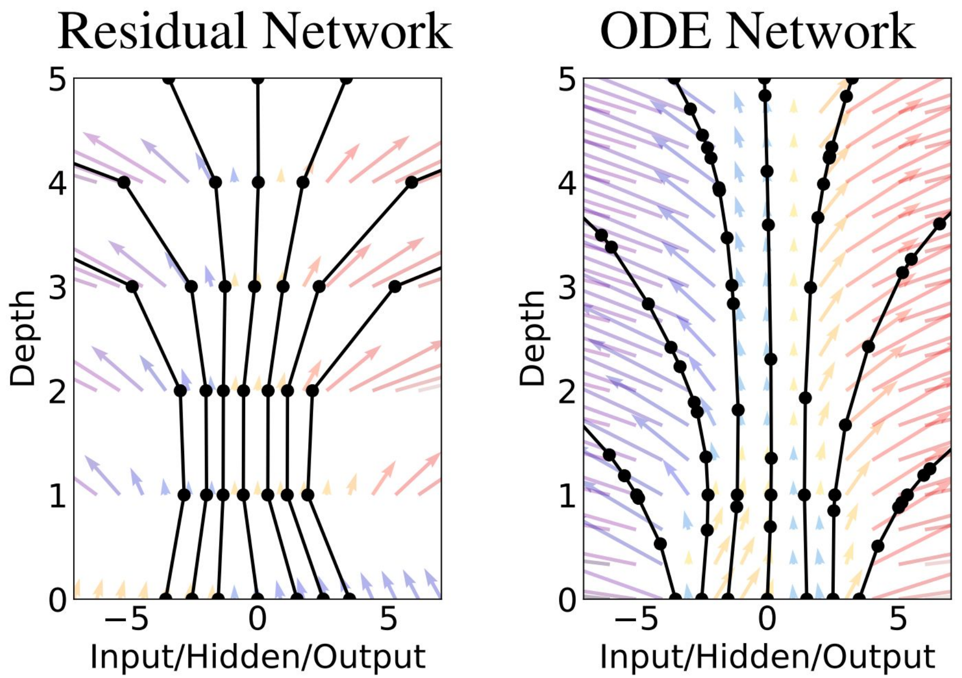
“Neural” Ordinary Differential Equations

Instead of **y = F(x)**, solve **y = z(T)** given the initial condition **z(0) = x**.

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

Solve the dynamic using **any black-box ODE solver**.

- Adaptive step size.
- Error estimate.
- O(1) memory learning.



Backprop without knowledge of the ODE Solver

Ultimately want to optimize some loss

$$L(z(T)) = L\left(z(t_0) + \int_{t_0}^{T} f(z(t), t, \theta)dt\right) = L\left(\text{ODESolve}(z(t_0), t_0, T, \theta)\right)$$

$$\frac{\partial L}{\partial \theta} = ?$$

Backprop without knowledge of the ODE Solver

Ultimately want to optimize some loss

$$L(z(T)) = L\left(z(t_0) + \int_{t_0}^{T} f(z(t), t, \theta)dt\right) = L\left(\text{ODESolve}(z(t_0), t_0, T, \theta)\right)$$

Naive approach: Know the solver. Backprop through the solver.

- Memory-intensive.
- Family of “implicit” solvers perform inner optimization.

Backprop without knowledge of the ODE Solver

Ultimately want to optimize some loss

$$L(z(T)) = L\left(z(t_0) + \int_{t_0}^{T} f(z(t), t, \theta)dt\right) = L\left(\text{ODESolve}(z(t_0), t_0, T, \theta)\right)$$

Naive approach: Know the solver. Backprop through the solver.

- Memory-intensive.
- Family of “implicit” solvers perform inner optimization.

Our approach: **Adjoint sensitivity analysis**. (Reverse-mode Autodiff.)

- Pontryagin (1962).
  + Automatic differentiation.
  + O(1) memory in backward pass.

Continuous-time Backpropagation

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

Backward: $a_t = a_{t+h} + h a_{t+h} \frac{\partial f(z_t)}{\partial z_t}$

Params: $\frac{\partial L}{\partial \theta} = h a_{t+h} \frac{\partial f(z(t), \theta)}{\partial \theta}$

Adjoint method.

Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Continuous-time Backpropagation

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $$z_{t+h} = z_t + hf(z_t)$$

Backward: $$a_t = a_{t+h} + ha_{t+h}\frac{\partial f(z_t)}{\partial z_t}$$

Params: $$\frac{\partial L}{\partial \theta} = ha_{t+h}\frac{\partial f(z(t), \theta)}{\partial \theta}$$

Adjoint method.

Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $$z(t+1) = z(t) + \int_t^{t+1} f(z(t))\, dt$$

Continuous-time Backpropagation

Residual network. $a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

Backward: $a_t = a_{t+h} + ha_{t+h}\frac{\partial f(z_t)}{\partial z_t}$

Params: $\frac{\partial L}{\partial \theta} = ha_{t+h}\frac{\partial f(z(t),\theta)}{\partial \theta}$

Adjoint method. Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $z(t+1) = z(t) + \int_t^{t+1} f(z(t))\, dt$

Backward: $a(t) = a(t+1) + \int_{t+1}^{t} a(t)\frac{\partial f(z(t))}{\partial z(t)} dt$

Adjoint State

Adjoint DiffEq

Continuous-time Backpropagation

Residual network.

$a_t := \frac{\partial L}{\partial z_t}$

Forward: $z_{t+h} = z_t + hf(z_t)$

Backward: $a_t = a_{t+h} + ha_{t+h}\frac{\partial f(z_t)}{\partial z_t}$

Params: $\frac{\partial L}{\partial \theta} = ha_{t+h}\frac{\partial f(z(t),\theta)}{\partial \theta}$

Adjoint method.

Define: $a(t) := \frac{\partial L}{\partial z(t)}$

Forward: $z(t+1) = z(t) + \int_t^{t+1} f(z(t))\, dt$

Backward: $a(t) = a(t+1) + \int_{t+1}^{t} a(t)\frac{\partial f(z(t))}{\partial z(t)} dt$

Adjoint State

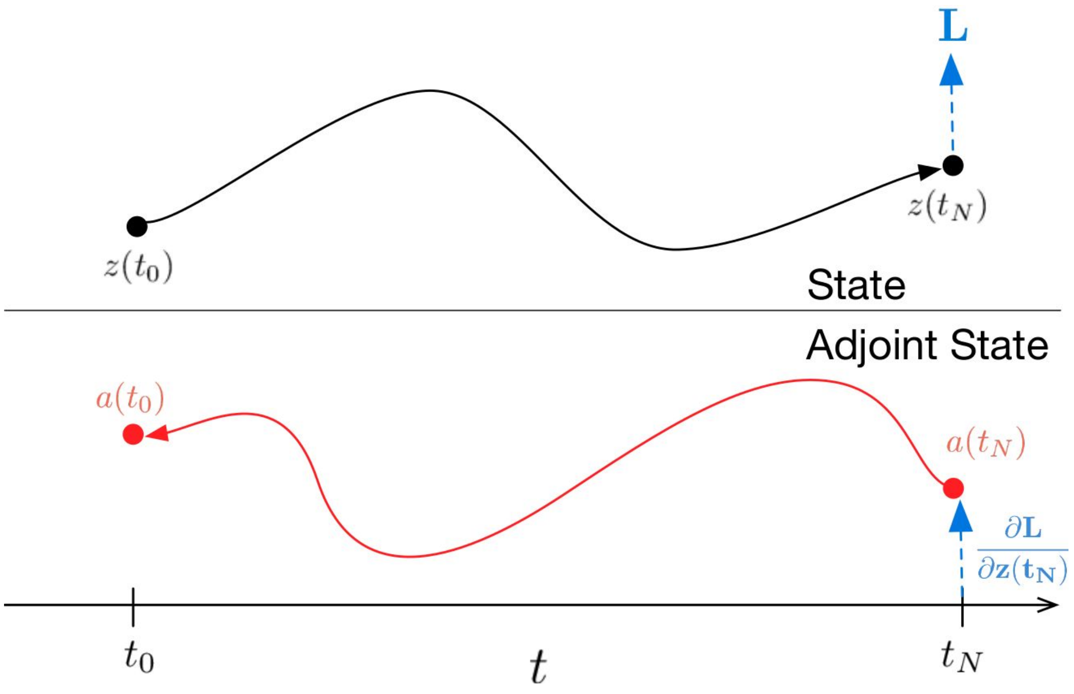
Adjoint DiffEq

Params: $\frac{\partial L}{\partial \theta} = \int_t^{t+1} a(t)\frac{\partial f(z(t),\theta)}{\partial \theta}\, dt$

A Differentiable Primitive for AutoDiff

Forward:

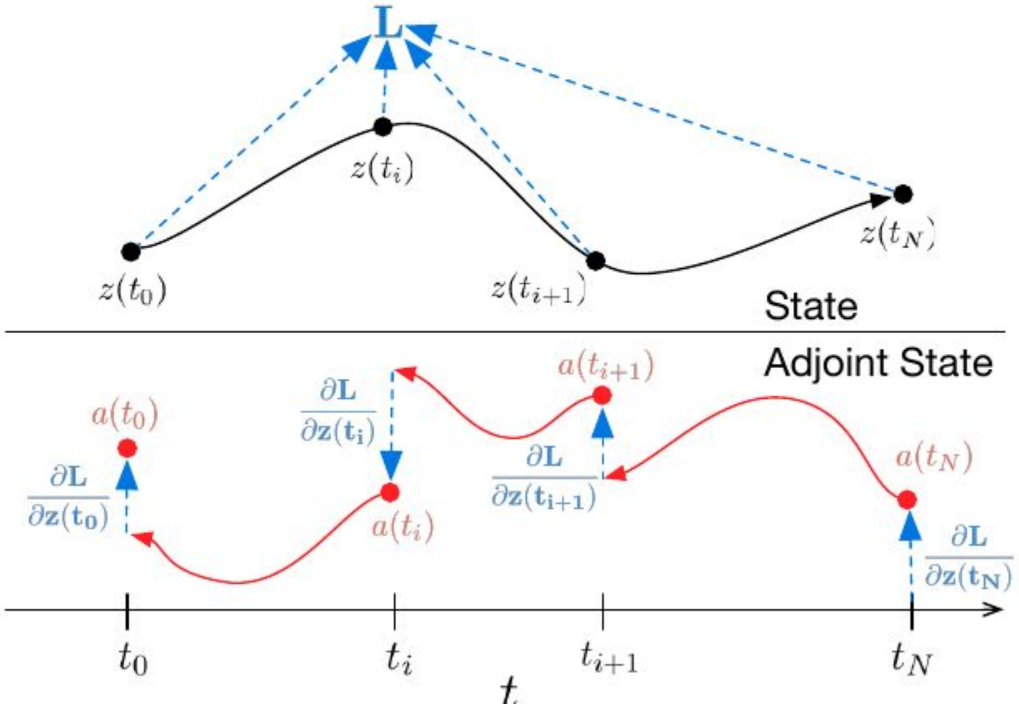
Backward:



A Differentiable Primitive for AutoDiff

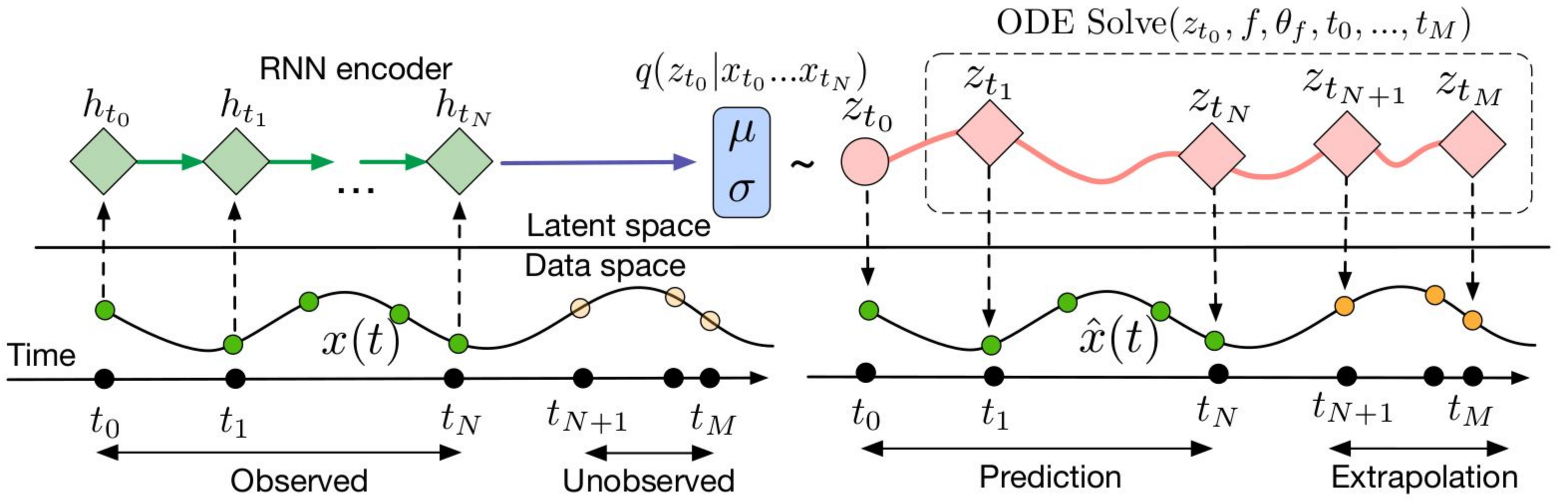
Forward:

Backward:



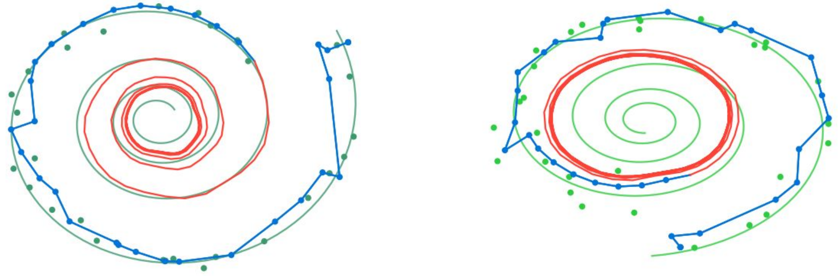
Continuous-time RNNs for Time Series Modeling

- We often want arbitrary measurement times, ie. irregular time intervals.
- Can do VAE-style inference with a latent ODE.

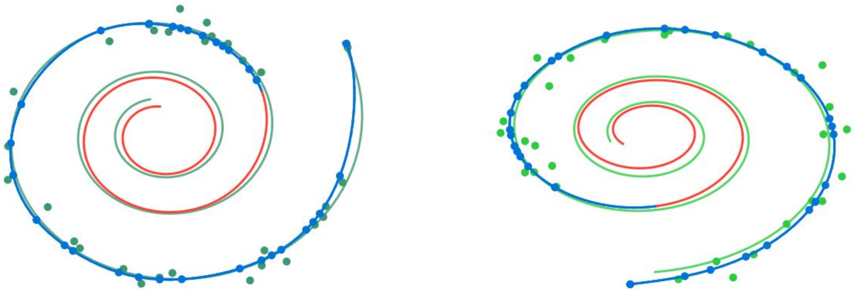


ODEs vs Recurrent Neural Networks (RNNs)

- RNNs learn very stiff dynamics, have exploding gradients.
- Whereas ODEs are guaranteed to be smooth.



(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation