Neural Ordinary Differential Equations

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

NIPS 2018 Best Paper

Present by Yurong You (yy785) at ORIE 7191

Apr 16, 2019

Quiz

- What is the memory cost of calculating gradients of an L-layer neural network?

  A. $O(1)$ B. $O(L)$ C. $O(log(L))$ D. $O(L\ log(L))$

- Which of the following is the most comparable quantity of Neural ODE (say, integrate from 0 to T) to the *depth* of a normal neural network?

  A. the number of evaluations of dynamics
  B. T
  C. the depth of dynamics function

Outline

- Neural ODE
- Training
- Applications
- Limitations

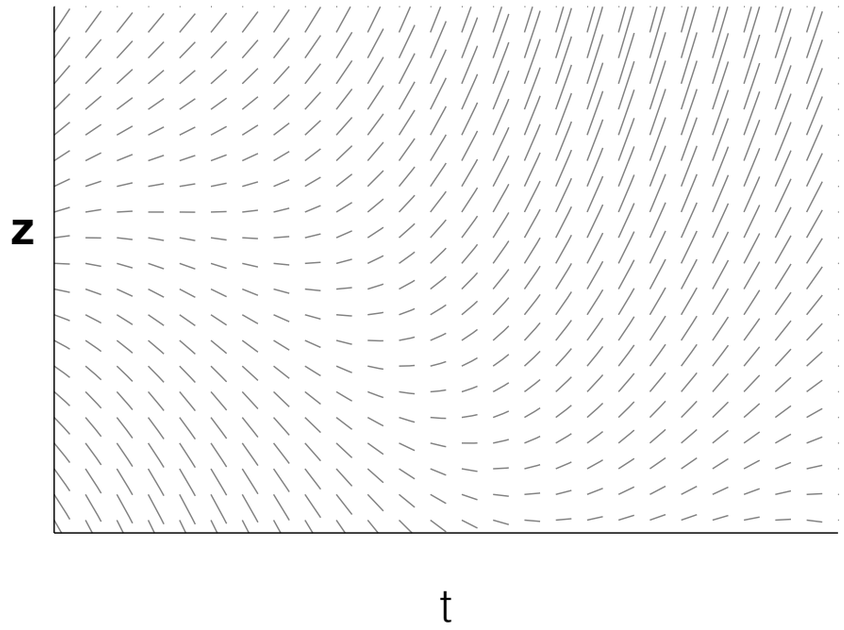
Outline

- Neural ODE
- Training
- Applications
- Limitations

ODE 101: quick review

- Ordinary Differential Equations

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



- Solving ODE with initial-value constraint

$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \int_{t_s}^{t_e} f(\mathbf{z}, t)\ \mathrm{d}t$$

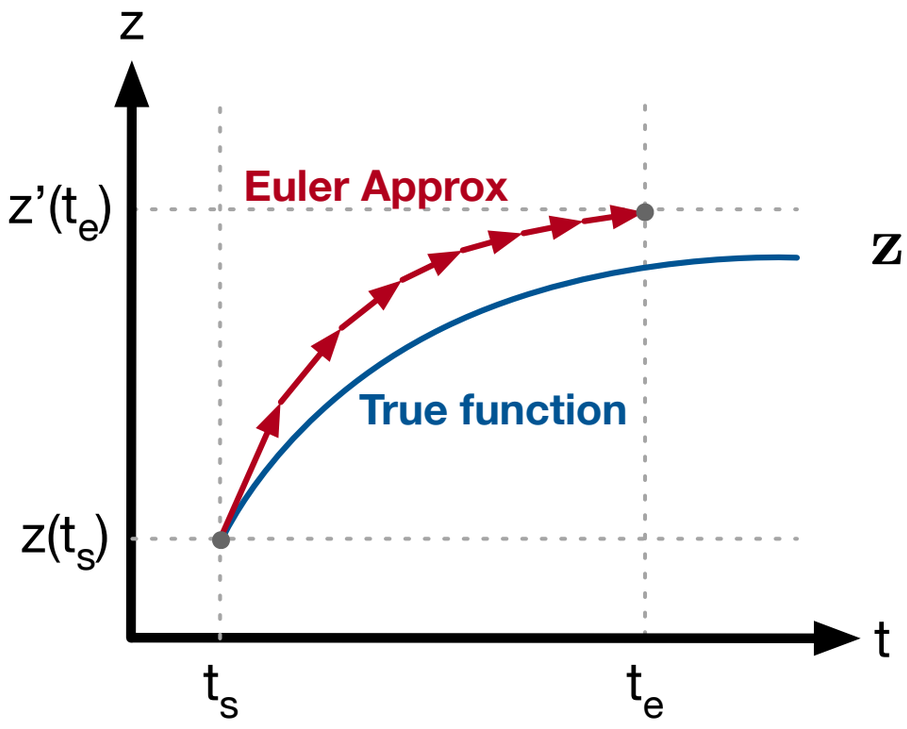
Most cases, no close-form for the integral!

ODE 101: quick review

- Approximately solving ODE with initial-value constraint

$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \int_{t_s}^{t_e} f(\mathbf{z}, t)\ \mathrm{d}t$$

- Euler method (~1800)



$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + f(\mathbf{z}(t), t)\Delta t$$

$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \sum_i f(\underbrace{\mathbf{z}(t_s + i\Delta t)}_{\mathbf{z}_i}, \underbrace{t_s + i\Delta t}_{t_i})\Delta t$$

ODE 101: quick review

- (Approximately) Solving ODE with initial-value constraint

$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \int_{t_s}^{t_e} f(\mathbf{z}, t)\ \mathrm{d}t$$

  - Euler method (~1800)

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + f(\mathbf{z}(t), t)\Delta t$$

  - Runge-Kutta Method (1900)

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + \frac{1}{6}\Delta t(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

  - .....

$f, \mathbf{z}(t_s)$ → `ODESolve(f, z, t0, t1)` → $\mathbf{z}(t_e)$

ResNet

- Basic ODE Solver (Euler method)

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + f(\mathbf{z}(t), t, \theta)\Delta t$$

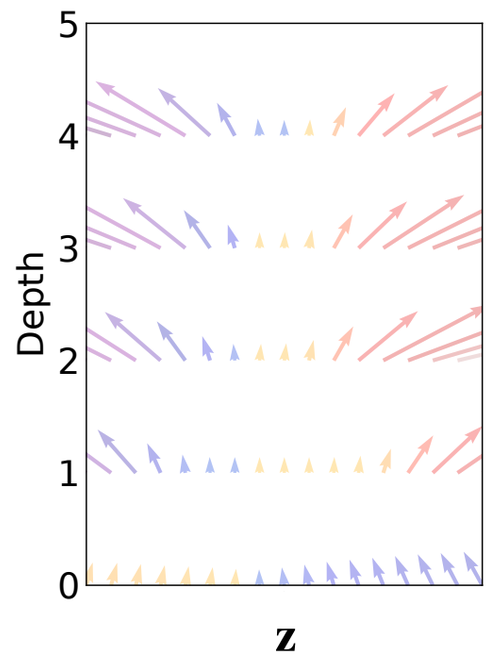
$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \sum_i f(\mathbf{z}_i, t_i, \theta)\Delta t$$

- ResNet

```
z_{l+1} = z_l + nnet(z_l, θ[l])
z       = z_0 + sum([nnet(z_l, θ[l]), l=1:T])
```

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

def resnet(z):
    for l in range(T):
        z = z + f(z, l, θ)
    return z
```



ResNet = an Euler Integrator

- Basic ODE Solver (Euler method)

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + f(\mathbf{z}(t), t)\Delta t$$

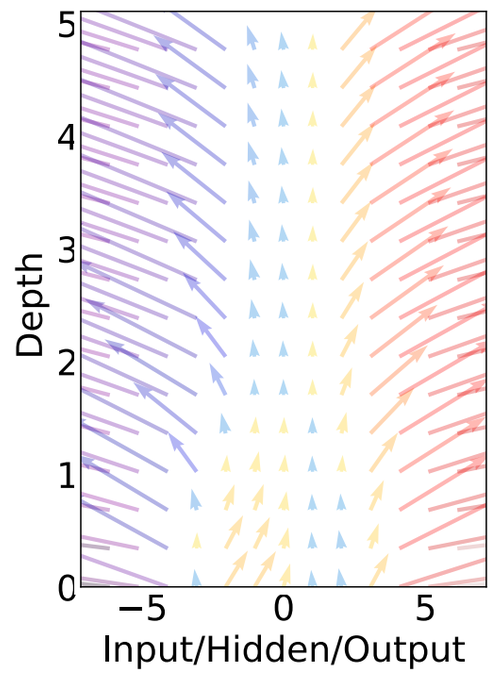
$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \sum_i f(\mathbf{z}_i, t_i)\Delta t$$

- ResNet

```
z_{l+1} = z_l + nnet([z, l], θ)
z       = z_0 + sum([nnet([z, l], θ), l=1:T])
```

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

def resnet(z):
    for l in range(T):
        z = z + f(z, l, θ)
    return z
```



Neural ODE

- Basic ODE Solver (Euler method)

$$\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + f(\mathbf{z}(t), t)\Delta t$$

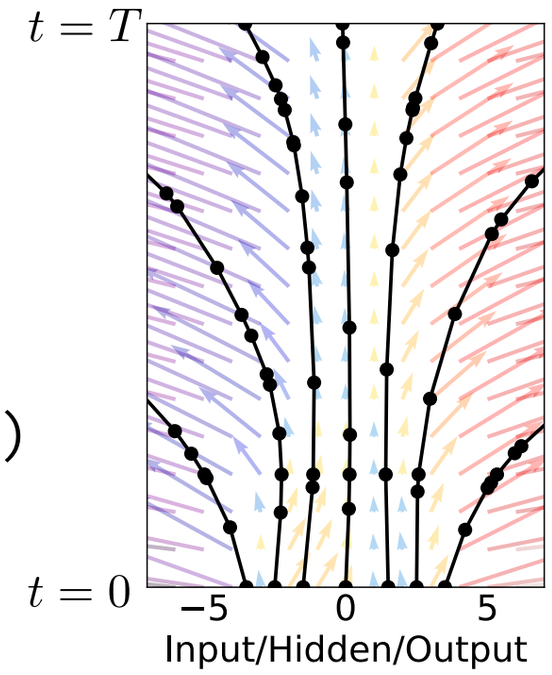
$$\mathbf{z}(t_e) = \mathbf{z}(t_s) + \sum_i f(\mathbf{z}_i, t_i)\Delta t$$

- ODENet

```
z(t+dt) = z(t) + nnet([z(t), t], θ)
z       = ODESolve(f, z, 0, T, θ)
```

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

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



Outline

- Neural ODE
- Training
- Applications
- Limitations

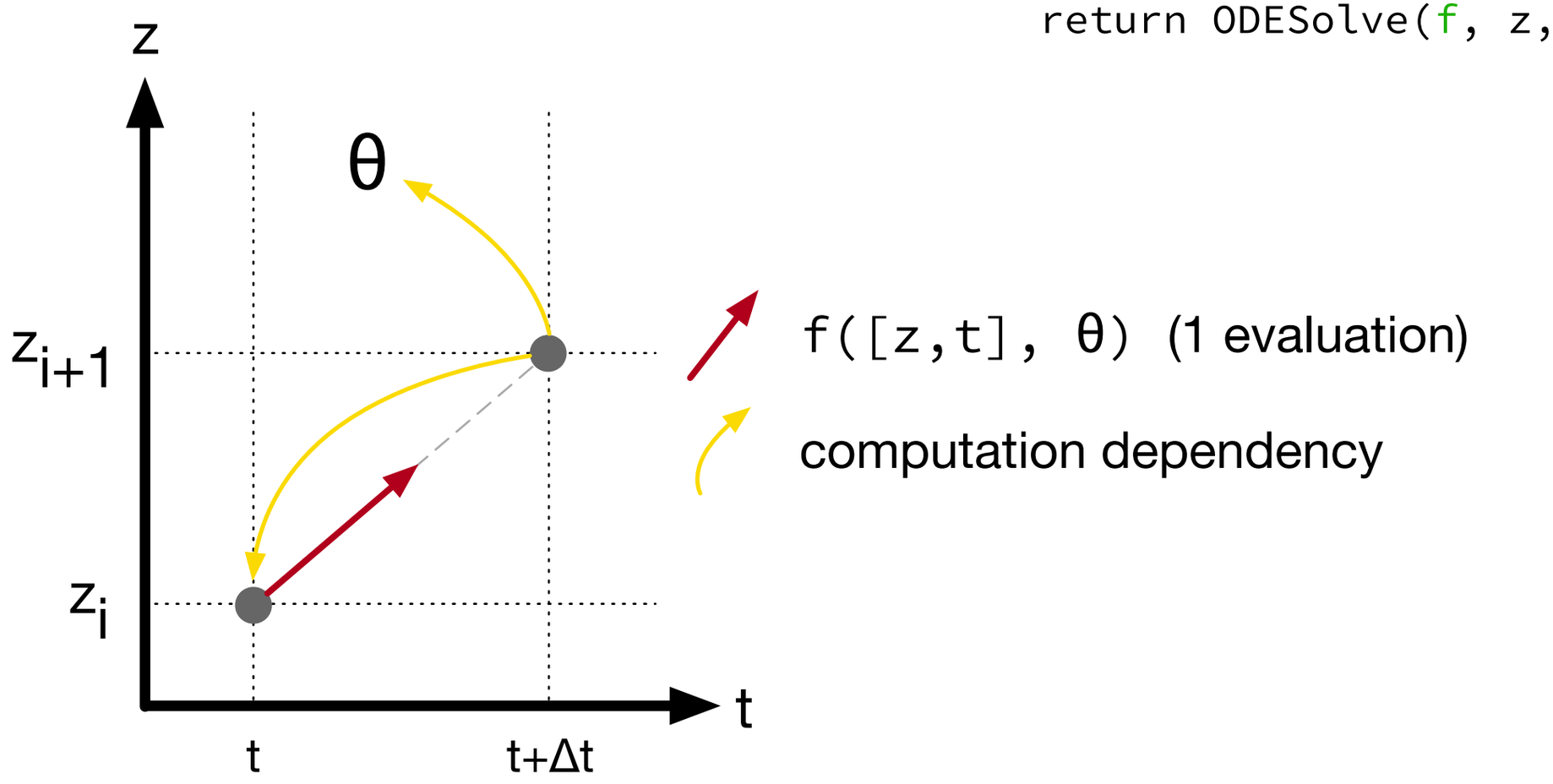
Training - is it possible?

- Method 1:

  Use white-box ODESolvers

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

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

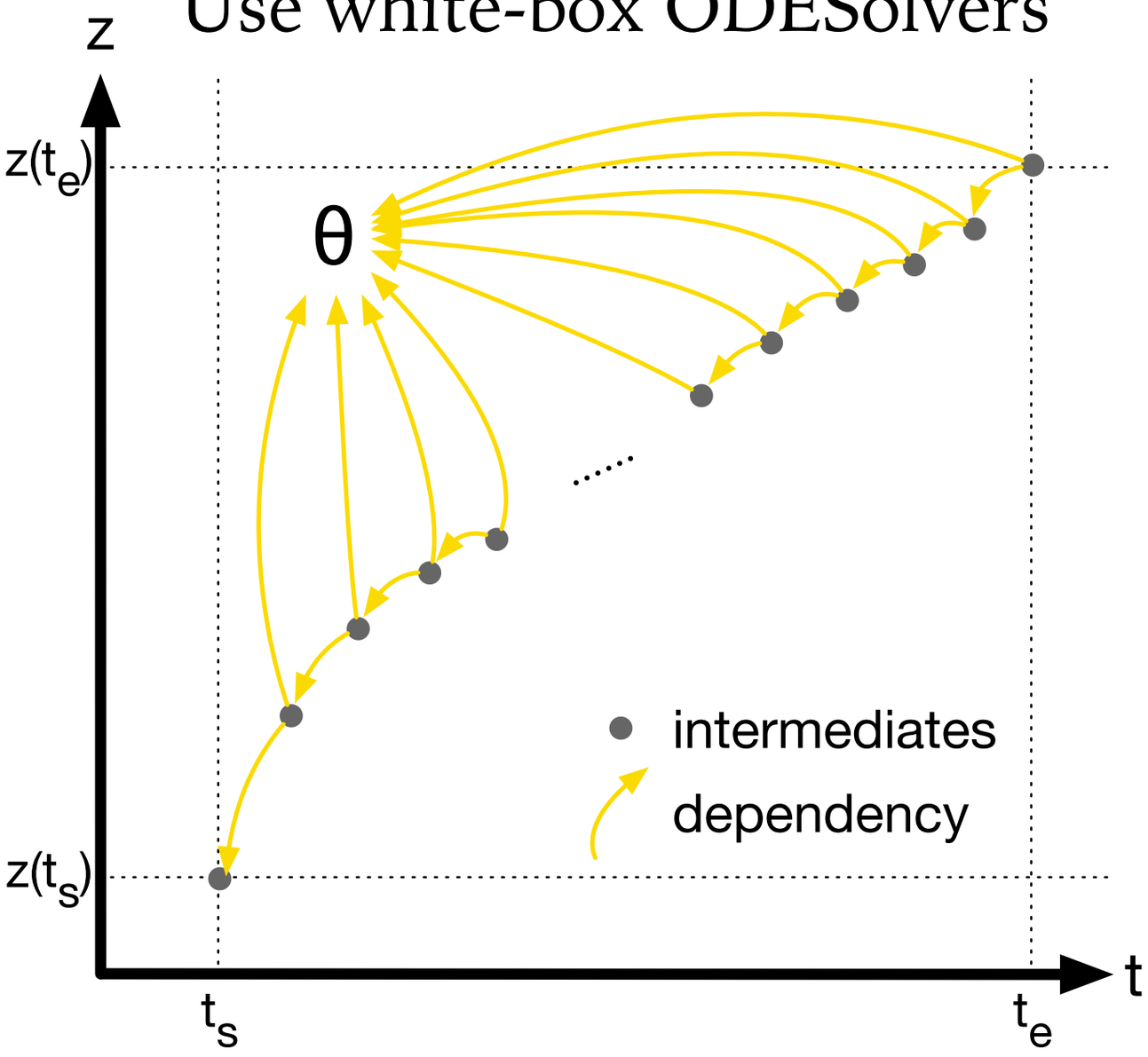


Euler Method: $\mathbf{z}(t + \Delta t) = \mathbf{z}(t) + f(\mathbf{z}(t), t, \theta)\Delta t$

Training - is it possible?

- Method 1:

  Use white-box ODESolvers



Euler Computation Graph

- Store all intermediate states

  $\#\text{states} = O(1 / \Delta t)$

- Backprop through them all

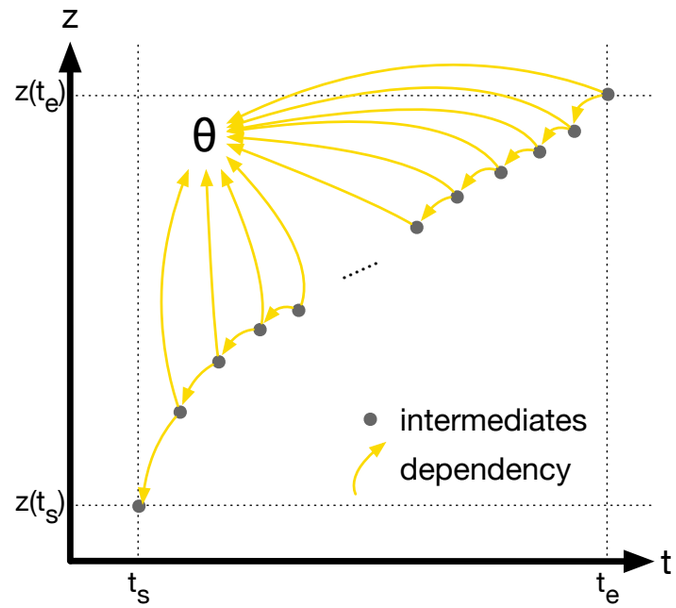
Drawbacks:

- Step size (Δt) must be large, o/w insufficient GPU memory
- Cannot use pre-existing ODE-solver codes without auto-grad.

Training - is it possible?

- Method 2: Backprop as another ODE

$$\mathbf{z}(t+\Delta t) = \mathbf{z}(t) + f_\theta(\mathbf{z}(t), t)\Delta t$$



$$L(\mathbf{z}(t_e)) = L\left(\mathbf{z}(t_s) + \sum_i f_\theta(\mathbf{z}_i, t_i)\Delta t\right)$$

$$\frac{\partial L}{\partial \theta} = \sum_i \frac{\partial L}{\partial \mathbf{z}_i}\frac{\partial \mathbf{z}_i}{\partial \theta} = \sum_i \frac{\partial L}{\partial \mathbf{z}_i}\frac{\partial f_\theta}{\partial \theta}\Delta t$$

$$\frac{\partial L}{\partial \mathbf{z}_i} = \frac{\partial L}{\partial \mathbf{z}_{i+1}}\frac{\partial \mathbf{z}_{i+1}}{\partial \mathbf{z}_i}$$

$$= \frac{\partial L}{\partial \mathbf{z}_{i+1}}\left(1 + \frac{\partial f_\theta}{\partial \mathbf{z}_i}\Delta t\right)$$

$$\Rightarrow \frac{\frac{\partial L}{\partial \mathbf{z}_{i+1}} - \frac{\partial L}{\partial \mathbf{z}_i}}{\Delta t} = -\frac{\partial L}{\partial \mathbf{z}_{i+1}}\frac{\partial f_\theta}{\partial \mathbf{z}_i}$$

to continuous

$$L(\mathbf{z}(t_e)) = L\left(\mathbf{z}(t_s) + \int_{t_s}^{t_e} f_\theta(\mathbf{z}(t), t)\ \mathrm{d}t\right)$$

$$\frac{\partial L}{\partial \theta} = \int_{t_s}^{t_e} \frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \theta}\ \mathrm{d}t$$

$$\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \theta}$$

$$\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \mathbf{z}(t)} = -\frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \mathbf{z}(t)}$$

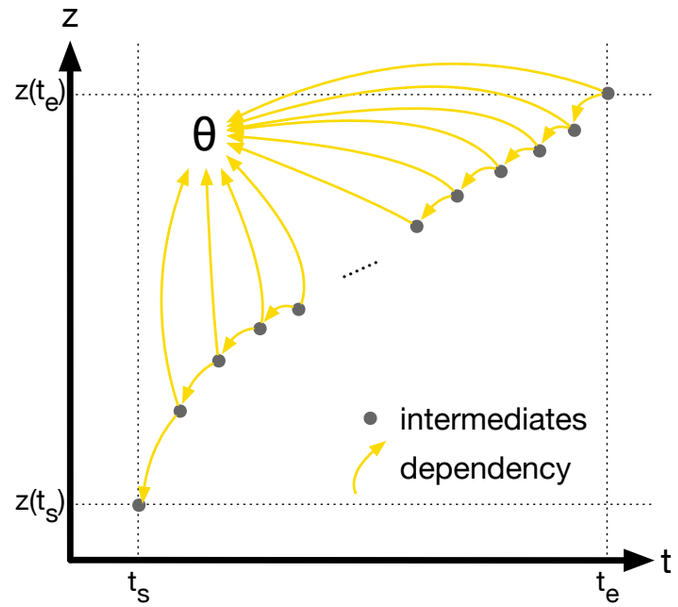
Training - is it possible?

- Method 2: Backprop as another ODE

$$L(\mathbf{z}(t_e)) = L\left(\mathbf{z}(t_s) + \int_{t_s}^{t_e} f_\theta(\mathbf{z}(t), t)\ \mathrm{d}t\right)$$

$$\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \theta}$$

$$\Rightarrow \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \mathbf{z}(t)} = -\frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \mathbf{z}(t)}$$



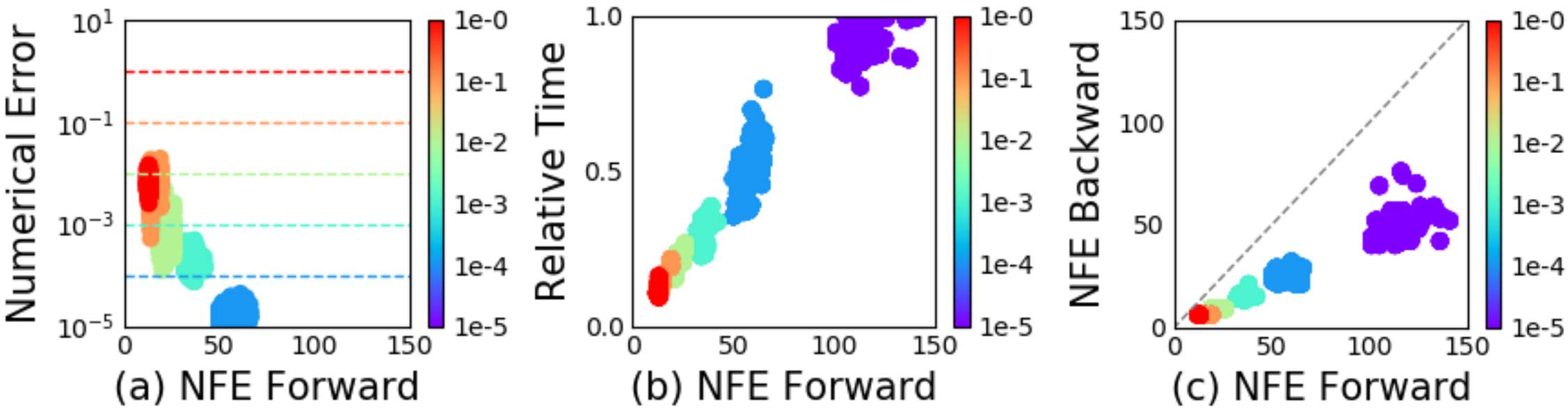
Advantage:

- Backprop GPU memory *O(1)*
- Can use pre-existing ODE-solver codes without auto-grad, with advanced error control

$$\left[\frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial \mathbf{z}(t_s)}\right] = \left[\mathbf{0}, \frac{\partial L}{\partial \mathbf{z}(t_e)}\right] + \int_{t_e}^{t_s} \left[\frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \theta}, -\frac{\partial L}{\partial \mathbf{z}(t)}\frac{\partial f_\theta}{\partial \mathbf{z}(t)}\right]\ \mathrm{d}t$$

One call to `ODESolve` !

Error control in Neural ODE



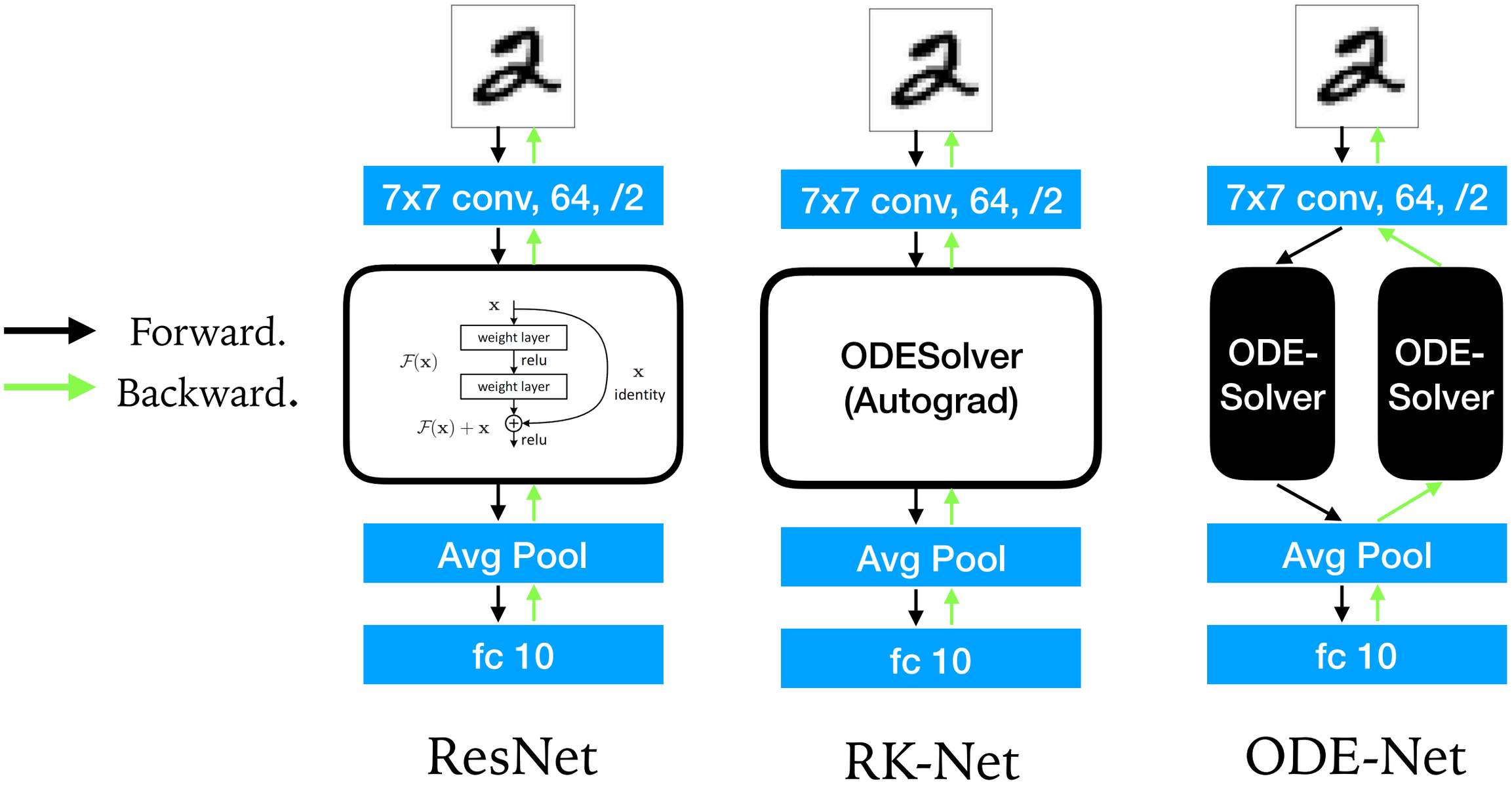
NFE = number of function evaluations

Outline

- Neural ODE
- Training
- Applications
  - Supervised Learning
  - Continuous Normalizing Flow
  - Generative Latent Time-series Model
- Limitations

Application - Supervised Learning

- Set-up : MNIST classification



Application - Supervised Learning

- Set-up : MNIST classification

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

Applications - Continuous Normalizing Flows

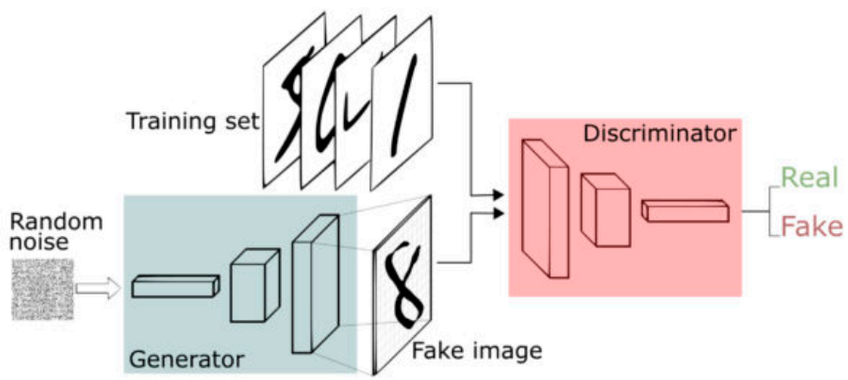
- Set-up:
  - Goal: want to learn a generative model for the data distribution :

$$Q(X)$$

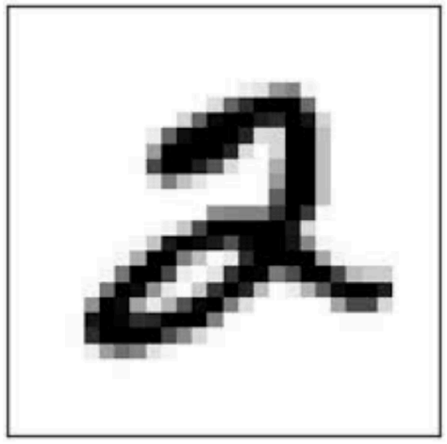
  - Given a dataset sampled from the ground truth distribution:

$$\mathcal{X}, \quad \forall X \in \mathcal{X}, X \sim Q(X)$$

- Output a model that could sample from the distribution.
  - e.g. GAN



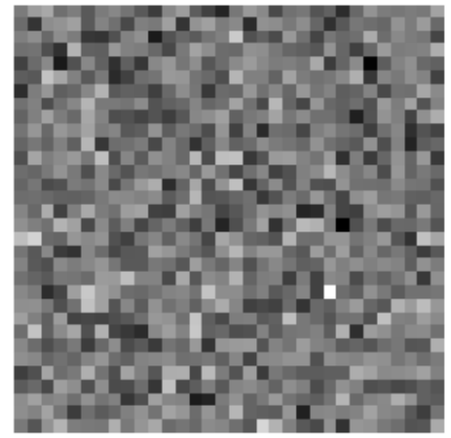
Normalizing Flow



$X \sim Q(X)$

$$X = F_\theta(Z)$$

$$Z = F_\theta^{-1}(X)$$

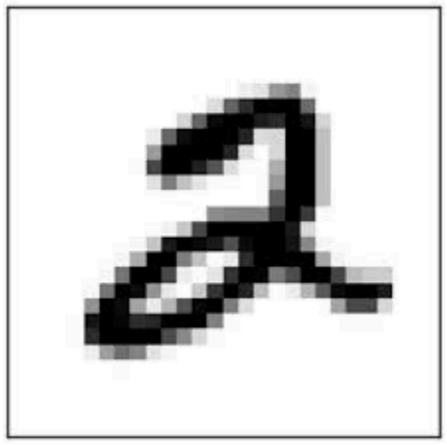


$Z \sim \mathcal{N}(0,I)$

$$\log p(X;\theta) = \log p(Z) - \log \left| \det \frac{\partial F_\theta}{\partial Z} \right|$$

$$\log p(X;\theta) = \log p(F_\theta^{-1}(X)) - \log \left| \det \frac{\partial F_\theta}{\partial (F_\theta^{-1}(X))} \right|$$

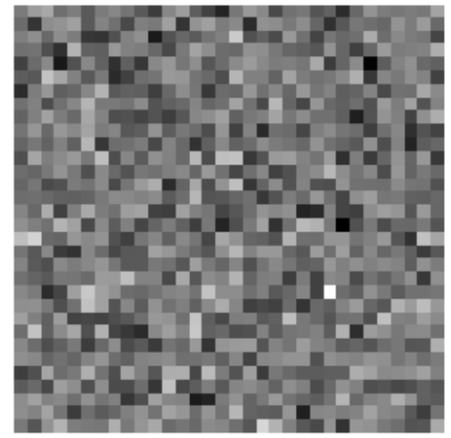
Normalizing Flow



$X \sim Q(X)$

$$X = F_\theta(Z)$$

$$Z = F_\theta^{-1}(X)$$



$Z \sim \mathcal{N}(0,I)$

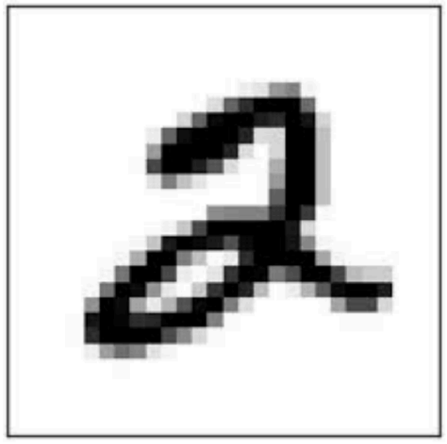
$$\log p(X;\theta) = \log p(F_\theta^{-1}(X)) - \log \left| \det \frac{\partial F_\theta}{\partial (F_\theta^{-1}(X))} \right|$$

$$\text{Training:} \quad \theta^* = \text{argmax}_\theta \sum_{X \in \mathcal{X}} \log p(X;\theta)$$

$$\text{Sample:} \qquad \hat{X} = F(\hat{Z};\theta^*), \quad \hat{Z} \sim \mathcal{N}(0,I)$$

$$\text{Likelihood estimation:} \qquad p(X_{test};\theta^*)$$

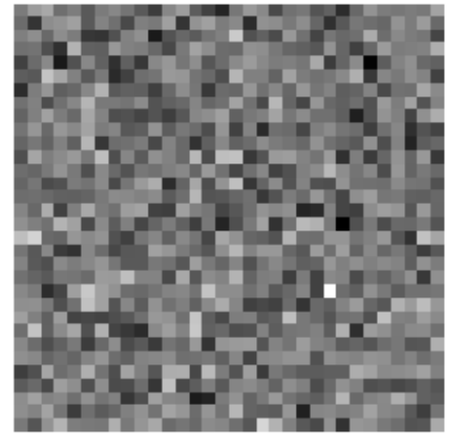
Normalizing Flow



$$X \sim Q(X)$$

$$X = F_\theta(Z) = f_{n,\theta} \circ \ldots \circ f_{1,\theta}(Z)$$

$$Z = F_\theta^{-1}(X) = f_{1,\theta}^{-1} \circ \ldots \circ f_{n,\theta}^{-1}(X)$$



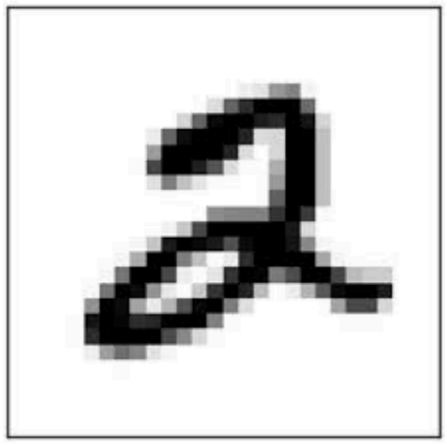
$$Z \sim \mathcal{N}(0, I)$$

$$\log p(X;\theta) = \log p(F_\theta^{-1}(X)) - \log \left| \det \frac{\partial F_\theta}{\partial (F_\theta^{-1}(X))} \right|$$

$$Z_k = f_{k,\theta}^{-1} \circ \ldots \circ f_{1,\theta}^{-1}(Z), \quad \frac{\partial F_\theta(Z)}{\partial Z} = \frac{\partial f_{n,\theta}(Z_n)}{\partial Z_n} \ldots \frac{\partial f_{1,\theta}(Z_1)}{\partial Z_1}$$

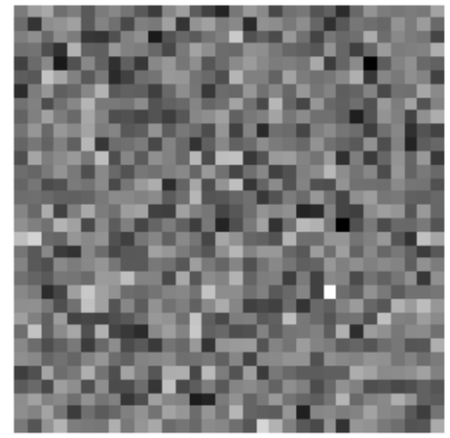
$$\log p(X) = \log p(f_1^{-1} \circ \ldots \circ f_n^{-1}(X)) - \sum_{i=1}^{n} \log \left| \det \frac{\partial f_{i,\theta}(Z_i)}{\partial Z_i} \right|$$

Continuous Normalizing Flow



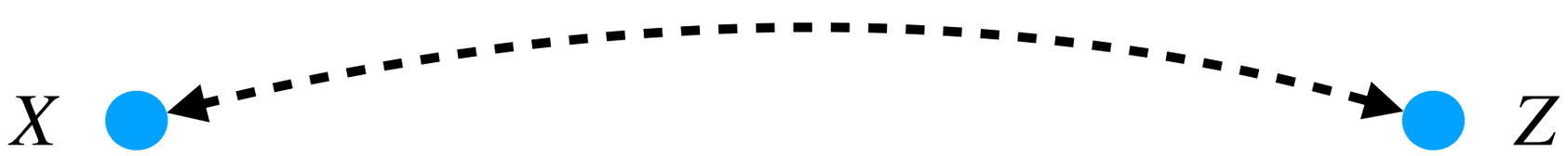
$$X = F_\theta(Z) = Z + \int_{t_0}^{t_1} f_\theta(Z(t), t)dt, \quad Z(t_0) = Z$$

$$Z = F_\theta^{-1}(X) = X + \int_{t_1}^{t_0} f_\theta(Z(t), t)dt, \quad Z(t_1) = X$$



$X \sim Q(X)$

$Z \sim \mathcal{N}(0, I)$



$$\log p(X;\theta) = \log p(F_\theta^{-1}(X)) - \log \left| \det \frac{\partial F_\theta}{\partial (F_\theta^{-1}(X))} \right|$$

How to compute this term efficiently?

Continuous Normalizing Flow

Theorem 1 (Instantaneous Change of Variables)**.** *Let $\mathbf{z}(t)$ be a finite continuous random variable with probability $p(\mathbf{z}(t))$ dependent on time. Let $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$ be a differential equation describing a continuous-in-time transformation of $\mathbf{z}(t)$. Assuming that f is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, then the change in log probability also follows a differential equation,*

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\text{tr}\left(\frac{df}{d\mathbf{z}(t)}\right) \tag{8}$$

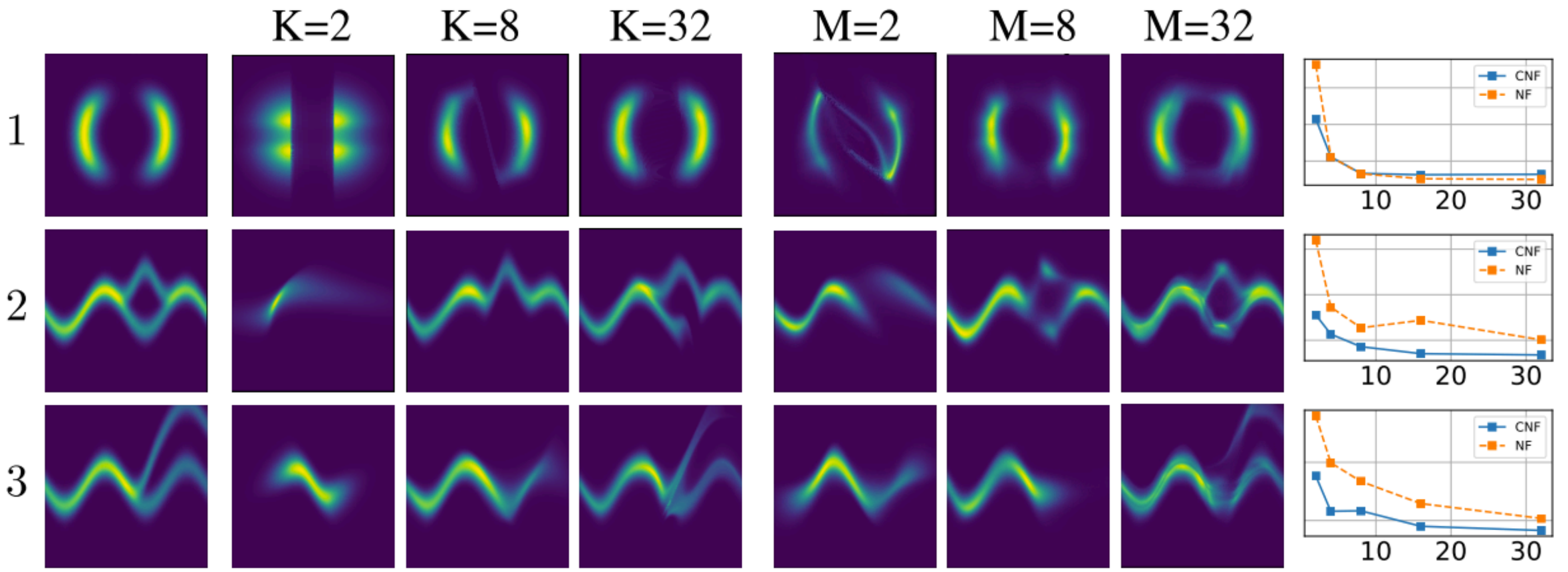
$$\log p(Z(t_1)) - \log p(Z(t_0)) = \int_{t_0}^{t_1} \frac{d \log p(Z(t))}{dt} dt = \int_{t_0}^{t_1} -\text{tr}\left(\frac{\partial f_\theta}{\partial Z(t)}\right) dt$$

$$\log p(X) - \log p(Z) = -\int_{t_0}^{t_1} \text{tr}\left(\frac{\partial f_\theta}{\partial Z(t)}\right) dt, \quad X = Z(t_1), Z = Z(t_0)$$

$$\log p(X) = \log p(Z) - \int_{t_0}^{t_1} \text{tr}\left(\frac{\partial f_\theta}{\partial Z(t)}\right) dt$$

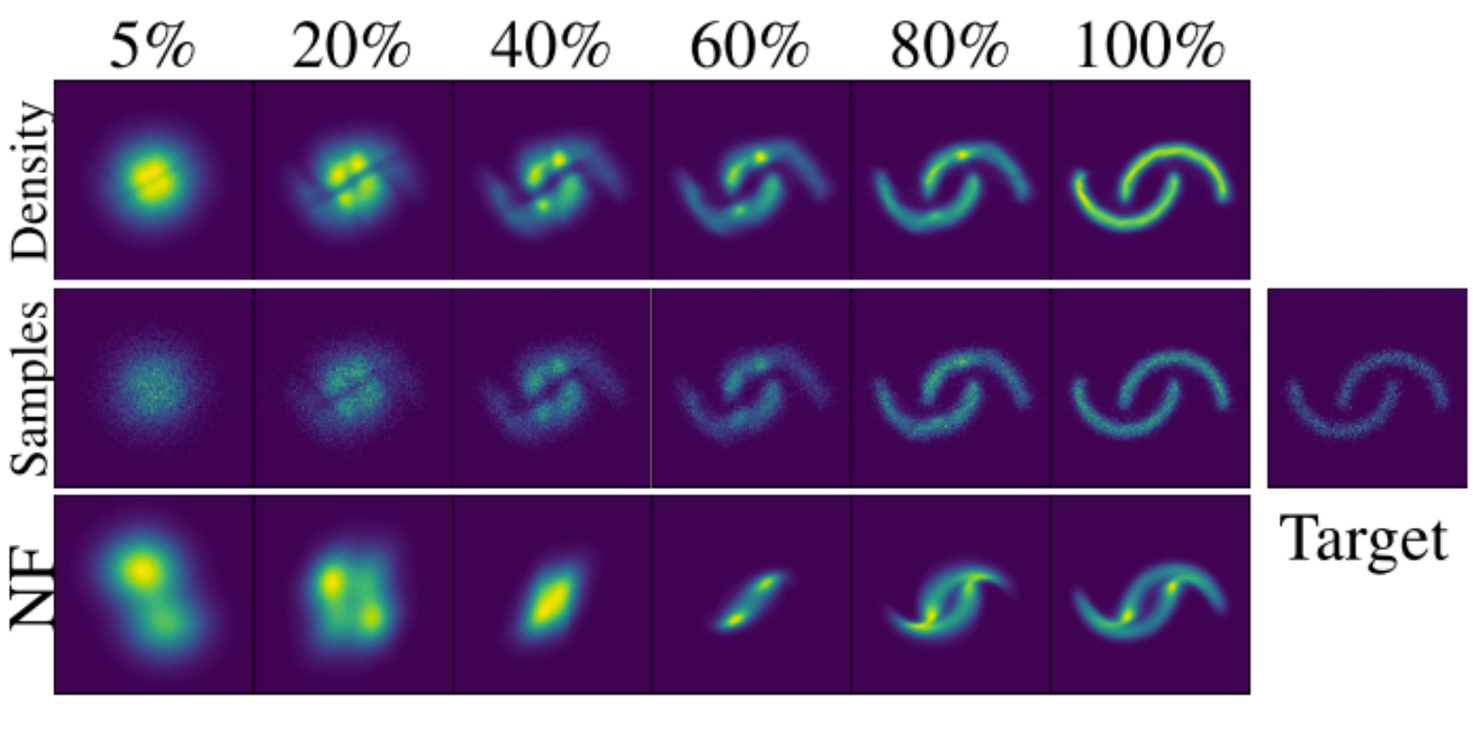
$$\log p(X) = \log p\left(X + \int_{t_1}^{t_0} f_\theta(Z(t), t) dt\right) - \int_{t_0}^{t_1} \text{tr}\left(\frac{\partial f_\theta(Z(t))}{\partial Z(t)}\right) dt$$

Experiment - Continuous Normalizing Flows

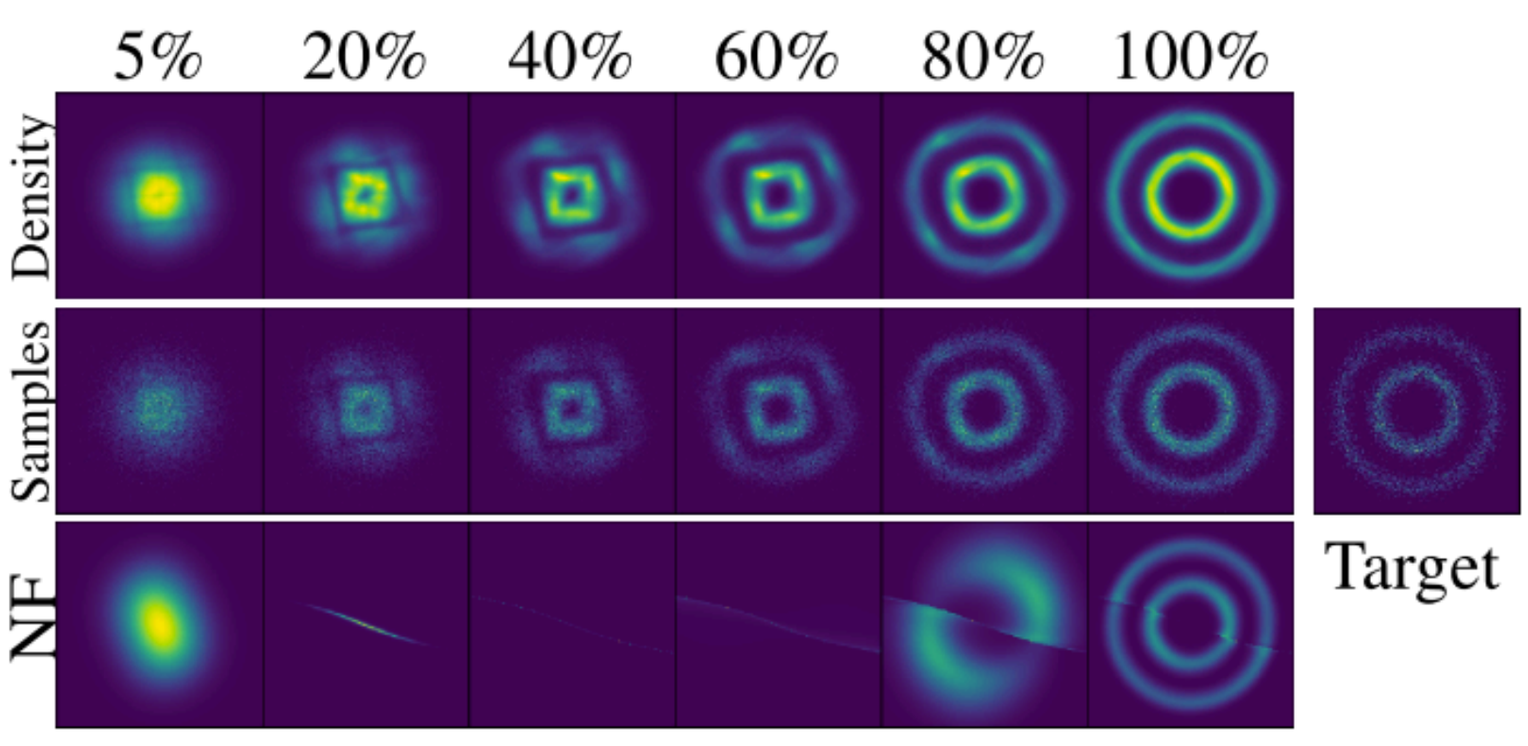


(a) Target (b) NF (c) CNF (d) Loss vs. K/M

Experiment - Continuous Normalizing Flows

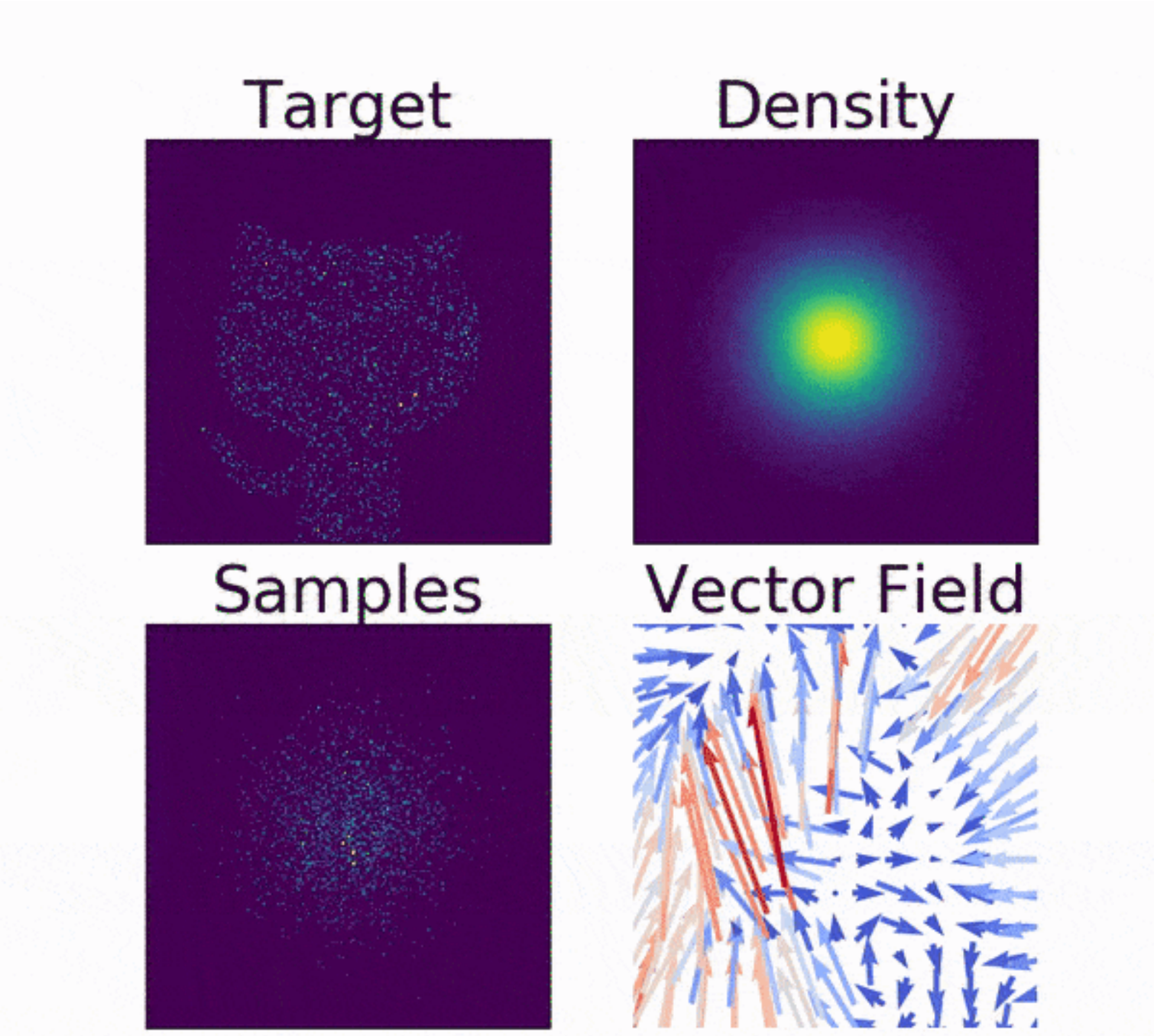


(b) Two Moons

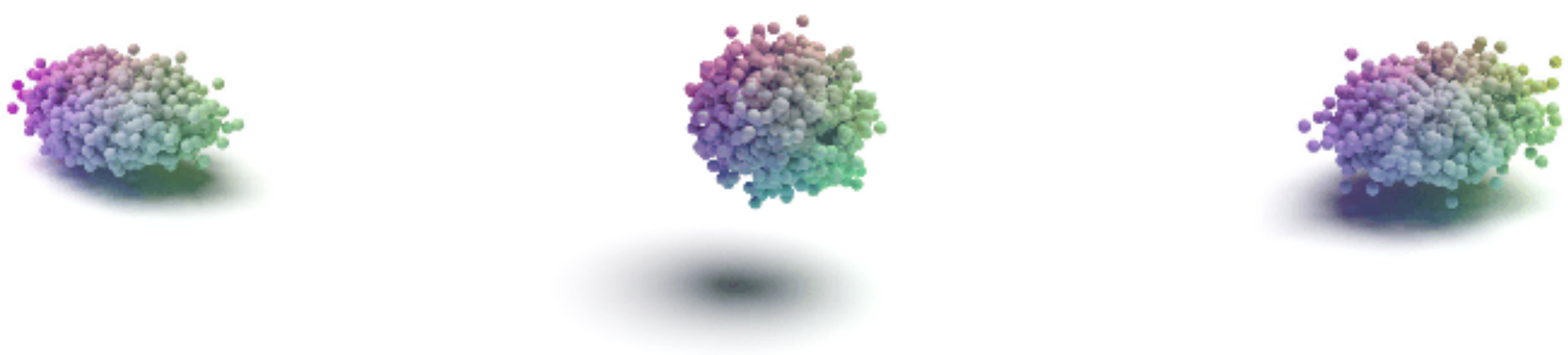


(a) Two Circles

Experiment - Continuous Normalizing Flows



Experiment - Continuous Normalizing Flows



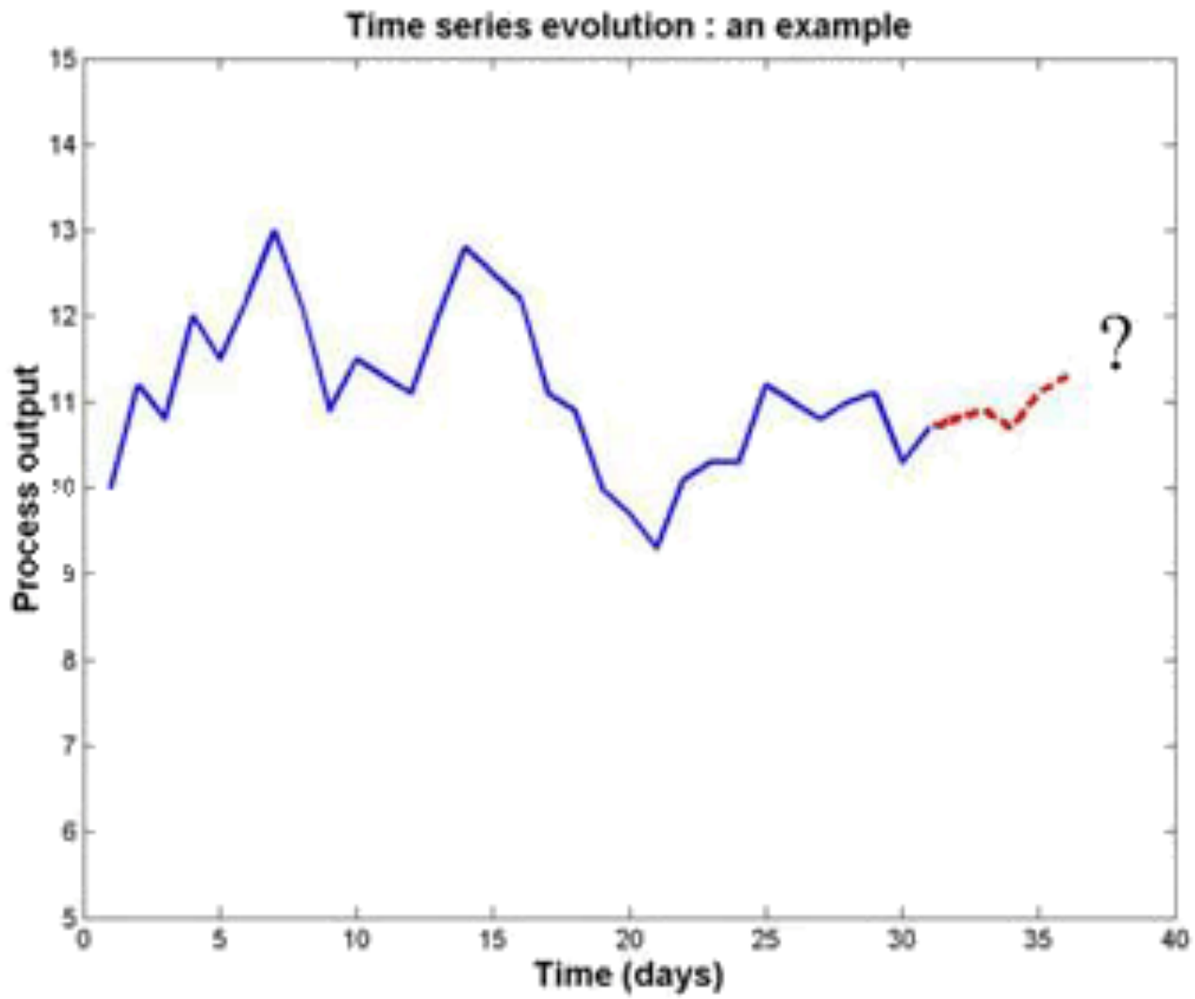
Applications - Generative Latent Time-series Model

- Time-series extrapolation
  - Input: a series of observations :

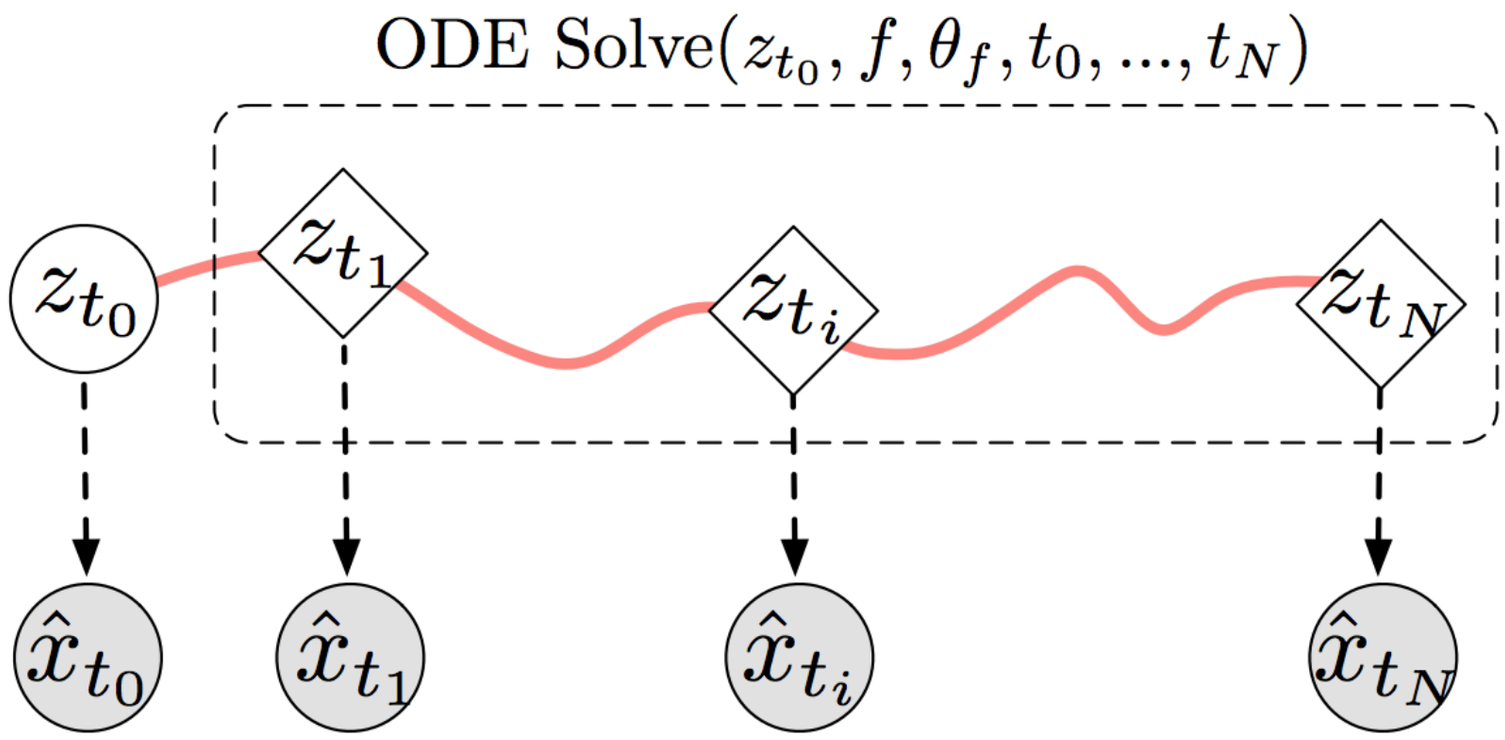
$$x_{t_1}, \ldots, x_{t_N}$$

  - Goal: predict observations in future time:

$$y_{t_{N+1}}, \ldots, y_{t_{N+M}}$$



Applications - Generative Latent Time-series Model

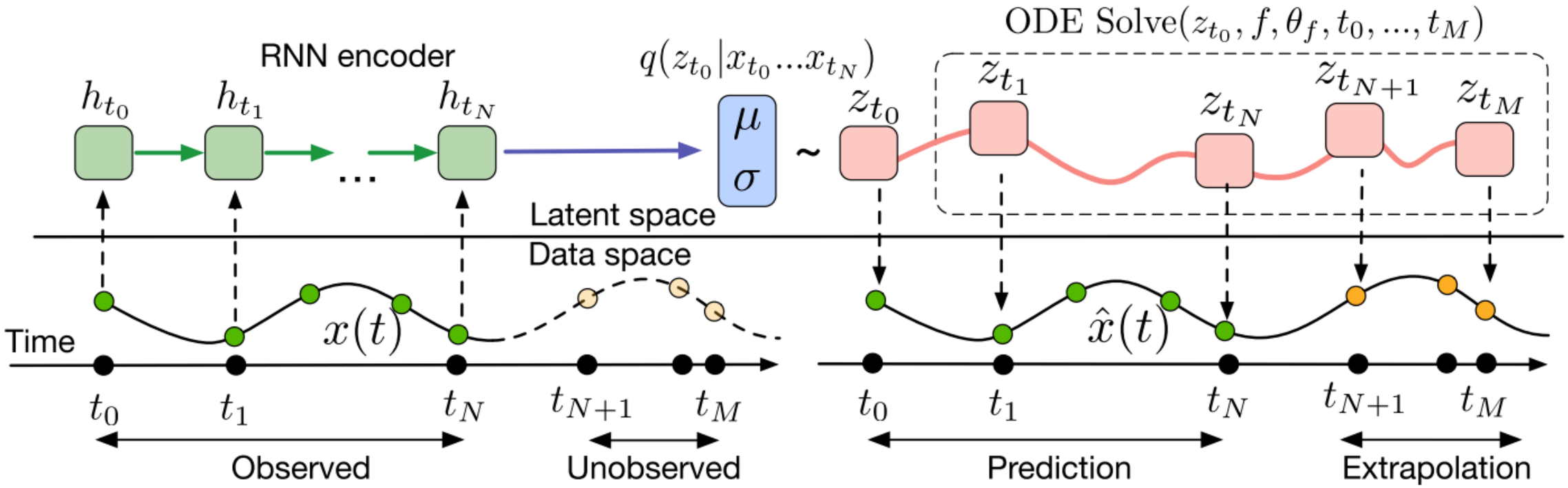


$$z_{t_0} \sim p(z_{t_0})$$

$$z_{t_i} = z_{t_0} + \int_{t_0}^{t_i} f(z(t); \theta_f)dt, \quad \forall 1 \leq i \leq N + M$$

$$x_{t_i} \sim p(x \mid z_{t_i}; \theta_x), \quad 1 \leq i \leq N + M$$

Applications - Generative Latent Time-series Model



$$z_{t_0} \sim p(z_{t_0}), \quad \text{approx. by } q(z_{t_0} \mid \{x_{t_i}\}_{1 \le i \le N})$$

$$z_{t_i} = z_{t_0} + \int_{t_0}^{t_i} f(z(t); \theta_f) dt, \quad \forall 1 \le i \le N + M$$

$$x_{t_i} \sim p(x \mid z_{t_i}; \theta_x), \quad 1 \le i \le N + M$$

Applications - Generative Latent Time-series Model

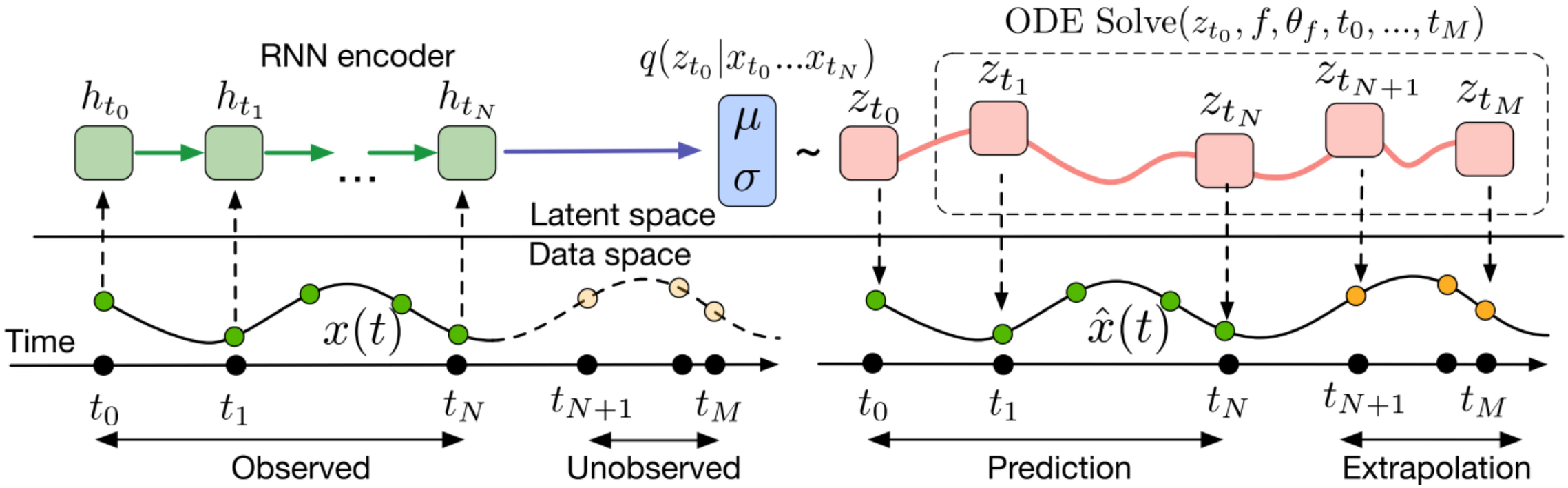


Table 2: Predictive RMSE on test set

# Observations	30/100	50/100	100/100
RNN	0.3937	0.3202	0.1813
Latent ODE	**0.1642**	**0.1502**	**0.1346**

Outline

- Neural ODE
- Training
- Applications
- Limitations
  - Estimate determinant of Jacobians
  - Reversibility
  - Scalability

Computing Jacobian

- It's usually hard to estimate determinant of the Jacobean for CNF or NF

$$\log p(X) = \log p\left(X + \int_{t_1}^{t_0} f_\theta(Z(t), t)dt\right) - \int_{t_0}^{t_1} \mathrm{tr}\left(\frac{df_\theta(Z(t))}{dZ(t)}\right) dt$$

$O(D^2)$

- Network architectures of invertible models (NF, CNF) are designed to make the computation of Jacobian matrix easier.

- These restricted architectures usually limit the model capacity.

Computing Jacobian - FFJORD

- Use an unbiased estimator to compute the trace

$$\mathbb{E}[\epsilon] = 0, \quad Cov(\epsilon) = I \implies Tr(A) = \mathbb{E}_\epsilon[\epsilon^T A \epsilon]$$

$$\mathbb{E}_\epsilon[\epsilon^T A \epsilon] = \sum_i \mathbb{E}_\epsilon[\epsilon_i^2] A_{ii} + \sum_{i \neq j} \mathbb{E}_\epsilon[\epsilon_i]\mathbb{E}_\epsilon[\epsilon_j] A_{ij}$$

$$= \sum_i A_{ii} \quad (\mathbb{E}_\epsilon[\epsilon_i^2] = Cov(\epsilon_i) = 1, \mathbb{E}_\epsilon[\epsilon_i] = 0)$$

- Auto-grad framework can compute $\epsilon^T \frac{\partial f(x)}{\partial x}$ efficiently. (i.e., it takes approximately one evaluation of [f])

- Dot-product $\left(\epsilon^T \frac{\partial f(x)}{\partial x}\right) \epsilon$ is efficient (O(D) time)

Computing Jacobian - FFJORD

- Final formula:

$$\log p(X) = \log p\left(X + \int_{t_1}^{t_0} f_\theta(Z(t), t)dt\right) - \int_{t_0}^{t_1} \mathrm{tr}\left(\frac{df_\theta(Z(t))}{dZ(t)}\right) dt$$

$$= \log p\left(X + \int_{t_1}^{t_0} f_\theta(Z(t), t)dt\right) - \mathbb{E}_\epsilon\left[\epsilon\left(\int_{t_0}^{t_1} \frac{df_\theta(Z(t))}{dZ(t)}dt\right)\epsilon\right]$$

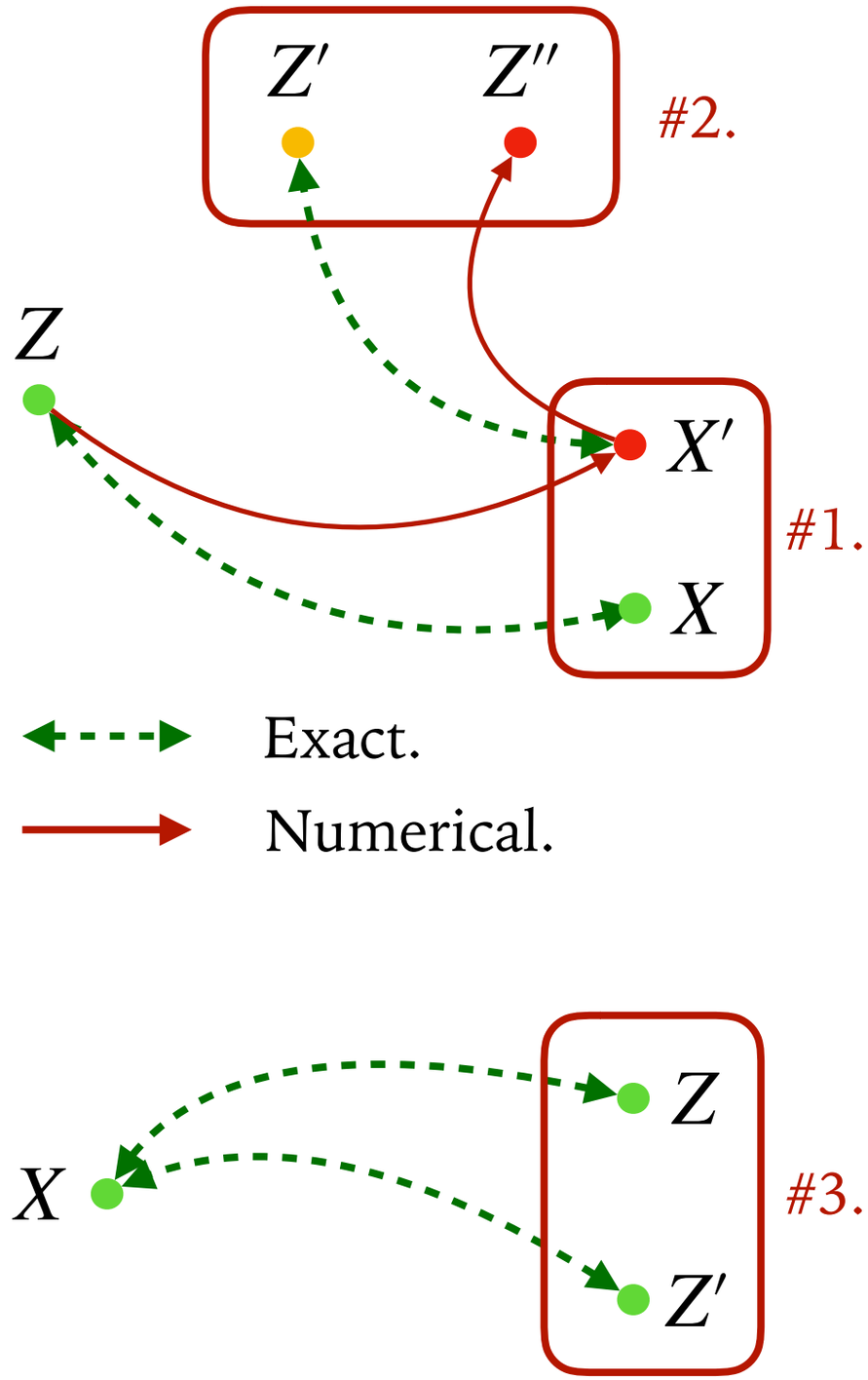
- The expectation can be computed with mini-batch (Monte Carlo Estimator).

Reversibility - 3 types of errors

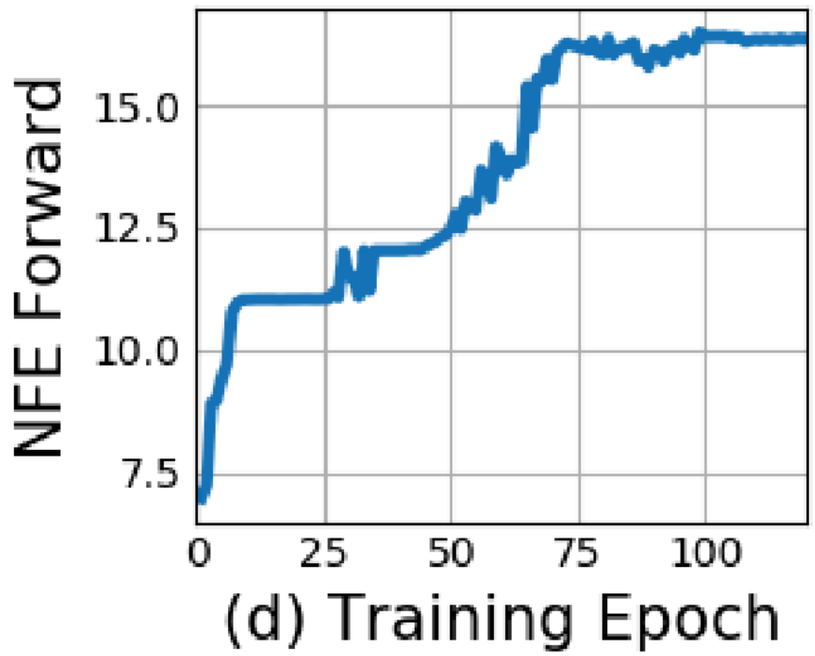
$$X = Z + \int_{t_0}^{t_1} f_\theta(Z(t), t)dt, \quad Z(t_0) = Z$$

$$Z = X + \int_{t_1}^{t_0} f_\theta(Z(t), t)dt, \quad Z(t_1) = X$$

1. Numerical error in forward pass.
2. Numerical error in backward pass.
3. Information lost due to multiple initial values mapping to the same final state.



Scalability



- Number of function evaluation can be prohibitive.
- NFE increases as training proceed since more numerically precision gradients information are needed as learning rate decreases.
- Decrease numerical precision for the ODESolver leads to slower convergence.

Questions?

- Possible ways to speed up?
- The advantages of Neural ODE over vanilla NN?
- More possible applications of Neural ODE?
- .....

References

- Chen, Tian Qi, et al. "Neural ordinary differential equations." Advances in Neural Information Processing Systems. 2018.
- Authors' Talk Slides: https://vectorinstitute.ai/wp-content/uploads/2019/03/ode-talk-vector-symposium.pdf