Neural Ordinary Differential Equations

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

*Equal Contribution University of Toronto, Vector Institute

Contributions

Black-box ODE solvers as a differentiable modeling component.

- Continuous-time recurrent neural nets and continuous-depth feedforward nets.
- Adaptive computation with explicit control over tradeoff between speed and numerical precision.
- ODE-based change of variables for automatically-invertible normalizing flows.
- Open-sourced ODE solvers with O(1)-memory backprop: `https://github.com/rtqichen/torchdiffeq`

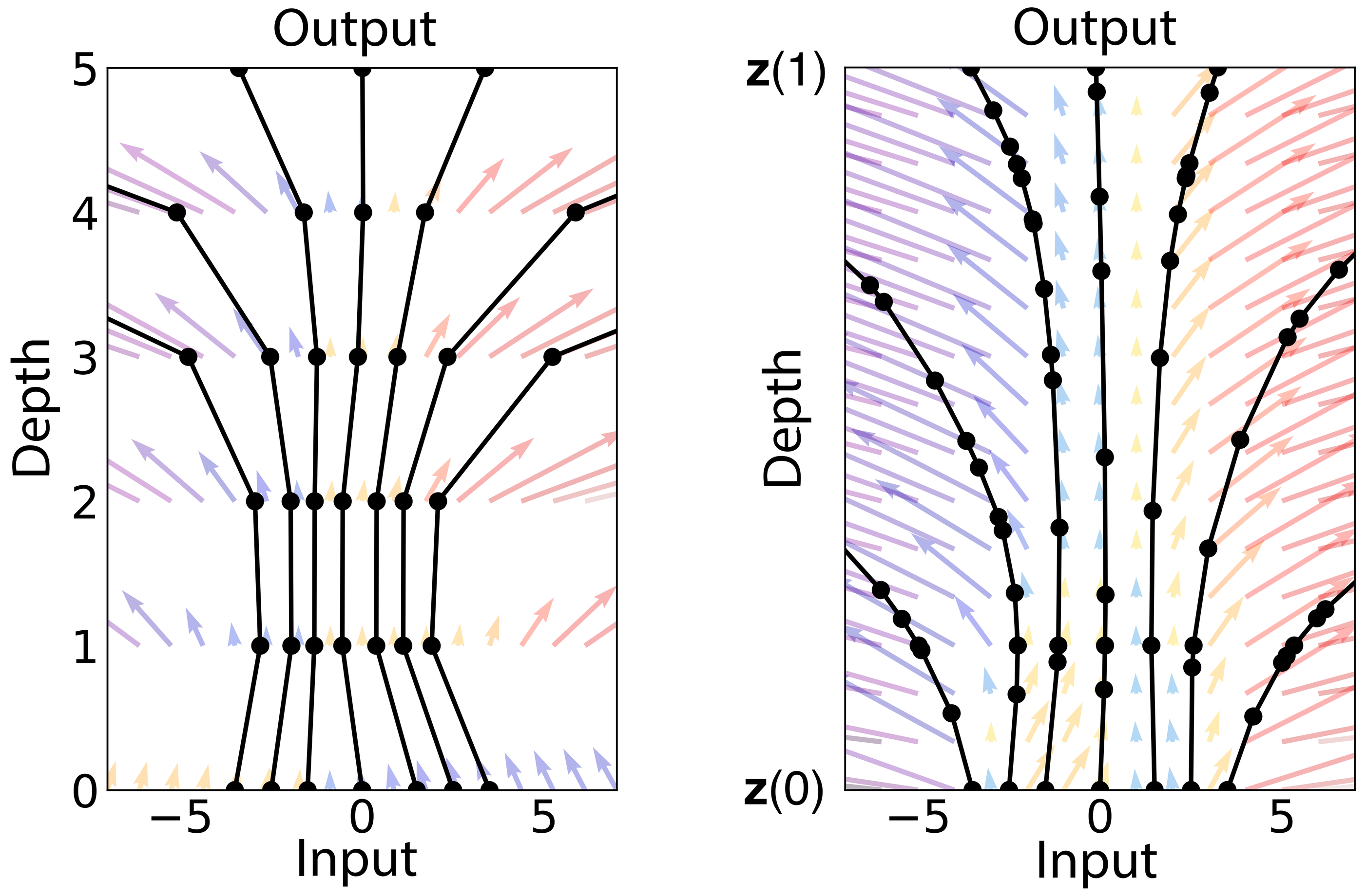
ODE Solvers: How Do They Work?

- $\mathbf{z}(t)$ changes in time, defines an infinite set of trajectories.
- Define a differential equation: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t, \theta)$.
- Initial-value problem: given $\mathbf{z}(t_0)$, find $\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(\mathbf{t}), \mathbf{t}, \theta)$.
- Approximate solution with discrete steps, e.g. $\mathbf{z}(t+h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$.
- Higher-order solvers are more accurate and use larger step sizes.
- Can adapt step size h given error tolerance level.

Continuous version of ResNets

ODE-Net replaces ResNet blocks with ODESolve$(f, \mathbf{z}(t_0), t_0, t_1, \theta)$, where f is a neural net with parameters θ.

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



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

ResNet

```
def resnet(x, θ):
  h1 = x + NeuralNet(x, θ[0])
  h2 = h1 + NeuralNet(h1, θ[1])
  h3 = h2 + NeuralNet(h2, θ[2])
  h4 = h3 + NeuralNet(h3, θ[3])
  return h4
```

ODE-Net

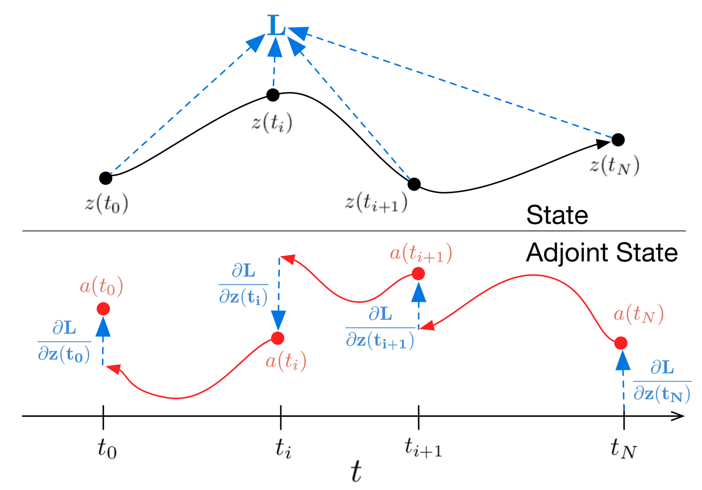
```
def f(z, t, θ):
  return NeuralNet([z, t], θ)

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

'Depth' is automatically chosen by an adaptive ODE solver.

Computing gradient for ODE solutions

O(1) memory cost when training.
Don't store activations, follow dynamics in reverse.
No backpropagation through the ODE solver – compute the gradient through another call to ODESolve.



Define adjoint state:
$a(t) = -\partial L / \partial \mathbf{z}(t)$

Adjoint state dynamics:
$\frac{\mathbf{a}(t)}{dt} = -a(t)\frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$

Solve ODE backwards in time:
$\frac{d\mathcal{L}}{d\theta} = \int_{t_0}^{t_1} a(t)^T \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$

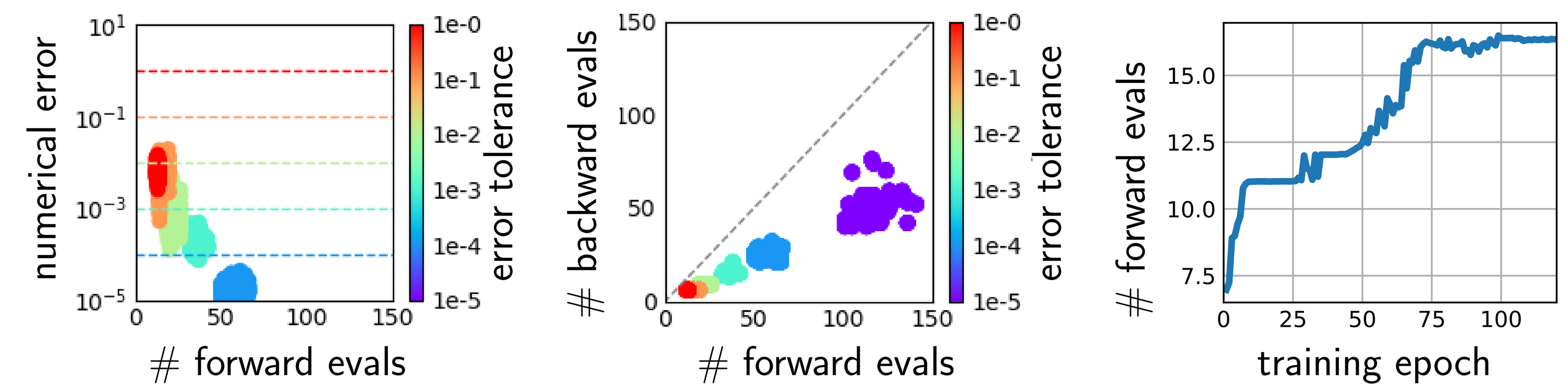
```
def f_and_a([z,a,grad], t):
    return [f, -a*df/da, -a*df/d theta]

[z0, dL/dx, dL/d theta] =
    ODESolve(f_and_a, [z(t1), dL/dz(t), 0], t0, t1)
```

ODE Nets for Supervised Learning

Adaptive computation: can adjust speed vs precision.
We can specify the error tolerance of ODE solution:

$$\text{ODESolve}(f_\theta, \mathbf{z}(t_0), t_0, t_1, \theta, \textit{rtol}, \textit{atol})$$



Performance on MNIST

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

Instantaneous Change of Variables

Change of variables theorem to compute exact changes in probability of samples transformed through bijective F:

$$\mathbf{z}_1 = \mathbf{z}_0 + f(\mathbf{z}_0) \implies \log p(\mathbf{z}_1) - \log p(\mathbf{z}_0) = -\log \left| \det \frac{\partial F}{\partial \mathbf{z}_0} \right|$$

Requires invertible F. Cost O(D^3).

Theorem: Assuming that f is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, then:

$$\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t) \implies \frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\text{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

Function f does not have to be invertible. Cost O(D^2).

Continuous Normalizing Flows (CNF)

Automatically-invertible Normalizing Flows.
Planar CNF is smooth and much easier to train than planar NF.

Planar normalizing flow (Rezende and Mohamed, 2015)	Continuous analog of planar flow
$\mathbf{z}(t+1) = \mathbf{z}(t) + uh(w^\top \mathbf{z}(t) + b)$	$\frac{d\mathbf{z}(t)}{dt} = uh(w^\top \mathbf{z}(t) + b)$
$\log p(\mathbf{z}(t+1)) = \log p(\mathbf{z}(t)) - \log \left|1 + u^\top \frac{\partial h}{\partial \mathbf{z}}\right|$	$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -u^\top \frac{\partial h}{\partial \mathbf{z}(t)}$

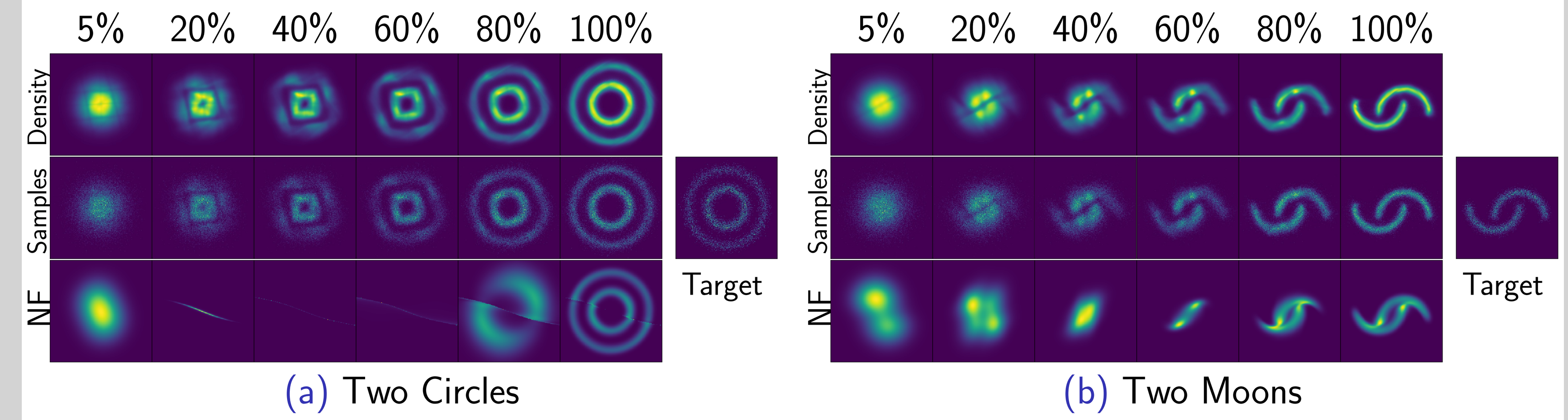


Figure: **Visualizing the transformation from noise to data.** Continuous normalizing flows are efficiently reversible, so we can train on a density estimation task and still be able to sample from the learned density efficiently.

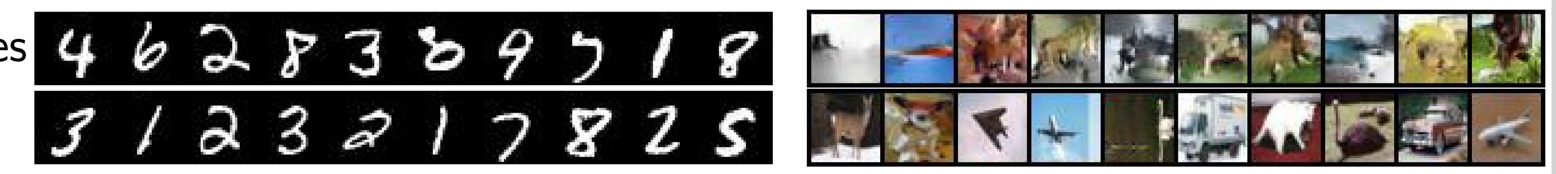
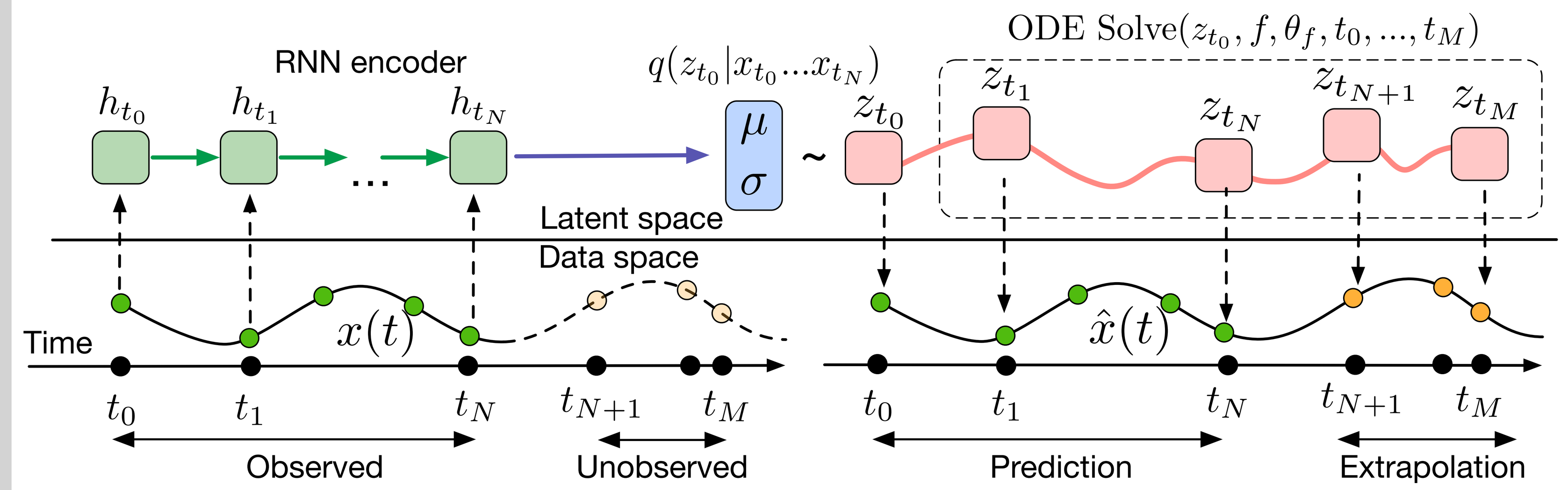


Figure: Have since scaled CNFs to images using Hutchinson's estimator (Grathwohl et al. 2018).

Continuous-time Generative Model for Time Series

Time series with irregular observation times.
No discretization of the timeline is needed.



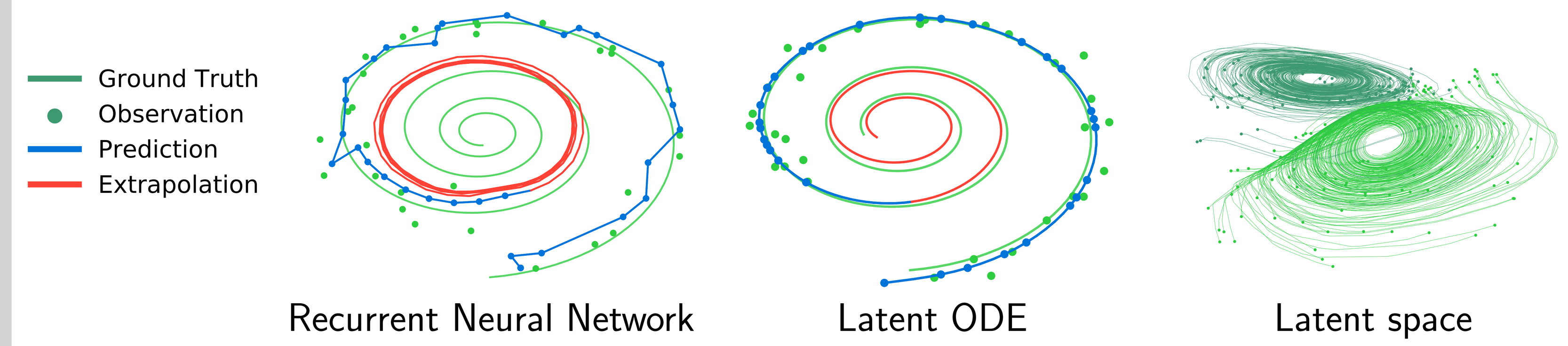


Figure: Latent ODE learns smooth latent dynamics from noisy observations.

Prior Works on ODE+DL

LeCun. "A theoretical framework for back-propagation." (1988)
Pearlmutter. "Gradient calculations for dynamic recurrent neural networks: a survey." (1993)
Haber & Ruthotto. "Stable Architectures for Deep Neural Networks." (2017)
Chang et al. "Multi-level Residual Networks from Dynamical Systems View." (2018)