# Neural Ordinary Differential Equations

Ricky Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud

Presented by
Chaitanya

# Convention

Number of layers are referred to as time (or t) in the following slides except in the case of RNNs

# Resnets

$$\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$

Where h=1

# Resnets

$$\mathbf{z}(t+h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$

Where h=1

OR

```
def f(z, t, θ):
  return nnet(z, θ[t])

def resnet(z):
  for t in [1:T]:
    z = z + f(z, t, θ)
  return z
```

# If time was continuous?

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

# If time was continuous?

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

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt$$

# If time was continuous?

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

```
def f(z, t, θ):
  return nnet([z, t], θ)

def ODEnet(z, θ):
  return ODESolve(f, z, 0, 1, θ)
```

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt$$

# Training

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

1) This seems familiar. We can probably backprob through the ODESolver.

# Training

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

1) This seems familiar. We can probably backprob through the ODESolver.

   Not a good idea. ODESolvers are not perfect. Backprob will take accumulate mor losses over epochs

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta))$$

Adjoint sensitivity method (Pontryagin et al., 1962)

$$\mathbf{a}(t) = \partial L / \partial \mathbf{z}(t)$$

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

Adjoint sensitivity method (Pontryagin et al., 1962)

$$\mathbf{a}(t) = \partial L / \partial \mathbf{z}(t)$$

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

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\mathbf{a}(t) = {\partial L}/{\partial \mathbf{z}(t)}$$

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

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\mathbf{a}(t) = \partial L / \partial \mathbf{z}(t)$$

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

Can be solved using an ODE solver

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\mathbf{a}(t) = {\partial L}/{\partial \mathbf{z}(t)}$$

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

Instantaneous analog of chain rule

Can be solved using an ODE solver

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\mathbf{a}(t) = {}^{\partial L}/_{\partial \mathbf{z}(t)}$$

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

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

Instantaneous analog of chain rule

Can be solved using an ODE solver

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\mathbf{a}(t) = {}^{\partial L}/_{\partial \mathbf{z}(t)}$$

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

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

Instantaneous analog of chain rule

Can be solved using an ODE solver

Can be solved using an ODE solver.

*Integrated from 0 to 1 in experiments

# How do we train this model then?

$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

$$\mathbf{a}(t) = {\partial L}/{\partial \mathbf{z}(t)}$$

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

Instantaneous analog of chain rule

Can be solved using an ODE solver

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

Used to update parameters of **f**

Can be solved using an ODE solver.

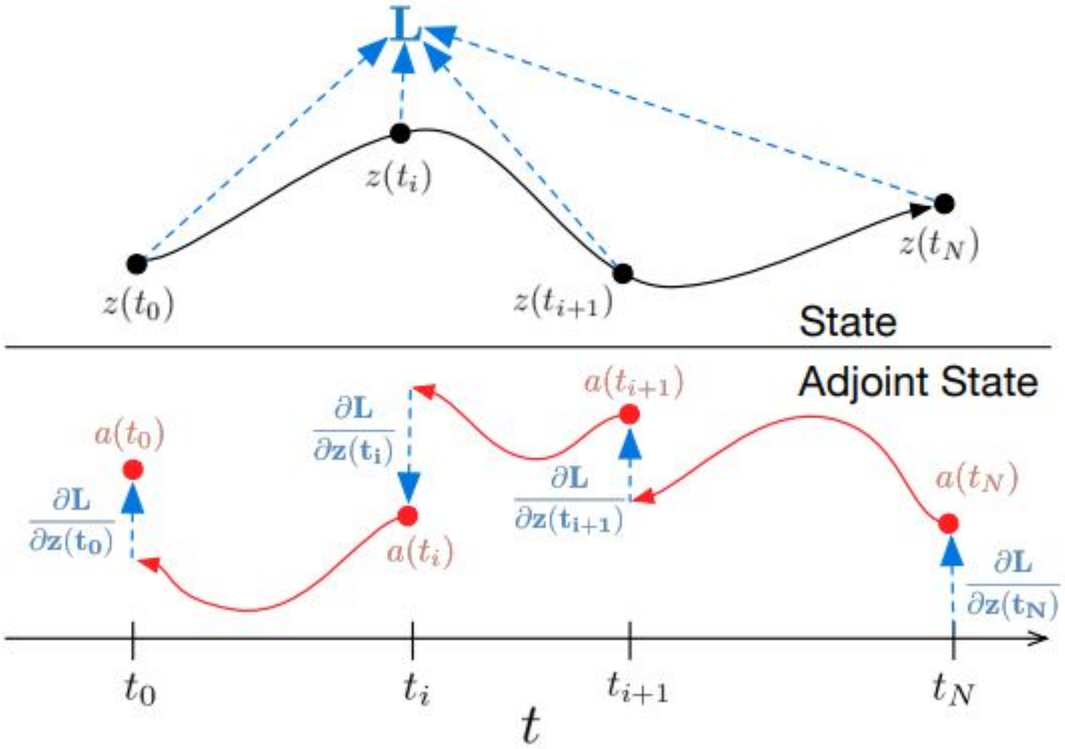
*Integrated from 0 to 1 in experiments

# Advantages

- O(1) Memory gradients: as the activations are not stored at each layer, instead the dynamics are run backwards from the output to input

# Advantages

- O(1) Men … ach layer, instead th … put



# Advantages

- O(1) Memory gradients: as the activations are not stored at each layer, instead the dynamics are run backwards from the output to input

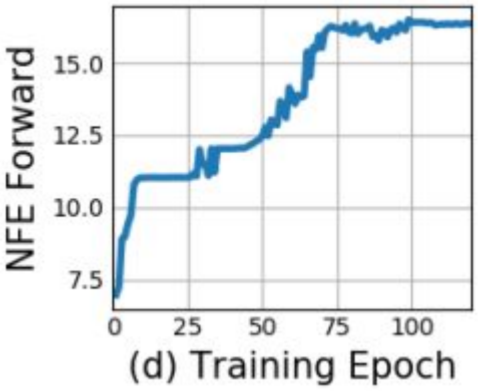
Table 1: Performance on MNIST. †From LeCun et al. (1998).

| | 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 | $\boldsymbol{\mathcal{O}(1)}$ | $\mathcal{O}(\tilde{L})$ |

# Advantages

Depth of ODEs

- It is left to the ODE solver. Hence it can change during training
- Approximately 2-4x the depth of resnet architectures - (shown empirically)

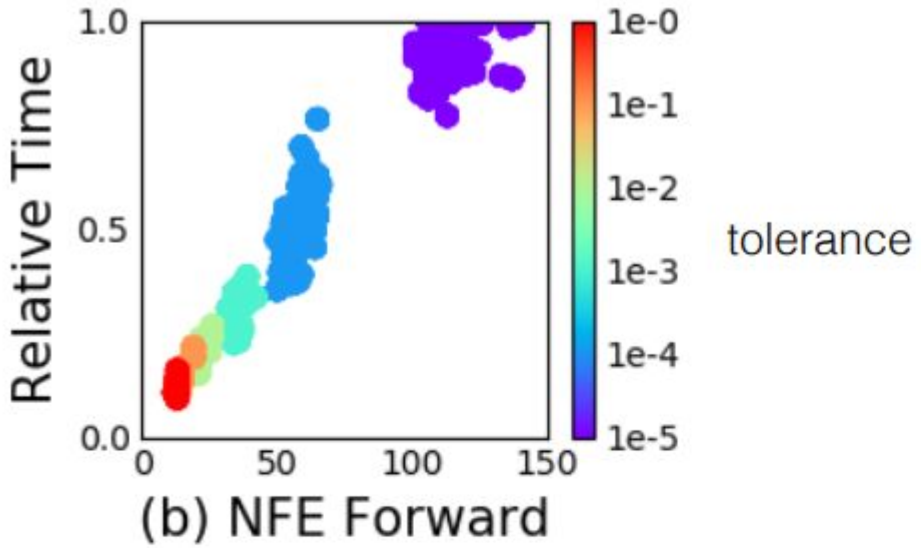



NFE- Number of forward evaluations

# Advantages

Explicit Error Control.

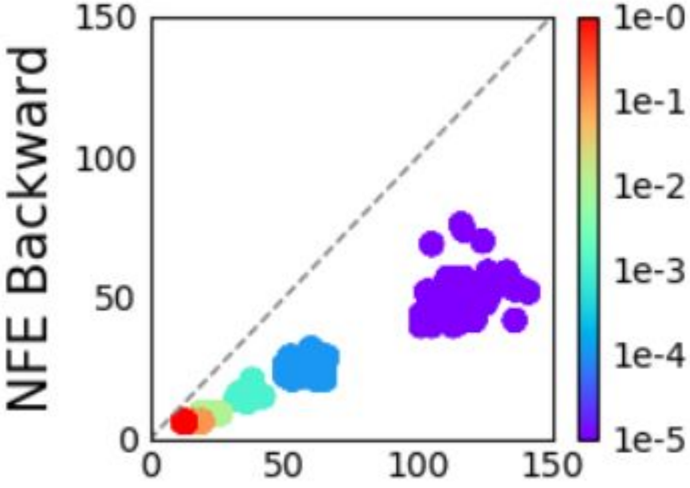
ODEsolver’s error can be explicitly controlled. Hence trade-off between speed and error is in the user’s hand



# Advantages

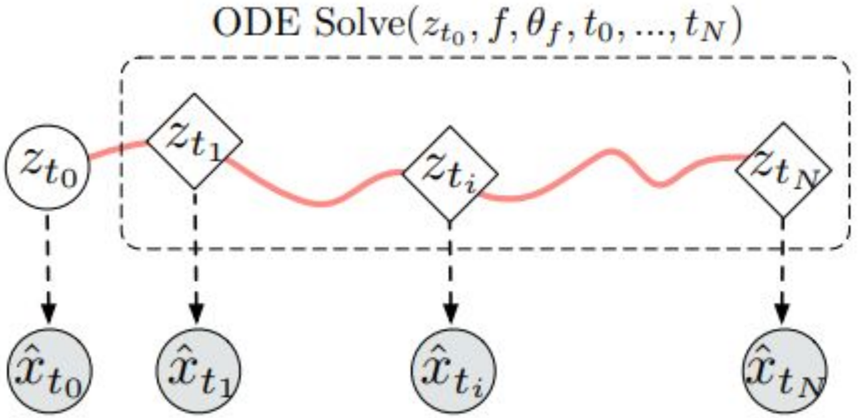
Cost of evaluation (Forward pass vs Backward Pass)

Unlike, SGD based training methods for neural networks, NeuralODEs have a faster backward pass




(c) NFE Forward

# Continuous-time models




- Well-defined state at all times
- Dynamics separate from inference
- Irregularly-timed observations.

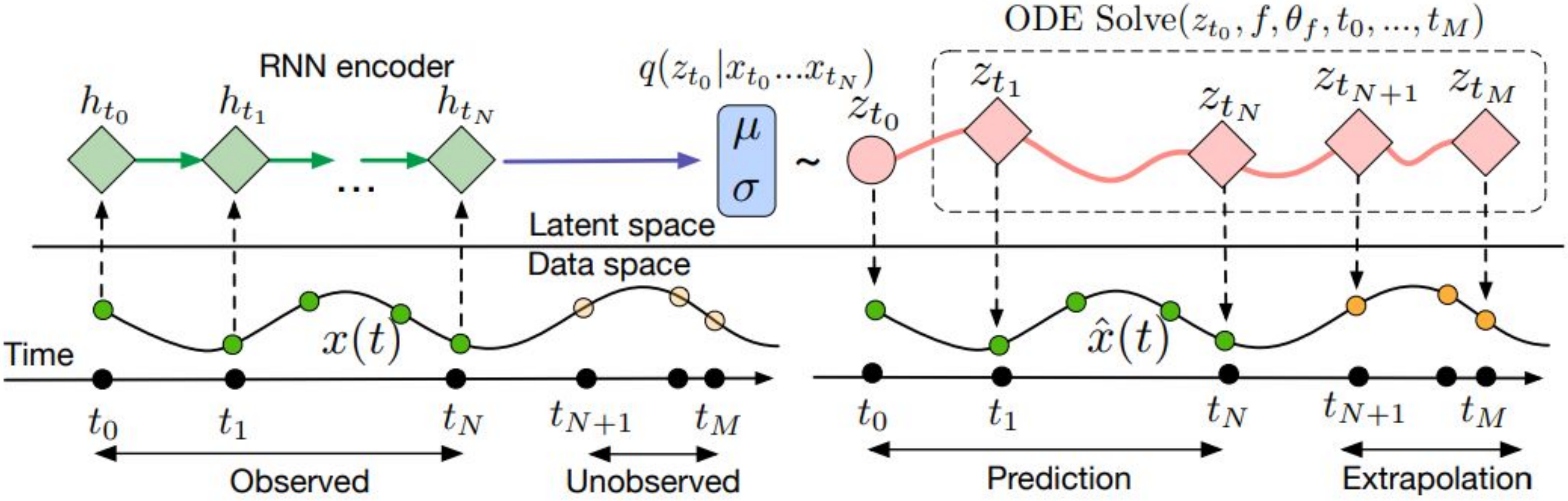
$$\mathbf{z}_{t_0} \sim p(\mathbf{z}_{t_0})$$

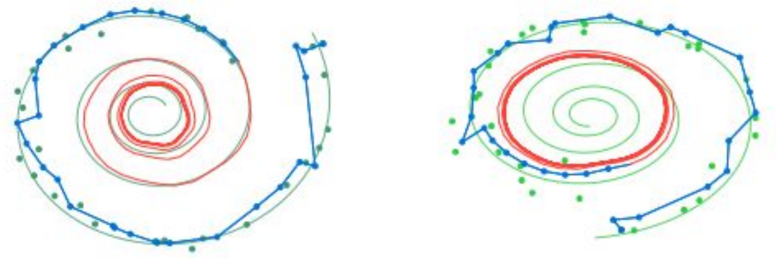
$$\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} = \text{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N)$$

$$\text{each} \quad \mathbf{x}_{t_i} \sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}})$$

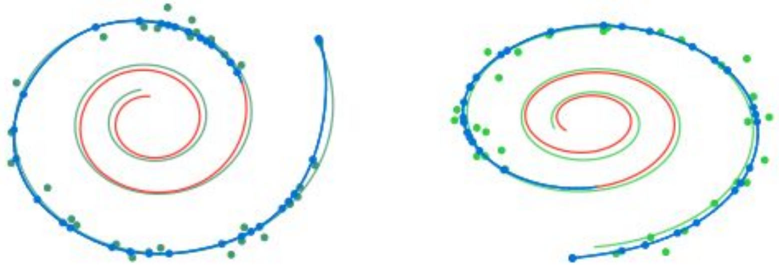
http://www.cs.toronto.edu/~rtqichen/pdfs/neural_ode_slides.pdf

# Handles unobserved variables and can extrapolate





(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation