

# Deep Generative Models

## Lecture 5

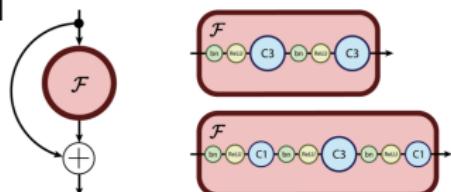
Roman Isachenko



Ozon Masters

Spring, 2021

- ▶ Modern neural networks are trained via backpropagation.
- ▶ Residual networks are state of the art in image classification.
- ▶ Backpropagation requires storing the network activations.



## Problem

Storing the activations imposes an increasing memory burden.

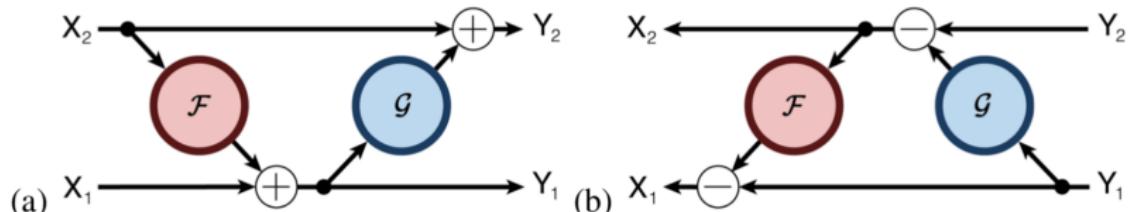
GPUs have limited memory capacity, leading to constraints often exceeded by state-of-the-art architectures (with thousand layers).

## NICE

$$\begin{cases} \mathbf{z}_1 = \mathbf{x}_1; \\ \mathbf{z}_2 = \mathbf{x}_2 + \mathcal{F}(\mathbf{x}_1, \theta); \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_1 = \mathbf{z}_1; \\ \mathbf{x}_2 = \mathbf{z}_2 - \mathcal{F}(\mathbf{z}_1, \theta). \end{cases}$$

## RevNet

$$\begin{cases} \mathbf{y}_1 = \mathbf{x}_1 + \mathcal{F}(\mathbf{x}_2, \theta); \\ \mathbf{y}_2 = \mathbf{x}_2 + \mathcal{G}(\mathbf{y}_1, \theta); \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_2 = \mathbf{y}_2 - \mathcal{F}(\mathbf{y}_1, \theta); \\ \mathbf{x}_1 = \mathbf{y}_1 - \mathcal{G}(\mathbf{x}_2, \theta). \end{cases}$$



Architecture	CIFAR-10 [15]		CIFAR-100 [15]	
	ResNet	RevNet	ResNet	RevNet
32 (38)	<b>7.14%</b>	7.24%	29.95%	<b>28.96%</b>
110	<b>5.74%</b>	5.76%	26.44%	<b>25.40%</b>
164	5.24%	<b>5.17%</b>	<b>23.37%</b>	23.69%

- ▶ If the network contains non-reversible blocks (poolings, strides), activations for these blocks should be stored.
- ▶ To avoid storing activations in the modern frameworks, the backward pass should be manually redefined.

## Hypothesis

The success of deep convolutional networks is based on progressively discarding uninformative variability about the input with respect to the problem at hand.

- ▶ It is difficult to recover images from their hidden representations.
- ▶ Information bottleneck principle: an optimal representation must reduce the MI between an input and its representation to reduce uninformative variability + maximize the MI between the output and its representation to preserve each class from collapsing onto other classes.

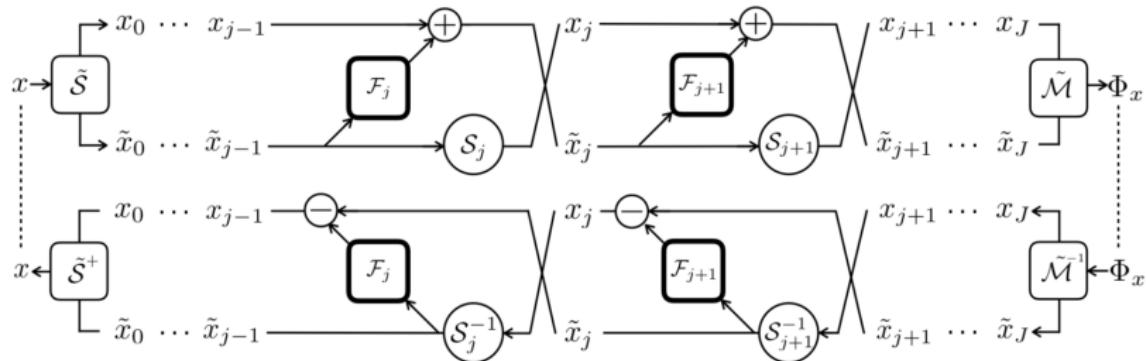
## Hypothesis

The success of deep convolutional networks is based on progressively discarding uninformative variability about the input with respect to the problem at hand.

## Idea

Build a cascade of homeomorphic layers (i-RevNet), a network that can be fully inverted up to the final projection onto the classes, i.e. no information is discarded.

# i-RevNet, 2018



Architecture	Injective	Bijective	Top-1 error	Parameters
ResNet	-	-	24.7	26M
RevNet	-	-	25.2	28M
<i>i</i> -RevNet (a)	yes	-	24.7	181M
<i>i</i> -RevNet (b)	yes	yes	26.7	29M

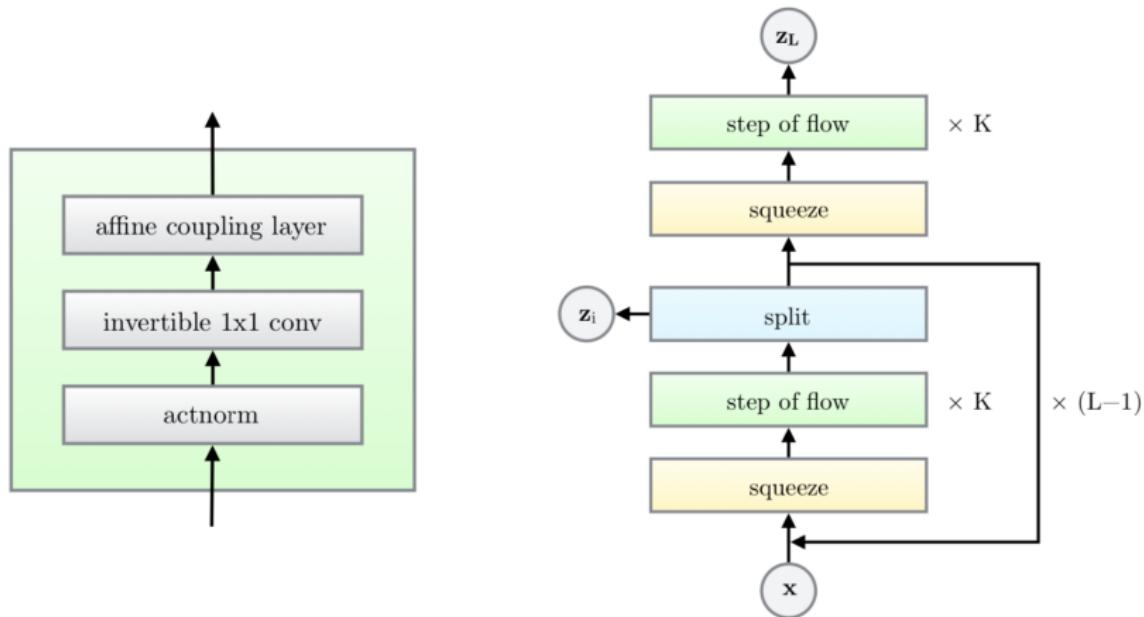
# Glow, 2018



---

*Kingma D. P., Dhariwal P. Glow: Generative Flow with Invertible 1x1 Convolutions, 2018*

## Model architecture



## NICE

$$\begin{cases} \mathbf{z}_1 = \mathbf{x}_1; \\ \mathbf{z}_2 = \mathbf{x}_2 + \mathcal{F}(\mathbf{x}_1, \theta); \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_1 = \mathbf{z}_1; \\ \mathbf{x}_2 = \mathbf{z}_2 - \mathcal{F}(\mathbf{z}_1, \theta). \end{cases}$$

First step is **split** operator which decouples a variable into 2 subparts (usually channel-wise). The order of decoupling should be manually changed between layers.

Could we use more general operator?

Let's use rotation matrix via  $1 \times 1$  invertible convolution.

$\mathbf{W} \in \mathbb{R}^{c \times c}$  - kernel of  $1 \times 1$  convolution with  $c$  input and output channels.

The cost of computing or differentiating  $\det(\mathbf{W})$  is  $O(c^3)$ .

# Glow, 2018

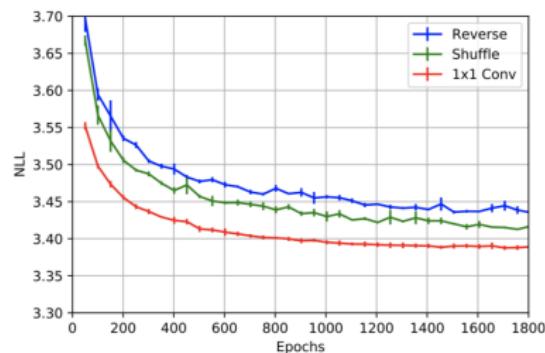
Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i, j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \text{sum}(\log  \mathbf{s} )$
Invertible $1 \times 1$ convolution. $\mathbf{W} : [c \times c]$ . See Section 3.2.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i, j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log  \det(\mathbf{W}) $ or $h \cdot w \cdot \text{sum}(\log  \mathbf{s} )$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \text{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \text{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \text{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \text{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \text{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \text{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\text{sum}(\log( \mathbf{s} ))$

## Invertible 1x1 conv

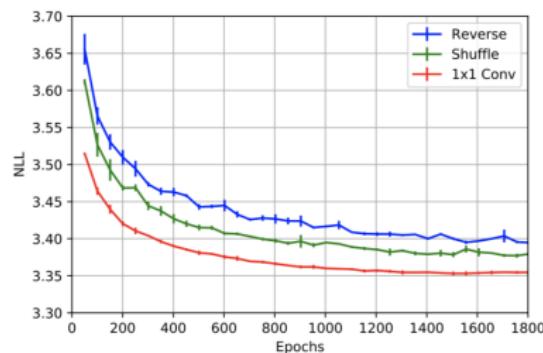
Cost to compute  $\det(\mathbf{W})$  is  $O(c^3)$ . LU-decomposition reduces the cost to  $O(c)$ :

$$\mathbf{W} = \mathbf{P}\mathbf{L}(\mathbf{U} + \text{diag}(\mathbf{s})),$$

where  $\mathbf{P}$  is a permutation matrix,  $\mathbf{L}$  is a lower triangular matrix with ones on the diagonal,  $\mathbf{U}$  is an upper triangular matrix with zeros on the diagonal, and  $\mathbf{s}$  is a vector.



(a) Additive coupling.



(b) Affine coupling.

# Glow, 2018

## Face interpolation



# Glow, 2018

## Face attributes manipulation



(a) Smiling

(b) Pale Skin



(c) Blond Hair

(d) Narrow Eyes



(e) Young

(f) Male

## Summary

- ▶ Flows are generative models with tractable likelihood and latent representation.
- ▶ Flows transform simple distributions into the complex ones via sequences of invertible transformations.
- ▶ The goal is to achieve tractable Jacobian for efficient learning and density estimation.

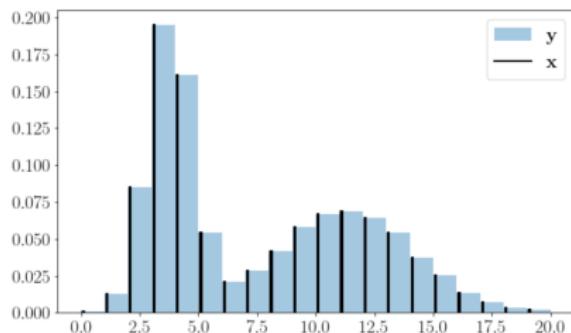
# Dequantization

- ▶ Images are discrete data (pixels lies in the [0, 255] integer domain).
- ▶ Flow is a continuous model.

Fitting a continuous density model to discrete data, produces a degenerate solution with all probability mass on discrete values.  
How to convert discrete data distribution to the continuous one?

## Uniform dequantization

$$\mathbf{y} = \mathbf{x} + \mathbf{u}, \quad \mathbf{u} \sim U[0, 1]$$



## Uniform dequantization

### Statement

Fitting continuous model  $p(\mathbf{y}|\theta)$  on uniformly dequantized data  $\mathbf{y} = \mathbf{x} + \mathbf{u}$ ,  $\mathbf{u} \sim U[0, 1]$  is equivalent to maximization of a lower bound on the log-likelihood for a discrete model:

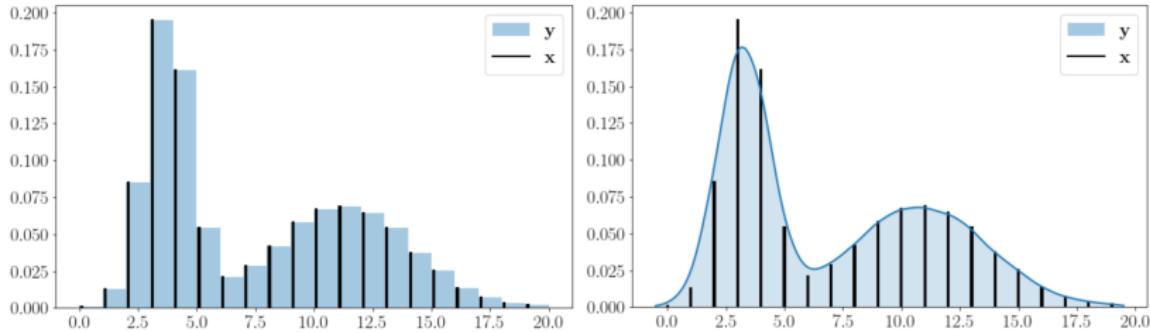
$$P(\mathbf{x}|\theta) = \int_{U[0,1]} p(\mathbf{x} + \mathbf{u}|\theta) d\mathbf{u}$$

Thus, maximizing the log-likelihood of the continuous model on  $\mathbf{y}$  cannot lead to the collapsing onto the discrete data (objective is bounded above by the log-likelihood of a discrete model).

### Proof

$$\begin{aligned} \log p(\mathbf{Y}|\theta) &= \sum_{i=1}^n \log p(\mathbf{y}_i|\theta) = \sum_{i=1}^n \int_{U[0,1]} \log p(\mathbf{x}_i + \mathbf{u}|\theta) d\mathbf{u} \\ &\leq \sum_{i=1}^n \log \int_{U[0,1]} p(\mathbf{x}_i + \mathbf{u}|\theta) d\mathbf{u} = \sum_{i=1}^n \log P(\mathbf{x}_i|\theta). \end{aligned}$$

# Variational dequantization



- ▶  $p(y|\theta)$  assign uniform density to unit hypercubes  $x + U[0, 1]$  (left fig).
- ▶ Neural network density models is a smooth function approximator (right fig).
- ▶ Smooth dequantization is more natural.

How to make the smooth dequantization?

# Flow++

## Variational dequantization

Introduce variational dequantization noise distribution  $q(\mathbf{u}|\mathbf{x})$  and treat it as an approximate posterior.

## Variational lower bound

$$\begin{aligned}\log P(\mathbf{X}|\theta) &= \sum_{i=1}^n \log P(\mathbf{x}_i|\theta) = \sum_{i=1}^n \left[ \log \int q(\mathbf{u}|\mathbf{x}) \frac{p(\mathbf{x} + \mathbf{u}|\theta)}{q(\mathbf{u}|\mathbf{x})} d\mathbf{u} \right] \geq \\ &\geq \sum_{i=1}^n \left[ \int q(\mathbf{u}|\mathbf{x}) \log \frac{p(\mathbf{x} + \mathbf{u}|\theta)}{q(\mathbf{u}|\mathbf{x})} d\mathbf{u} \right] = \\ &= \int q(\mathbf{U}|\mathbf{X}) \log \frac{p(\mathbf{X} + \mathbf{U}|\theta)}{q(\mathbf{U}|\mathbf{X})} d\mathbf{U} = \mathcal{L}(q, \theta).\end{aligned}$$

# Flow++

## Variational lower bound

$$\mathcal{L}(q, \theta) = \int q(\mathbf{U}|\mathbf{X}) \log \frac{p(\mathbf{X} + \mathbf{U}|\theta)}{q(\mathbf{U}|\mathbf{X})} d\mathbf{U}.$$

Let  $\mathbf{u} = h(\epsilon)$  is a flow model with base distribution  $\epsilon \sim p(\epsilon) = \mathcal{N}(0, \mathbf{I})$ :

$$q(\mathbf{u}|\mathbf{x}) = p(h^{-1}(\mathbf{u})) \cdot \left| \det \frac{\partial h^{-1}(\mathbf{u})}{\partial \mathbf{u}} \right|.$$

Then

$$\log P(\mathbf{X}|\theta) \geq \sum_{i=1}^n \int \log \left( \frac{p(\mathbf{x} + h(\epsilon))}{p(\epsilon) \cdot \left| \det \frac{\partial h(\epsilon)}{\partial \epsilon} \right|^{-1}} \right) d\epsilon.$$

## Flow++

$$\log P(\mathbf{X}|\theta) \geq \sum_{i=1}^n \int \log \left( \frac{p(\mathbf{x} + h(\epsilon))}{p(\epsilon) \cdot \left| \det \frac{\partial h(\epsilon)}{\partial \epsilon} \right|^{-1}} \right) d\epsilon.$$

If  $p(\mathbf{x} + \mathbf{u}|\theta)$  is also a flow model, it is straightforward to calculate stochastic gradient of this ELBO.

**Note:** Uniform dequantization is a special case of variational dequantization ( $q(\mathbf{u}|\mathbf{x}) = U[0, 1]$ ). The gap between  $\log P(\mathbf{X}|\theta)$  and the derived ELBO is

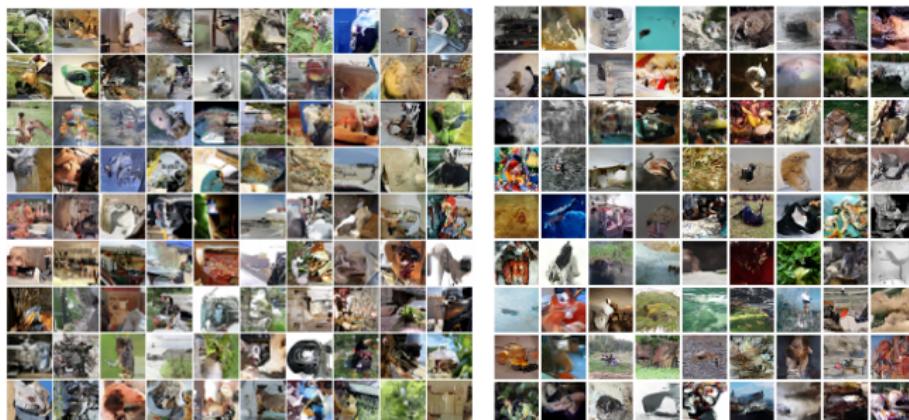
$$KL(q(\mathbf{U}|\mathbf{X})||p(\mathbf{U}|\mathbf{X})).$$

In the case of uniform dequantization the model unnaturally places uniform density over each hypercube  $\mathbf{x} + U[0, 1]$  due to inexpressive distribution  $q$ .

# Flow++

Table 1. Unconditional image modeling results in bits/dim

Model family	Model	CIFAR10	ImageNet 32x32	ImageNet 64x64
Non-autoregressive	RealNVP (Dinh et al., 2016)	3.49	4.28	—
	Glow (Kingma & Dhariwal, 2018)	3.35	4.09	3.81
	IAF-VAE (Kingma et al., 2016)	3.11	—	—
	<b>Flow++ (ours)</b>	<b>3.08</b>	<b>3.86</b>	<b>3.69</b>
Autoregressive	Multiscale PixelCNN (Reed et al., 2017)	—	3.95	3.70
	PixelCNN (van den Oord et al., 2016b)	3.14	—	—
	PixelRNN (van den Oord et al., 2016b)	3.00	3.86	3.63
	Gated PixelCNN (van den Oord et al., 2016c)	3.03	3.83	3.57
	PixelCNN++ (Salimans et al., 2017)	2.92	—	—
	Image Transformer (Parmar et al., 2018)	2.90	3.77	—
	PixelSNAIL (Chen et al., 2017)	2.85	3.80	3.52



(a) PixelCNN

(b) Flow++

# Likelihood-based models

## Exact likelihood evaluation

- ▶ Autoregressive models (PixelCNN, WaveNet);
- ▶ Flow models (NICE, RealNVP, Glow).

## Approximate likelihood evaluation

- ▶ Latent variable models (VAE).

What are the pros and cons of each of them?

# VAE recap

$$p(\mathbf{x}|\theta) \geq \mathcal{L}(\phi, \theta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \phi)} \log \frac{p(\mathbf{x}, \mathbf{z}|\theta)}{q(\mathbf{z}|\mathbf{x}, \phi)} \rightarrow \max_{\phi, \theta}.$$

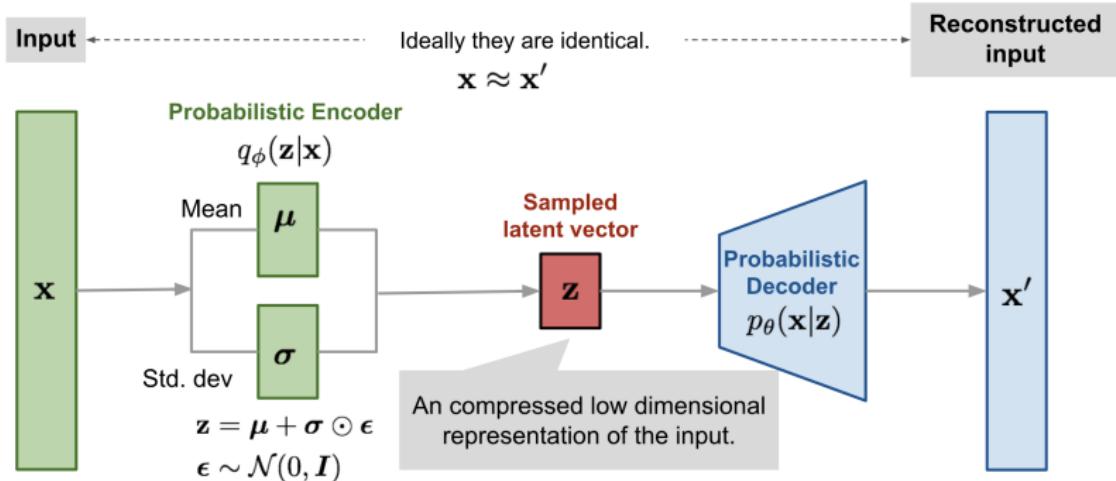


image credit:

<https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html>

## VAE limitations

- ▶ Poor variational posterior distribution (encoder)

$$q(\mathbf{z}|\mathbf{x}, \phi) = \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), \sigma_\phi^2(\mathbf{x})).$$

- ▶ Poor prior distribution

$$p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I}).$$

- ▶ Poor probabilistic model (decoder)

$$p(\mathbf{x}|\mathbf{z}, \theta) = \mathcal{N}(\mathbf{x}|\mu_\theta(\mathbf{z}), \sigma_\theta^2(\mathbf{z})).$$

- ▶ Loose lower bound

$$p(\mathbf{x}|\theta) - \mathcal{L}(q, \theta) = (?).$$

## Variational posterior

We wish  $KL(q(\mathbf{z}|\mathbf{x}, \phi) || p(\mathbf{z}|\mathbf{x}, \theta)) = 0$ .

(In this case the lower bound is tight  $p(\mathbf{x}|\theta) = \mathcal{L}(q, \theta)$ ).

Normal variational distribution  $q(\mathbf{z}|\mathbf{x}, \phi) = \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), \sigma_\phi^2(\mathbf{x}))$  is poor (e.g. has only one mode).

Flows models convert a simple base distribution to a complex one using invertible transformation with simple Jacobian.

How to use flows in VAE?

## Flows in VAE

Apply a sequence of transformations to the random variables

$$\mathbf{z}_0 \sim q(\mathbf{z}|\mathbf{x}, \phi) = \mathcal{N}(\mathbf{z}|\mu_\phi(\mathbf{x}), \sigma_\phi^2(\mathbf{x})).$$

Here,  $q(\mathbf{z}|\mathbf{x}, \phi)$  (which is a VAE encoder) plays a role of a base distribution.

$$\mathbf{z}_0 \xrightarrow{g_1} \mathbf{z}_1 \xrightarrow{g_2} \dots \xrightarrow{g_K} \mathbf{z}_K, \quad \mathbf{z}_K = g(\mathbf{z}_0), \quad g = g_K \circ \dots \circ g_1.$$

Each  $g_k$  is a flow transformation (e.g. planar, radial, coupling layer) parameterized by  $\phi_k$ .

$$\begin{aligned} \log q_K(\mathbf{z}_K|\mathbf{x}, \phi, \{\phi\}_{k=1}^K) &= \log q(\mathbf{z}_0|\mathbf{x}, \phi) \\ &\quad - \sum_{k=1}^K \log \left| \det \left( \frac{\partial g_k(\mathbf{z}_{k-1}, \phi_k)}{\partial \mathbf{z}_{k-1}} \right) \right|. \end{aligned}$$

# Flows in VAE

## Flow model in latent space

$$\log q_K(\mathbf{z}_K | \mathbf{x}, \phi, \{\phi\}_{k=1}^K) = \log q(\mathbf{z}_0 | \mathbf{x}, \phi) - \sum_{k=1}^K \log \left| \det \left( \frac{\partial g_k(\mathbf{z}_{k-1}, \phi_k)}{\partial \mathbf{z}_{k-1}} \right) \right|.$$

Let use  $q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)$ ,  $\phi_* = \{\phi, \phi_1, \dots, \phi_K\}$  as a variational distribution.

Here  $\phi$  – encoder parameters,  $\{\phi\}_{k=1}^K$  – flow parameters.

## ELBO objective

$$\begin{aligned}\mathcal{L}(\phi, \theta) &= \mathbb{E}_{q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)} \log \frac{p(\mathbf{x}, \mathbf{z}_K | \theta)}{q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)} \\ &= \mathbb{E}_{q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)} [\log p(\mathbf{x}, \mathbf{z}_K | \theta) - \log q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)]\end{aligned}$$

# Flows in VAE

## Variational distribution

$$\log q_K(\mathbf{z}_K | \mathbf{x}, \phi_*) = \log q(\mathbf{z}_0 | \mathbf{x}, \phi) - \sum_{k=1}^K \log \left| \det \left( \frac{\partial g_k(\mathbf{z}_{k-1}, \phi_k)}{\partial \mathbf{z}_{k-1}} \right) \right|.$$

## ELBO objective

$$\begin{aligned}\mathcal{L}(\phi, \theta) &= \mathbb{E}_{q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)} [\log p(\mathbf{x}, \mathbf{z}_K | \theta) - \log q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)] \\ &= \mathbb{E}_{q(\mathbf{z}_0 | \mathbf{x}, \phi)} [\log p(\mathbf{x}, \mathbf{z}_K | \theta) - \log q_K(\mathbf{z}_K | \mathbf{x}, \phi_*)] \Big|_{\mathbf{z}_K = g(\mathbf{z}_0, \{\phi\}_{k=1}^K)} \\ &= \mathbb{E}_{q(\mathbf{z}_0 | \mathbf{x}, \phi)} \left[ \log p(\mathbf{x}, \mathbf{z}_K | \theta) - \log q(\mathbf{z}_0 | \mathbf{x}, \phi) + \right. \\ &\quad \left. + \sum_{k=1}^K \log \left| \det \left( \frac{\partial g_k(\mathbf{z}_{k-1}, \phi_k)}{\partial \mathbf{z}_{k-1}} \right) \right| \right].\end{aligned}$$

## Gaussian autoregressive model

Consider autoregressive model

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{i=1}^m p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta}),$$

with conditionals

$$p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta}) = \mathcal{N}(\hat{\mu}_i(\mathbf{x}_{1:i-1}), \hat{\sigma}_i^2(\mathbf{x}_{1:i-1})).$$

## Sampling

$$x_i = \hat{\sigma}_i(\mathbf{x}_{1:i-1}) \cdot z_i + \hat{\mu}_i(\mathbf{x}_{1:i-1}), \quad z_i \sim \mathcal{N}(0, 1).$$

Sampling from autoregressive model is sequential.

Note that we could interpret this sampling as a transformation  $\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta})$ , where  $\mathbf{z}$  comes from base distribution  $\mathcal{N}(0, 1)$ .

# Gaussian autoregressive model

## Sampling

$$x_i = \hat{\sigma}_i(\mathbf{x}_{1:i-1}) \cdot z_i + \hat{\mu}_i(\mathbf{x}_{1:i-1}), \quad z_i \sim \mathcal{N}(0, 1).$$

## Jacobian

$$\log \left| \det \left( \frac{\partial f(\mathbf{x}, \theta)}{\partial \mathbf{x}} \right) \right| = -\log \left| \det \left( \frac{\partial g(\mathbf{z}, \theta)}{\partial \mathbf{z}} \right) \right| = -\sum_{i=1}^m \log \hat{\sigma}_i(\mathbf{x}_{1:i-1}).$$

## Inverse transform

$$z_i = (x_i - \hat{\mu}_i(\mathbf{x}_{1:i-1})) \cdot \frac{1}{\hat{\sigma}_i(\mathbf{x}_{1:i-1})}.$$

We get an autoregressive model with tractable (triangular) Jacobian, which is easily invertible. It is a flow!

## Inverse autoregressive flow (IAF)

### Gaussian autoregressive model ( $\mathbf{z} \rightarrow \mathbf{x}$ )

$$x_i = \hat{\sigma}_i(\mathbf{x}_{1:i-1}) \cdot z_i + \hat{\mu}_i(\mathbf{x}_{1:i-1}).$$

$$z_i = (x_i - \hat{\mu}_i(\mathbf{x}_{1:i-1})) \cdot \frac{1}{\hat{\sigma}_i(\mathbf{x}_{1:i-1})}.$$

This process is sequential.

Let use the following reparametrization:  $\sigma = \frac{1}{\hat{\sigma}}$ ;  $\mu = -\frac{\hat{\mu}}{\hat{\sigma}}$ .

### Inverse transform ( $\mathbf{x} \rightarrow \mathbf{z}$ )

$$z_i = \sigma_i(\mathbf{x}_{1:i-1}) \cdot x_i + \mu_i(\mathbf{x}_{1:i-1}).$$

$$x_i = (z_i - \mu_i(\mathbf{x}_{1:i-1})) \cdot \frac{1}{\sigma_i(\mathbf{x}_{1:i-1})}.$$

This process is **not** sequential.

## Inverse autoregressive flow (IAF)

### Inverse transform ( $\mathbf{x} \rightarrow \mathbf{z}$ )

$$z_i = \sigma_i(\mathbf{x}_{1:i-1}) \cdot x_i + \mu_i(\mathbf{x}_{1:i-1}).$$

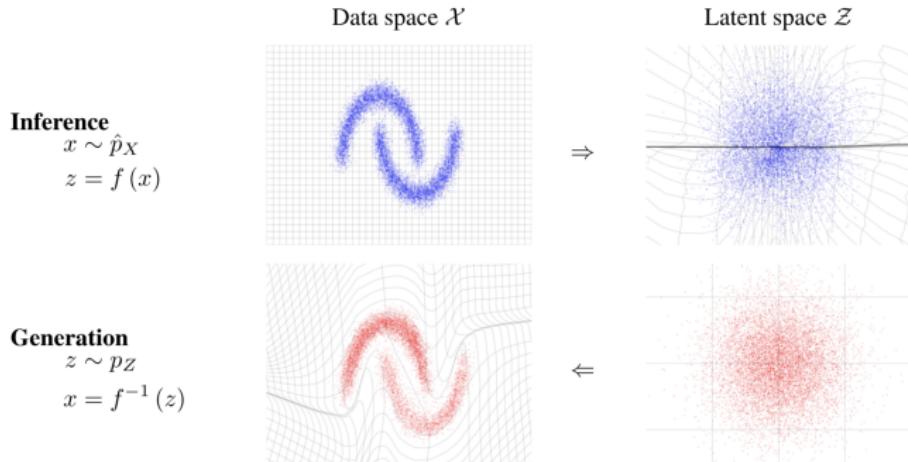
$$x_i = (z_i - \mu_i(\mathbf{x}_{1:i-1})) \cdot \frac{1}{\sigma_i(\mathbf{x}_{1:i-1})}.$$

Inverse autoregressive flow use such inverted autoregressive model as a flow in VAE:

$$\mathbf{z}_0 = \boldsymbol{\sigma}(\mathbf{x}) \cdot \epsilon + \boldsymbol{\mu}(\mathbf{x}), \quad \epsilon \sim \mathcal{N}(0, 1); \quad \sim q(\mathbf{z}_0 | \mathbf{x}, \boldsymbol{\phi}).$$

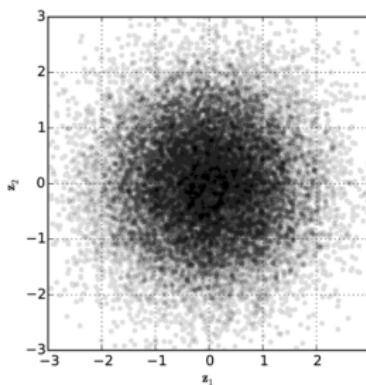
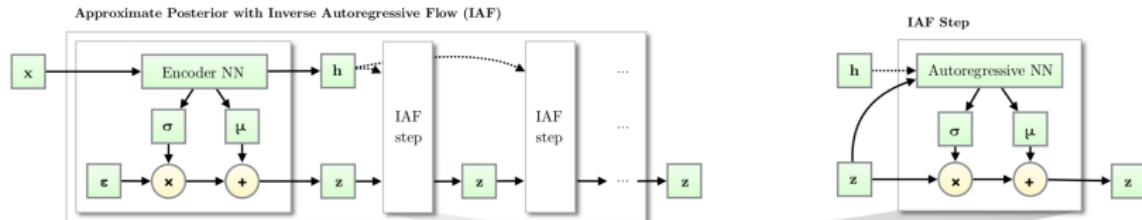
$$\mathbf{z}_k = \boldsymbol{\sigma}_k(\mathbf{z}_{k-1}) \cdot \mathbf{z}_{k-1} + \boldsymbol{\mu}_k(\mathbf{z}_{k-1}), \quad k \geq 1; \quad \sim q_k(\mathbf{z}_k | \mathbf{x}, \boldsymbol{\phi}, \{\boldsymbol{\phi}_j\}_{j=1}^k).$$

# Flows

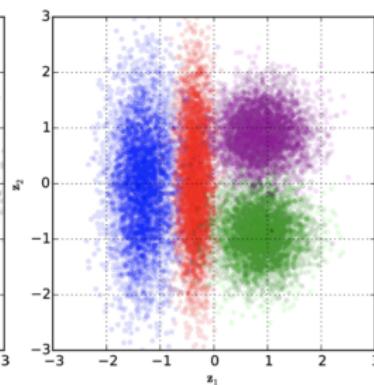


- ▶ Inference mode in autoregressive flows is used for density estimation tasks.
- ▶ Generation mode in autoregressive flows (IAF) is used for stochastic variational inference to get a more flexible posterior distribution.

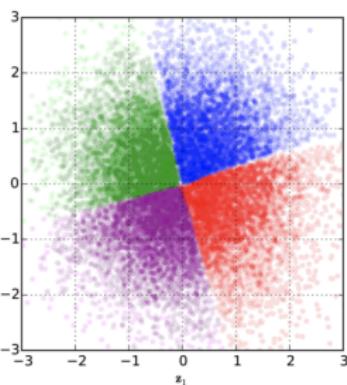
# Inverse autoregressive flow (IAF)



(a) Prior distribution



(b) Posteriors in standard VAE



(c) Posteriors in VAE with IAF

# Summary