

July 10th, 2020



A brief tutorial on Neural ODEs

Vikram Voleti

PhD student - Mila, University of Montreal

Visiting Researcher - University of Guelph

Prof. Christopher Pal

Prof. Graham Taylor

1. Ordinary Differential Equations (ODEs)

- Initial Value Problems
- Numerical Integration methods
- Fundamental theorem of ODEs

2. Neural ODEs

3. Later research

1st order Ordinary Differential Equation:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta)$$

x is a variable we are interested in,

t is (typically) time,

f is a function of x and t , it is the differential,

θ parameterizes f (optionally).

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Many physical processes follow this template!

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

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

Example:

$$\frac{dx}{dt} = 2t; \quad x(0) = 2; \quad x(1) = ?$$

$$\begin{aligned} \Rightarrow x(1) &= x(0) + \int_0^1 2t dt \\ &= x(0) + (t^2|_{t=1} - t^2|_{t=0}) \\ &= 2 + 1^2 - 0^2 \\ &= 3 \end{aligned}$$

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

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

What if this cannot be
analytically integrated?

Example:

$$\begin{aligned} \frac{dx}{dt} &= 2xt; \quad x(0) = 3 \\ \Rightarrow \int \frac{1}{2x} dx &= \int t dt \\ \Rightarrow \frac{1}{2} \log x &= \frac{1}{2} t^2 + c_0 \\ \Rightarrow x(t) &= ce^{t^2} \\ x(0) = 3 &\Rightarrow c = 3 \\ \therefore x(t) &= 3e^{t^2} \\ \Rightarrow x(1) &= 5.436 \end{aligned}$$

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

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

Approximations to $\int_{t_0}^{t_1} f(x(t), t, \theta) dt$

i.e. **Numerical Integration :**

- Euler method
- Runge-Kutta methods
- ...

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

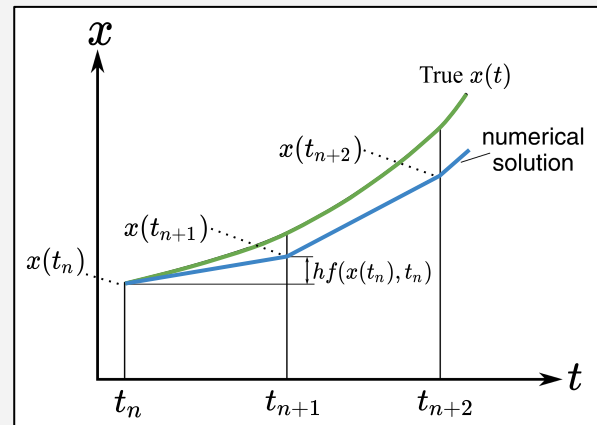
Solution:

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

1st-order Runge-Kutta / Euler's method:

$$t_{n+1} = t_n + h \quad \text{-----} \rightarrow \text{Step size } h$$

$$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n) \rightarrow \text{Update using derivative } f$$



<https://guide.freecodecamp.org/mathematics/differential-equations/eulers-method/>

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

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

1st-order Runge-Kutta / Euler's method:

$$t_{n+1} = t_n + h$$

$$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)$$

Example:

$$\frac{dx}{dt} = f(x, t) = 2xt; \quad x(0) = 3; \quad x(1) = ?$$

(Solution: $x(t) = 2e^{t^2}$; $x(1) = 5.436$)

$h = 0.25$

$$\begin{aligned} x(0.25) &= x(0) + 0.25 * f(x(0), 0) \\ &= 3 + 0.25 * (2 * 3 * 0) \\ &= 3 \end{aligned}$$

$$\begin{aligned} x(0.5) &= x(0.25) + 0.25 * f(x(0.25), 0.25) \\ &= 3 + 0.25 * (2 * 3 * 0.25) \\ &= 3.375 \end{aligned}$$

$$\begin{aligned} x(0.75) &= x(0.5) + 0.25 * f(x(0.5), 0.5) \\ &= 3.375 + 0.25 * (2 * 3.375 * 0.5) \\ &= 4.21875 \end{aligned}$$

$$\begin{aligned} x(1) &= x(0.75) + 0.25 * f(x(0.75), 0.75) \\ &= 4.21875 + 0.25 * (2 * 4.21875 * 0.75) \\ &= 5.8008 \end{aligned}$$

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

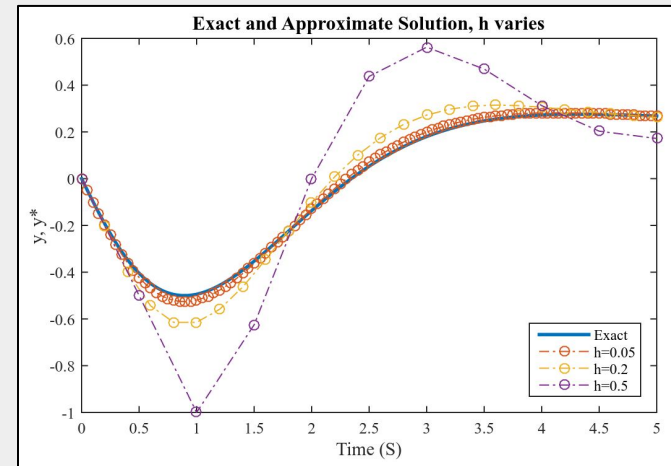
$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t, \theta) dt$$

1st-order Runge-Kutta / Euler's method:

$$t_{n+1} = t_n + h$$

$$x(t_{n+1}) = x(t_n) + hf(x(t_n), t_n)$$

Step size matters!



<https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html>

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

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

★ 4th-order Runge-Kutta method:

$$t_{n+1} = t_n + h$$

$$s_1 = f(x(t_n), t_n)$$

$$s_2 = f\left(x(t_n) + \frac{h}{2}s_1, t_n + \frac{h}{2}\right)$$

$$s_3 = f\left(x(t_n) + \frac{h}{2}s_2, t_n + \frac{h}{2}\right)$$

$$s_4 = f(x(t_n) + hs_3, t_n + h)$$

$$x(t_{n+1}) = x(t_n) + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4)$$

--- ➡ Default ODE solver used in MATLAB:
<https://blogs.mathworks.com/loren/2015/09/23/ode-solver-selection-in-matlab/>

Initial value problem:

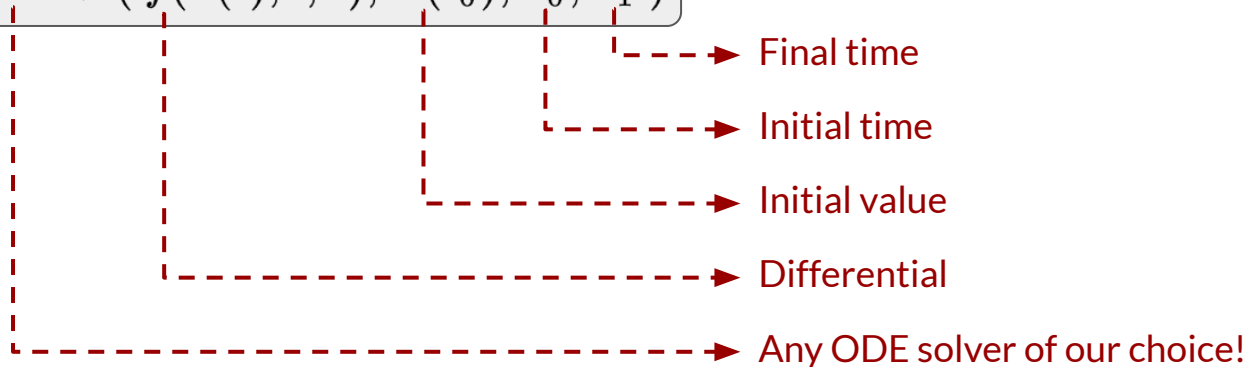
$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

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



`x(t1) = ODEsolve(f(x(t), t, θ), x(t0), t0, t1)`



Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

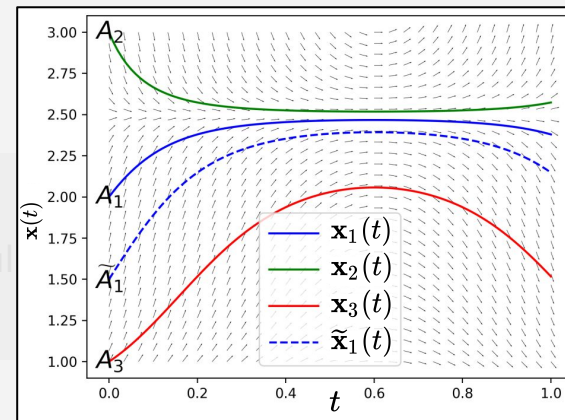
$$x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)$$

Fundamental Theorem of ODEs

Suppose f is continuously differentiable.

1. The solution is unique and exists for a certain interval.
2. Geometrically, $x(t)$ is a flow!
3. Solution curves for different solutions do not intersect.

<http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf>



<https://openreview.net/pdf?id=B1e9Y2NYvS>

1. Ordinary Differential Equations (ODEs)

- Initial Value Problems
- Numerical Integration methods
- Fundamental theorem of ODEs

2. **Neural ODEs** (Chen et al., NeurIPS 2018)

- **Adjoint method**
- **Applications**

3. Later research

Initial value problem:

$$\frac{dx(t)}{dt} = f(x(t), t, \theta); \quad x(t_0) \text{ is given}; \quad x(t_1) = ?$$

Solution:

$$x(t_1) = \text{ODESolve}(f(x(t), t, \theta), x(t_0), t_0, t_1)$$

f is a neural network!

Paradigm shift: whereas earlier f was pre-defined/hand-designed according to the domain, here we would like to estimate an f that suits our objective.

ODEs

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

Euler discretization

Vector
notation

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h f(\mathbf{x}_n, t_n, \theta)$$

Residual networks

$$\mathbf{x}_{l+1} = \text{ResBlock}(\mathbf{x}_l, \theta)$$

$$\mathbf{x}_{l+1} = \mathbf{x}_l + g(\mathbf{x}_l, \theta)$$

Skip connection

ODEs

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

Euler discretization

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h f(\mathbf{x}_n, t_n, \theta)$$

Forward propagation:

$$\mathbf{x}(t_1) = \text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1)$$

$$L(\mathbf{x}(t_1)) \rightarrow \frac{\partial L}{\partial \theta}$$

How to compute this?

Update θ to reduce L

<https://arxiv.org/pdf/1806.07366.pdf>

Residual networks

$$\mathbf{x}_{l+1} = \text{ResBlock}(\mathbf{x}_l, \theta)$$

$$\mathbf{x}_{l+1} = \mathbf{x}_l + g(\mathbf{x}_l, \theta)$$

Skip connection

$$\mathbf{y}_{pred} = \text{ResNet}(\mathbf{x})$$

Stacked ResBlocks

$$L(\mathbf{y}_{pred}) \rightarrow \frac{\partial L}{\partial \theta}$$

Update θ to reduce L

<https://arxiv.org/pdf/1512.03385.pdf>

ODEs

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

Euler discretization

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h f(\mathbf{x}_n, t_n, \theta)$$

Forward propagation:

$$\mathbf{x}(t_1) = \text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1)$$

$$L(\mathbf{x}(t_1)) \rightarrow \boxed{\frac{\partial L}{\partial \theta}}$$

Update θ to reduce L

Back-propagate through the ODE Solver!

High memory cost -

need to save all activations of all iterations of ODESolve.

Can we do better?

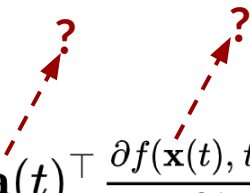
Yes.

$$L(\text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1)) \rightarrow \frac{\partial L}{\partial \theta}$$

Adjoint method (Pontryagin et al., 1962)

$$\text{adjoint } \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}} ; \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}$$

Forward propagation: $\mathbf{x}(t_1) = \text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1) \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$

$$\boxed{\frac{\partial L}{\partial \theta}} = \int_{t_1}^{t_0} -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta} dt$$


<https://arxiv.org/pdf/1806.07366.pdf>

$$L(\text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1)) \rightarrow \frac{\partial L}{\partial \theta}$$

Adjoint method (Pontryagin et al., 1962)

$$\text{adjoint } \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}} ; \frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}$$

Forward propagation: $\mathbf{x}(t_1) = \text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1) \Rightarrow \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$

Back-propagation: $x(t_0) = \text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_1), t_1, t_0)$

$$\Rightarrow \mathbf{a}(t_0) = \frac{\partial L}{\partial \mathbf{x}(t_0)} = \text{ODESolve}(-\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}}, \frac{\partial L}{\partial \mathbf{x}(t_1)}, t_1, t_0)$$

$$\therefore \boxed{\frac{\partial L}{\partial \theta}} = \int_{t_1}^{t_0} -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta} dt = \text{ODESolve}(-\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta}, \mathbf{0}_{|\theta|}, t_1, t_0)$$

Initial value is 0

<https://arxiv.org/pdf/1806.07366.pdf>

Forward propagation:

$$\mathbf{x}(t_1) = \text{ODESolve}(f(\mathbf{x}(t), t, \theta), \mathbf{x}(t_0), t_0, t_1)$$

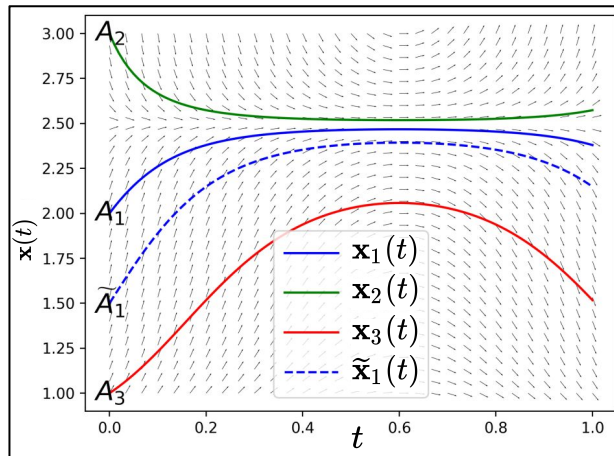
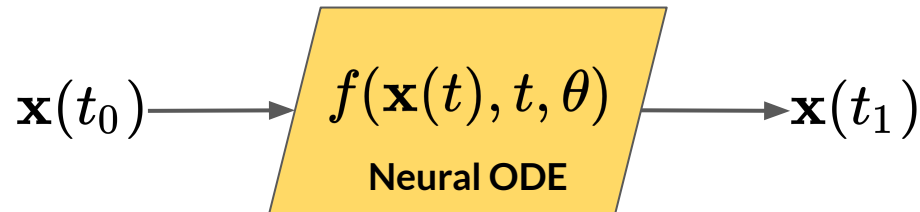
Compute $L(\mathbf{x}(t_1))$.

$$\mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}$$

Back-propagation:

$$\begin{bmatrix} \mathbf{x}(t_0) \\ \frac{\partial L}{\partial \mathbf{x}(t_0)} \\ \boxed{\frac{\partial L}{\partial \theta}} \end{bmatrix} = \text{ODESolve} \left(\begin{bmatrix} f(\mathbf{x}(t), t, \theta) \\ -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \mathbf{x}} \\ -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t), t, \theta)}{\partial \theta} \end{bmatrix}, \begin{bmatrix} \mathbf{x}(t_1) \\ \frac{\partial L}{\partial \mathbf{x}(t_1)} \\ \mathbf{0}_{|\theta|} \end{bmatrix}, t_1, t_0 \right)$$

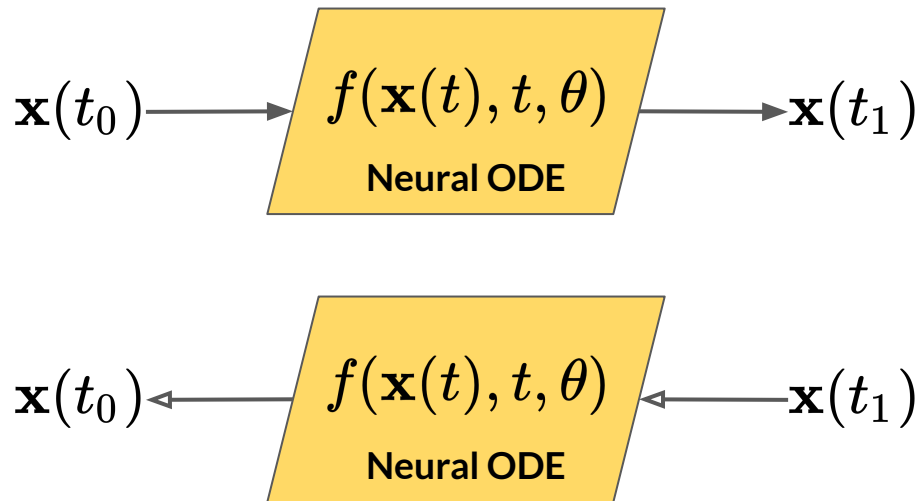
Update θ to reduce L



<https://openreview.net/pdf?id=B1e9Y2NYvS>

Neural ODEs describe a homeomorphism (flow).

- They preserve dimensionality.
- They form non-intersecting trajectories.



Neural ODEs are **reversible** models!
Just integrate forward/backward in time.

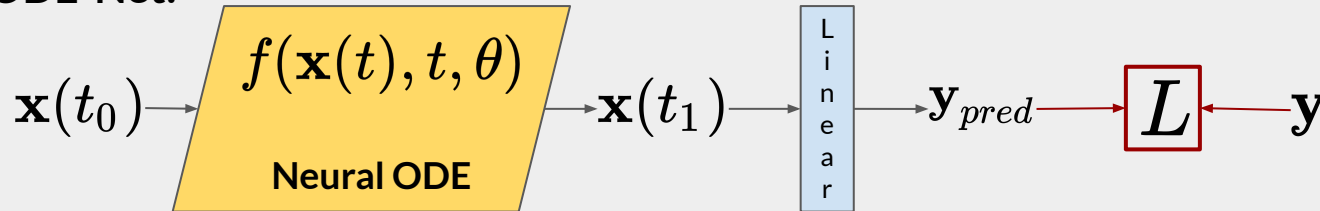
Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

ODE-Net:



~ Replacement for ResNets

Table 1: Performance on MNIST.

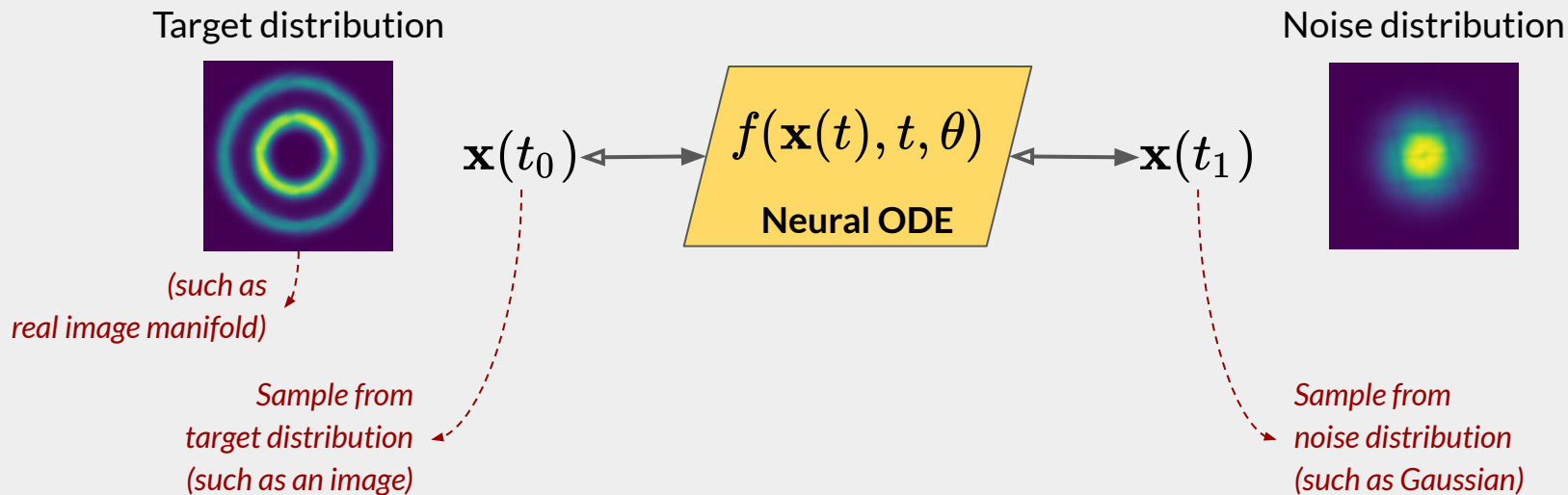
	Test error	# Params	Memory	Time
1-layer MLP	1.60%	0.24 M	-	-
ResNet	0.41%	0.60 M	$\mathcal{O}(L)$	$\mathcal{O}(L)$
RK-Net	0.47%	0.22 M	$\mathcal{O}(\tilde{L})$	$\mathcal{O}(\tilde{L})$
ODE-Net	0.42%	0.22 M	$\mathcal{O}(1)$	$\mathcal{O}(\tilde{L})$

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models



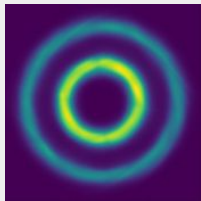
Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Target distribution



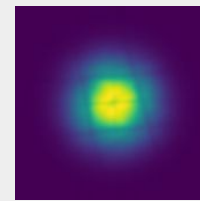
$\mathbf{x}(t_0)$

$f(\mathbf{x}(t), t, \theta)$

Neural ODE

$\mathbf{x}(t_1)$

Noise distribution



Likelihood estimation

using Change of Variables formula

$$\mathbf{x}_1 = g(\mathbf{x}_0) \Rightarrow \log p(\mathbf{x}_0) = \log p(\mathbf{x}_1) + \log \left| \det \frac{\partial g}{\partial \mathbf{x}_0} \right|$$

Train f to maximize the likelihood of the samples from target distribution $\log p(\mathbf{x}_0)$

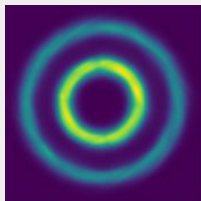
Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Target distribution



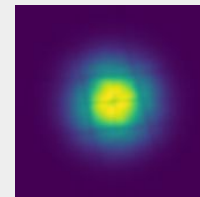
$\mathbf{x}(t_0)$

$f(\mathbf{x}(t), t, \theta)$

Neural ODE

$\mathbf{x}(t_1)$

Noise distribution



Likelihood estimation
using Change of Variables formula

Generate samples

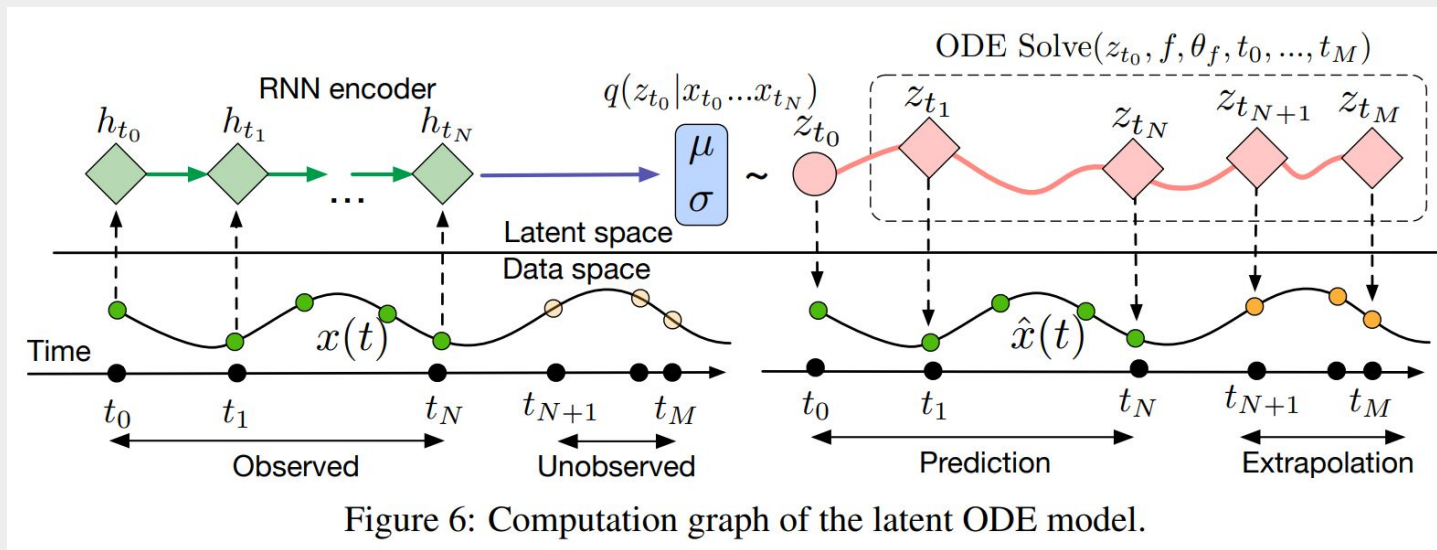
Sample from the noise distribution, transform it into a sample from the target distribution
using the trained Neural ODE.

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models



Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

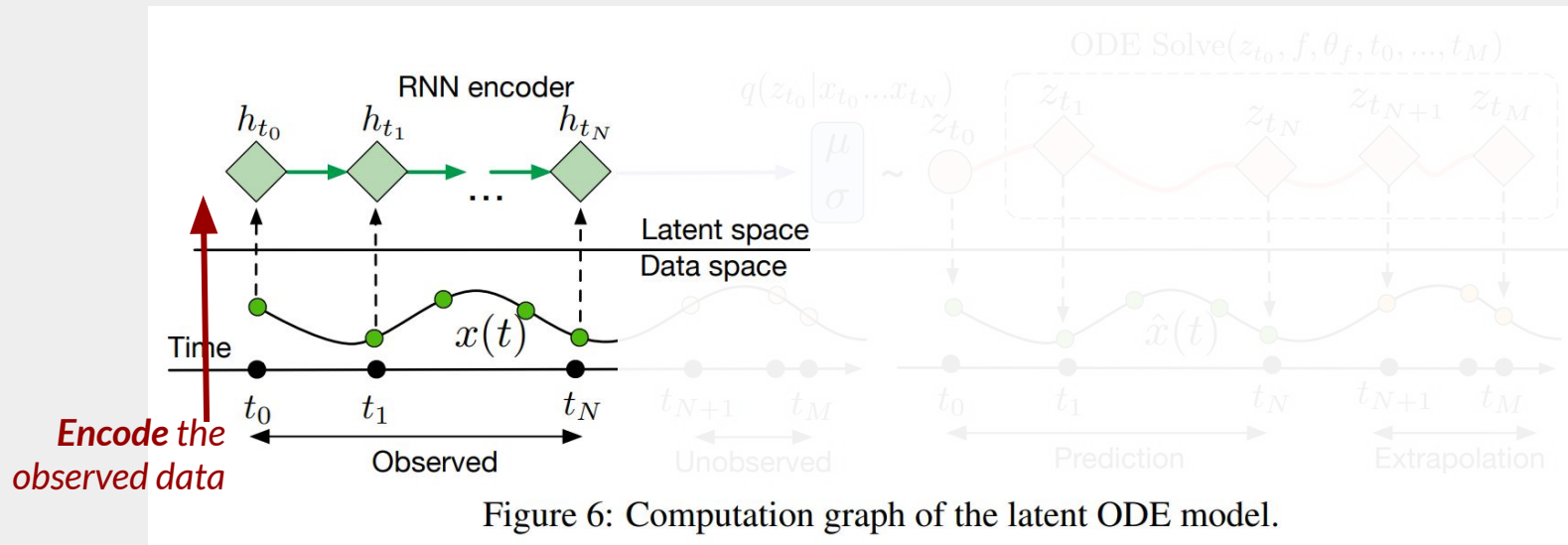


Figure 6: Computation graph of the latent ODE model.

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

Encode into a latent distribution (such as Gaussian) →

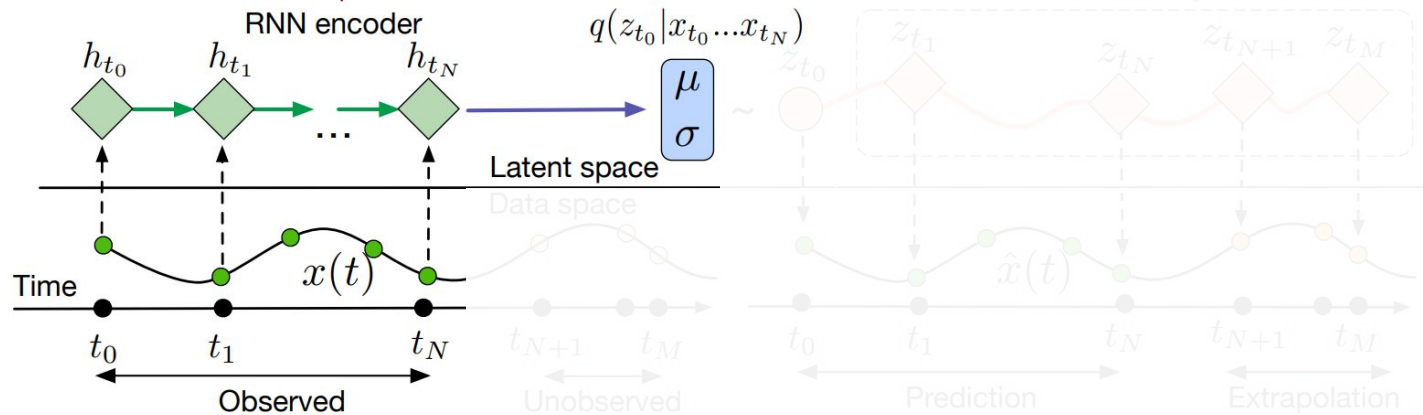


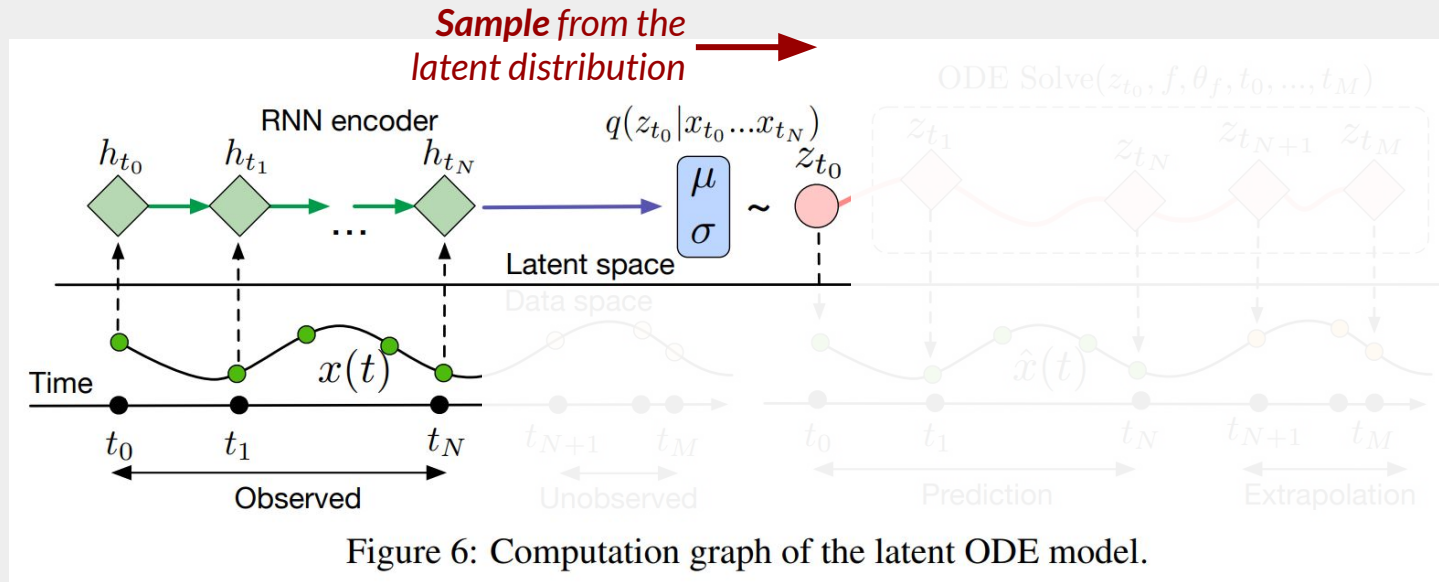
Figure 6: Computation graph of the latent ODE model.

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

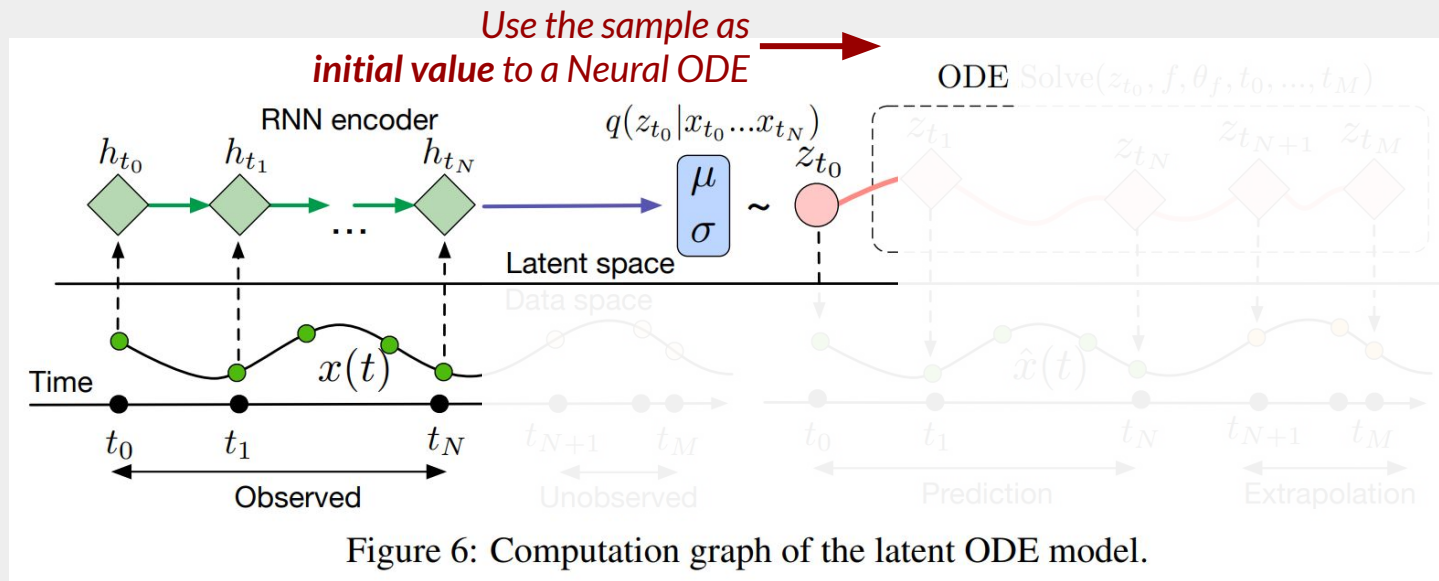


Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

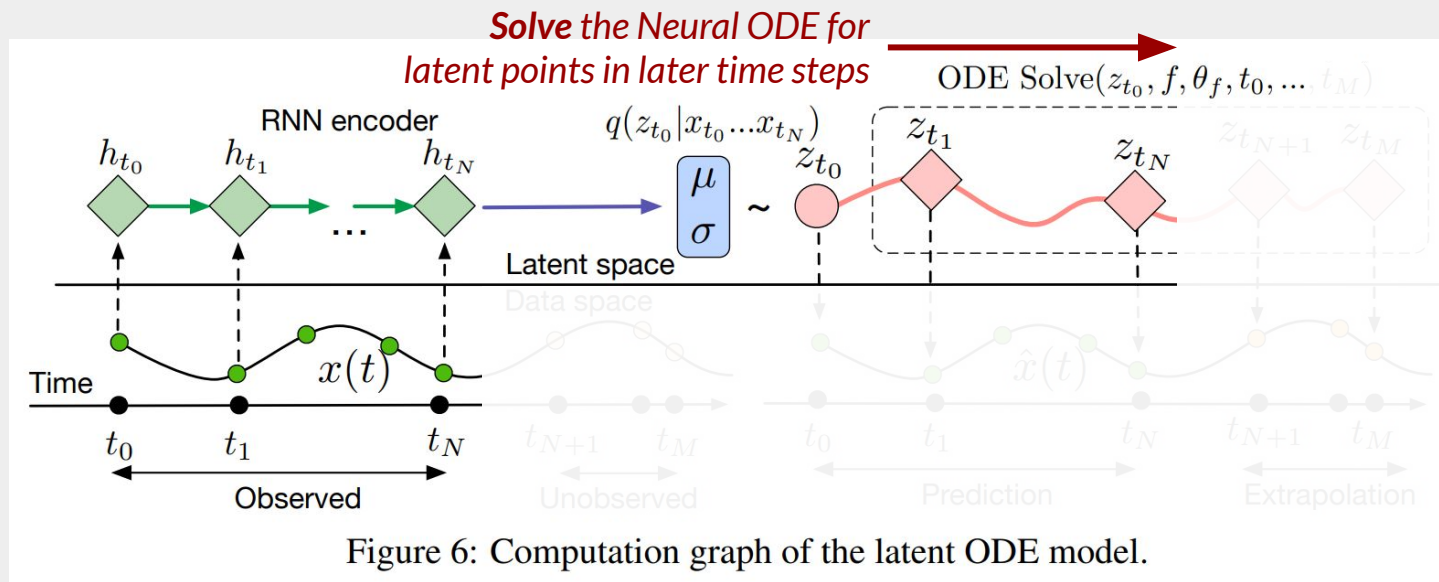


Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models

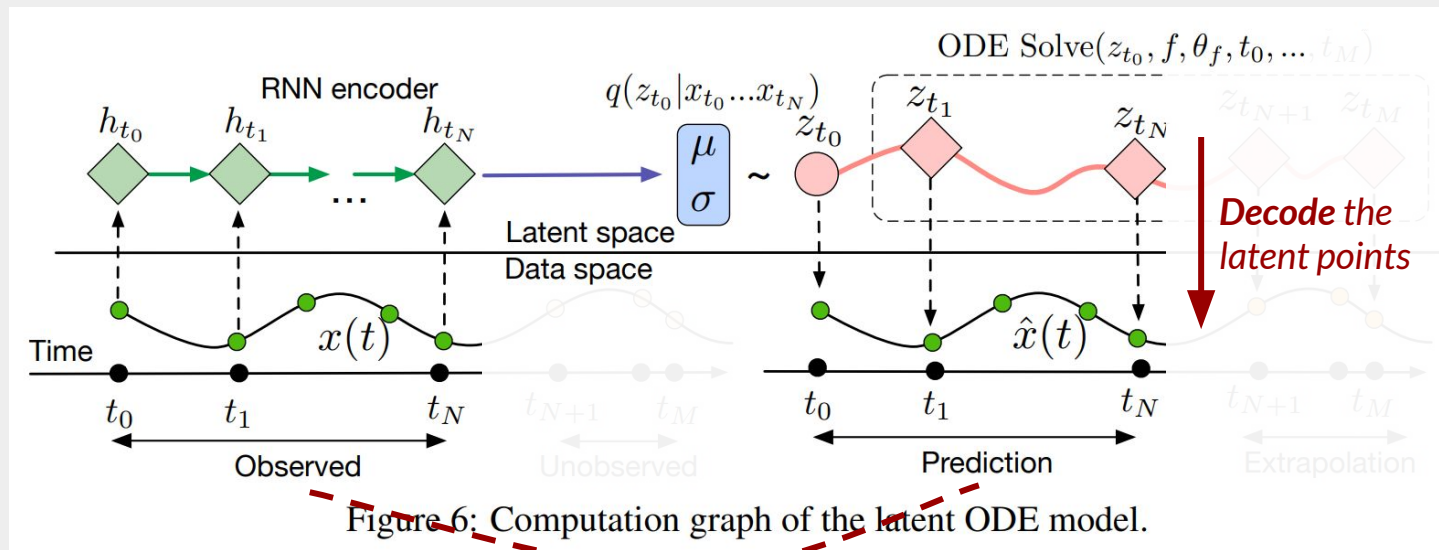


Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models



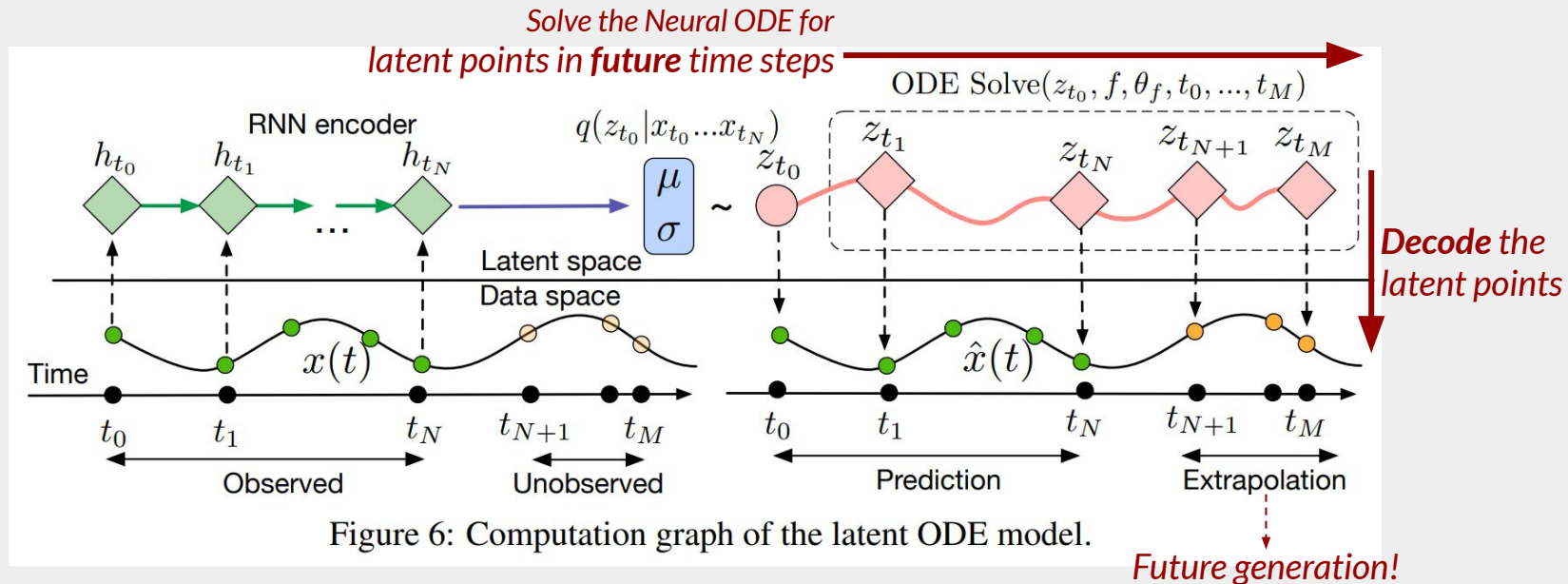
Compute loss

Applications

Supervised Learning

Continuous Normalizing Flows

Generative Latent Models



1. Ordinary Differential Equations (ODEs)

- Initial Value Problems
- Numerical Integration methods
- Fundamental theorem of ODEs

2. Neural ODEs (Chen et al., NeurIPS 2018)

- Adjoint method
- Applications

3. Later research

FFJORD: Free-form Continuous Dynamics For Scalable Reversible Generative Models (Grathwohl et al., ICLR 2019)

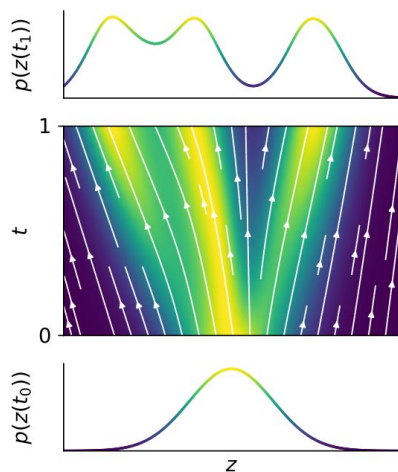
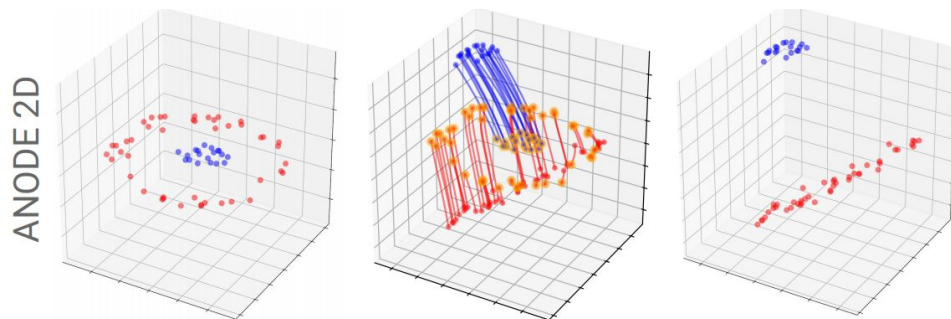


Figure 1: FFJORD transforms a simple base distribution at t_0 into the target distribution at t_1 by integrating over learned continuous dynamics.

- Essentially a better Continuous Normalizing Flow.
- Makes a better estimate for the log determinant term.
- “We demonstrate our approach on high-dimensional density estimation, image generation, and variational inference, achieving the state-of-the-art among exact likelihood methods with efficient sampling.”

<https://arxiv.org/pdf/1810.01367.pdf>

Augmented Neural ODEs (Dupont et al., NeurIPS 2019)



- Shows that Neural ODEs cannot model non-homeomorphisms (non-flows)
- **Augments** the state with additional dimensions to cover non-homeomorphisms
- Performs ablation study on toy examples and image classification

<https://arxiv.org/pdf/1904.01681.pdf>

ANODEV2: A Coupled Neural ODE Evolution Framework

(Zhang et al., NeurIPS 2019)

$$\begin{cases} z(1) = z(0) + \int_0^1 f(z(t), \theta(t)) dt & \text{“parent network”,} \\ \theta(t) = \theta(0) + \int_0^t q(\theta(t), p) dt, \quad \theta(0) = \theta_0 & \text{“weight network”.} \end{cases}$$

- Network weights are also a function of time
- Separate “weight network” generates the weights of the function network at a given time

<https://arxiv.org/pdf/1906.04596.pdf>

Latent ODEs for Irregularly-Sampled Time Series (Rubanova et al., NeurIPS 2019)

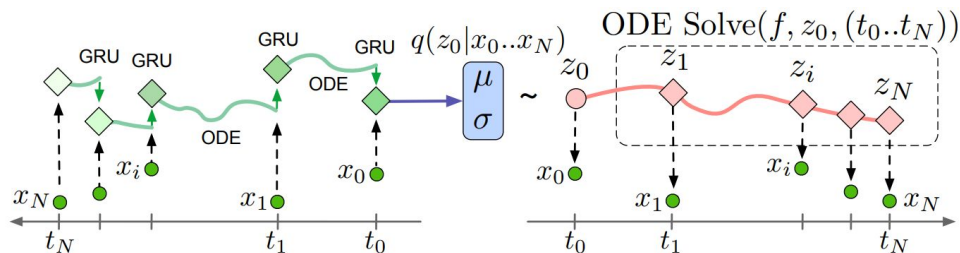


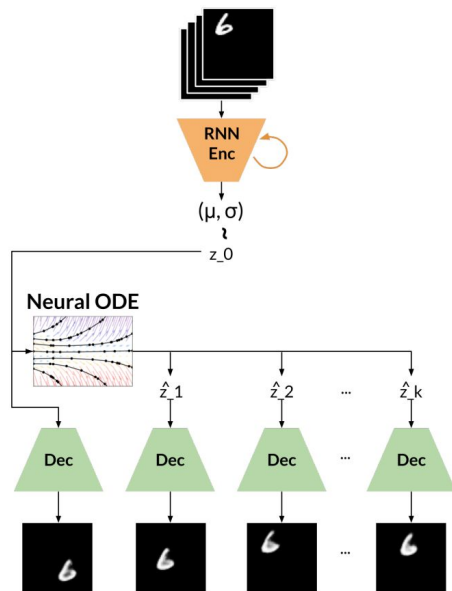
Figure 2: The Latent ODE model with an ODE-RNN encoder. To make predictions in this model, the ODE-RNN encoder is run backwards in time to produce an approximate posterior over the initial state: $q(z_0 | \{x_i, t_i\}_{i=0}^N)$. Given a sample of z_0 , we can find the latent state at any point of interest by solving an ODE initial-value problem. Figure adapted from Chen et al. [2018].

- Improves the generative latent variable framework for irregularly-sampled time series
- Essentially uses an ODE in the encoder where samples are missing
- Shows results on toy data, MuJoCo, PhysioNet

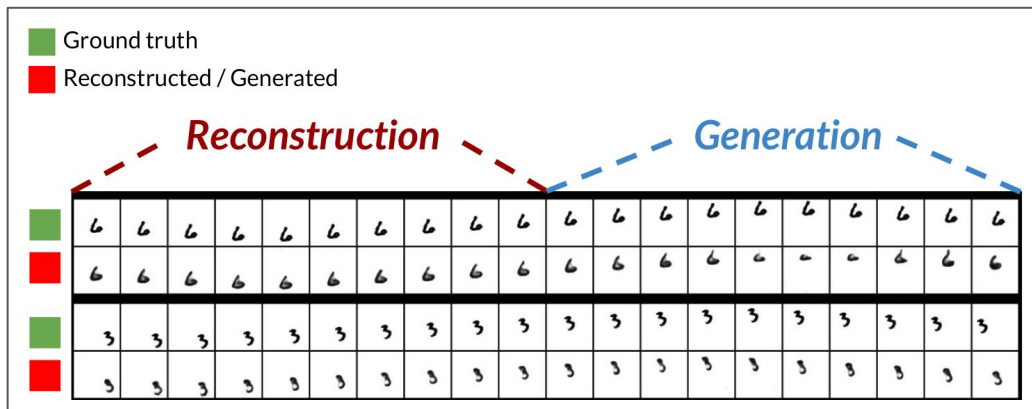
<https://arxiv.org/pdf/1907.03907.pdf>

Simple Video Generation using Neural ODEs

(David Kanaa*, Vikram Voleti*, Samira Kahou, Christopher Pal; NeurIPS 2019 Workshop)



- Video generation as a generative latent variable model using Neural ODEs



<https://sites.google.com/view/neurips2019lire/accepted-papers?authuser=0>

ODE2VAE: Deep generative second order ODEs with Bayesian neural networks (Yildiz et al., NeurIPS 2019)

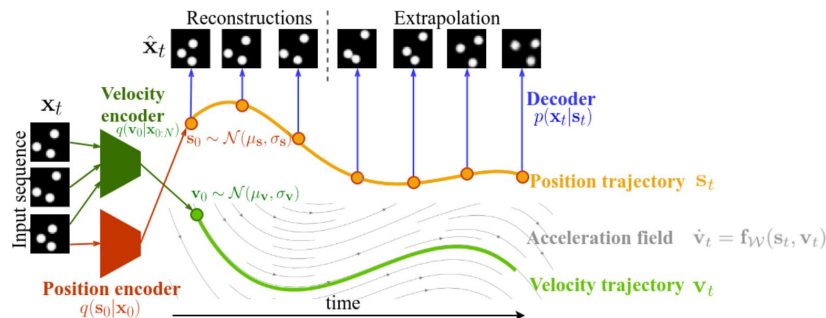


Figure 2: A schematic illustration of ODE²VAE model. Position encoder (μ_s, σ_s) maps the first item \mathbf{x}_0 of a high-dimensional data sequence into a distribution of the initial position \mathbf{s}_0 in a latent space. Velocity encoder (μ_v, σ_v) maps the first m high-dimensional data items $\mathbf{x}_{0:m}$ into a distribution of the initial velocity \mathbf{v}_0 in a latent space. Probabilistic latent dynamics are implemented by a second order ODE model $\mathbf{f}_{\mathcal{W}}$ parameterised by a Bayesian deep neural network (\mathcal{W}). Data points in the original data domain are reconstructed by a decoder.

- Uses 2nd-order Neural ODE
- Uses a Bayesian Neural Network
- Showed results modelling video generation as a generative latent variable model using (2nd-order Bayesian) Neural ODE

<https://papers.nips.cc/paper/9497-ode2vae-deep-generative-second-order-odes-with-bayesian-neural-networks.pdf>

Neural Jump Stochastic Differential Equations (Jia et al., NeurIPS 2019)

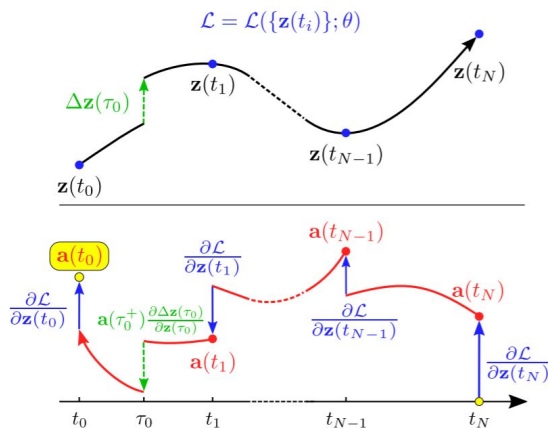


Figure 1: Reverse-mode differentiation of an ODE with discontinuities. Each jump $\Delta \mathbf{z}(\tau_j)$ in the latent vector (green, top panel) also introduces a discontinuity for adjoint vectors (green, bottom panel).

- Models continuous + discrete dynamics of a hybrid system
- Discontinuities are modelled as stochastic events
- Show results on real-world and synthetic point process datasets

<https://papers.nips.cc/paper/9177-neural-jump-stochastic-differential-equations.pdf>

Neural SDE: Stabilizing Neural ODE Networks with Stochastic Noise (Liu et al., 2019)

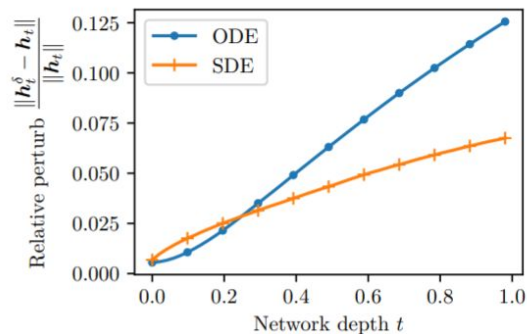


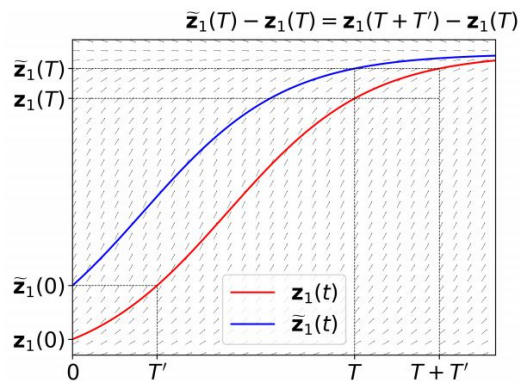
Figure 4: Comparing the perturbations of hidden states, ϵ_t , on both ODE and SDE (we choose dropout-style noise).

- Random noise injection into Neural ODEs
- Adds a diffusion term into the Neural ODE formulation, denoting a continuous time stochastic process
- Makes a case for robustness

<https://arxiv.org/pdf/1906.02355.pdf>

On Robustness of Neural Ordinary Differential Equations

(Yan et al., ICLR 2020)



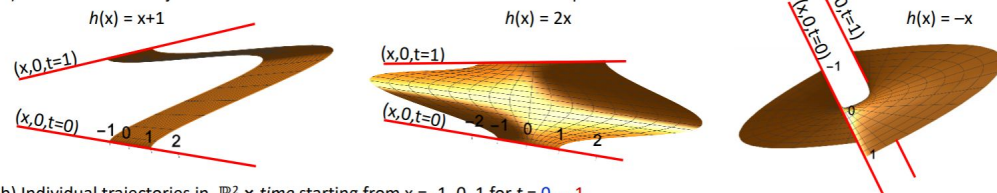
- Ablation study on adversarial attacks on ODE-Nets
- Introduces new regularization term to improve robustness

Figure 3: An illustration of the time-invariant property of ODEs. We can see that the curve $\tilde{z}_1(t)$ is exactly the horizontal translation of $z_1(t)$ on the interval $[T', \infty)$.

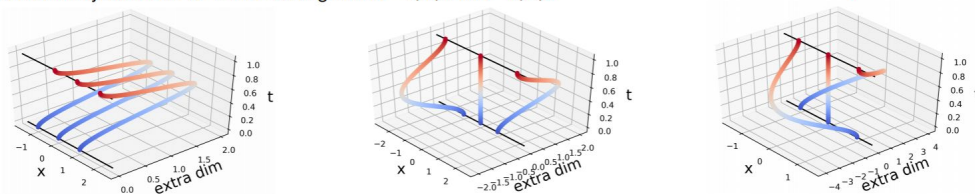
<https://arxiv.org/pdf/1910.05513.pdf>, <https://openreview.net/pdf?id=B1e9Y2NYvS>

Approximation Capabilities of Neural ODEs and Invertible Residual Networks (Zhang et al., ICML 2020)

a) Surface view of trajectories in $\mathbb{R}^2 \times \text{time}$ for three different homeomorphisms $h: \mathbb{R} \rightarrow \mathbb{R}$



b) Individual trajectories in $\mathbb{R}^2 \times \text{time}$ starting from $x = -1, 0, 1$ for $t = 0, \dots, 1$

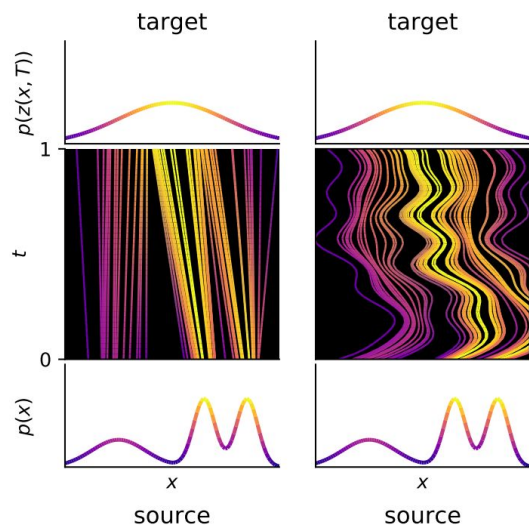


- Provides guarantees on modelling capability of homeomorphisms v/s the capacity of the Neural ODE

Figure 1: Trajectories in \mathbb{R}^{2p} that embed an $\mathbb{R}^p \rightarrow \mathbb{R}^p$ homeomorphism, using $f(\tau) = (1 - \cos \pi \tau)/2$ and $g(\tau) = (1 - \cos 2\pi \tau)$. Three examples for $p = 1$ are shown, including the mapping $h(x) = -x$ that cannot be modeled by Neural ODE on \mathbb{R}^p , but can in \mathbb{R}^{2p} .

<https://arxiv.org/pdf/1907.12998.pdf>

How to Train Your Neural ODE : the world of Jacobian and kinetic regularization (Finlay et al., ICML 2020)



(a) Optimal transport map

(b) generic flow

Figure 1. Optimal transport map and a generic normalizing flow.

- Makes a link between the flow in Neural ODEs and optimal transport
- Introduces two new regularization terms to constrain flows to straight lines
- Speeds up training of Neural ODEs

<https://arxiv.org/pdf/2002.02798.pdf>

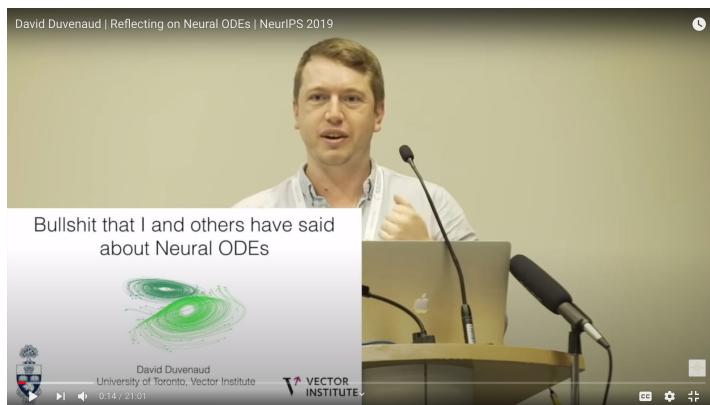
Scalable Gradients for Stochastic Differential Equations

(Li et al., AISTATS 2020)

- Generalizes the adjoint method to stochastic dynamics defined by SDEs :
“stochastic adjoint sensitivity method.”
- PyTorch Implementation of Differentiable SDE Solvers:
<https://github.com/google-research/torchsde>

<https://arxiv.org/pdf/2001.01328.pdf>

“Bullshit that I and others have said about Neural ODEs” (David Duvenaud, ML Retrospectives Workshop @ NeurIPS 2019)



- Summarizes what went right and wrong in the publication of the Neural ODEs paper

<https://www.youtube.com/watch?v=YZ-E7A3V2w>

- <http://faculty.bard.edu/belk/math213/InitialValueProblems.pdf>
- ODE Solvers: <https://math.temple.edu/~queisser/assets/files/Talk3.pdf>
- Textbook : <https://users.math.msu.edu/users/gnagy/teaching/ode.pdf>
- <https://lpsa.swarthmore.edu/NumInt/NumIntFirst.html>
- Excellent blog post on ODE solvers: <https://blogs.mathworks.com/loren/2015/09/23/ode-solver-selection-in-matlab/>
- Autodiff tutorial:
http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/readings/L06%20Automatic%20Differentiation.pdf
- Course on Neural Networks & Deep Learning by Roger Grosse & Jimmy Ba, University of Toronto -
http://www.cs.toronto.edu/~rgrosse/courses/csc421_2019/
- Official Neural ODE code torchdiffeq : <https://github.com/rtqichen/torchdiffeq>
- DiffEqML's torchdyn : <https://github.com/DiffEqML/torchdyn>
- TorchSDE : <https://github.com/google-research/torchsde>

Thank you!