Deep Generative Models
Lecture 12

Roman Isachenko

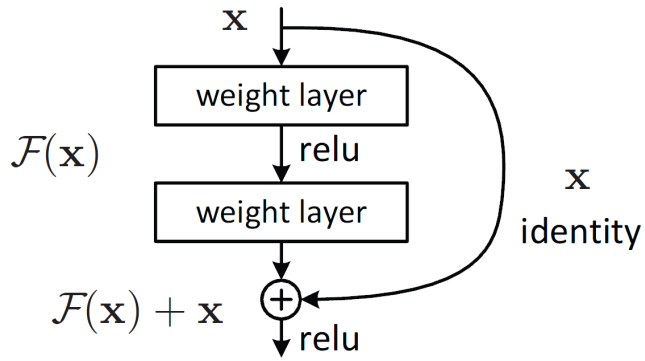
Moscow Institute of Physics and Technology

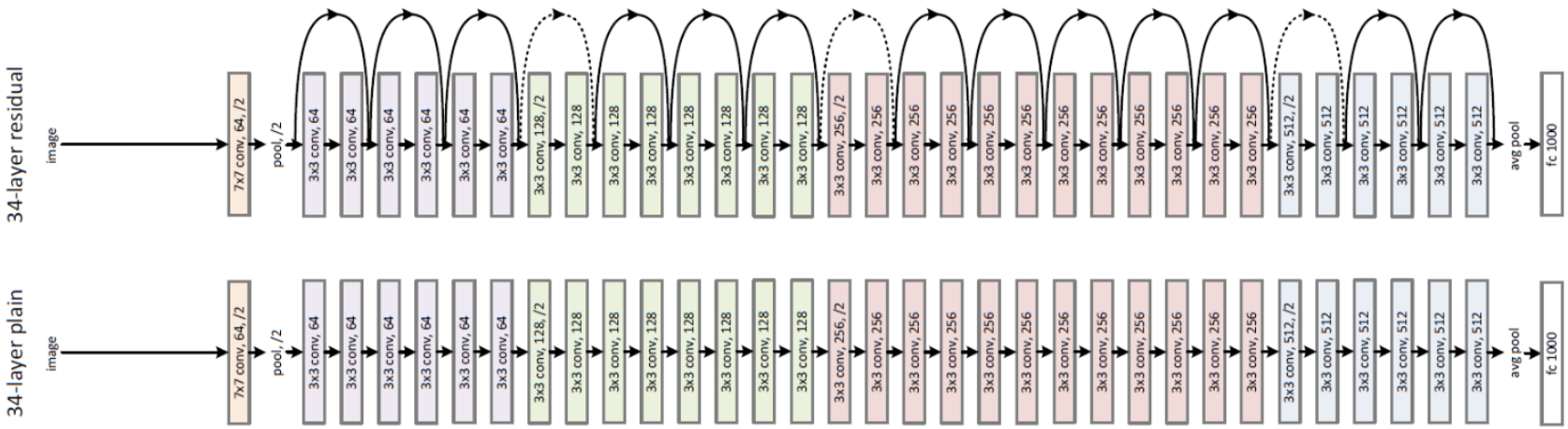
2020

Neural ODE

- Neural network for classification task has hundreds of layers.
- Skip connections eliminates exploding/vanishing gradients.

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \boldsymbol{\theta})$$





https://arxiv.org/abs/1806.07366

Neural ODE

Consider Ordinary Differential Equation

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \boldsymbol{\theta}); \quad \text{with initial condition } \mathbf{z}(t_0) = \mathbf{z}_0.$$

$$\mathbf{z}(t_1) = \int_{t_0}^{t_1} f(\mathbf{z}(t), \boldsymbol{\theta}) dt + \mathbf{z}_0 \quad \Leftarrow \quad \text{solution.}$$

Euler update step

$$\frac{\mathbf{z}(t + \Delta t) - \mathbf{z}(t)}{\Delta t} = f(\mathbf{z}(t), \boldsymbol{\theta}) \quad \Rightarrow \quad \mathbf{z}(t+\Delta t) = \mathbf{z}(t) + \Delta t f(\mathbf{z}(t), \boldsymbol{\theta}).$$

Residual block

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \boldsymbol{\theta}).$$

It is exactly Euler update step for solving ODE with $\Delta t = 1$!
Euler update step is unstable and trivial.

https://arxiv.org/abs/1806.07366

Neural ODE

Residual block

$$\mathbf{z}_{t+1} = \mathbf{z}_t + f(\mathbf{z}_t, \boldsymbol{\theta}).$$

What happens as we add more layers and take smaller steps?
In the limit, we parameterize the continuous dynamics of hidden units using an ODE specified by a neural network:

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), t, \boldsymbol{\theta}); \quad \mathbf{z}(t_0) = \mathbf{x}; \quad \mathbf{z}(t_1) = \mathbf{y}.$$

Loss function (forward pass)

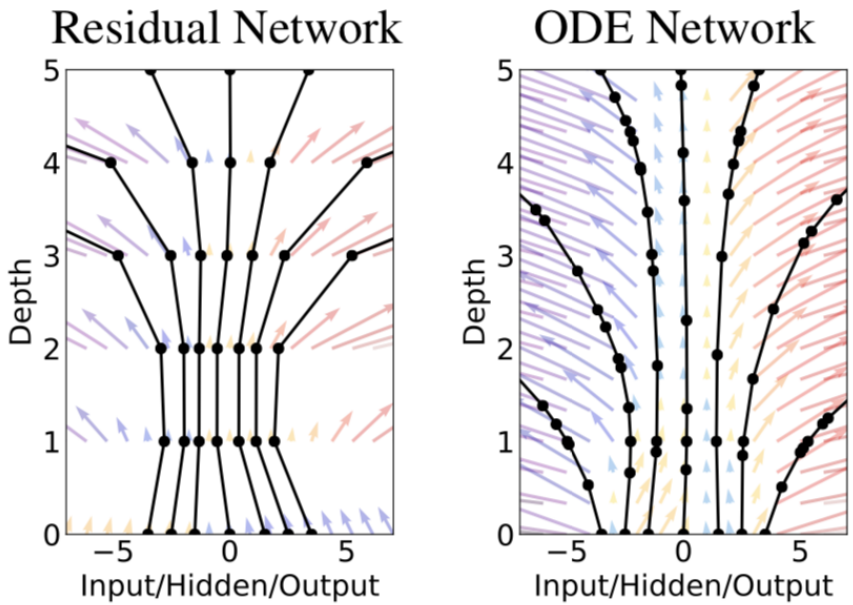
$$\begin{aligned} L(\mathbf{y}) = L(\mathbf{z}(t_1)) &= L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \boldsymbol{\theta}) dt\right) \\ &= L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \boldsymbol{\theta})\right) \end{aligned}$$

If we could get $\frac{\partial L(\mathbf{y})}{\partial \boldsymbol{\theta}}$, the problem will be solved.

Neural ODE

Benefits

- memory efficient (there is no need to store activations);
- adaptive computation (depth is not defined explicitly);
- parameter efficient (there is only parameters of function $f(\mathbf{z}(t), \theta)$);
- scalable and invertible normalizing flows (we will discuss it today).



https://arxiv.org/abs/1806.07366

Neural ODE

Loss function

$$L(\mathbf{y}) = L(\mathbf{z}(t_1)) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \boldsymbol{\theta})\right)$$

How to train such model? How to fit $\boldsymbol{\theta}$? How to compute efficiently $\frac{\partial L}{\partial \boldsymbol{\theta}}$? – **Pontryagin theorem!**

Adjoint function

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

Shows how the gradient of the loss depends on the hidden state $\mathbf{z}(t)$.

Theorem

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \mathbf{z}(t)}$$

Do we know any initilal condition?

Neural ODE

Theorem

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \mathbf{z}(t)}; \quad \mathbf{a}(t) = \frac{\partial L(\mathbf{z}(t))}{\partial \mathbf{z}(t)}$$

To obtain $\mathbf{a}(t)$ along the trajectory we could solve this ODE backward in time, starting from the initial value $\mathbf{a}(t_1) = \frac{\partial L(\mathbf{z}(t_1))}{\partial \mathbf{z}(t_1)}$.

Solution for adjoint function

$$\mathbf{a}(t_0) = -\int_{t_1}^{t_0} \mathbf{a}(t)^T \frac{\partial f(\mathbf{z}(t), t, \boldsymbol{\theta})}{\partial \mathbf{z}(t)} dt + \mathbf{a}(t_1)$$

Solution for hidden state

$$\mathbf{z}(t_1) = \int_{t_0}^{t_1} f(\mathbf{z}(t), \boldsymbol{\theta}) dt + \mathbf{z}_0.$$

---

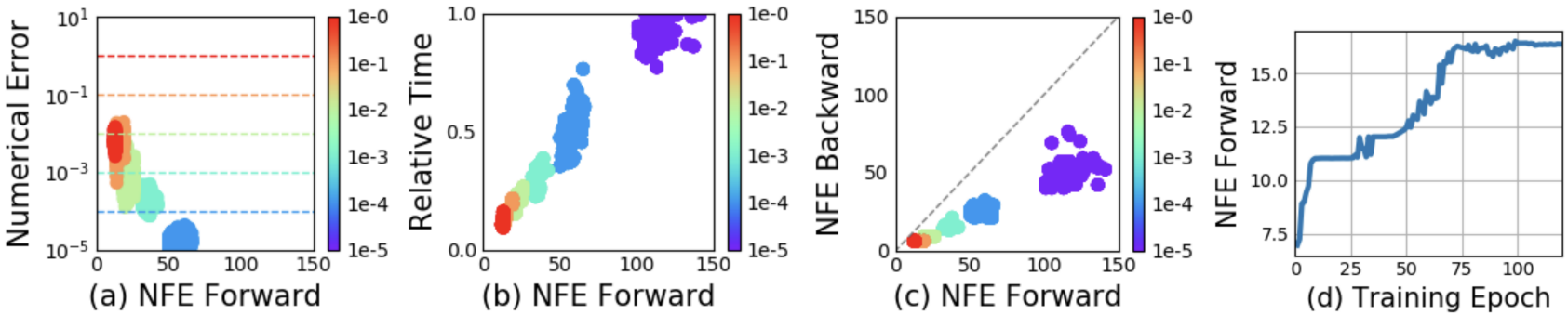
https://arxiv.org/abs/1806.07366

Neural ODE

Final gradient

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

Initial condition is $\frac{\partial L}{\partial \boldsymbol{\theta}(t_1)} = 0$ (why?).
All these ODEs could be computed simultaneiusly using augmented dynamic.



https://arxiv.org/abs/1806.07366

Continuous Normaling Flows

Discrete Normalizing Fflows

$$\mathbf{z}_{t+1} = f(\mathbf{z}_t, \theta); \quad \log p(\mathbf{z}_{t+1}) = \log p(\mathbf{z}_t) - \log \left| \det \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \mathbf{z}_t} \right|.$$

Planar flows

$$\mathbf{z}_{t+1} = g(\mathbf{z}_t, \theta) = \mathbf{z}_t + \mathbf{u} h(\mathbf{w}^T \mathbf{z}_t + b) = \mathbf{z}_t + f(\mathbf{z}_t, \theta).$$

Exactly residual learning.

Let consider continuous-in-time dynamic transformation of $\mathbf{z}_t$

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

Continuous dynamic for planar flows

$$\frac{d\mathbf{z}(t)}{dt} = \mathbf{u} h(\mathbf{w}^T \mathbf{z}_t + b); \quad \frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathbf{u}^T \frac{\partial h}{\partial \mathbf{z}(t)}.$$

Continuous Normaling Flows

Theorem

Consider the continuous dynamic $\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \boldsymbol{\theta})$. if function f is uniformly Lipschitz continuous in $\mathbf{z}$ and continuous in t, then the change in log probability follows a differential equation

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\text{trace}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right).$$

Continuity of function f guarantees that the solution of ODE exists and unique (Piccard theorem).

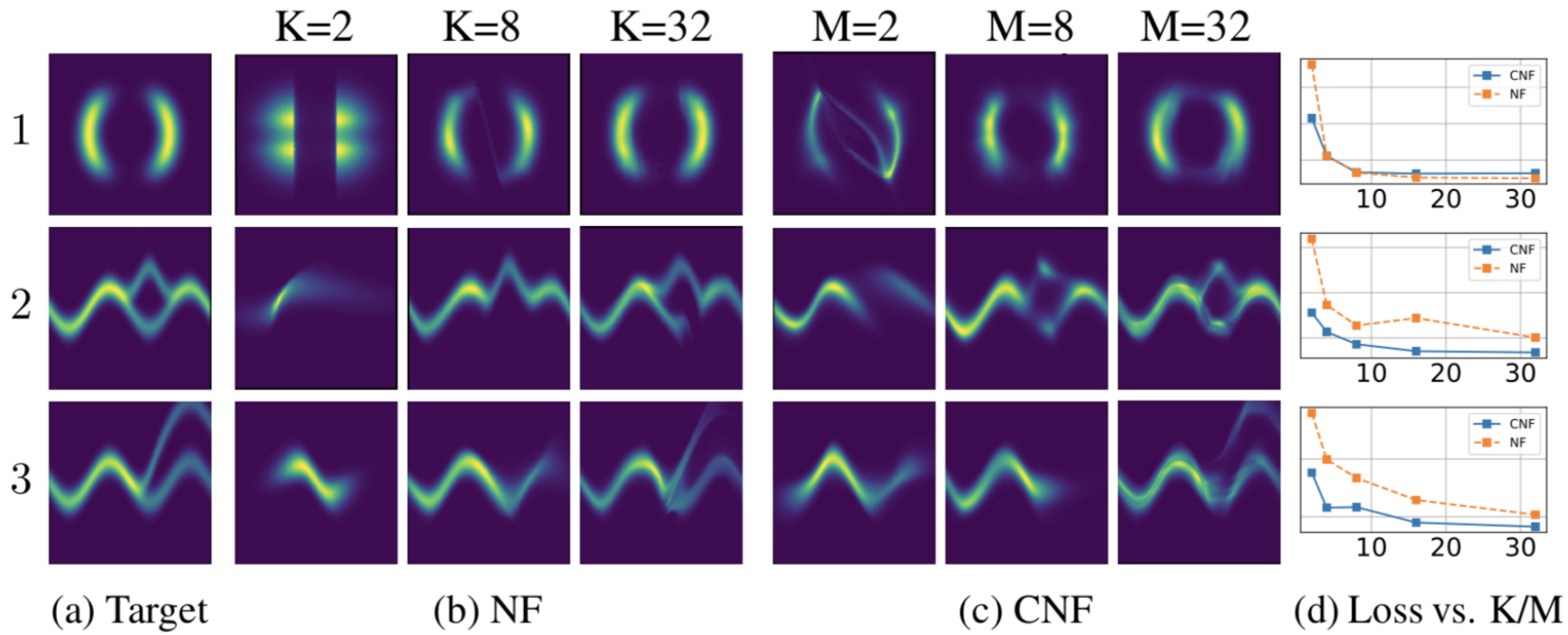
- Unlike standard finite flows, the differential equation f does not need to be bijective, since if uniqueness is satisfied, then the entire transformation is automatically bijective.
- trace$(\cdot)$ is a linear op instead of det$(\cdot)$

$$\frac{d\mathbf{z}(t)}{dt} = \sum_{j=1}^{M} f_j(\mathbf{z}(t), \boldsymbol{\theta}); \quad \frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\sum_{j=1}^{M} \text{trace}\left(\frac{\partial f_j}{\partial \mathbf{z}(t)}\right).$$

Continuous NF

Solution for continuous NF

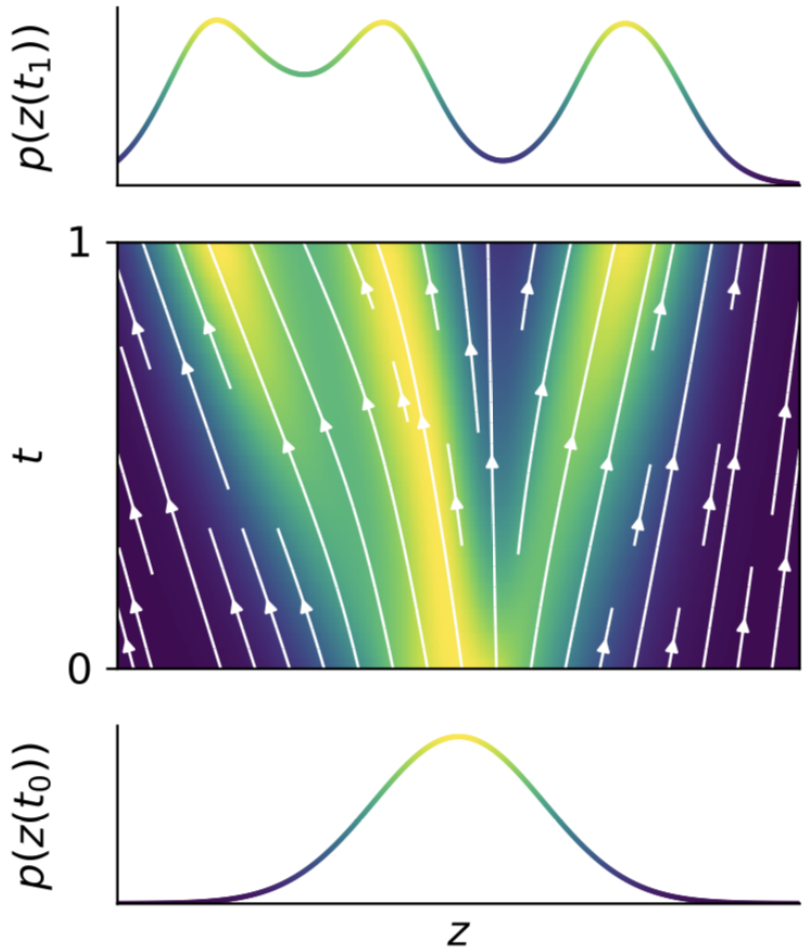
$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{trace}\left(\frac{\partial f}{\partial \mathbf{z}}\right) dt.$$



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

https://arxiv.org/abs/1806.07366

FFJORD



https://arxiv.org/abs/1810.01367

FFJORD

Hutchinson's trace estimator

$$\text{trace}(A) = \mathbb{E}_{p(\epsilon)}\left[\epsilon^T A \epsilon\right]; \quad \mathbb{E}[\epsilon] = 0; \quad \text{Cov}(\epsilon) = I.$$

$$\begin{aligned}
\log p(\mathbf{z}(t_1)) &= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{trace}\left(\frac{\partial f}{\partial \mathbf{z}}\right) dt \\
&= \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \mathbb{E}_{p(\epsilon)}\left[\epsilon^T \frac{\partial f}{\partial \mathbf{z}} \epsilon\right] dt \\
&= \log p(\mathbf{z}(t_0)) - \mathbb{E}_{p(\epsilon)} \int_{t_0}^{t_1} \left[\epsilon^T \frac{\partial f}{\partial \mathbf{z}} \epsilon\right] dt.
\end{aligned}$$

This reduces the cost from quadratic to linear.

---

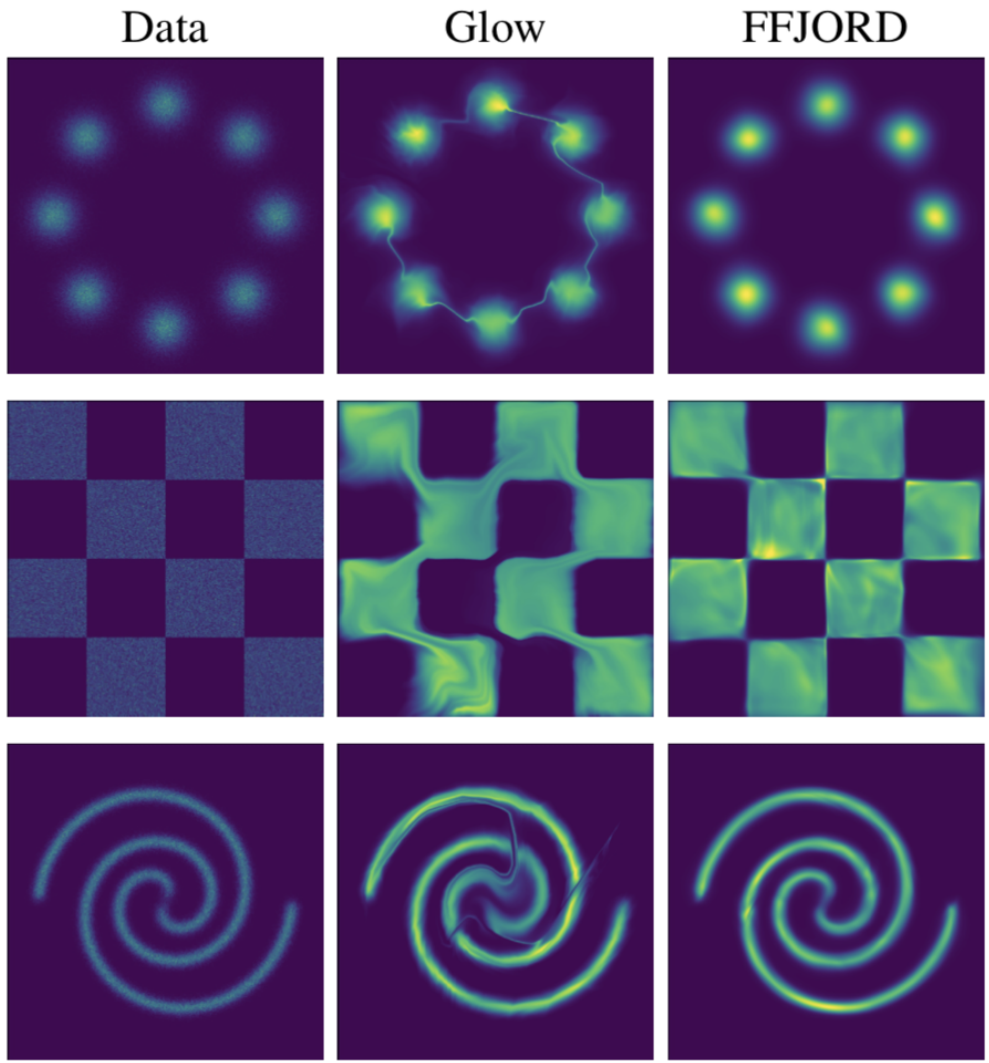
https://arxiv.org/abs/1810.01367

FFJORD

	Method	Train on data	One-pass Sampling	Exact log-likelihood	Free-form Jacobian
	Variational Autoencoders	✓	✓	✗	✓
	Generative Adversarial Nets	✓	✓	✗	✓
	Likelihood-based Autoregressive	✓	✗	✓	✗
Change of Variables	Normalizing Flows	✗	✓	✓	✗
Change of Variables	Reverse-NF, MAF, TAN	✓	✗	✓	✗
Change of Variables	NICE, Real NVP, Glow, Planar CNF	✓	✓	✓	✗
Change of Variables	**FFJORD**	✓	✓	✓	✓

https://arxiv.org/abs/1810.01367

FFJORD



https://arxiv.org/abs/1810.01367

Discrete VAE

- Before we have discussed VAE with **continuous** latent variables.
- VAE suffers from posterior collapse if the decoder too powerful (PixelVAE, VLAE tries to solve this problem).
- **Discrete** representations are potentially a more natural fit for many of the modalities.
- Powerful autoregressive models (like PixelCNNN) have been developed for modelling distributions over discrete variables.
- However, to construct the model with discrete representations is not so easy (e.g. variance of such estimators is a problem).

---

https://arxiv.org/abs/1711.00937

Vector Quantized VAE

- Latent embedding space $\{\mathbf{e}_j\}_{j=1}^{K}$, where $\mathbf{e}_j \in \mathbb{R}_D$, K is the size of the discrete latent space.
- $\mathbf{z}_e(\mathbf{x})$ is the encoder output.
- z is the discrete random variable calculated by a nearest neighbour look-up using the shared embedding space. The posterior categorical distribution is defined as

$$q(z = k|\mathbf{x}) = \begin{cases} 1, & \text{for } k = \arg\min_j \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j\| \\ 0, & \text{otherwise.} \end{cases}$$

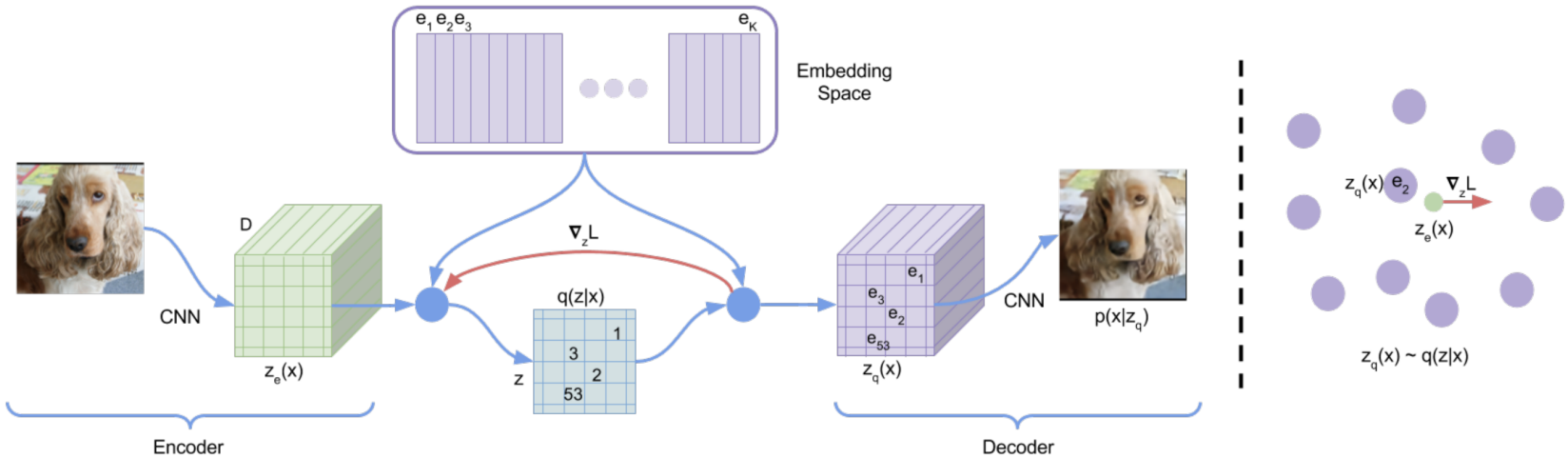
- VAE proposal distribution $q(z|\mathbf{x})$ is deterministic. If prior $p(z)$ is a uniform then $KL(q(z|\mathbf{x})||p(z))$ term in ELBO is constant (equals to $\log K$).
- Quantized representation is defined as follows

$$\mathbf{z}_q(\mathbf{x}) = \mathbf{e}_k, \quad \text{where } k = \arg\min_j \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_j\|$$

---

https://arxiv.org/abs/1711.00937

Vector Quantized VAE



Objective

$$\log p(\mathbf{x}|\mathbf{z}_q(\mathbf{x})) + \|\text{sg}(\mathbf{z}_e(\mathbf{x})) - \mathbf{z}_q(\mathbf{x})\| + \beta\|\mathbf{z}_e(\mathbf{x}) - \text{sg}(\mathbf{z}_q(\mathbf{x}))\|$$

- Quantization operation is not differentiable.
- Straight-through gradient estimation is used to backpropagate the quantization operation.

---

https://arxiv.org/abs/1711.00937

Vector Quantized VAE

https://arxiv.org/abs/1711.00937

References

- Neural Ordinary Differential Equations
https://arxiv.org/abs/1806.07366
Summary: New interpretation of resnets as special case of ode. Discrete sequence of layers are replaced with continuous dynamic. ODESolver is used for backpropagation. Pontryagin theorem gives the analog of the chain rule. Continuous version of normalizing flow is constructed.
- **FFJORD:** Free-form Continuous Dynamics for Scalable Reversible Generative Models
https://arxiv.org/abs/1810.01367
Summary: Continuous version of NF is investigated. Jacobian computation cost is reduced to O(D) by using Hutchinson's trace estimator.
- **VQ-VAE:** Neural discrete representation learning
https://arxiv.org/abs/1711.00937
Summary: Discrete latent representation for VAE. Nearest neighbor lookup table quantization is used. Learned powerful PixelCNN prior in the latent space.