Neural Ordinary Differential Equations

Paper by

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

University of Toronto, Vector Institute

Presentation by Raphaël Cohen

Motivations

We have been using neural networks to solve vaguely defined problems.

Current	What is new with this paper ?
• Great for discrete classification • Time series with even intervals • Some data generation • Bad parameter efficiency • Very static	• Adaptive Network / Dynamic • Better fit for continuous problems • New field of study for neural networks • Trade numerical precision for speed • Input prior knowledge

What do differential equations have to do with machine learning?

There are three common ways to define a nonlinear transform:

- Direct modeling
  - You know the exact function
- Machine learning
  - You don't know anything
- Differential equations
  - You know the structure

What do differential equations have to do with machine learning?

$$\text{rabbits tomorrow} = \text{Model}(\text{rabbits today}).$$

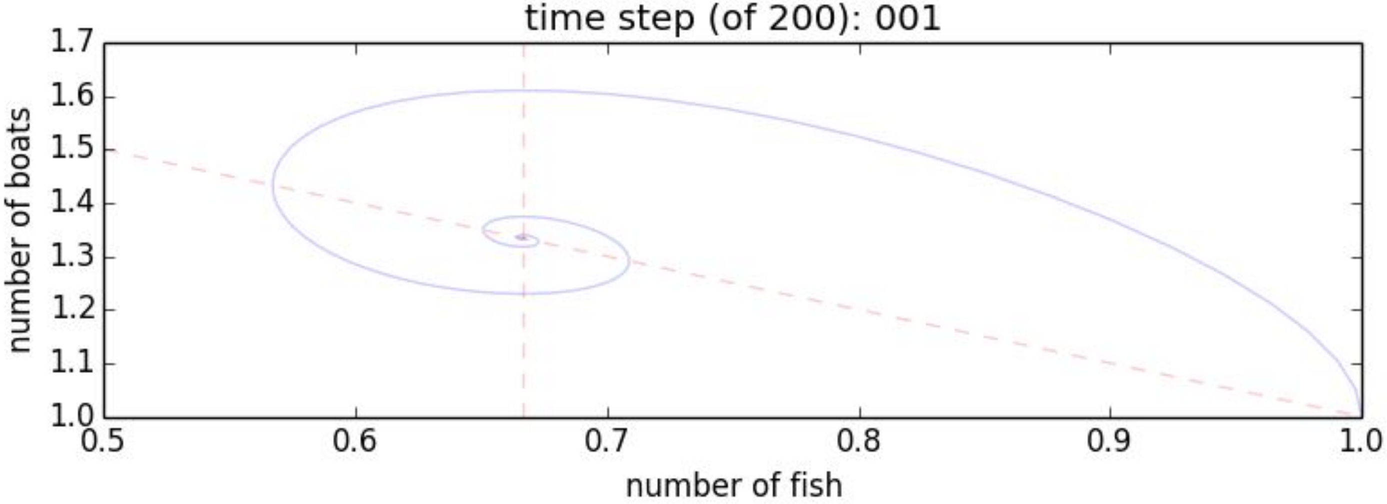
$$\text{rabbits'}(t) = \alpha \cdot \text{rabbits}(t)$$

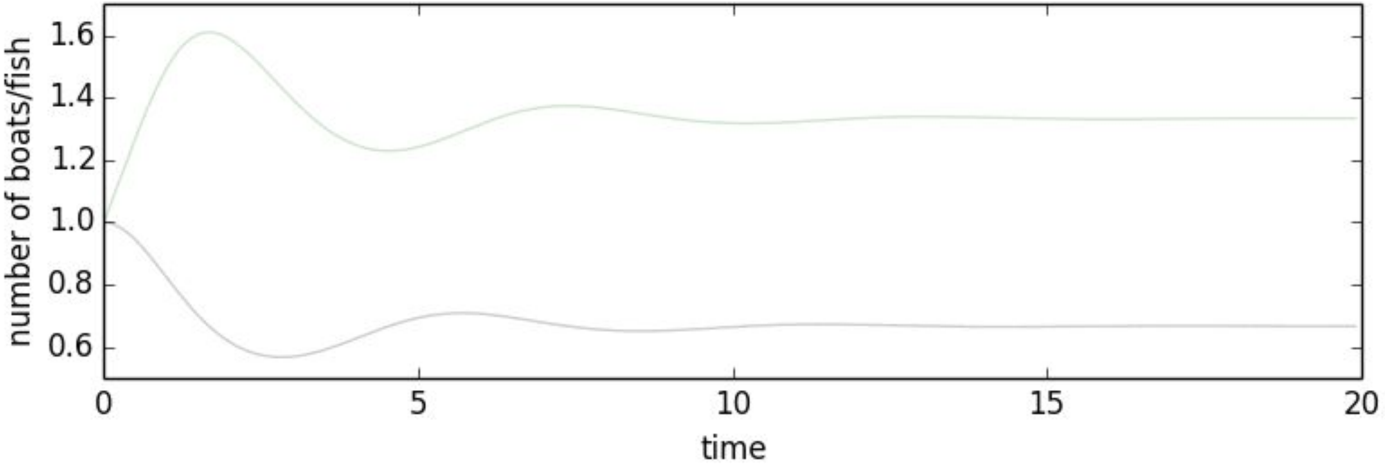
What is an ODE ?

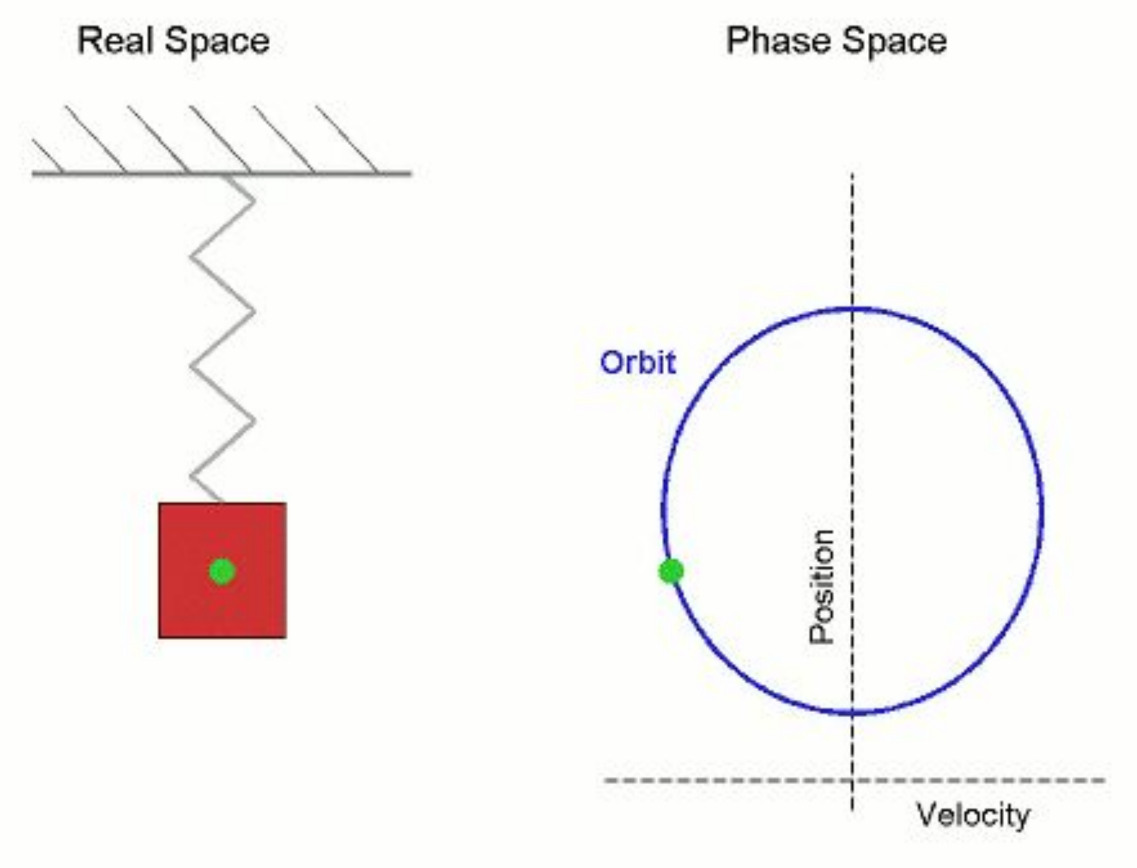
These are essentially equations of how things change and thus “where things will be” is the solution to a differential equation.

$$y' = F(x, y)\,, \quad y_0 = y(x_0)$$

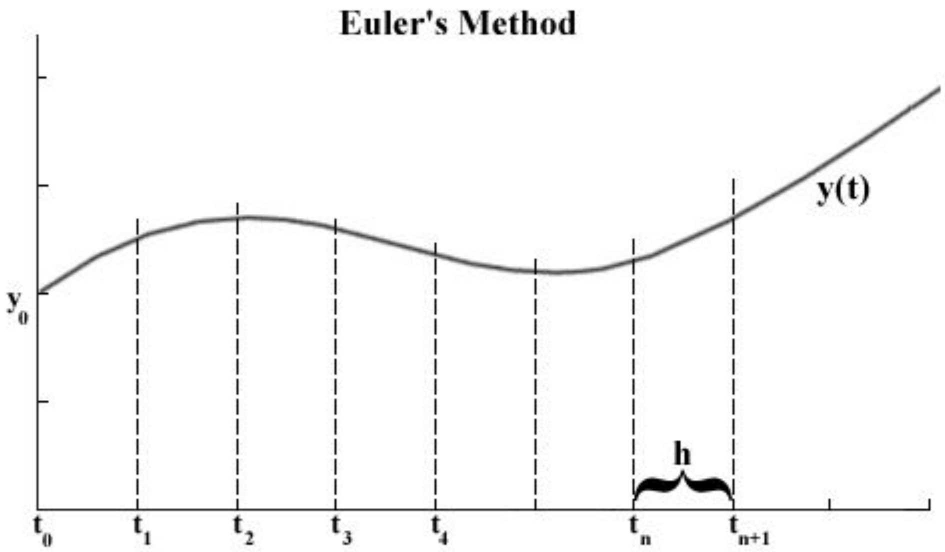
$$\frac{dx}{dt} = x(2 - y - x)$$
$$\frac{dy}{dt} = -y(1 - 1.5x)$$






Real Space
Phase Space
Orbit
Position
Velocity

Numerical solutions



$$\frac{dy}{dt} = f(t,y)$$

$$y(t_0) = y_0$$

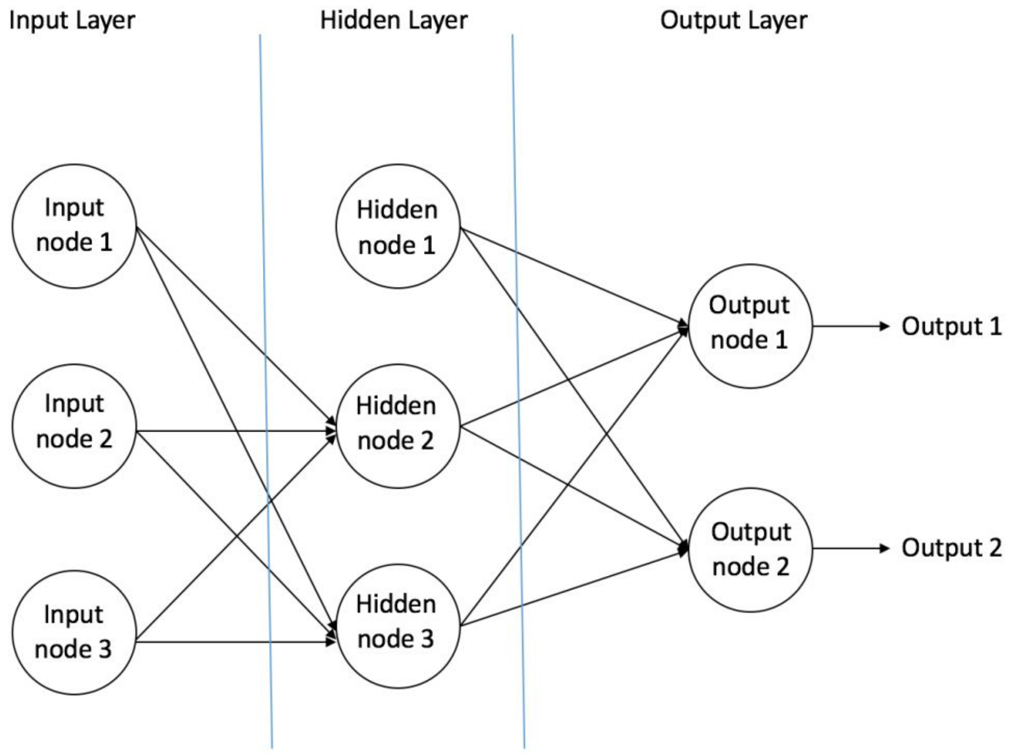
y(t) is the solution of this differential equation

This is Euler's formula to approximate the solutions.

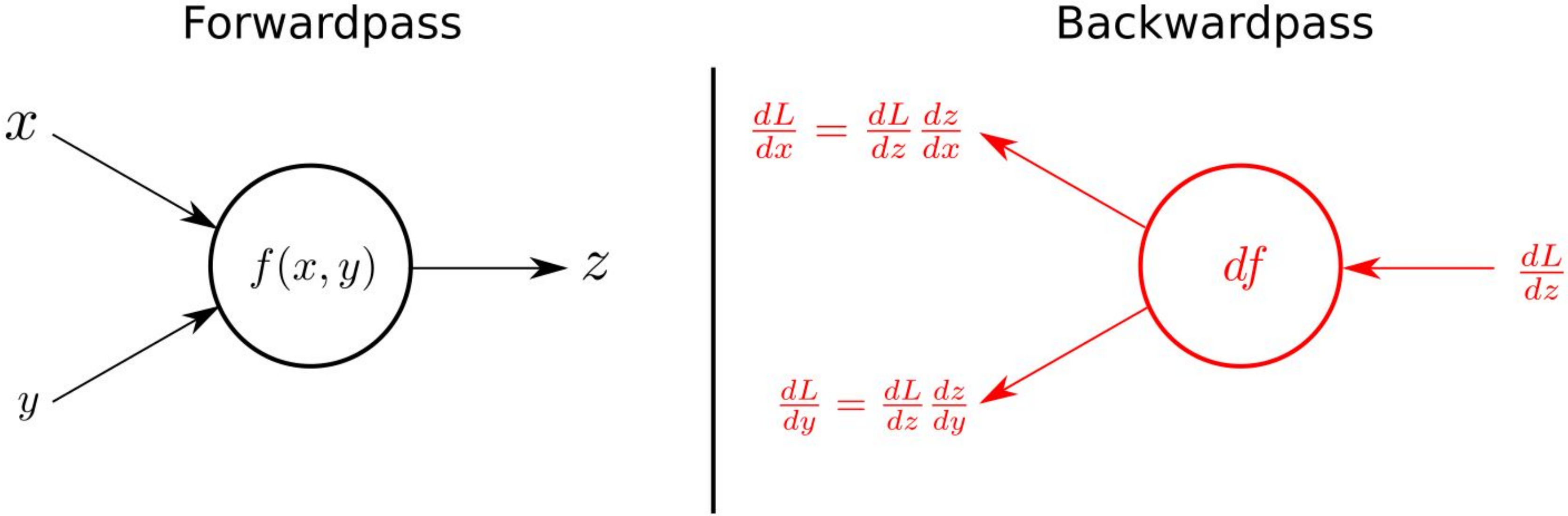
$$\boxed{y_{n+1} = y_n + hf(t_n, y_n)}$$

Brief introduction - Neural Networks

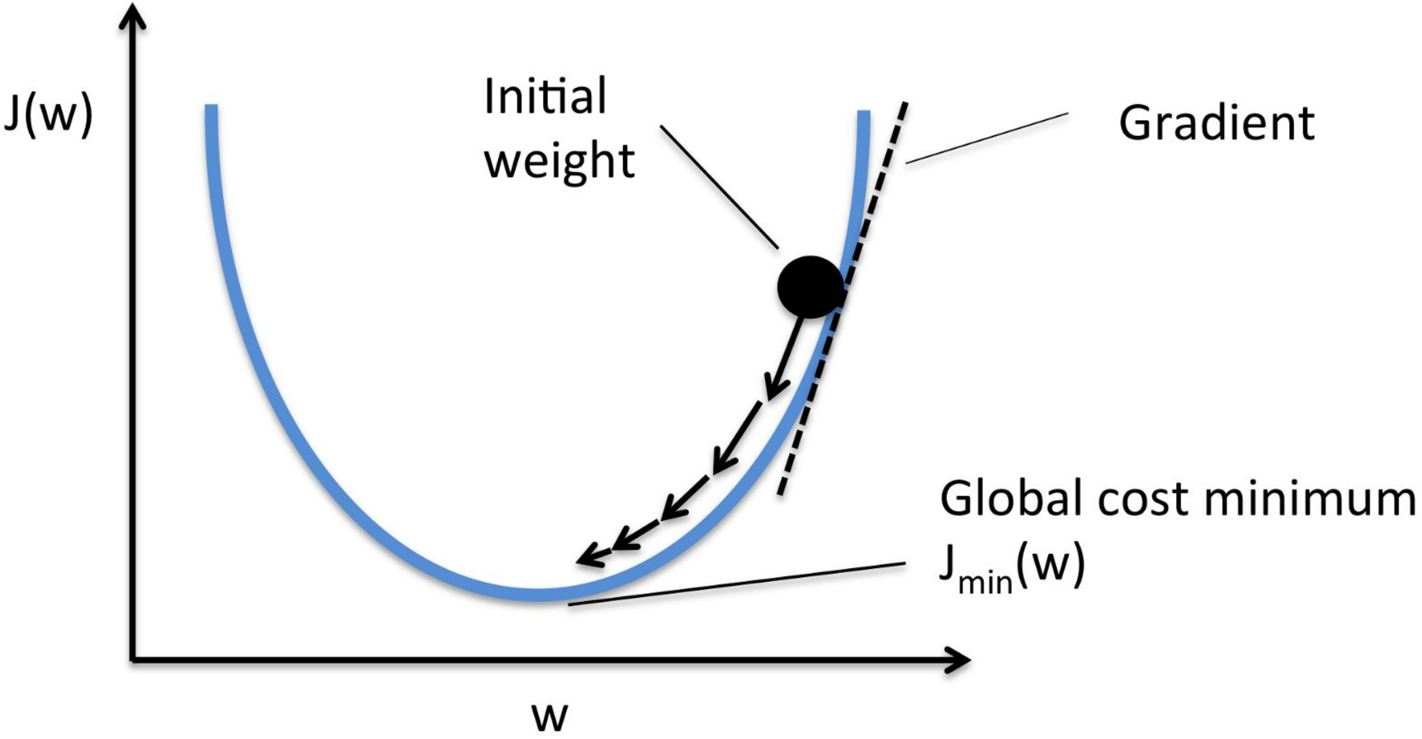
$$y = ML(x)$$



Backward propagation



Gradient descent



Residual Networks

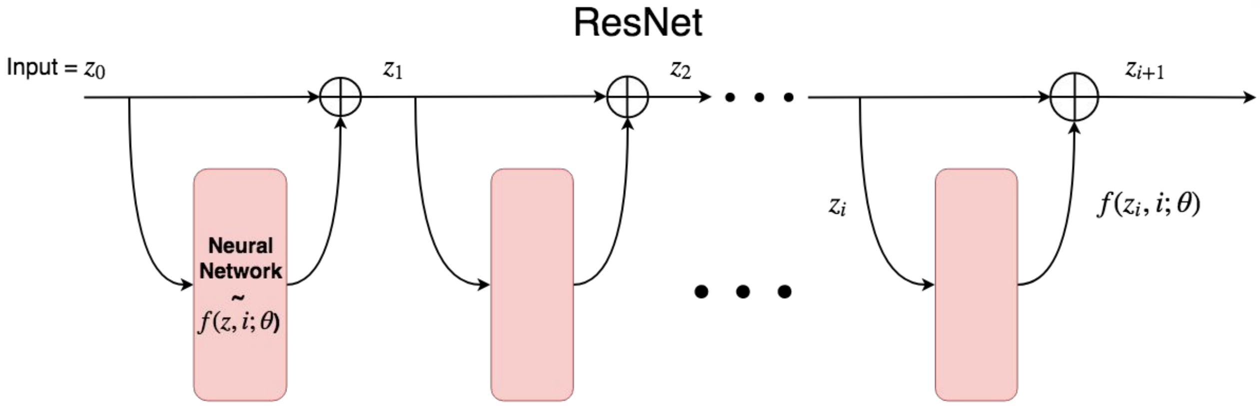
How to partially avoid choosing the number of layers ?

$$z_1 = f_0(x) + x$$

$$z_2 = f_1(z_1) + z_1$$

$$z_3 = f_2(z_2) + z_2$$

$$z_4 = f_3(z_3) + z_3$$



Neural Ordinary Differential Equations

What would we have to do to transform residual networks into ODE-Nets ?

$$z_1 = f_0(x) + x$$

$$z_2 = f_1(z_1) + z_1$$

$$z_3 = f_2(z_2) + z_2$$

$$z_4 = f_3(z_3) + z_3$$

$$z(t=0) = x$$

$$z(1) - z(0) = f(z(0), t=0)$$

$$z(2) - z(1) = f(z(1), t=1)$$

$$z(3) - z(2) = f(z(2), t=2)$$

$$z(4) - z(3) = f(z(3), t=3)$$

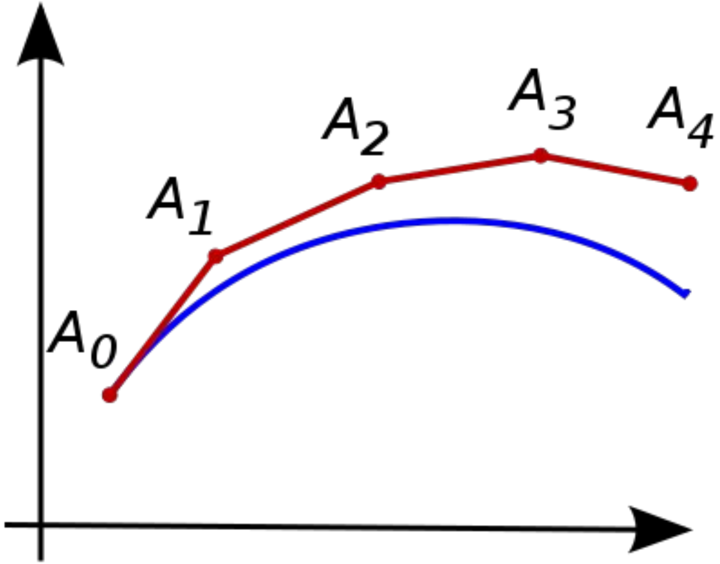
$$\Delta y = (y_{\text{next}} - y_{\text{prev}}) = \Delta x \cdot ML(x)$$

Neural Ordinary Differential Equations

We can now go to the continuous domain and write an ODE:

$$\frac{dz}{dt} = f(z(t), t; \theta)$$

$$y_{i+1} = y_i + \Delta x \cdot ML(x_i).$$



Solve ODE - Adaptive Methods

Advanced ODE solvers differ from the simple Euler method in multiple aspects:

- They are higher order methods
- More importantly, they have adaptive step-sizes

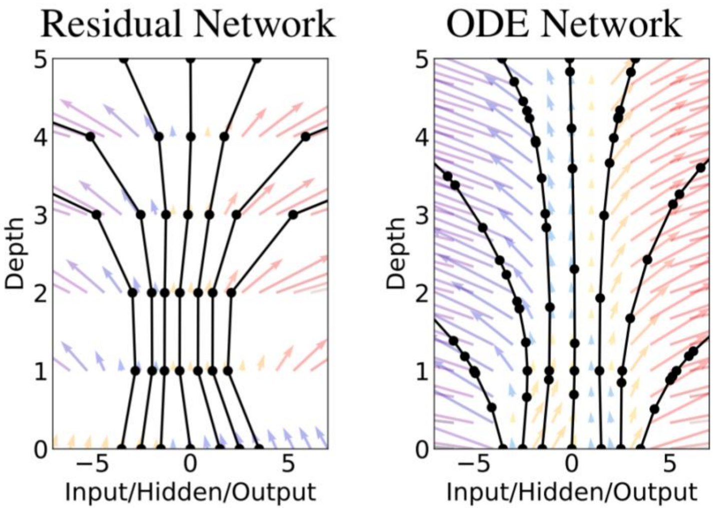
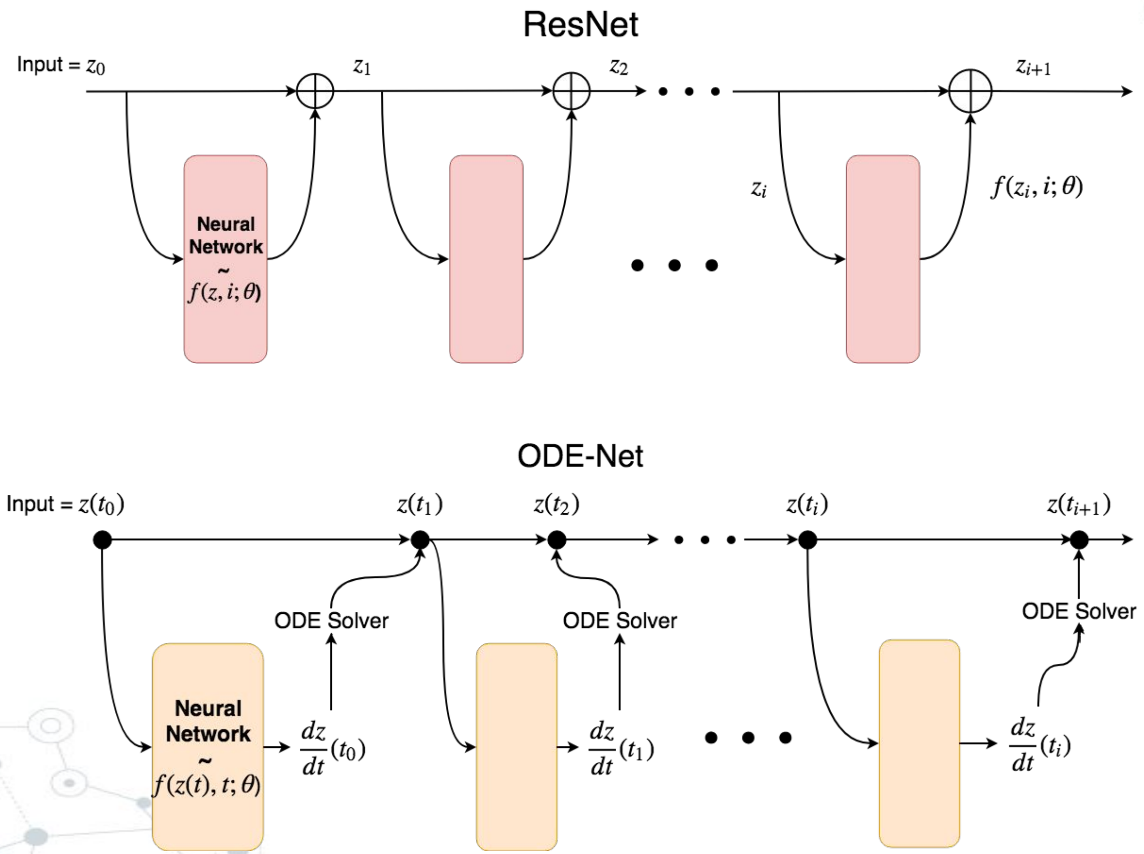


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

ResNet vs ODE-Net



Reverse-mode automatic differentiation of ODE solutions

What about backpropagation ?

"Differentiating through the operations of the forward pass is straightforward but incurs a high memory cost and introduces additional numerical error"

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

Adjoint Method

To optimize L, we require gradients with respect to its parameters: z(t0), t0 , t1, and θ.

The first step is to determining how the gradient of the loss depends on the hidden state z(t) at each instant.

Adjoint:

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

Adjoint Method

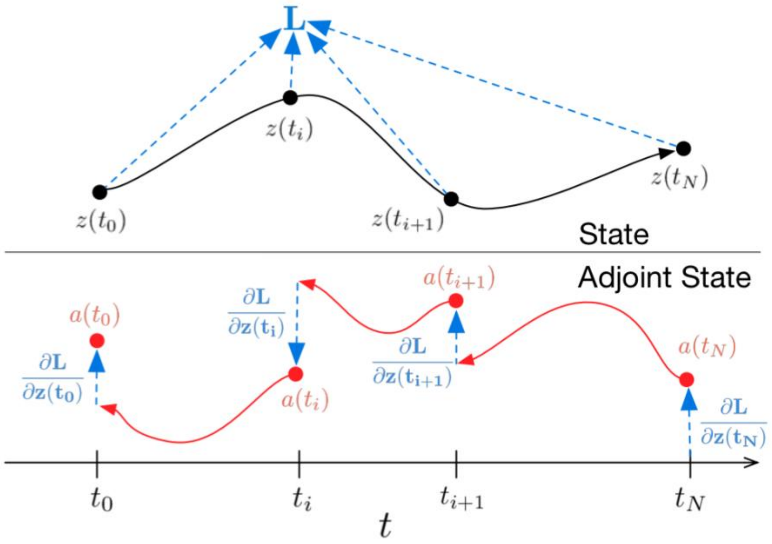


Figure 2: Reverse-mode differentiation through an ODE solver requires solving an augmented system backwards in time. This adjoint state is updated by the gradient at each observation.

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

$$\frac{da(t)}{dt} = -a(t)^{\top} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

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

Pseudocode for Automatic Gradient Computing

Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient ${\partial L}/{\partial \mathbf{z}(t_1)}$

$\frac{\partial L}{\partial t_1} = \frac{\partial L}{\partial \mathbf{z}(t_1)}^{\intercal} f(\mathbf{z}(t_1), t_1, \theta)$ ▷ Compute gradient w.r.t. t_1

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}, -\frac{\partial L}{\partial t_1}]$ ▷ Define initial augmented state

def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), -, -], t, \theta$): ▷ Define dynamics on augmented state

    return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^{\intercal}\frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^{\intercal}\frac{\partial f}{\partial \theta}, -\mathbf{a}(t)^{\intercal}\frac{\partial f}{\partial t}]$ ▷ Concatenate time-derivatives

$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial t_0}, \frac{\partial L}{\partial t_1}$ ▷ Return all gradients

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

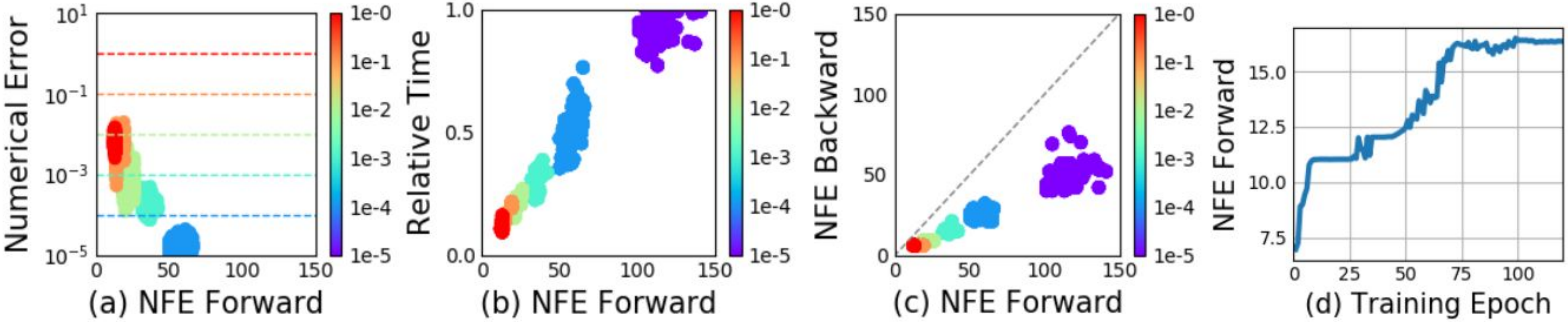
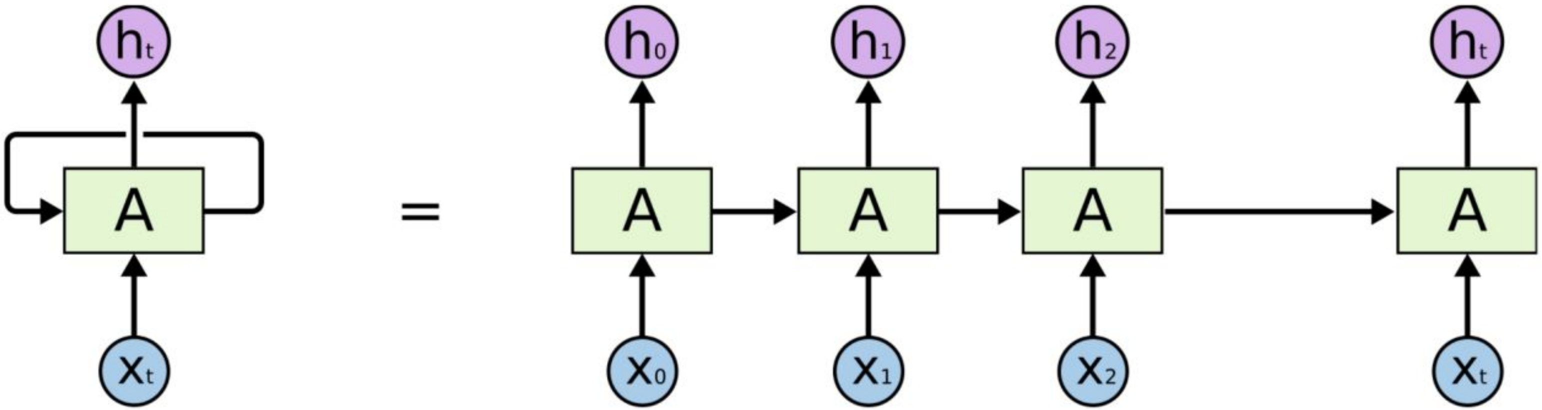


Figure 3: Statistics of a trained ODE-Net. (NFE = number of function evaluations.)

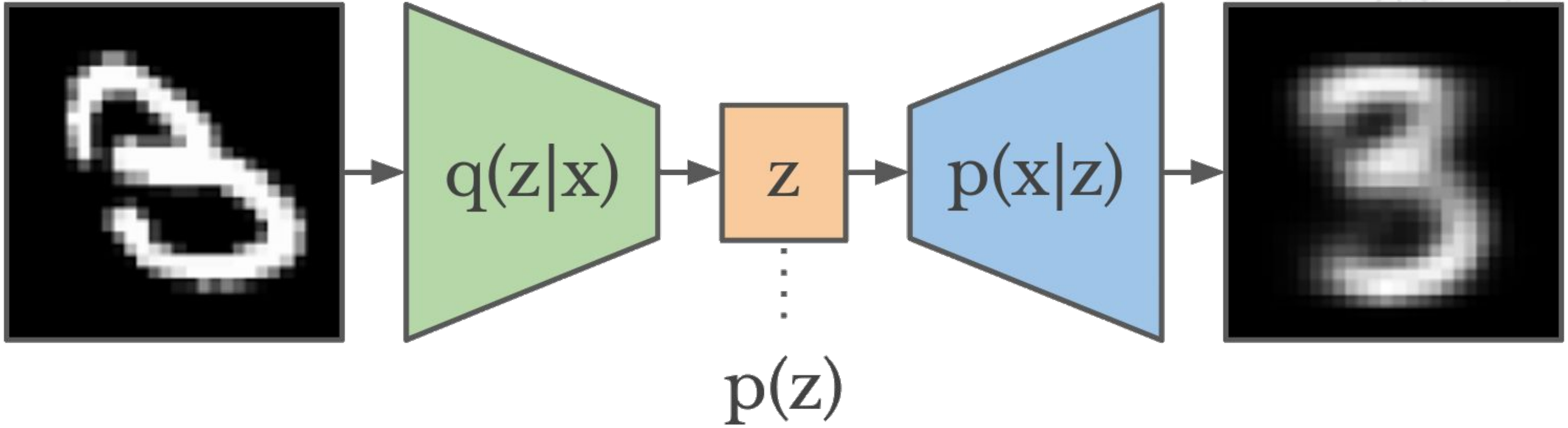
Benefits

- Memory efficiency
- Adaptive computation
- Parameter efficiency
- Scalable and invertible normalizing flows
- Continuous time-series models

Recurrent neural networks

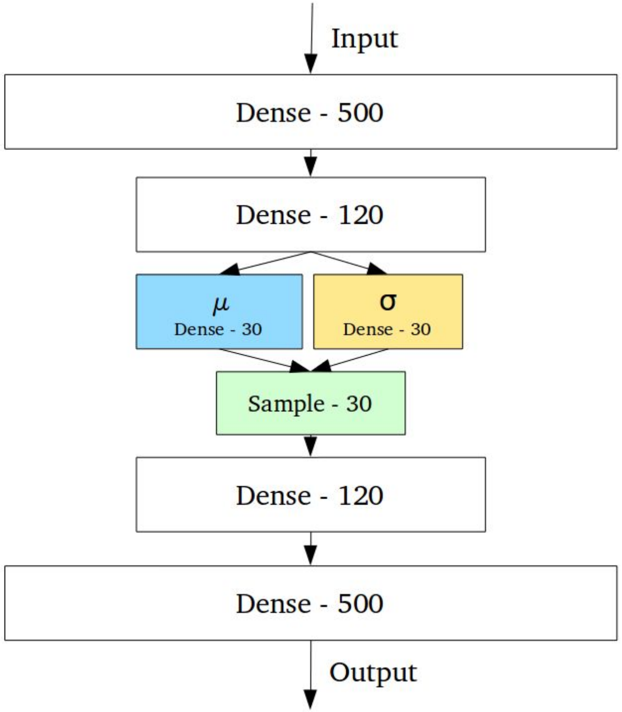


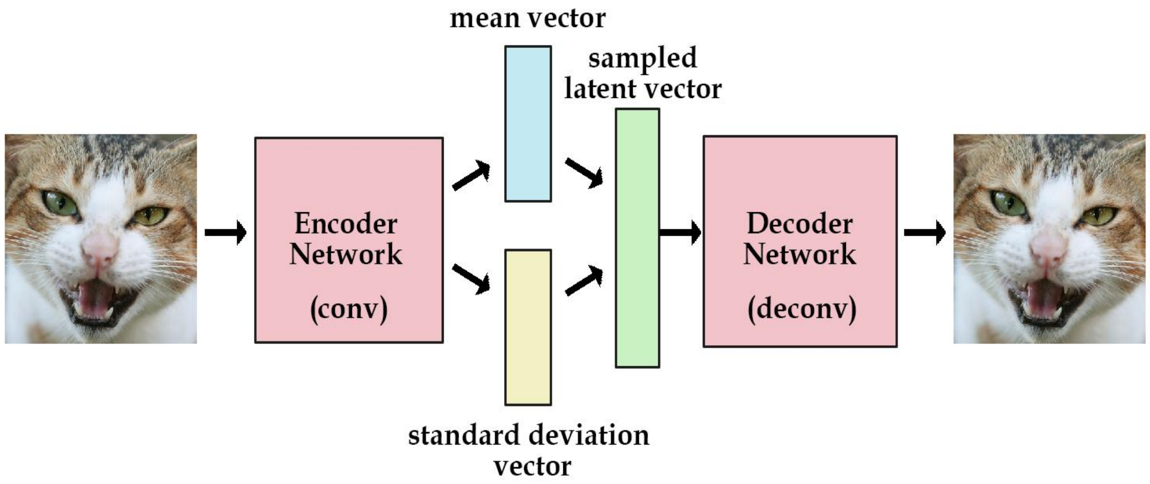
Variational Autoencoders



1. X: data that we want to model a.k.a the animal
2. z: latent variable a.k.a our imagination
3. $P(X)$: probability distribution of the data, i.e. that animal kingdom
4. $P(z)$: probability distribution of latent variable, i.e. our brain, the source of our imagination
5. $P(X|z)$: distribution of generating data given latent variable, e.g. turning imagination into real animal

Variational Autoencoders

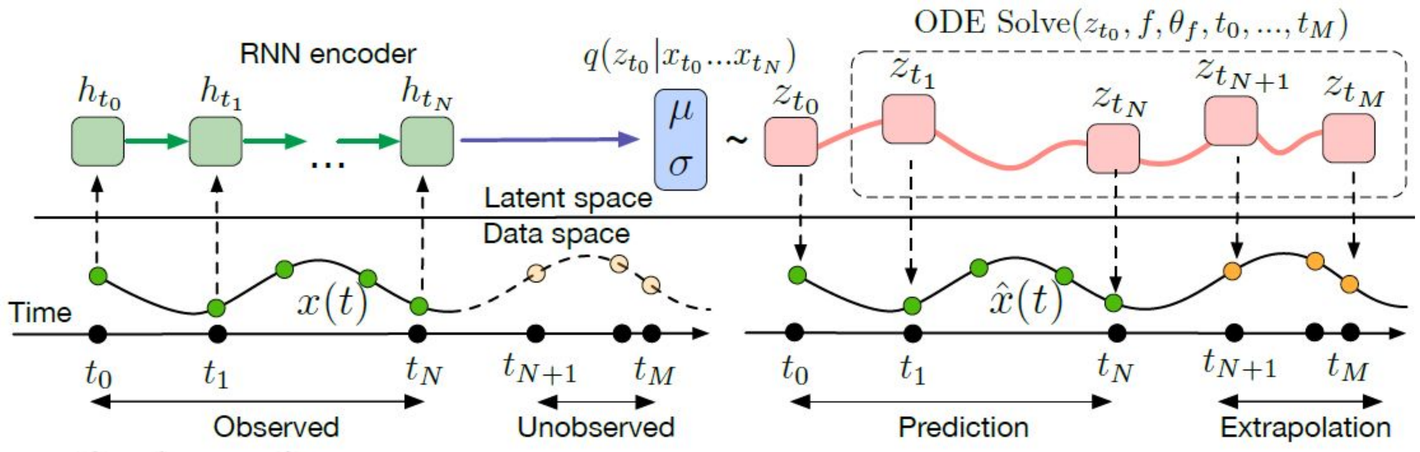




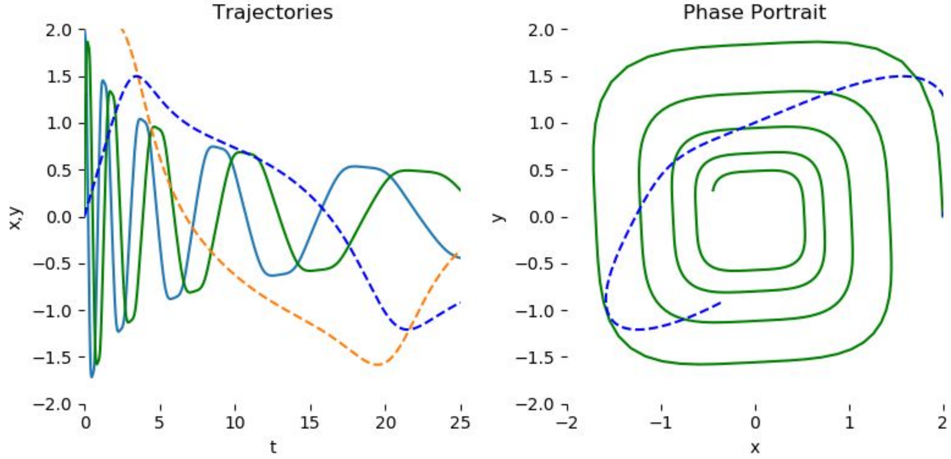
A generative latent function time-series model

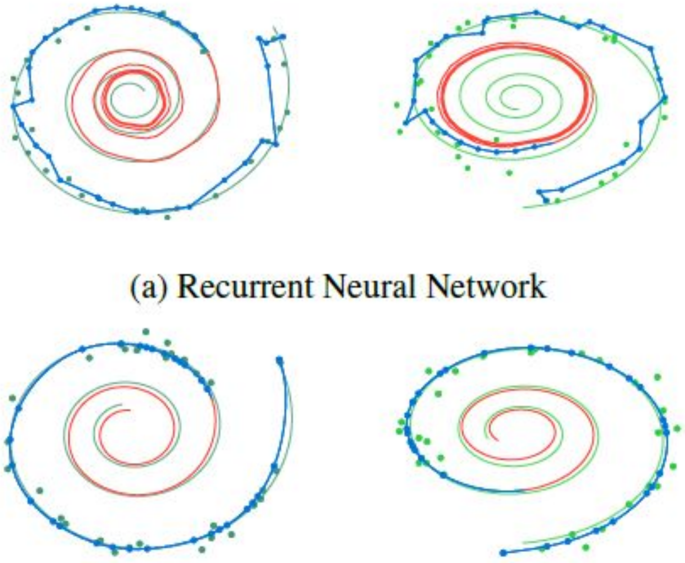
The model

$$\mathbf{z}_{t_0} \sim p(\mathbf{z}_{t_0})$$
$$\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} = \mathrm{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N)$$
$$\text{each} \quad \mathbf{x}_{t_i} \sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}})$$



Some experiments

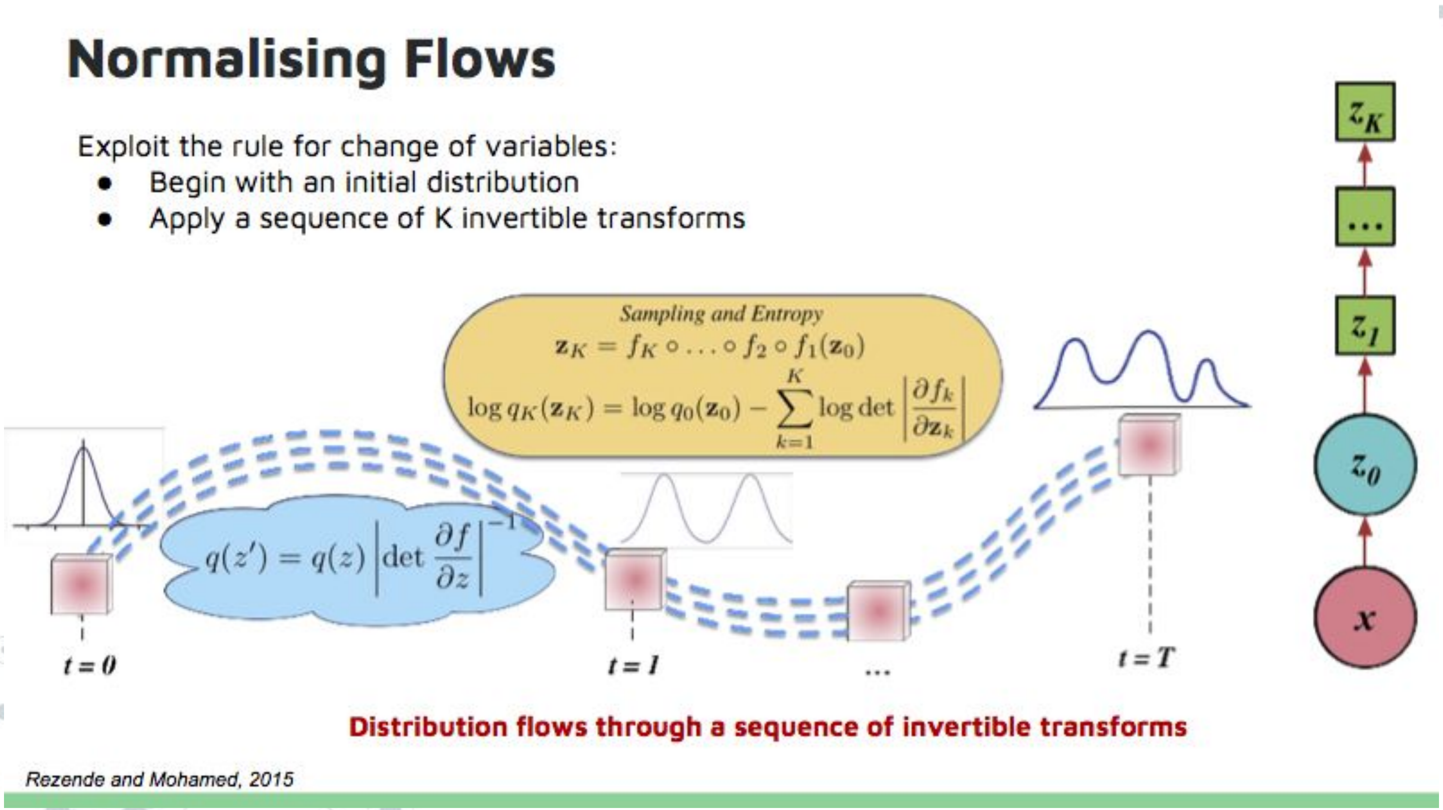




(a) Recurrent Neural Network

(b) Latent Neural Ordinary Differential Equation

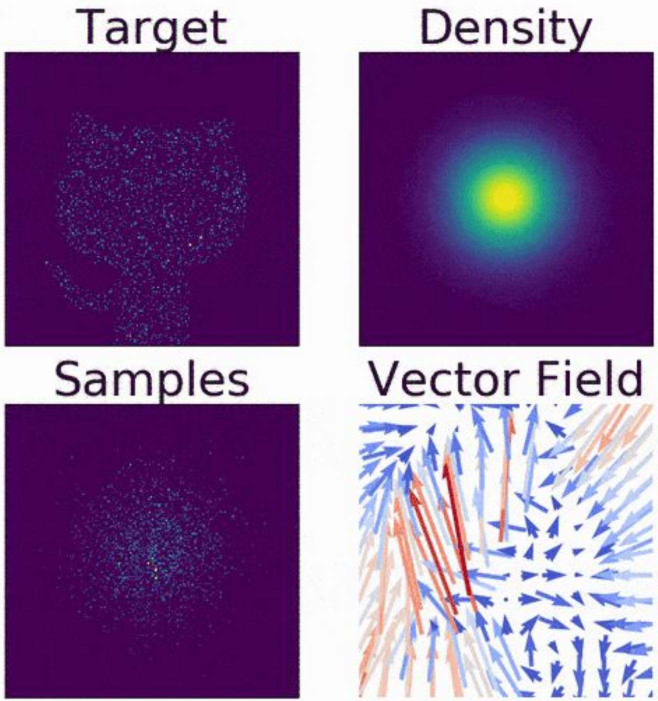
Normalizing flows



Continuous Normalizing Flows

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathrm{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

$$\ln q_K(z_K) = \ln q_0(z_0) - \sum_{k=1}^{K} \ln \left| \frac{\partial f_k}{\partial z_{k-1}} \right|$$



Continuous Normalizing Flows

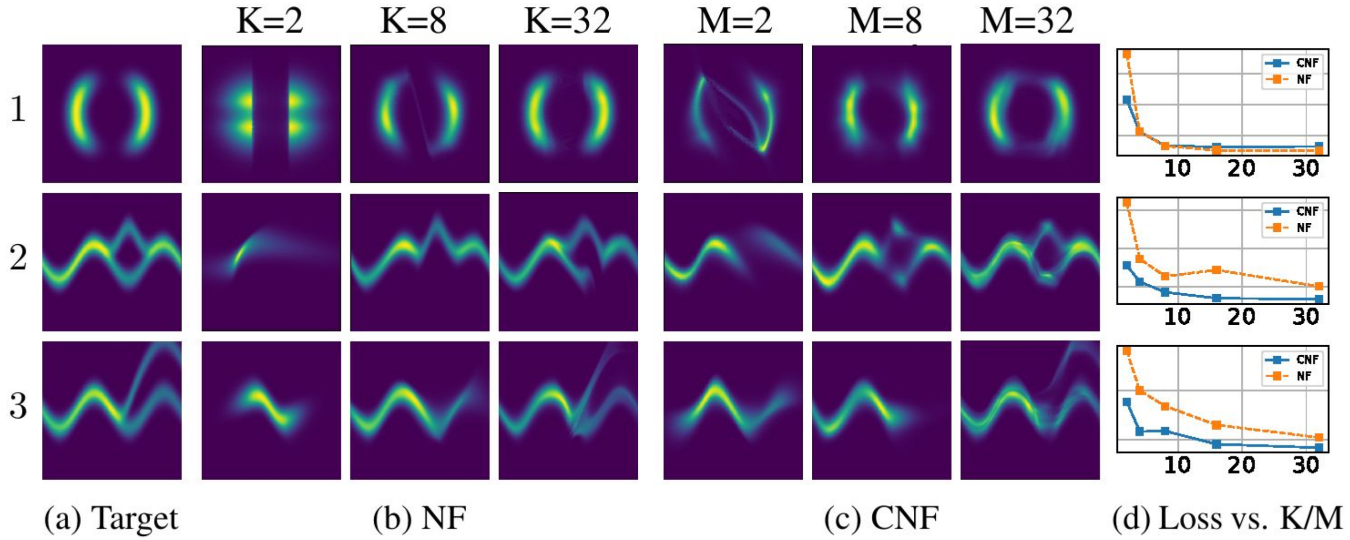


Figure 4: Comparison of normalizing flows versus continuous normalizing flows. The model capacity of normalizing flows is determined by their depth (K), while continuous normalizing flows can also increase capacity by increasing width (M), making them easier to train.

Scope and limitations

- **Uniqueness** When do continuous dynamics have a unique solution?
    - the solution to an initial value problem exists and is unique if the differential equation is uniformly Lipschitz continuous in z and continuous in t.
      → finite weights and Lipshitz nonlinearities, such as tanh or relu.
- **Reversibility** Even if the forward trajectory is invertible in principle, in practice there will be three compounding sources of error in the gradient:
    - Numerical error introduced in the forward ODE solver
    - Information lost due to multiple initial values mapping to the same final state
    - Numerical error introduced in the reverse ODE solver.

My research project - GAN Neural-ODE

Thank you !