

# Deep Generative Models

## Lecture 12

Roman Isachenko

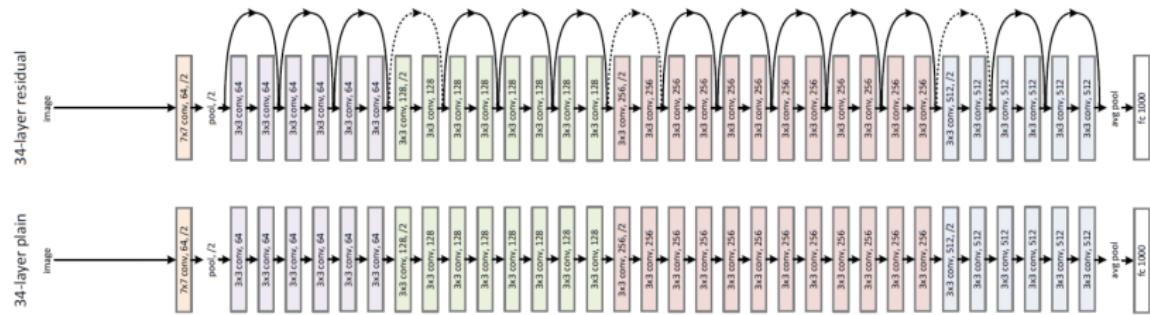
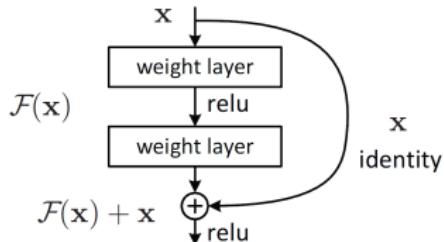
Moscow Institute of Physics and Technology

2020

# Neural ODE

How did it become possible to train neural networks with hundreds of layers?

Skip connections eliminates exploding/vanishing gradients.



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

# Neural ODE

Consider ODE

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

Euler update step

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

Residual block

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

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

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

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:

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

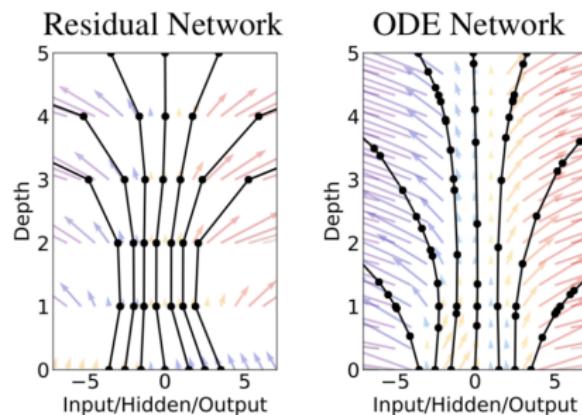
## Loss function

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

## Neural ODE

## Benefits

- ▶ memory efficient;
  - ▶ adaptive computation;
  - ▶ parameter efficient;
  - ▶ scalable and invertible normalizing flows.



# Neural ODE

## Loss function

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

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

## Adjoint function

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

## Theorem

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

# Neural ODE

## Theorem

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

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

## Theorem

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

All these gradients could be computed at once.

---

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

# Continuous NF

## Discrete NF

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

Function  $f$  should be bijective!

## Theorem

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

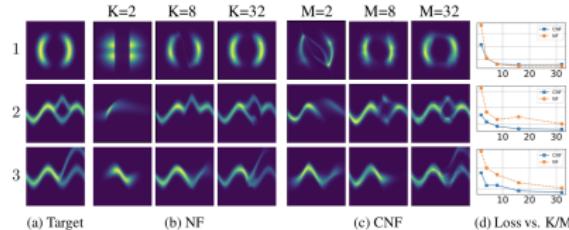
Function  $f$  is not necessary bijective! (uniformly Lipschitz continuous in  $\mathbf{z}$  and continuous in  $t$ ).

---

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

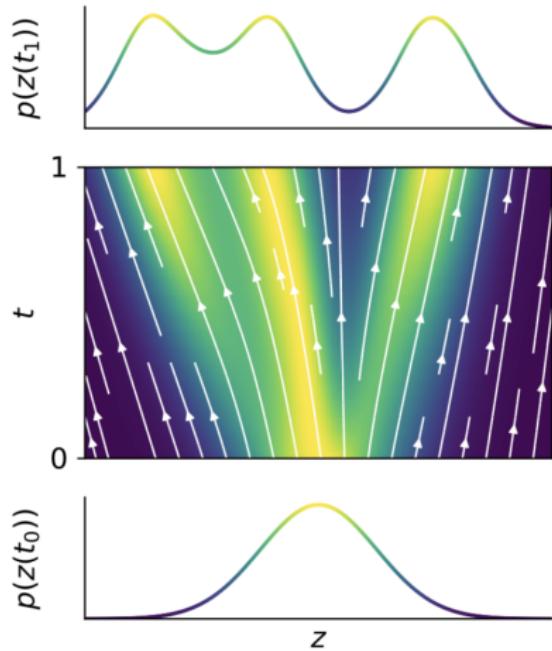
# Continuous NF

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



---

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



## Hutchinson's trace estimator

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

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

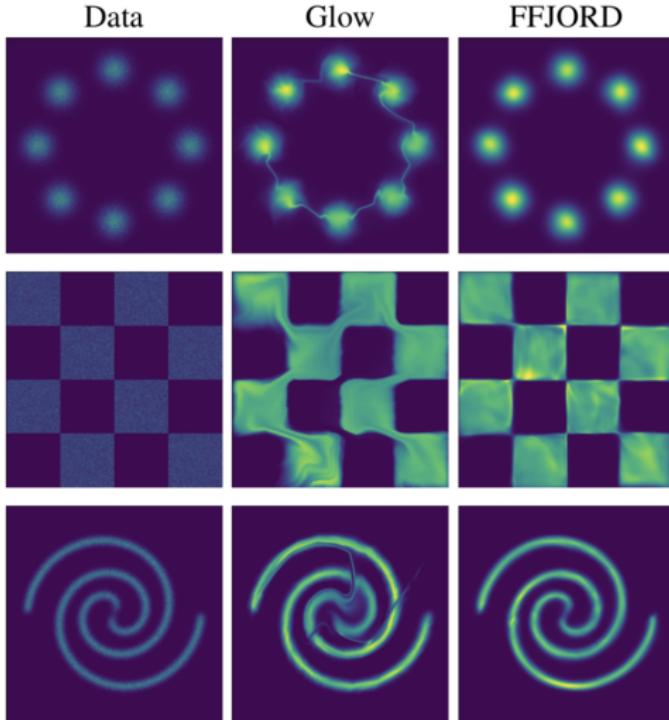
This reduces the cost from quadratic to linear.

# 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	✗	✓	✓
	Reverse-NF, MAF, TAN	✓	✗	✓
	NICE, Real NVP, Glow, Planar CNF	✓	✓	✓
	<b>FFJORD</b>	✓	✓	✓

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

# FFJORD



## Discrete VAE

- ▶ Before we have discussed VAE with **continuous** latent variables.
- ▶ VAE suffers from posterior collapse if the decoder too powerful (PixelVAE, VLAЕ 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

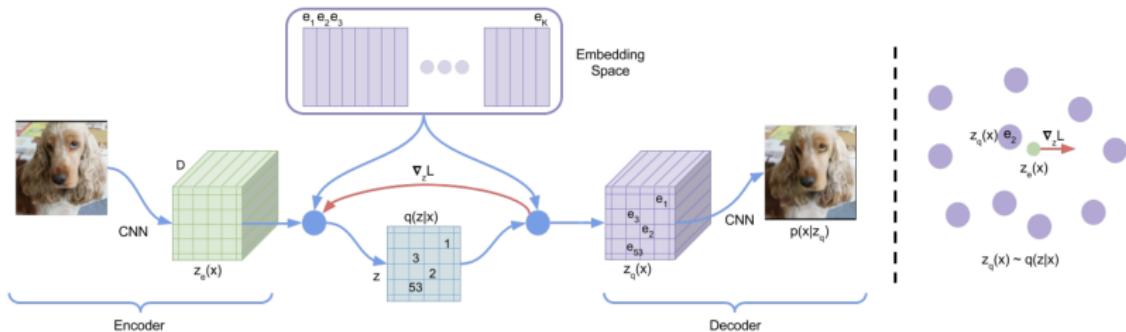
- ▶ Latent embedding space  $\{\mathbf{e}_j\}_{j=1}^K$ , where  $\mathbf{e}_j \in \mathbb{R}_D$ ,  $K$  is the size of the discrete latent space.
- ▶  $\mathbf{z}_e(\mathbf{x})$  is the encoder output.
- ▶  $z$  is the discrete random variable calculated by a nearest neighbour look-up using the shared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

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

# Vector Quantized VAE

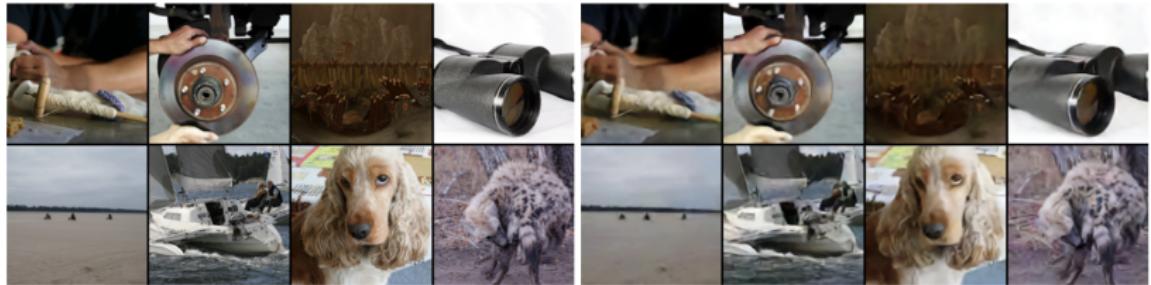


## Objective

$$\log p(x|z_q(x)) + \|sg(z_e(x)) - z_q(x)\| + \beta \|z_e(x) - sg(z_q(x))\|$$

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

# 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.