Neural Ordinary Differential Equations

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud

Presented by Simon Haile and Elliott Skomski

Preliminaries

Ordinary Differential Equations (ODEs)

A DIFFERENTIAL EQUATION is an equation that relates a function to its derivative

ORDINARY DIFFERENTIAL EQUATIONS are defined with respect to one independent variable (contrast with partial differential equations)

$$e.g. \qquad \frac{df}{dx} = -f(x)^2$$

Initial Value Problems

An INITIAL VALUE PROBLEM is an ODE paired with an initial condition, or some point in the domain of the function

Example

$$\frac{df}{dx} = -f(x)^2; \quad f(0) = 5$$
$$\Rightarrow f(x) = \frac{5}{5x+1}$$

Simple IVPs can be solved using general solutions

Most of the time, they must be solved using numerical methods

Introduction

Neural Networks as Continuous Transformations

Neural networks traditionally compose a series of transformations

$$\mathsf{h}_{t+1} = f(\mathsf{h}_t, \theta_t),$$
$$\text{where t} \in \{0, ..., T\} \text{ and } \mathsf{h}_t \in \mathbb{R}^D$$

What if we reduce the step size between layers and increase the number of layers *infinitely*?

Neural Networks as Continuous Transformations

In the limit, as $dt \to 0$ approaches zero and $T \to \infty$, we obtain a differential:

$$\tfrac{d\mathsf{h}(t)}{dt} = f(\mathsf{h}(t), t, \theta)$$

This ODE parameterizes the continuous dynamics of hidden units

Neural Networks as Initial Value Problems

Now that we have an ODE, what else do we know?

- $\mathsf{h}(0)$ is the input layer: $\mathsf{h}(0) = \mathsf{x}$ (an initial condition!)
- $\mathsf{h}(T)$ is the output layer: $\mathsf{h}(T) = \mathsf{y}_{pred}$

Given this IVP, we can now use a numeric solver to find $\mathsf{h}(T)$

Motivation

This formulation has some benefits over traditional neural networks:

- Memory efficiency (gradient computation in $O(1)$ space!)
- Adaptive computation
- Parameter efficiency
- Scalable and invertible normalizing flows
- Continuous time-series modeling

How It Works

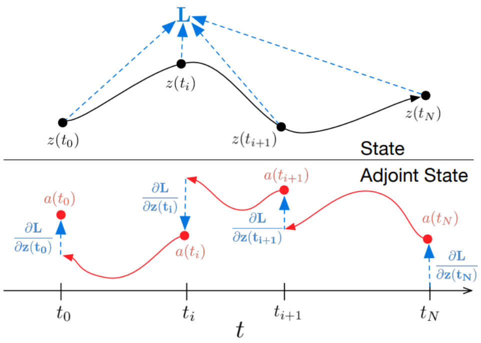

$\mathbf{L}$
$z(t_i)$
$z(t_0)$
$z(t_{i+1})$
$z(t_N)$
State
Adjoint State
$a(t_0)$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_0})}$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_i})}$
$a(t_i)$
$a(t_{i+1})$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_{i+1}})}$
$a(t_N)$
$\frac{\partial \mathbf{L}}{\partial \mathbf{z}(\mathbf{t_N})}$
t_0
t_i
t_{i+1}
t_N
t

Figure 2: Reverse-mode differentiation of an ODE solution. The adjoint sensitivity method solves an augmented ODE backwards in time. The augmented system contains both the original state and the sensitivity of the loss with respect to the state. If the loss depends directly on the state at multiple observation times, the adjoint state must be updated in the direction of the partial derivative of the loss with respect to each observation.

Experiments

ODEs vs. Residual Networks

To demonstrate supervised learning capabilities, they compare a ResNet with six blocks to a single ODE-Net

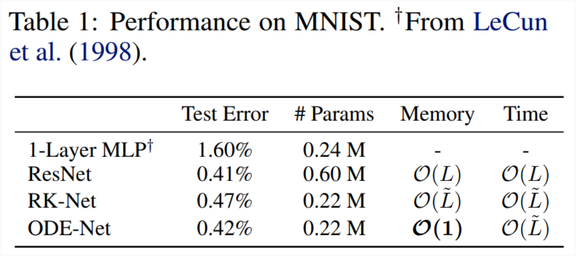


Table 1: Performance on MNIST. †From LeCun et al. (1998).

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

ODE-Net found to achieve similar performance to ResNet, using fewer parameters

Error Control

ODE solvers have adjustable tolerance that trades off accuracy and computational cost

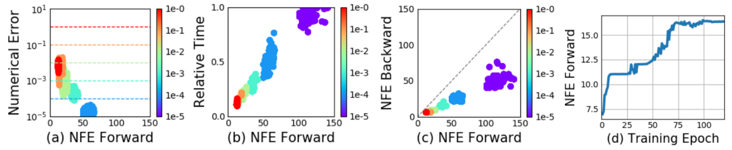


Figure 3: Statistics of a trained ODE-Net. (NFE = number of function evaluations.)

Reducing error tolerance reduces number of function evaluations but increases numerical error

Interestingly, backward pass requires ~50% fewer evaluations than forward

Continuous Normalizing Flows

Normalizing flows are a handy technique to transform simple densities into complex distributions

- Multivariate Gaussians with diagonal covariance matrices used in continuous control tasks

Typically represented in a discrete fashion, but can be represented as a continuous transformation

Coincidentally makes computation easier

Continuous Normalizing Flows

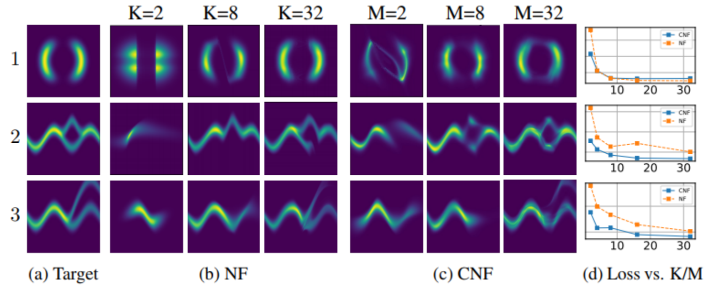


Figure 4: Comparison of normalizing flows versus continuous normalizing flows. The model capacity of normalizing flows is determined by their depth (K), while continuous normalizing flows can also increase capacity by increasing width (M), making them easier to train.

Continuous, Generative Time Series Modeling

- Modeling irregularly-sampled data is hard: network traffic.
  - We're forced to discretize inputs as well as outputs
- The data in between bins is missing and generative time-series models can interpolate or extrapolate
- This allows us to extrapolate or interpolate continuous values using discrete, irregularly sampled inputs

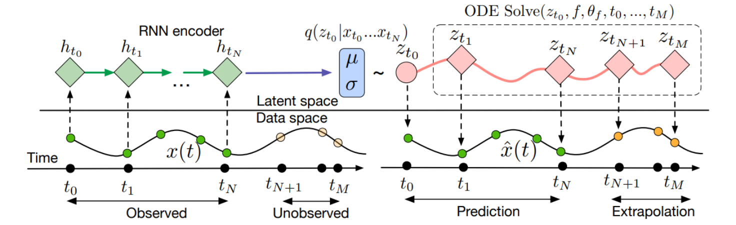
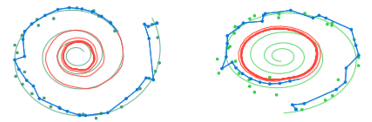
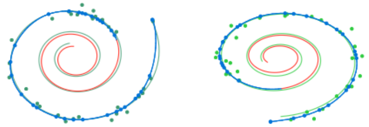


Figure 6: Computation graph of the latent ODE model.

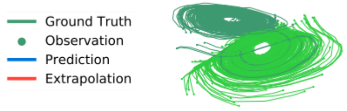
Continuous, Generative Time Series Modeling



(a) Recurrent Neural Network



(b) Latent Neural Ordinary Differential Equation



(c) Latent Trajectories

Scope and Limitations

- Unique solution of IVP theoretically guaranteed for certain nonlinearities
- Framework allows trading off speed for precision, but requires choosing error tolerances for forward and backward passes
- Minibatching is less straightforward
- Reconstructing state trajectories may introduce numerical error

Conclusion

- Black-box ODE solvers, new time-series modeling/supervised learning/density estimation
- Adaptive evaluation: speed vs accuracy trade-off
- Continuous-time normalizing flows
- Reconstructing state trajectories may introduce numerical error

Questions?