

Deep Generative Models

Lecture 12

Roman Isachenko

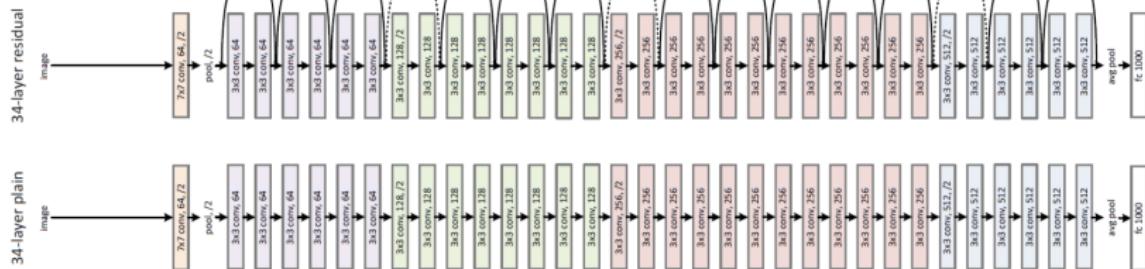
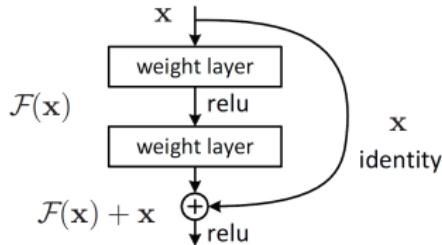
Moscow Institute of Physics and Technology

2020

Neural ODE

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

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



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

Neural ODE

Consider Ordinary Differential Equation

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

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

Euler update step

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

Residual block

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

- ▶ Residual learning is equivalent to Euler update step for solving ODE with $\Delta t = 1$!
- ▶ Euler update step is unstable and trivial. There are much more sophisticated methods.

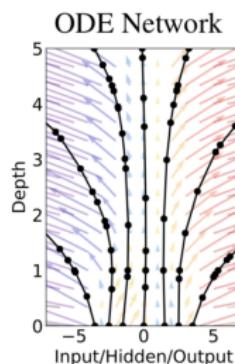
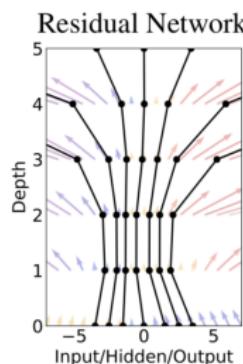
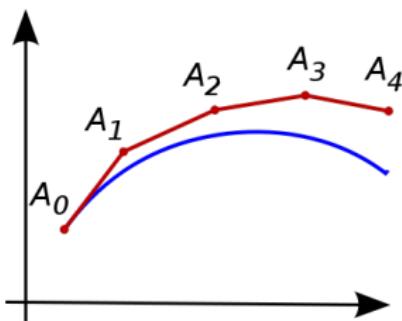
Neural ODE

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:

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



Neural ODE

Forward pass (loss function)

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

Note: ODESolve could be any method (not necessary Euler).

Backward pass (gradients computation)

For fitting parameters we need gradients:

$$\mathbf{a}_\mathbf{z}(t) = \frac{\partial L(\mathbf{y})}{\partial \theta(t)}; \quad \mathbf{a}_\theta(t) = \frac{\partial L(\mathbf{y})}{\partial \mathbf{z}(t_0)}.$$

In theory of optimal control these functions called **adjoint** functions.
They show how the gradient of the loss depends on the hidden state $\mathbf{z}(t)$ and parameters θ .

Neural ODE

Loss function (forward pass)

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

Adjoint functions

$$\mathbf{a}_z(t) = \frac{\partial L(\mathbf{z}(t))}{\partial \mathbf{z}(t)}; \quad \mathbf{a}_\theta(t) = \frac{\partial L(\mathbf{z}(t))}{\partial \theta}$$

Theorem (Pontryagin)

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

Do we know any initial condition?

Neural ODE

Theorem (Pontryagin)

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

Solution for adjoint function

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

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

$$\mathbf{z}(t_0) = \int_{t_0}^{t_1} f(\mathbf{z}(t), \theta) dt + \mathbf{z}_1.$$

Neural ODE

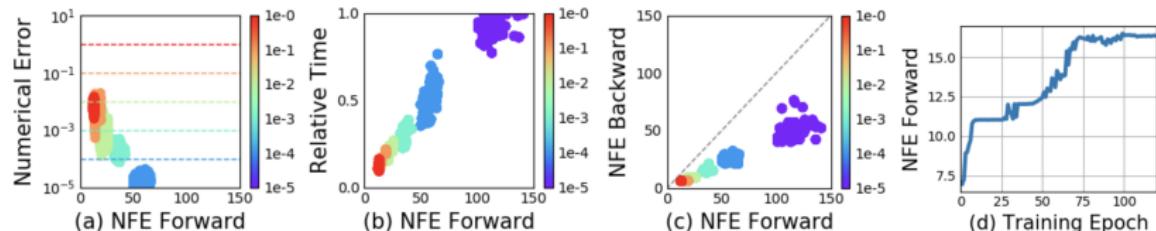
Forward pass

$$\mathbf{z}(t_1) = \int_{t_0}^{t_1} f(\mathbf{z}(t), \theta) dt + \mathbf{z}_0 \quad \Rightarrow \quad \text{ODE Solver}$$

Backward pass

$$\left. \begin{aligned} \frac{\partial L}{\partial \mathbf{z}(t_0)} &= \mathbf{a}_{\mathbf{z}}(t_0) = - \int_{t_1}^{t_0} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}(t)} dt + \frac{\partial L}{\partial \mathbf{z}(t_1)} \\ \frac{\partial L}{\partial \theta(t_0)} &= \mathbf{a}_{\theta}(t_0) = - \int_{t_1}^{t_0} \mathbf{a}_{\mathbf{z}}(t)^T \frac{\partial f(\mathbf{z}(t), \theta)}{\partial \theta(t)} dt \\ \mathbf{z}(t_0) &= \int_{t_0}^{t_1} f(\mathbf{z}(t), \theta) dt + \mathbf{z}_1. \end{aligned} \right\} \Rightarrow \text{ODE Solver}$$

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>

Continuous Normalizing Flows

Discrete Normalizing Flows

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

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 Normalizing Flows

Theorem

Consider the continuous dynamic $\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \theta)$. 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.
- ▶ $\text{trace}(\cdot)$ is a linear op instead of $\det(\cdot)$

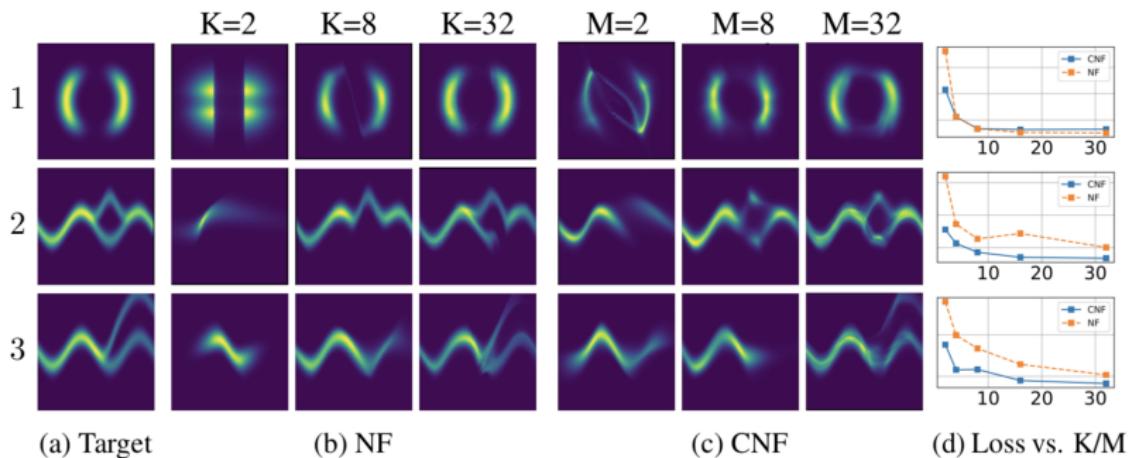
$$\frac{d\mathbf{z}(t)}{dt} = \sum_{j=1}^M f_j(\mathbf{z}(t), \theta); \quad \frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\sum_{j=1}^M \text{trace}\left(\frac{\partial f_j}{\partial \mathbf{z}(t)}\right).$$

Continuous NF

Solution for continuous NF

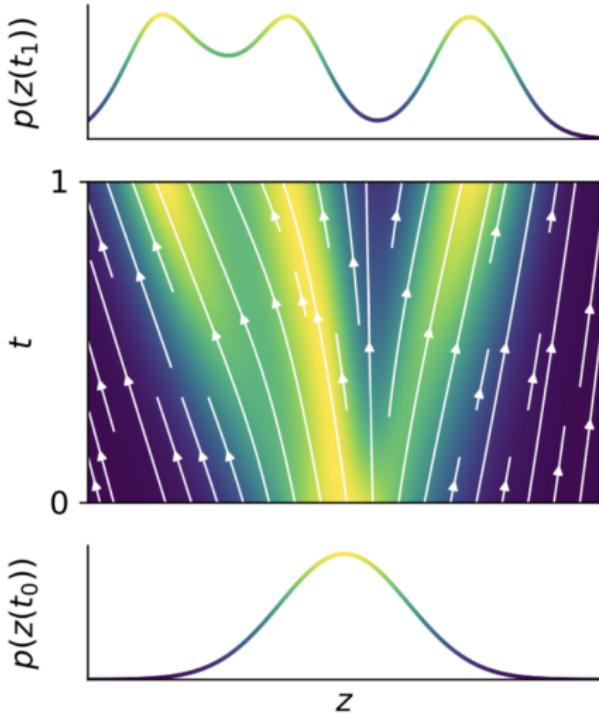
$$\mathbf{z}(t_0) = \int_{t_1}^{t_0} f(\mathbf{z}(t), \theta) dt + \mathbf{z}(t_1);$$

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



Continuous NF

- ▶ Standard normalizing flows need invertible f .
 - ▶ In general, it costs $O(d^3)$ to get det of Jacobian.
 - ▶ Continuous flows need smooth f .
 - ▶ In general, it costs $O(d^2)$ to get trace of Jacobian.



FFJORD

It is possible to reduce cost from $O(d^2)$ to $O(d)$.

Hutchinson's trace estimator

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

Unbiased estimation

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

Comparison of generative modelling approaches

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

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

FFJORD

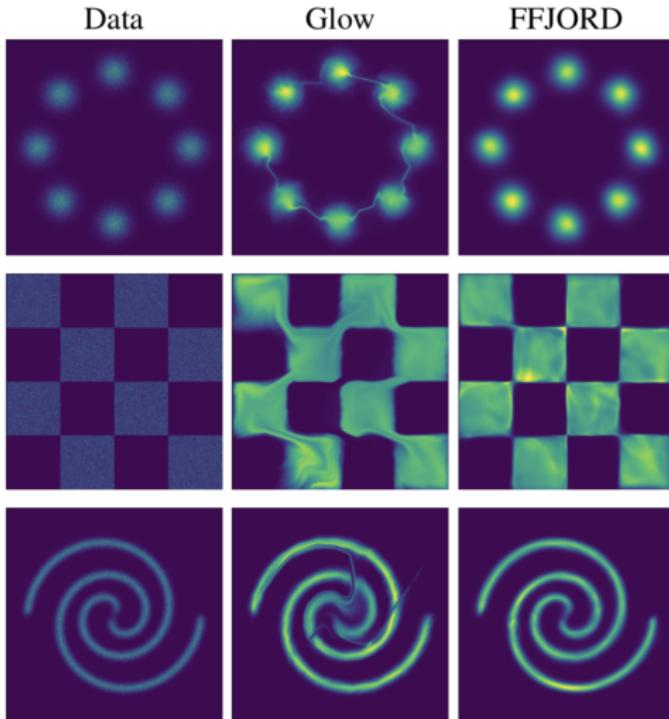
Density estimation

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	CIFAR10
Real NVP	-0.17	-8.33	18.71	13.55	-153.28	1.06*	3.49*
Glow	-0.17	-8.15	18.92	11.35	-155.07	1.05*	3.35*
FFJORD	-0.46	-8.59	14.92	10.43	-157.40	0.99* (1.05 [†])	3.40*
MADE	3.08	-3.56	20.98	15.59	-148.85	2.04	5.67
MAF	-0.24	-10.08	17.70	11.75	-155.69	1.89	4.31
TAN	-0.48	-11.19	15.12	11.01	-157.03	-	-
MAF-DDSF	-0.62	-11.96	15.09	8.86	-157.73	-	-

Flows for variational inference

	MNIST	Omniglot	Frey Faces	Caltech Silhouettes
No Flow	$86.55 \pm .06$	$104.28 \pm .39$	$4.53 \pm .02$	$110.80 \pm .46$
Planar	$86.06 \pm .31$	$102.65 \pm .42$	$4.40 \pm .06$	$109.66 \pm .42$
IAF	$84.20 \pm .17$	$102.41 \pm .04$	$4.47 \pm .05$	$111.58 \pm .38$
Sylvester	$83.32 \pm .06$	$99.00 \pm .04$	$4.45 \pm .04$	$104.62 \pm .29$
FFJORD	$82.82 \pm .01$	$98.33 \pm .09$	$4.39 \pm .01$	$104.03 \pm .43$

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

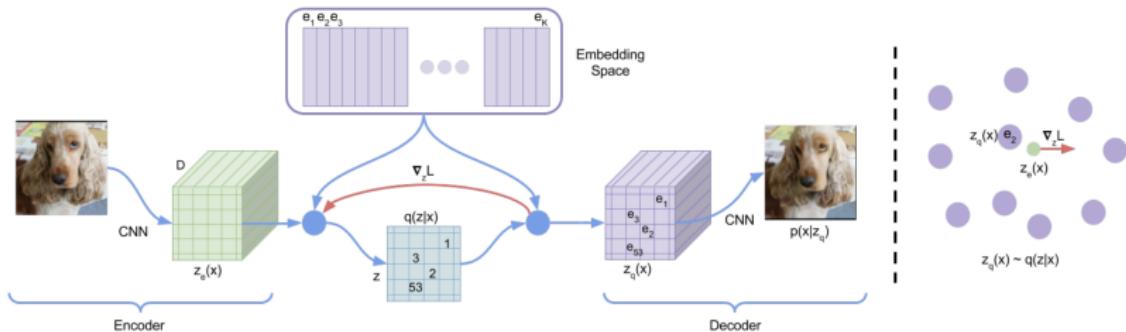
- ▶ Latent embedding space $\{\mathbf{e}_j\}_{j=1}^K$, where $\mathbf{e}_j \in \mathbb{R}_D$, K is the size of the discrete latent space.
- ▶ $\mathbf{z}_e(\mathbf{x})$ is the encoder output.
- ▶ z is the discrete random variable calculated by a nearest neighbour look-up using the shared embedding space.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



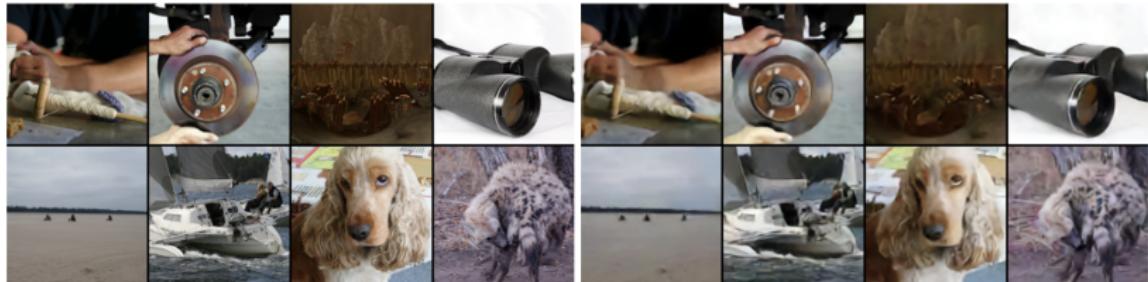
Objective

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

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

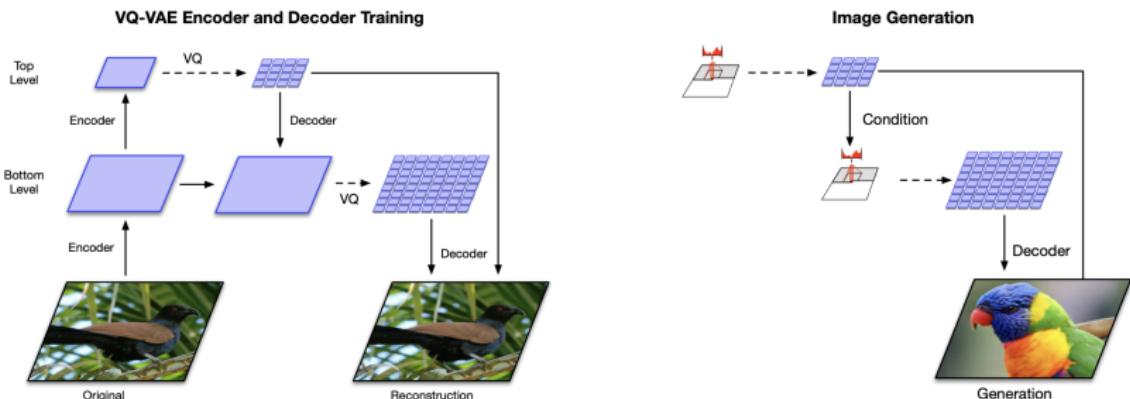
Vector Quantized VAE

- ▶ The prior distribution over the discrete latents $p(\mathbf{z})$ is a categorical distribution.
 - ▶ It could be made autoregressive by depending on other \mathbf{z} in the feature map.
 - ▶ Whilst training the VQ-VAE, the prior is kept constant and uniform.
 - ▶ After training, fit an autoregressive distribution (using PixelCNN) over \mathbf{z} .



Vector Quantized VAE-2

- ▶ Use multi-scale hierarchical model.
 - ▶ Use autoregressive prior model in each scale of the hierarchy.
 - ▶ Improve autoregressive prior (PixelSNAIL with self-attention in bottom layer, PixelCNN++ in bottom layer).
 - ▶ Train the encoder and decoder at the first stage, train the priors at the second stage.



Vector Quantized VAE-2

Algorithm 1 VQ-VAE training (stage 1)

Require: Functions E_{top} , E_{bottom} , D , \mathbf{x}
 (batch of training images)

$$1: \mathbf{h}_{top} \leftarrow E_{top}(\mathbf{x})$$

▷ quantize with top codebook eq 1

2: $\mathbf{e}_{top} \leftarrow Quantize(\mathbf{h}_{top})$

3: $\mathbf{h}_{bottom} \leftarrow E_{bottom}(\mathbf{x}, \mathbf{e}_{top})$

▷ quantize with bottom codebook eq. 1

4: $\mathbf{e}_{bottom} \leftarrow Quantize(\mathbf{h}_{bottom})$

$$5: \hat{\mathbf{x}} \leftarrow D(\mathbf{e}_{t_{\text{last}}}, \mathbf{e}_{k+t_{\text{last}}})$$

▷ Loss according to eq 2

6. $\theta \leftarrow \text{Update}(f(\mathbf{x}, \hat{\mathbf{x}}))$

Algorithm 2 Prior training (stage 2)

1: $\mathbf{T}_{top}, \mathbf{T}_{bottom} \leftarrow \emptyset$ ▷ training set

2: for $x \in$ training set do

$$3: \quad \mathbf{e}_{top} \leftarrow Quantize(E_{top}(\mathbf{x}))$$

4: $\mathbf{e}_{bottom} \leftarrow Quantize$

$$5: \quad \underline{\mathbf{T}}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$$

6: T_{bo}

7: **end for**

9: $p_{top} = \text{TrainPixelCNN}(T_{top})$

▷ Sampling procedure

10: while true do

$$11: \quad \mathbf{e}_{top} \sim p_{top}$$

$$12: \quad \mathbf{e}_{bottom} \sim p_{bottom}(\mathbf{e}_{top})$$

$$13: \quad \mathbf{x} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$$

14: **end while**



h_{top}

$h_{\text{top}}, h_{\text{middle}}$

$h_{\text{top}}, h_{\text{middle}}, h_{\text{bottom}}$

Original

Vector Quantized VAE-2

FFHQ 1024x1024



Vector Quantized VAE-2



VQ-VAE (Proposed)

BigGAN deep

Feature Quantized GAN

<https://arxiv.org/abs/2004.02088>

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.
- ▶ VQ-VAE-2: Generating Diverse High-Fidelity Images with VQ-VAE-2
<https://arxiv.org/abs/1906.00446>
Summary: The extension of VQ-VAE model for generating highly realistic images (comparable with sota gans). Use hierarchical latent representation with PixelSnail prior. Comparison with BigGan.
- ▶ FQ-VAE: Feature Quantization Improves GAN Training
<https://arxiv.org/abs/2004.02088>
Summary: Lots of "feature matching" techniques for GANN was introduced to stabilize training. Vector quantization has a property of implicit feature matching. Adding vector quantization module to GAN discriminator improves quality.