# Deep Generative Models

## Lecture supplementary

Roman Isachenko

Moscow Institute of Physics and Technology

2024, Autumn

# Outline I

1. Autoregressive models
   - Masked Autoencoder (MADE)
   - WaveNet
   - GatedPixelCNN
2. Variational inference
   - ELBO gradient, Log derivative trick
   - Mean field approximation
3. VAE-related topics
   - Posterior collapse and decoder weakening techniques
   - IWAE
   - PixelVAE, Hierarchical VAE
4. Normalizing Flows
   - Flows intuition
   - Residual Flows (planar flows)
   - Autoregressive flows
   - Inverse gaussian autoregressive flows

# Outline II

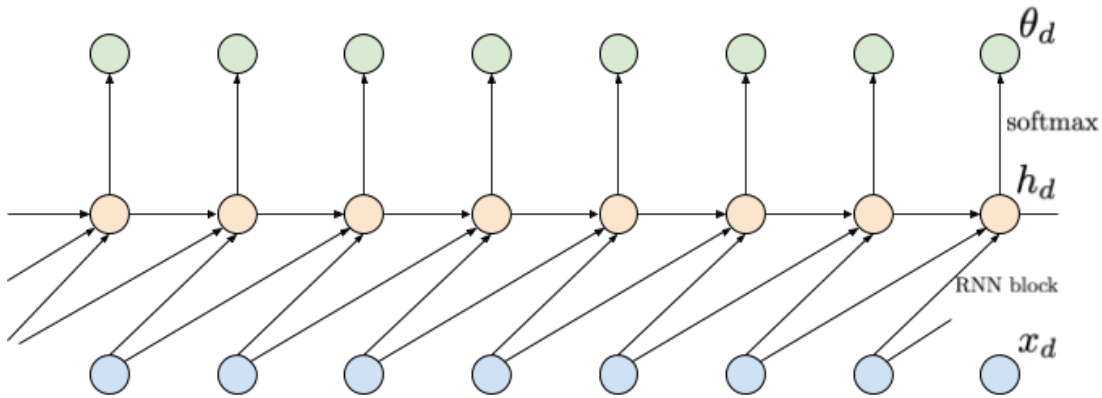
# Outline III

# Outline

# Autoregressive models

- Previous model has **limited** memory $d$. It is insufficient for many modalities (e.g. for images and text).
- Recurrent NN fixes this problem and potentially could learn long-range dependencies:

$$p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}) = p(x_j|\mathbf{h}_j, \boldsymbol{\theta}), \quad \mathbf{h}_j = \text{RNN}(\mathbf{x}_{j-d:j-1}, \mathbf{h}_{j-1})$$



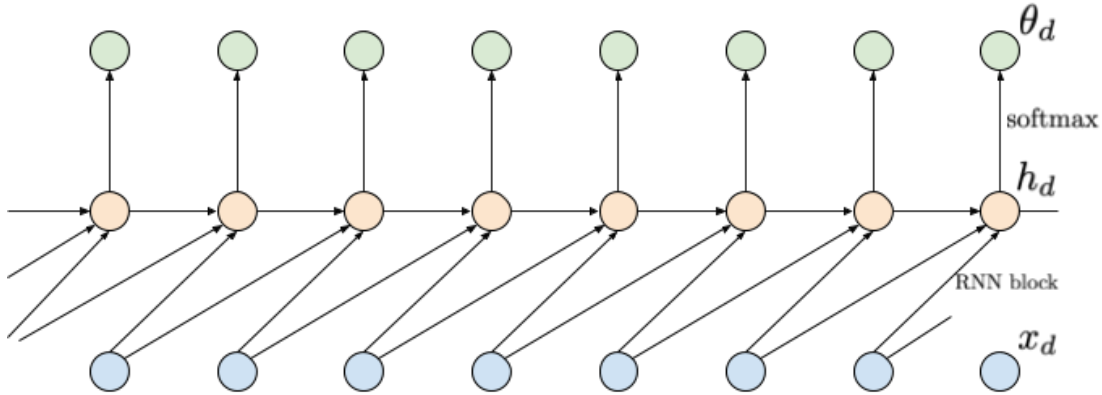

- Sequential computation of all conditionals $p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta})$, hence, the training is slow.
- RNN suffers from vanishing and exploding gradients.

*image credit: https://jmtomczak.github.io/blog/2/2_ARM.html*

# Autoregressive models: RNN

- Previous model has **limited** memory $d$. It is insufficient for many modalities (e.g. for images and text).
- Recurrent NN fixes this problem and potentially could learn long-range dependencies:

$$p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}) = p(x_j|\mathbf{h}_j, \boldsymbol{\theta}), \quad \mathbf{h}_j = \text{RNN}(\mathbf{x}_{j-d:j-1}, \mathbf{h}_{j-1})$$



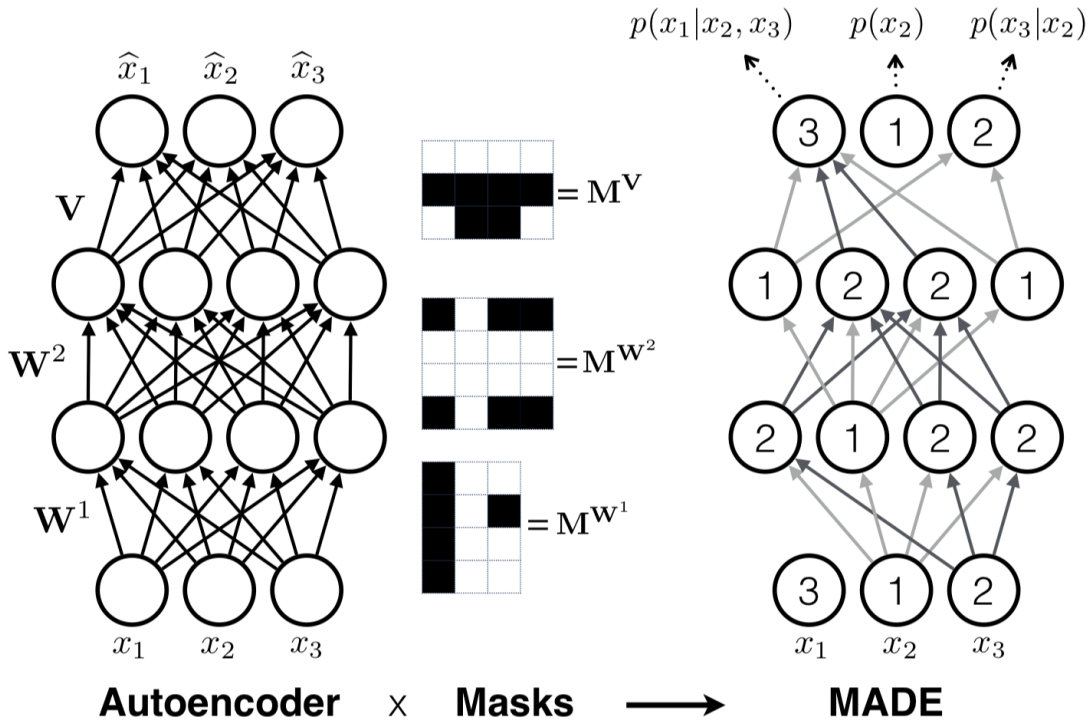

- Sequential computation of all conditionals $p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta})$, hence, the training is slow.
- RNN suffers from vanishing and exploding gradients.

*image credit: https://jmtomczak.github.io/blog/2/2_ARM.html*

# Outline

# MADE

- Vanila autoencoder is not a generative model.
- Let mask the weight matrices to make the model generative: $\mathbf{W}_M = \mathbf{W} \cdot \mathbf{M}$.




- The question is how to create matrices $\mathbf{M}$ which produce the autoregressive property?

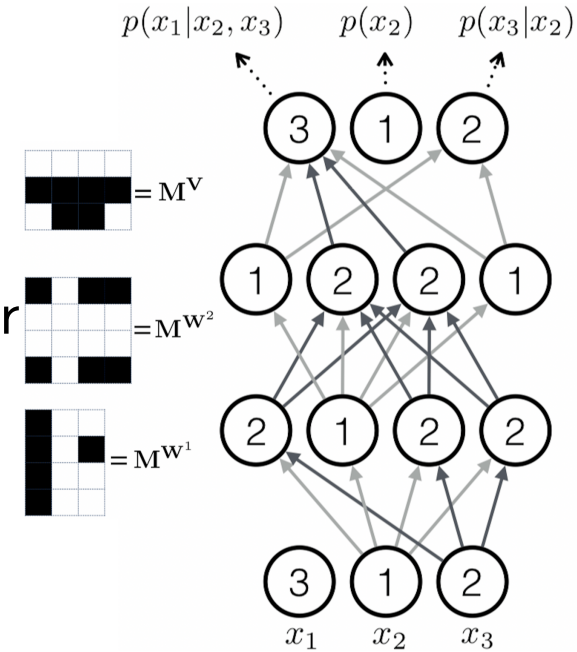
---

*Germain M. et al. Made: Masked autoencoder for distribution estimation, 2015*

# MADE

## Masks generation

- Define the ordering of input elements from 1 to $m$.
- Assign the random number $k$ from 1 to $m-1$ to each hidden unit. The number gives the maximum value of input units to which the unit can be connected.
- Connect each hidden unit with number $k$ with the previous layer units which has the number is **less or equal** than $k$.
- Connect each output unit with number $k$ with the previous layer units which has the number is **less** than $k$.

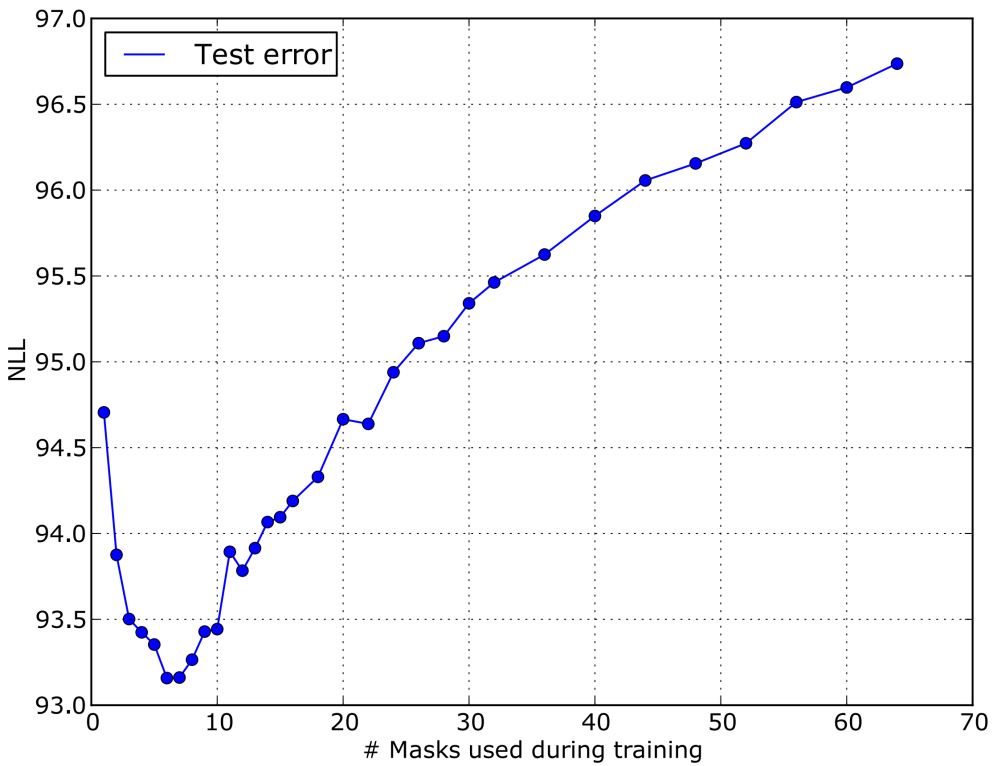


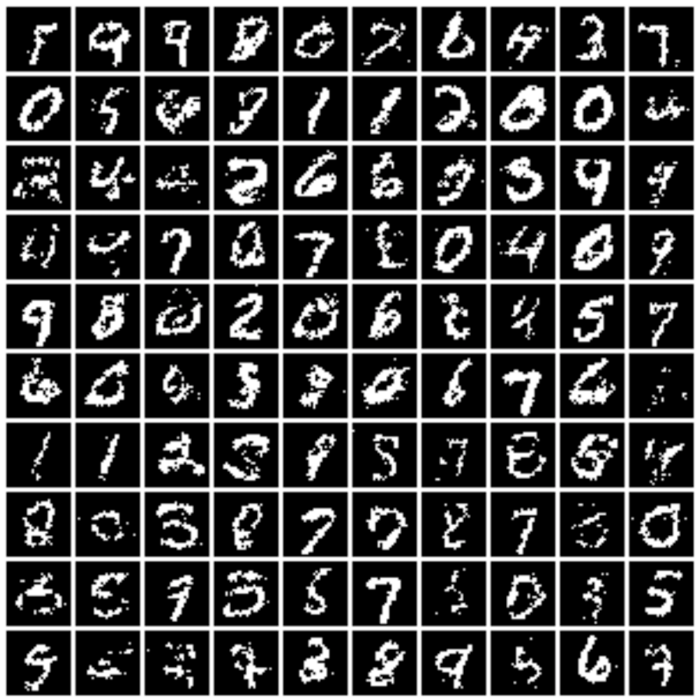

*Germain M. et al. Made: Masked autoencoder for distribution estimation, 2015*

# MADE

## Possible variations

- Order agnostic training (missing values in partially observed input vectors can be imputed efficiently);
- Connectivity-agnostic training (cheap ensembling).



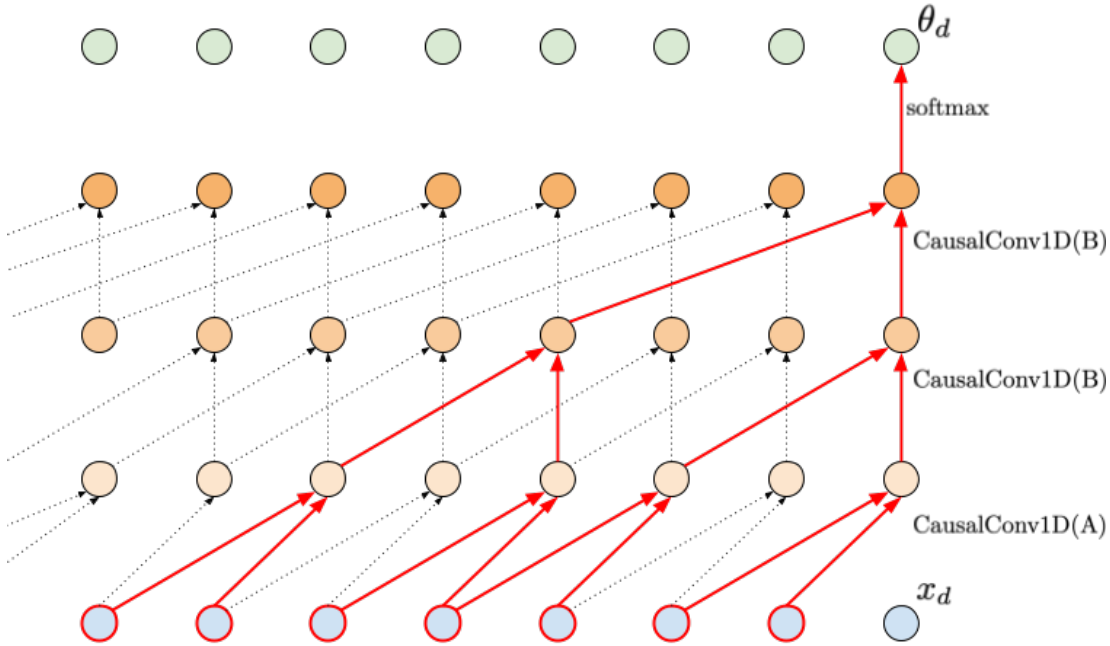



*Germain M. et al. Made: Masked autoencoder for distribution estimation, 2015*

# Outline

# Autoregressive models

- Convolutions could be used for autoregressive models, but they have to be **causal**.
- Try to find and understand the difference between Conv A/B.




- Could learn long-range dependecies.
- Do not suffer from gradient issues.
- Easy to estimate probability for given input, but hard generation of new samples (the sequential process).

*image credit: https://jmtomczak.github.io/blog/2/2_ARM.html*

# WaveNet

## Goal

Efficient generation of raw audio waveforms with natural sounds.

## Solution

Autoregressive model

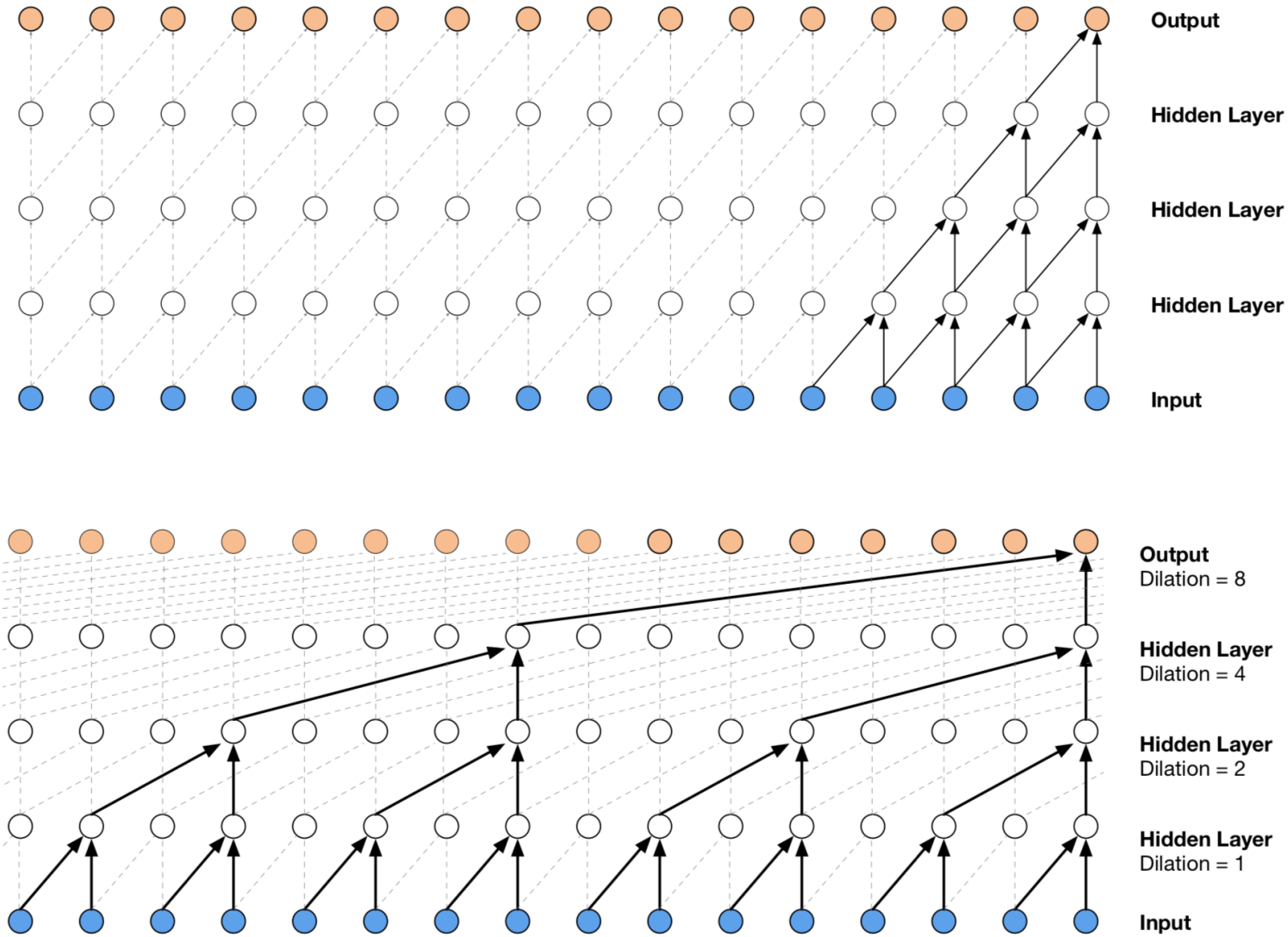
$$p(\mathbf{x}|\theta) = \prod_{t=1}^{T} p(x_t|\mathbf{x}_{1:t-1}, \theta).$$

- Each conditional $p(x_t|\mathbf{x}_{1:t-1}, \theta)$ models the distribution for the timestamp $t$.
- The model uses **causal** dilated convolutions.

---

*Oord A. et al. Wavenet: A generative model for raw audio, 2016*

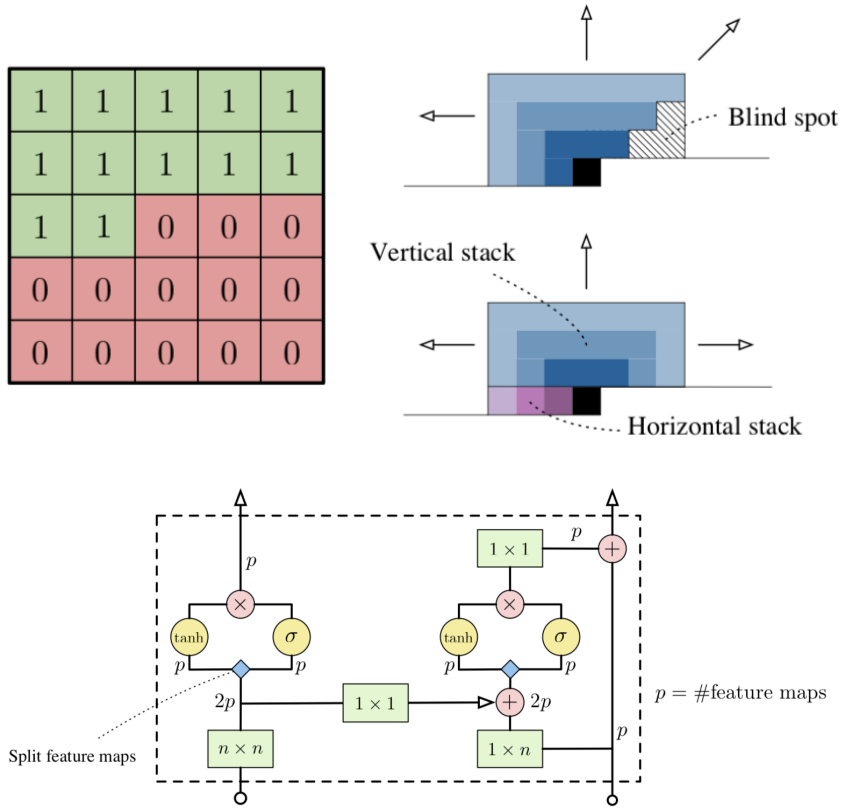
# WaveNet




*Oord A. et al. Wavenet: A generative model for raw audio, 2016*

# Outline

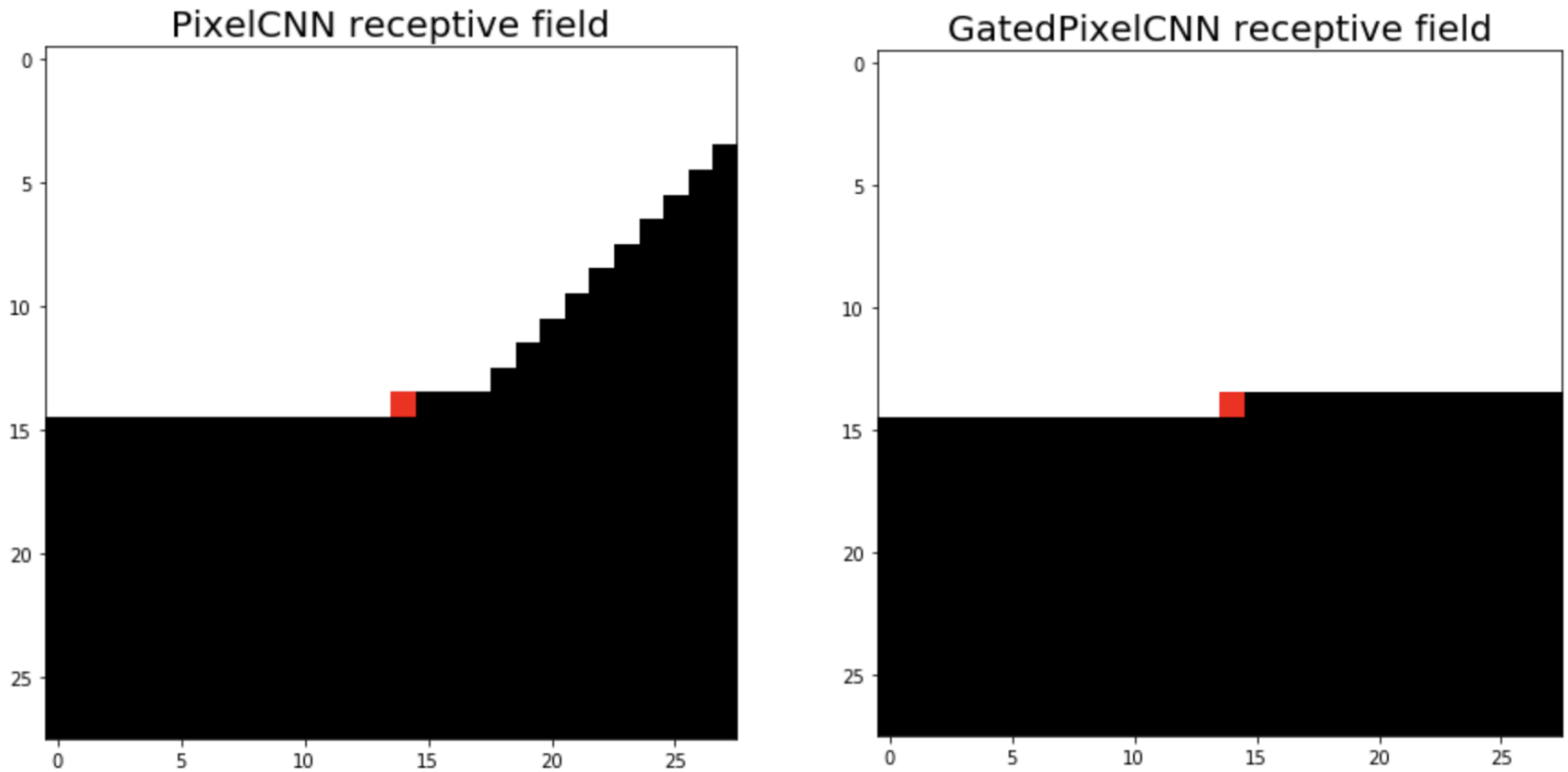
# GatedPixelCNN (2016)




Van den Oord A. et al. Conditional image generation with pixelcnn decoders

https://arxiv.org/pdf/1606.05328.pdf

# GatedPixelCNN (2016)




Van den Oord A. et al. Conditional image generation with pixelcnn decoders

https://arxiv.org/pdf/1606.05328.pdf

# Extensions

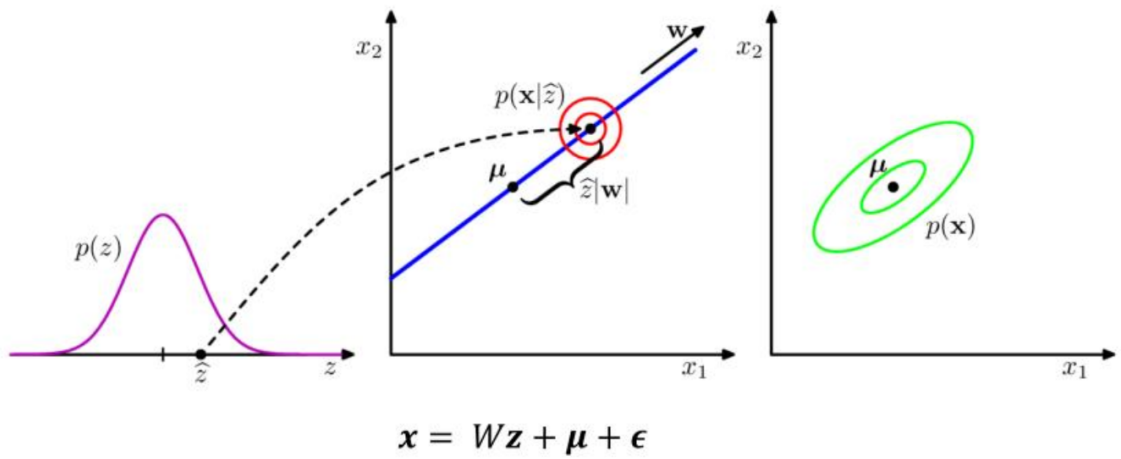
- **PixelCNN++**: *Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications*
  https://arxiv.org/pdf/1701.05517.pdf
  (mixture of logistics instead of softmax);
- **PixelSNAIL**: *An Improved Autoregressive Generative Model*
  https://arxiv.org/pdf/1712.09763.pdf
  (self-attention to learn optimal autoregression ordering).

# Outline

# Latent variable models (LVM)

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log \int p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) p(\mathbf{z}) d\mathbf{z} \to \max_{\boldsymbol{\theta}}$$

**PCA** projects original data $\mathbf{X}$ onto a low dimensional latent space while maximizing the variance of the projected data.




- $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$
- $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, \mathbf{I})$
- $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})$
- $p(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{M}^{-1}\mathbf{W}^T(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M}\right)$, where $\mathbf{M} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$

# Outline

# ELBO gradient (E-step, $\nabla_\phi \mathcal{L}(\phi, \theta)$)

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_q \log p(\mathbf{X}|\mathbf{Z}, \theta) - KL(q(\mathbf{Z}|\mathbf{X}, \phi)||p(\mathbf{Z})) \to \max_{\phi, \theta}.$$

Difference from M-step: density function $q(\mathbf{z}|\mathbf{x}, \phi)$ depends on the parameters $\phi$, it is impossible to use Monte-Carlo estimation:

$$\nabla_\phi \mathcal{L}(\phi, \theta) = \int \nabla_\phi q(\mathbf{Z}|\mathbf{X}, \phi) \log p(\mathbf{X}|\mathbf{Z}, \theta) d\mathbf{Z} - \nabla_\phi KL$$

## Log-derivative trick

$$\nabla_\xi q(\eta|\xi) = q(\eta|\xi) \left( \frac{\nabla_\xi q(\eta|\xi)}{q(\eta|\xi)} \right) = q(\eta|\xi) \nabla_\xi \log q(\eta|\xi).$$

$$\nabla_\phi q(\mathbf{Z}|\mathbf{X}, \phi) = q(\mathbf{Z}|\mathbf{X}, \phi) \nabla_\phi \log q(\mathbf{Z}|\mathbf{X}, \phi).$$

# ELBO gradient (E-step, $\nabla_\phi \mathcal{L}(\phi, \theta)$)

$$\nabla_\phi \mathcal{L}(\phi, \theta) = \int \nabla_\phi q(\mathbf{Z}|\mathbf{X}, \phi) \log p(\mathbf{X}|\mathbf{Z}, \theta) d\mathbf{Z} - \nabla_\phi KL =$$

$$= \int q(\mathbf{Z}|\mathbf{X}, \phi) \left[\nabla_\phi \log q(\mathbf{Z}|\mathbf{X}, \phi) \log p(\mathbf{X}|\mathbf{Z}, \theta)\right] d\mathbf{Z} - \nabla_\phi KL$$

After applying log-reparametrization trick, we are able to use Monte-Carlo estimation:

$$\nabla_\phi \mathcal{L}(\phi, \theta) \approx n \nabla_\phi \log q(\mathbf{z}_i^*|\mathbf{x}_i, \phi) \log p(\mathbf{x}_i|\mathbf{z}_i^*, \theta) - \nabla_\phi KL,$$

$$\mathbf{z}_i^* \sim q(\mathbf{z}_i|\mathbf{x}_i, \phi).$$

## Problem

Unstable solution with huge variance.

## Solution

Reparametrization trick

# Variational Autoencoder

Generated images for latent objects $\mathbf{z}$ sampled from prior $\mathcal{N}(0, \mathbf{I})$

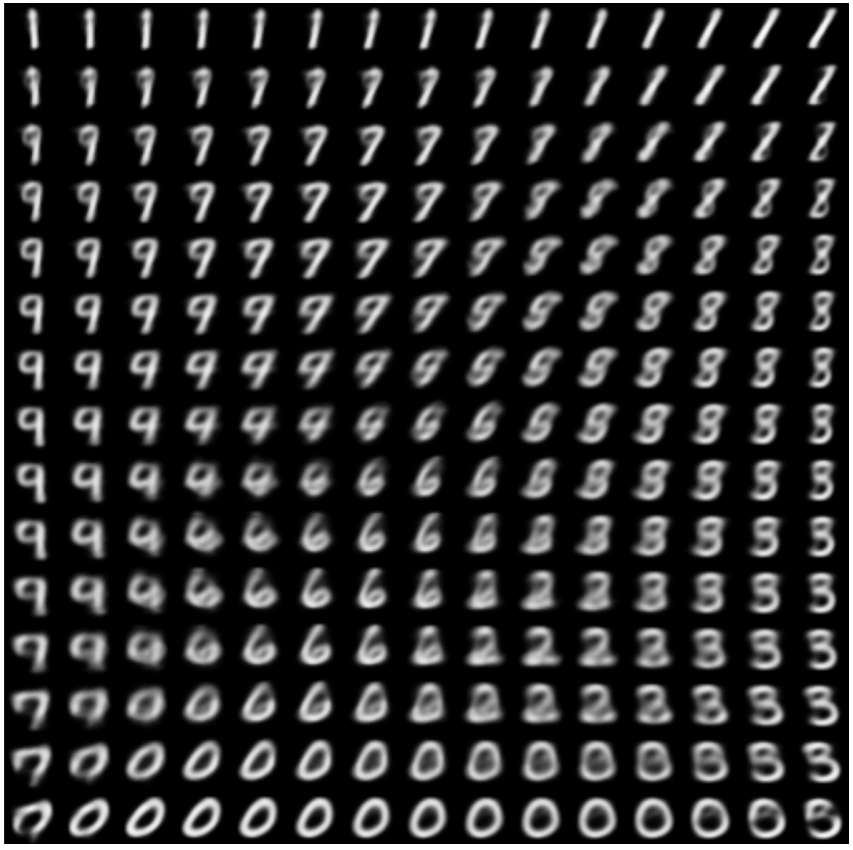


*image credit: https://habr.com/ru/post/331664*

# Outline

# VAE as Bayesian model

## Posterior distribution

$$p(\theta|\mathbf{X}) = \frac{p(\mathbf{X}|\theta)p(\theta)}{p(\mathbf{X})}$$

## ELBO

$$\begin{aligned}\log p(\theta|\mathbf{X}) &= \log p(\mathbf{X}|\theta) + \log p(\theta) - \log p(\mathbf{X}) \\ &= \mathcal{L}(q,\theta) + KL(q||p) + \log p(\theta) - \log p(\mathbf{X}) \\ &\geq [\mathcal{L}(q,\theta) + \log p(\theta)] - \log p(\mathbf{X}).\end{aligned}$$

## EM-algorithm

- E-step

$$q(\mathbf{z}) = \arg\max_q \mathcal{L}(q,\theta^*) = \arg\min_q KL(q||p) = p(\mathbf{z}|\mathbf{x},\theta^*);$$

- M-step

$$\theta^* = \arg\max_\theta \left[\mathcal{L}(q,\theta) + \log p(\theta)\right].$$

# Bayesian framework

## Bayes theorem

$$p(\mathbf{t}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{t})p(\mathbf{t})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{t})p(\mathbf{t})}{\int p(\mathbf{x}|\mathbf{t})p(\mathbf{t})d\mathbf{t}}$$

- $\mathbf{x}$ – observed variables, $\mathbf{t}$ – unobserved variables (latent variables/parameters);
- $p(\mathbf{x}|\mathbf{t})$ – likelihood;
- $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{t})p(\mathbf{t})d\mathbf{t}$ – evidence;
- $p(\mathbf{t})$ – prior distribution, $p(\mathbf{t}|\mathbf{x})$ – posterior distribution.

## Meaning

We have unobserved variables $\mathbf{t}$ and some prior knowledge about them $p(\mathbf{t})$. Then, the data $\mathbf{x}$ has been observed. Posterior distribution $p(\mathbf{t}|\mathbf{x})$ summarizes the knowledge after the observations.

# Variational Lower Bound

We have set of objects $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$. The goal is to perform Bayesian inference on the unobserved variables $\mathbf{T} = \{\mathbf{t}_i\}_{i=1}^n$.

## Evidence Lower Bound (ELBO)

$$\log p(\mathbf{X}) = \log \frac{p(\mathbf{X}, \mathbf{T})}{p(\mathbf{T}|\mathbf{X})} =$$

$$= \int q(\mathbf{T}) \log \frac{p(\mathbf{X}, \mathbf{T})}{p(\mathbf{T}|\mathbf{X})} d\mathbf{T} = \int q(\mathbf{T}) \log \frac{p(\mathbf{X}, \mathbf{T}) q(\mathbf{T})}{p(\mathbf{T}|\mathbf{X}) q(\mathbf{T})} d\mathbf{T} =$$

$$= \int q(\mathbf{T}) \log \frac{p(\mathbf{X}, \mathbf{T})}{q(\mathbf{T})} d\mathbf{T} + \int q(\mathbf{T}) \log \frac{q(\mathbf{T})}{p(\mathbf{T}|\mathbf{X})} d\mathbf{T} =$$

$$= \mathcal{L}(q) + KL(q(\mathbf{T})||p(\mathbf{T}|\mathbf{X})) \geq \mathcal{L}(q).$$

We would like to maximize lower bound $\mathcal{L}(q)$.

# Mean field approximation

## Independence assumption

$$q(\mathbf{T}) = \prod_{i=1}^{k} q_i(\mathbf{T}_i), \quad \mathbf{T} = [\mathbf{T}_1, \ldots, \mathbf{T}_k],\ \mathbf{T}_j = \{\mathbf{t}_{ij}\}_{i=1}^{n},\ \mathbf{t}_i = \{\mathbf{T}_{ij}\}_{j=1}^{k}.$$

## Block coordinate optimization of ELBO for $q_j(\mathbf{T}_j)$

$$\mathcal{L}(q) = \int q(\mathbf{T}) \log \frac{p(\mathbf{X}, \mathbf{T})}{q(\mathbf{T})} d\mathbf{T} = \int \left[\prod_{i=1}^{k} q_i(\mathbf{T}_i)\right] \log \frac{p(\mathbf{X}, \mathbf{T})}{\left[\prod_{i=1}^{k} q_i(\mathbf{T}_i)\right]} \prod_{i=1}^{k} d\mathbf{T}_i =$$

$$= \int \left[\prod_{i=1}^{k} q_i\right] \log p(\mathbf{X}, \mathbf{T}) \prod_{i=1}^{k} d\mathbf{T}_i - \sum_{i=1}^{k} \int \left[\prod_{j=1}^{k} q_j\right] \log q_i \prod_{j=1}^{k} d\mathbf{T}_j =$$

$$= \int q_j \left[\int \log p(\mathbf{X}, \mathbf{T}) \prod_{i \neq j} q_i d\mathbf{T}_i\right] d\mathbf{T}_j -$$

$$- \int q_j \log q_j d\mathbf{T}_j + \text{const}(q_j) \to \max_{q_j}$$

# Mean field approximation

## Block coordinate optimization of ELBO for $q_j(\mathbf{T}_j)$

$$\mathcal{L}(q) = \int q_j \left[ \int \log p(\mathbf{X}, \mathbf{T}) \prod_{i \neq j} q_i d\mathbf{T}_i \right] d\mathbf{T}_j - \int q_j \log q_j d\mathbf{T}_j + \text{const}(q_j) =$$

$$= \int q_j \log \hat{p}(\mathbf{X}, \mathbf{T}_j) d\mathbf{T}_j - \int q_j \log q_j d\mathbf{T}_j + \text{const}(q_j) \to \max_{q_j}.$$

Here we introduce

$$\log \hat{p}(\mathbf{X}, \mathbf{T}_j) = \int \log p(\mathbf{X}, \mathbf{T}) \prod_{i \neq j} q_i d\mathbf{T}_i = \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const}(q_j)$$

## Final ELBO derivation for $q_j(\mathbf{T}_j)$

$$\mathcal{L}(q) = \int q_j(\mathbf{T}_j) \log \hat{p}(\mathbf{X}, \mathbf{T}_j) d\mathbf{T}_j - \int q_j(\mathbf{T}_j) \log q_j(\mathbf{T}_j) d\mathbf{T}_j + \text{const}(q_j) =$$

$$\int q_j(\mathbf{T}_j) \log \frac{\hat{p}(\mathbf{X}, \mathbf{T}_j)}{q_j(\mathbf{T}_j)} d\mathbf{T}_j + \text{const}(q_j) =$$

$$= -KL(q_j(\mathbf{T}_j) || \hat{p}(\mathbf{X}, \mathbf{T}_j)) + \text{const}(q_j) \to \max_{q_j}.$$

# Mean field approximation

## Independence assumption

$$q(\mathbf{T}) = \prod_{i=1}^{k} q_i(\mathbf{T}_i), \quad \mathbf{T} = [\mathbf{T}_1, \ldots, \mathbf{T}_k], \quad \mathbf{T}_j = \{\mathbf{t}_{ij}\}_{i=1}^{n}.$$

## ELBO

$$\mathcal{L}(q) = -KL(q_j(\mathbf{T}_j)||\hat{p}(\mathbf{X}, \mathbf{T}_j)) + \text{const}(q_j) \to \max_{q_j}.$$

## Solution

$$q_j(\mathbf{T}_j) = \text{const} \cdot \hat{p}(\mathbf{X}, \mathbf{T}_j)$$

$$\log \hat{p}(\mathbf{X}, \mathbf{T}_j) = \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const}$$

$$\log q_j(\mathbf{T}_j) = \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const}$$

# Mean field approximation

## ELBO

$$\mathcal{L}(q) = -KL(q_j(\mathbf{T}_j)||\hat{p}(\mathbf{X}, \mathbf{T}_j)) + \text{const}(q_j) \to \max_{q_j}.$$

## Solution

$$\log q_j(\mathbf{T}_j) = \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const}$$

Assumptions:

- $\mathbf{T} = [\mathbf{T}_1, \mathbf{T}_2] = [\mathbf{Z}, \boldsymbol{\theta}]$, $q(\mathbf{T}) = q(\mathbf{T}_1) \cdot q(\mathbf{T}_2) = q(\mathbf{Z}) \cdot q(\boldsymbol{\theta})$.
- restrict a class of probability distributions for $\boldsymbol{\theta}$ to Dirac delta functions:

  $$q_2 = q(\mathbf{T}_2) = q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*).$$

  Under the restrictions the exact solution for $q_2$ is not reached (KL can be greater than 0).

# Mean field approximation

## General solution

$$\log q_j(\mathbf{T}_j) = \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const}$$

## Solution for $q_1 = q(\mathbf{Z})$

$$\log q(\mathbf{Z}) = \int q(\boldsymbol{\theta}) \log p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\theta}) d\boldsymbol{\theta} + \text{const} =$$

$$= \int \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \log p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\theta}) d\boldsymbol{\theta} + \text{const} =$$

$$= \log p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^*) + \text{const}.$$

## EM-algorithm (E-step)

$$q(\mathbf{Z}) = \arg\max_q \mathcal{L}(q, \boldsymbol{\theta}^*) = \arg\min_q KL(q||p) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^*).$$

# Mean field approximation

## ELBO

$$\mathcal{L}(q) = -KL(q_j(\mathbf{T}_j)||\hat{p}(\mathbf{X}, \mathbf{T}_j)) + \text{const}(q_j) \to \max_{q_j}.$$

## ELBO maximization w.r.t. $q_2 = q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$

$$\begin{aligned}
\mathcal{L}(q_1, q_2) &= -KL(q(\boldsymbol{\theta})||\hat{p}(\mathbf{X}, \boldsymbol{\theta})) + \text{const}(\boldsymbol{\theta}^*) \\
&= \int q(\boldsymbol{\theta}) \log \frac{\hat{p}(\mathbf{X}, \boldsymbol{\theta})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} + \text{const}(\boldsymbol{\theta}^*) \\
&= \int q(\boldsymbol{\theta}) \log \hat{p}(\mathbf{X}, \boldsymbol{\theta}) d\boldsymbol{\theta} - \int q(\boldsymbol{\theta}) \log q(\boldsymbol{\theta}) d\boldsymbol{\theta} + \text{const}(\boldsymbol{\theta}^*) \\
&= \int \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \log \hat{p}(\mathbf{X}, \boldsymbol{\theta}) d\boldsymbol{\theta} + \text{const}(\boldsymbol{\theta}^*) \to \max_{\boldsymbol{\theta}^*}
\end{aligned}$$

# Mean field approximation

## ELBO maximization w.r.t. $q_2 = q(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*)$

$$\mathcal{L}(q_1, q_2) = \int \delta(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \log \hat{p}(\mathbf{X}, \boldsymbol{\theta}) d\boldsymbol{\theta} + \text{const} = \log \hat{p}(\mathbf{X}, \boldsymbol{\theta}^*) + \text{const}$$

$$= \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const} = \mathbb{E}_{q_1} \log p(\mathbf{X}, \mathbf{Z}, \boldsymbol{\theta}^*) + \text{const}$$

$$= \int q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta}^*) d\mathbf{Z} + \log p(\boldsymbol{\theta}^*) + \text{const} \to \max_{\boldsymbol{\theta}^*}$$

## EM-algorithm (M-step)

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

$$= \int q(\mathbf{Z}) \log p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\theta}) d\mathbf{Z} + \text{const} \to \max_{\boldsymbol{\theta}}$$

# Mean field approximation

## Solution

$$\log q_j(\mathbf{T}_j) = \mathbb{E}_{i \neq j} \log p(\mathbf{X}, \mathbf{T}) + \text{const}$$

## EM algorithm (special case)

- Initialize $\boldsymbol{\theta}^*$;
- E-step

$$q(\mathbf{Z}) = \arg\max_q \mathcal{L}(q, \boldsymbol{\theta}^*) = \arg\min_q KL(q||p) = p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^*);$$

- M-step

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(q, \boldsymbol{\theta});$$

- Repeat E-step and M-step until convergence.

# Outline

# Outline

# VAE limitations

- **Poor generative distribution (decoder)**

$$p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\mathbf{z})) \quad \text{or} \quad = \text{Softmax}(\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z})).$$

- Loose lower bound

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

- Poor prior distribution

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

- Poor variational posterior distribution (encoder)

$$q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\mathbf{x})).$$

# Posterior collapse: toy example

Let define latent variable model in the following way:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z} = \int p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) p(\mathbf{z}) d\mathbf{z}$$

- prior distribution $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, \mathbf{I})$;
- probabilistic model $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\sigma}_{\boldsymbol{\theta}}(\mathbf{z}))$ (diagonal covariance);
- variational posterior $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \boldsymbol{\sigma}_{\phi}(\mathbf{x}))$ (diagonal covariance).

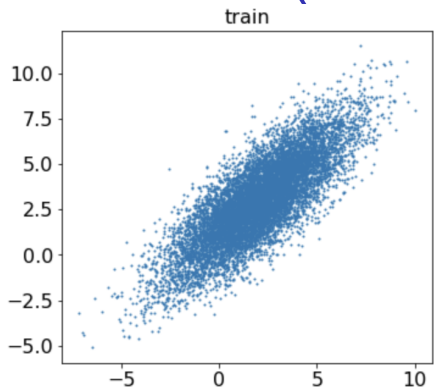
Let data distribution is $\pi(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Possible cases:

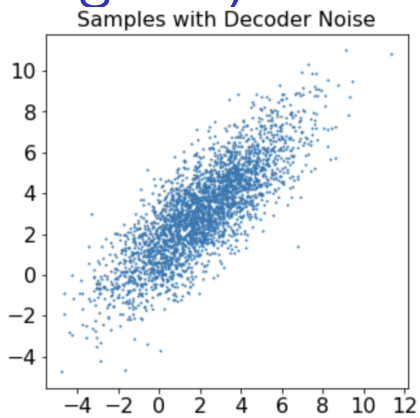
- covariance matrix $\boldsymbol{\Sigma}$ is diagonal (univariate case);
- covariance matrix $\boldsymbol{\Sigma}$ is **not** diagonal (multivariate case).

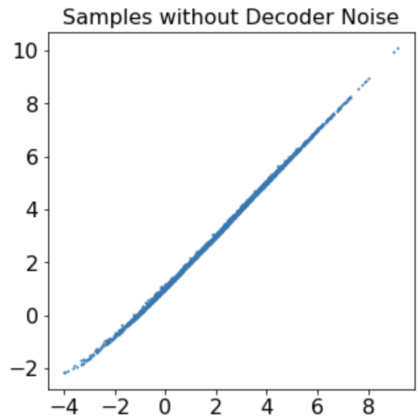
What is the difference?

# Posterior collapse: toy example + VLAE

## Multivariate ($\mathbf{\Sigma}$ is non-diagonal)

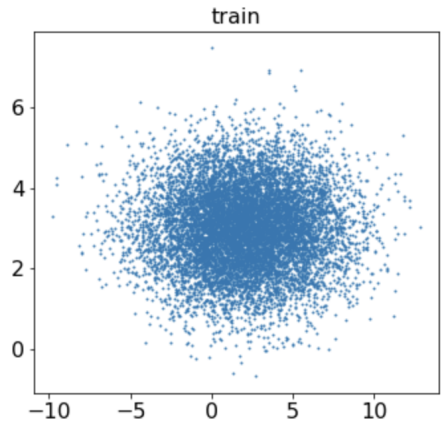


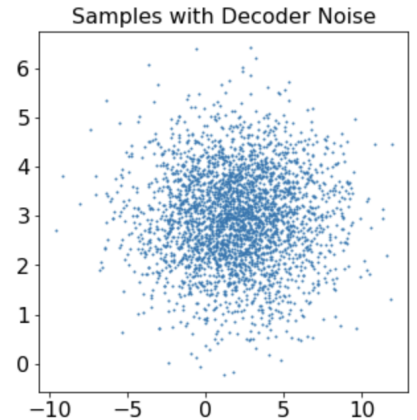



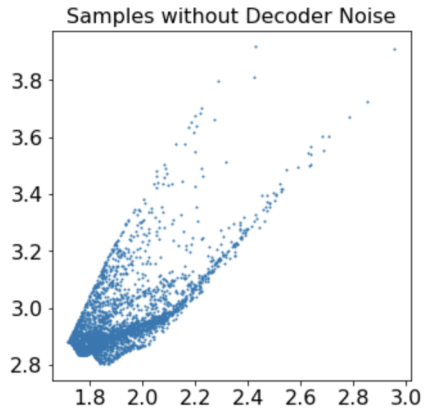




The encoder uses latent variables to model data.

## Univariate ($\mathbf{\Sigma}$ is diagonal)




Latent variables are not used, since the decoder could model the data without the encoder.

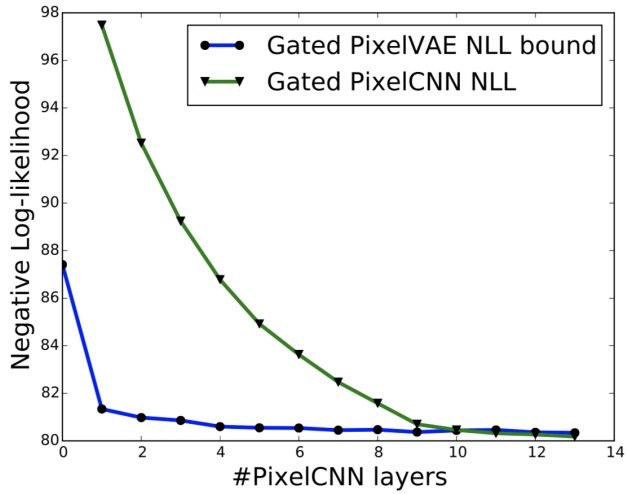
# PixelVAE

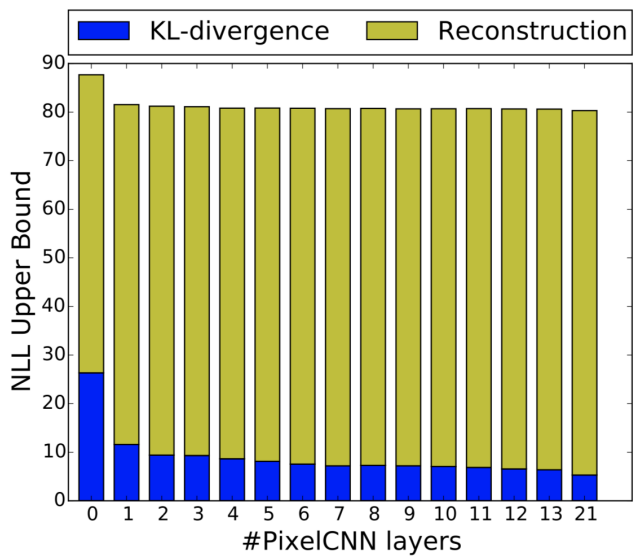
## Autoregressive decoder

$$p(\mathbf{x}|\mathbf{z},\theta) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1},\mathbf{z},\theta)$$

- Global structure is captured by latent variables.
- Local statistics are captured by limited receptive field autoregressive model.

## MNIST results




*Gulrajani I. et al. PixelVAE: A Latent Variable Model for Natural Images, 2016*

# Variational Lossy AutoEncoder

## Lossy code via explicit information placement

$$p(\mathbf{x}|\mathbf{z},\theta) = \prod_{i=1}^{m} p(x_i|\mathbf{z}, \mathbf{x}_{\text{WindowAround}(i)}, \theta).$$

- WindowAround($i$) restricts the receptive field (it forbids to represent arbitrarily complex distribution over $\mathbf{x}$ without dependence on $\mathbf{z}$).
- Local statistics of 2D images (texture) will be modeled by a small local window.
- Global structural information (shapes) is long-range dependency that can only be communicated through latent code $\mathbf{z}$.

---

*Chen X. et al. Variational Lossy Autoencoder, 2016*

# Variational Lossy AutoEncoder

- Can VLAE learn lossy codes that encode global statistics?
- Does using AF priors improves upon using IAF posteriors as predicted by theory?
- Does using autoregressive decoding distributions improve density estimation performance?

MNIST

| **Model** | **NLL Test** |
|---|---|
| Normalizing flows (Rezende & Mohamed, 2015) | 85.10 |
| DRAW (Gregor et al., 2015) | $< 80.97$ |
| Discrete VAE (Rolfe, 2016) | 81.01 |
| PixelRNN (van den Oord et al., 2016a) | 79.20 |
| IAF VAE (Kingma et al., 2016) | 79.88 |
| AF VAE | 79.30 |
| VLAE | **79.03** |

CIFAR10

| Method | bits/dim $\leq$ |
|---|---|
| *Results with tractable likelihood models*: | |
| Uniform distribution [1] | 8.00 |
| Multivariate Gaussian [1] | 4.70 |
| NICE [2] | 4.48 |
| Deep GMMs [3] | 4.00 |
| Real NVP [4] | 3.49 |
| PixelCNN [1] | 3.14 |
| Gated PixelCNN [5] | 3.03 |
| PixelRNN [1] | 3.00 |
| PixelCNN++ [6] | **2.92** |
| *Results with variationally trained latent-variable models*: | |
| Deep Diffusion [7] | 5.40 |
| Convolutional DRAW [8] | 3.58 |
| ResNet VAE with IAF [9] | 3.11 |
| ResNet VLAE | 3.04 |
| DenseNet VLAE | **2.95** |

*Chen X. et al. Variational Lossy Autoencoder, 2016*

# Posterior collapse

## LVM

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z} = \int p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) p(\mathbf{z}) d\mathbf{z}$$

## ELBO objective

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z})).$$

More powerful $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$ leads to more powerful generative model $p(\mathbf{x}|\boldsymbol{\theta})$.

## Extreme cast

$$p(\mathbf{x}|\boldsymbol{\theta}) \in \mathcal{P} = \{p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) | \, \forall \mathbf{z}, \boldsymbol{\theta}\}.$$

If the decoder $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$ is too powerful (it could model $p(\mathbf{x}|\boldsymbol{\theta})$), then ELBO avoids paying any cost $KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}))$ ($q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) \approx p(\mathbf{z})$), the variational posterior $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$ will not carry any information about $\mathbf{x}$, the latent variables $\mathbf{z}$ becomes irrelevant.

# Autoregressive VAE decoder

How to make the generative model $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$ more powerful?

## PixelVAE/VLAE

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

- Global structure is captured by latent variables $\mathbf{z}$.
- Local statistics are captured by limited receptive field of autoregressive context $\mathbf{x}_{1:j-1}$.

PixelVAE/VLAE models use the autoregressive PixelCNN decoder model with small number of layers to limit receptive field.

---

*Gulrajani I. et al. PixelVAE: A Latent Variable Model for Natural Images, 2016,*
*Chen X. et al. Variational Lossy Autoencoder, 2016*

# Decoder weakening techniques

How to force the model encode information about $\mathbf{x}$ into $\mathbf{z}$?

## KL annealing

$$\mathcal{L}(\phi, \theta, \beta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log p(\mathbf{x}|\mathbf{z}, \theta) - \beta \cdot KL(q(\mathbf{z}|\mathbf{x}, \phi)||p(\mathbf{z}))$$

Start training with $\beta = 0$, increase it until $\beta = 1$ during training.

## Free bits

$$\mathcal{L}(\phi, \theta, \lambda) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log p(\mathbf{x}|\mathbf{z}, \theta) - \max(\lambda, KL(q(\mathbf{z}|\mathbf{x}, \phi)||p(\mathbf{z}))).$$

It ensures the use of less than $\lambda$ bits of information and results in $KL(q(\mathbf{z}|\mathbf{x}, \phi)||p(\mathbf{z})) \geq \lambda$.

---

*Bowman S. R. et al. Generating Sentences from a Continuous Space, 2015*
*Kingma D. P. et al. Improving Variational Inference with Inverse Autoregressive Flow, 2016*

# Outline

# VAE limitations

- Poor generative distribution (decoder)

$$p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}), \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\mathbf{z})) \quad \text{or} \quad = \text{Softmax}(\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{z})).$$

- **Loose lower bound**

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

- Poor prior distribution

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

- Poor variational posterior distribution (encoder)

$$q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{\boldsymbol{\phi}}(\mathbf{x}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\mathbf{x})).$$

# Importance sampling

## LVM

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) d\mathbf{z} = \int \left[\frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})}\right] q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) d\mathbf{z}$$
$$= \int f(\mathbf{x}, \mathbf{z}) q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) d\mathbf{z} = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} f(\mathbf{x}, \mathbf{z})$$

Here $f(\mathbf{x}, \mathbf{z}) = \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})}$.

## ELBO: derivation 1

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

$f(\mathbf{x}, \mathbf{z})$ could be any function that satisfies $p(\mathbf{x}|\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z} \sim q} f(\mathbf{x}, \mathbf{z})$.
Could we choose better $f(\mathbf{x}, \mathbf{z})$?

# Importance Weighted Autoencoders (IWAE)

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int \left[\frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})}\right] q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} f(\mathbf{x}, \mathbf{z})$$

Let define

$$f(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_K) = \frac{1}{K}\sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})}$$

$$\mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K\sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} f(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_K) = p(\mathbf{x}|\boldsymbol{\theta})$$

## ELBO

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \log \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K\sim q(\mathbf{z}|\mathbf{x})} f(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_K) \geq$$

$$\geq \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K\sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log f(\mathbf{x}, \mathbf{z}_1, \ldots, \mathbf{z}_K) =$$

$$= \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K\sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \left[\frac{1}{K}\sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})}\right] = \mathcal{L}_K(q, \boldsymbol{\theta}).$$

---

*Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015*

# Importance Weighted Autoencoders (IWAE)

## VAE objective

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

$$\mathcal{L}(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K \sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \left( \frac{1}{K} \sum_{k=1}^{K} \log \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})} \right) \to \max_{q,\boldsymbol{\theta}}.$$

## IWAE objective

$$\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K \sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})} \right) \to \max_{q,\boldsymbol{\theta}}.$$

If $K = 1$, these objectives coincide.

---

*Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015*

# Importance Weighted Autoencoders (IWAE)

## Theorem

1. $\log p(\mathbf{x}|\boldsymbol{\theta}) \geq \mathcal{L}_K(q, \boldsymbol{\theta}) \geq \mathcal{L}_M(q, \boldsymbol{\theta}), \quad$ for $K \geq M$;
2. $\log p(\mathbf{x}|\boldsymbol{\theta}) = \lim_{K\to\infty} \mathcal{L}_K(q, \boldsymbol{\theta})$ if $\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}$ is bounded.

If $K > 1$ the bound could be tighter.

$$\mathcal{L}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})};$$

$$\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K\sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})} \right).$$

- $\mathcal{L}_1(q, \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta})$;
- $\mathcal{L}_\infty(q, \boldsymbol{\theta}) = \log p(\mathbf{x}|\boldsymbol{\theta})$.
- Which $q^*(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$ gives $\mathcal{L}(q^*, \boldsymbol{\theta}) = \log p(\mathbf{x}|\boldsymbol{\theta})$?

---

*Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015*

# IWAE

## Theorem

1. $\log p(\mathbf{x}|\boldsymbol{\theta}) \geq \mathcal{L}_K(q, \boldsymbol{\theta}) \geq \mathcal{L}_M(q, \boldsymbol{\theta}), \quad \text{for } K \geq M;$
2. $\log p(\mathbf{x}|\boldsymbol{\theta}) = \lim_{K\to\infty} \mathcal{L}_K(q, \boldsymbol{\theta})$ if $\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x})}$ is bounded.

## Proof of 1.

$$\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K} \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x})} \right) =$$

$$= \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K} \log \mathbb{E}_{k_1,\ldots,k_M} \left( \frac{1}{M} \sum_{m=1}^{M} \frac{p(\mathbf{x}, \mathbf{z}_{k_M}|\boldsymbol{\theta})}{q(\mathbf{z}_{k_m}|\mathbf{x})} \right) \geq$$

$$\geq \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K} \mathbb{E}_{k_1,\ldots,k_m} \log \left( \frac{1}{M} \sum_{m=1}^{M} \frac{p(\mathbf{x}, \mathbf{z}_{k_m}|\boldsymbol{\theta})}{q(\mathbf{z}_{k_m}|\mathbf{x})} \right) =$$

$$= \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_M} \log \left( \frac{1}{M} \sum_{m=1}^{M} \frac{p(\mathbf{x}, \mathbf{z}_m|\boldsymbol{\theta})}{q(\mathbf{z}_m|\mathbf{x})} \right) = \mathcal{L}_M(q, \boldsymbol{\theta})$$

$$\frac{a_1 + \cdots + a_K}{K} = \mathbb{E}_{k_1,\ldots,k_M} \frac{a_{k_1} + \cdots + a_{k_M}}{M}, \quad k_1, \ldots, k_M \sim U[1, K]$$

---

*Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015*

# IWAE

## Theorem

1. $\log p(\mathbf{x}|\boldsymbol{\theta}) \geq \mathcal{L}_K(q, \boldsymbol{\theta}) \geq \mathcal{L}_M(q, \boldsymbol{\theta}), \quad \text{for} K \geq M;$
2. $\log p(\mathbf{x}|\boldsymbol{\theta}) = \lim_{K\to\infty} \mathcal{L}_K(q, \boldsymbol{\theta})$ if $\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x})}$ is bounded.

## Proof of 2.

Consider r.v. $\xi_K = \frac{1}{K}\sum_{k=1}^{K} \frac{p(\mathbf{x},\mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x})}$.

If summands are bounded, then (from the strong law of large numbers)

$$\xi_K \xrightarrow[K\to\infty]{a.s.} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x})} = p(\mathbf{x}|\boldsymbol{\theta}).$$

Hence $\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E} \log \xi_K$ converges to $\log p(\mathbf{x}|\boldsymbol{\theta})$ as $K \to \infty$.

---

*Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015*

# Importance Weighted Autoencoders (IWAE)

## Theorem

1. $\log p(\mathbf{x}|\boldsymbol{\theta}) \geq \mathcal{L}_K(q, \boldsymbol{\theta}) \geq \mathcal{L}_M(q, \boldsymbol{\theta}), \quad$ for $K \geq M$;
2. $\log p(\mathbf{x}|\boldsymbol{\theta}) = \lim_{K\to\infty} \mathcal{L}_K(q, \boldsymbol{\theta})$ if $\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})}$ is bounded.

If $K > 1$ the bound could be tighter.

$$\mathcal{L}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})};$$

$$\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K\sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})} \right).$$

- $\mathcal{L}_1(q, \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta})$;
- $\mathcal{L}_\infty(q, \boldsymbol{\theta}) = \log p(\mathbf{x}|\boldsymbol{\theta})$.
- Which $q^*(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$ gives $\mathcal{L}(q^*, \boldsymbol{\theta}) = \log p(\mathbf{x}|\boldsymbol{\theta})$?
- Which $q^*(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$ gives $\mathcal{L}(q^*, \boldsymbol{\theta}) = \mathcal{L}_K(q, \boldsymbol{\theta})$?

---

*Burda Y., Grosse R., Salakhutdinov R. Importance Weighted Autoencoders, 2015*

# Importance Weighted Autoencoders (IWAE)

## Theorem

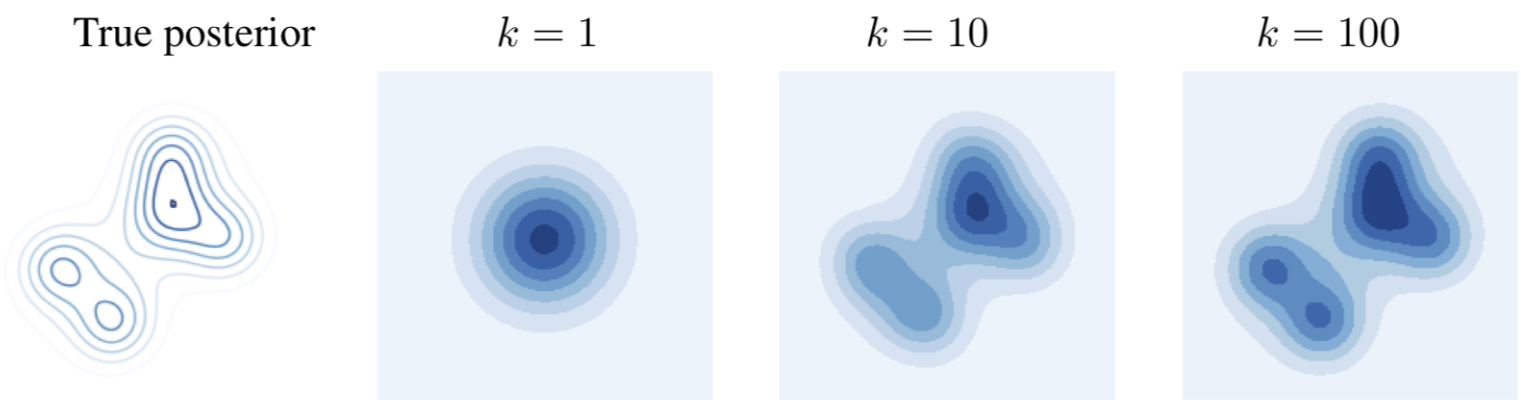
$\mathcal{L}(q^*, \boldsymbol{\theta}) = \mathcal{L}_K(q, \boldsymbol{\theta})$ for the following variational distribution

$$q^*(\mathbf{z}|\mathbf{x}, \phi) = \mathbb{E}_{\mathbf{z}_2,\ldots,\mathbf{z}_K \sim q(\mathbf{z}|\mathbf{x})} q_{IW}(\mathbf{z}|\mathbf{x}, \mathbf{z}_{2:K}),$$

where

$$q_{IW}(\mathbf{z}|\mathbf{x}, \mathbf{z}_{2:K}) = \frac{\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}}{\frac{1}{K}\sum_{k=1}^{K}\frac{p(\mathbf{x},\mathbf{z}_k)}{q(\mathbf{z}_k|\mathbf{x})}} q(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x},\mathbf{z})}{\frac{1}{K}\left(\frac{p(\mathbf{x},\mathbf{z})}{q(\mathbf{z}|\mathbf{x})} + \sum_{k=2}^{K}\frac{p(\mathbf{x},\mathbf{z}_k)}{q(\mathbf{z}_k|\mathbf{x})}\right)}.$$

## IWAE posterior




*Cremer C., Morris Q., Duvenaud D. Reinterpreting Importance-Weighted Autoencoders, 2017*

# IWAE, 2015

How to determine whether all VAE latent variables are informative?

$$A_i = \mathrm{cov}_{\mathbf{x}}\left(\mathbb{E}_{q(z_i|\mathbf{x})}[z_i]\right) > 0.01 \quad \Leftrightarrow \quad z_i \text{ is active}$$

| | | MNIST | | | | OMNIGLOT | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | VAE | | IWAE | | VAE | | IWAE | |
| # stoch. layers | $k$ | NLL | active units | NLL | active units | NLL | active units | NLL | active units |
| 1 | 1 | 86.76 | 19 | 86.76 | 19 | 108.11 | 28 | 108.11 | 28 |
| | 5 | 86.47 | 20 | 85.54 | 22 | 107.62 | 28 | 106.12 | 34 |
| | 50 | 86.35 | 20 | 84.78 | 25 | 107.80 | 28 | 104.67 | 41 |
| 2 | 1 | 85.33 | 16+5 | 85.33 | 16+5 | 107.58 | 28+4 | 107.56 | 30+5 |
| | 5 | 85.01 | 17+5 | 83.89 | 21+5 | 106.31 | 30+5 | 104.79 | 38+6 |
| | 50 | 84.78 | 17+5 | 82.90 | 26+7 | 106.30 | 30+5 | 103.38 | 44+7 |

https://arxiv.org/pdf/1509.00519.pdf

# Importance Weighted Autoencoders (IWAE)

## Objective

$$\mathcal{L}_K(q, \boldsymbol{\theta}) = \mathbb{E}_{\mathbf{z}_1,\ldots,\mathbf{z}_K \sim q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})} \right) \to \max_{\boldsymbol{\phi},\boldsymbol{\theta}}.$$

## Gradient

$$\Delta_K = \nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \log \left( \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathbf{x}, \mathbf{z}_k|\boldsymbol{\theta})}{q(\mathbf{z}_k|\mathbf{x}, \boldsymbol{\phi})} \right), \quad \mathbf{z}_k \sim q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}).$$

## Theorem

$$\text{SNR}_K = \frac{\mathbb{E}[\Delta_K]}{\sigma(\Delta_K)}; \quad \text{SNR}_K(\boldsymbol{\theta}) = O(\sqrt{K}); \quad \text{SNR}_K(\boldsymbol{\phi}) = O\left(\sqrt{\frac{1}{K}}\right).$$

Hence, increasing $K$ vanishes gradient signal of inference network $q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})$.

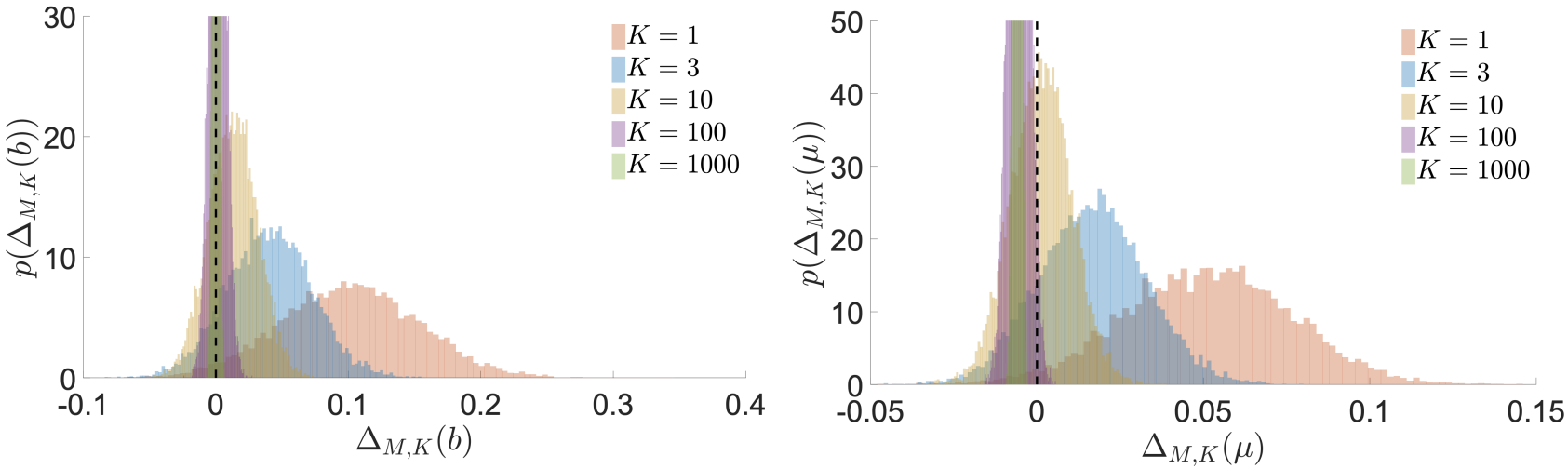
---

*Rainforth T. et al. Tighter variational bounds are not necessarily better, 2018*

# Importance Weighted Autoencoders (IWAE)

## Theorem

$$\mathrm{SNR}_K = \frac{\mathbb{E}[\Delta_K]}{\sigma(\Delta_K)}; \quad \mathrm{SNR}_K(\theta) = O(\sqrt{K}); \quad \mathrm{SNR}_K(\phi) = O\left(\sqrt{\frac{1}{K}}\right).$$




- IWAE makes the variational bound tighter and extends the class of variational distributions.
- Gradient signal becomes really small, training is complicated.
- IWAE is a standard quality measure for VAE models.

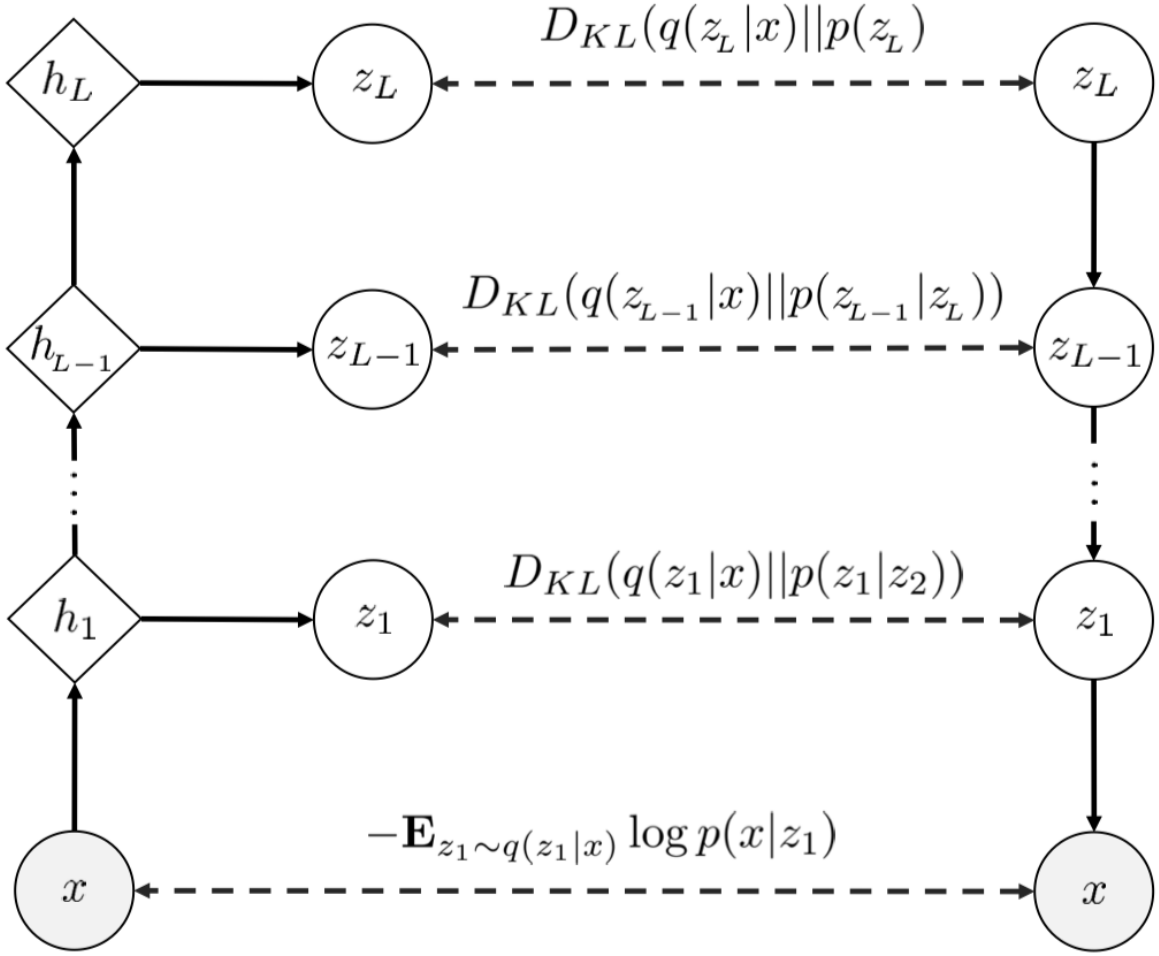
---

*Rainforth T. et al. Tighter variational bounds are not necessarily better, 2018*

# Outline

# PixelVAE, 2016

## Hierarchical VAE




https://arxiv.org/pdf/1611.05013.pdf

# PixelVAE, 2016

## Hierarchical decomposition

$$p(\mathbf{z}_1, \dots, \mathbf{z}_L) = p(\mathbf{z}_L)p(\mathbf{z}_{L-1}|\mathbf{z}_L) \dots p(\mathbf{z}_1, \mathbf{z}_2);$$
$$q(\mathbf{z}_1, \dots, \mathbf{z}_L|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x}) \dots q(\mathbf{z}_L|\mathbf{x}).$$

## ELBO

$$\begin{aligned}
\mathcal{L}(q, \boldsymbol{\theta}) &= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta}) - KL(q(\mathbf{z}_1, \dots, \mathbf{z}_L|\mathbf{x})||p(\mathbf{z}_1, \dots, \mathbf{z}_L)) \\
&= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta}) - \int \prod_{j=1}^{L} q(\mathbf{z}_j|\mathbf{x}) \sum_{i=1}^{L} \log \frac{q(\mathbf{z}_i|\mathbf{x})}{p(\mathbf{z}_i|\mathbf{z}_{i+1})} d\mathbf{z}_1 \dots d\mathbf{z}_L \\
&= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta}) - \sum_{i=1}^{L} \int \prod_{j=1}^{L} q(\mathbf{z}_j|\mathbf{x}) \log \frac{q(\mathbf{z}_i|\mathbf{x})}{p(\mathbf{z}_i|\mathbf{z}_{i+1})} d\mathbf{z}_1 \dots d\mathbf{z}_L \\
&= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta}) - \sum_{i=1}^{L} \int q(\mathbf{z}_{i+1}|\mathbf{x}) q(\mathbf{z}_i|\mathbf{x}) \log \frac{q(\mathbf{z}_i|\mathbf{x})}{p(\mathbf{z}_i|\mathbf{z}_{i+1})} d\mathbf{z}_i d\mathbf{z}_{i+1} \\
&= \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\theta}) - \sum_{i=1}^{L} \mathbb{E}_{q(\mathbf{z}_{i+1}|\mathbf{x})} \left[ KL(q(\mathbf{z}_i|\mathbf{x})||p(\mathbf{z}_i|\mathbf{z}_{i+1})) \right]
\end{aligned}$$

---

https://arxiv.org/pdf/1611.05013.pdf

# Outline

# Outline

# Flows intuition

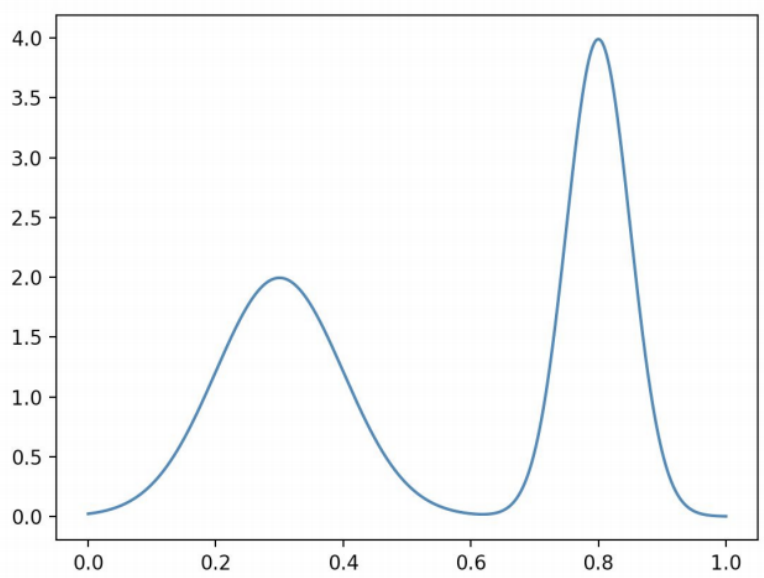
Let $\xi$ be a random variable with density $p(\xi)$. Then

$$\eta = F(\xi) = \int_{-\infty}^{\xi} p(t)dt \sim U[0,1].$$

$$P(\eta < y) = P(F(\xi) < y) = P(\xi < F^{-1}(y)) = F(F^{-1}(y)) = y$$

Hence

$$\eta \sim U[0,1]; \quad \xi = F^{-1}(\eta) \quad \Rightarrow \quad \xi \sim p(\xi).$$



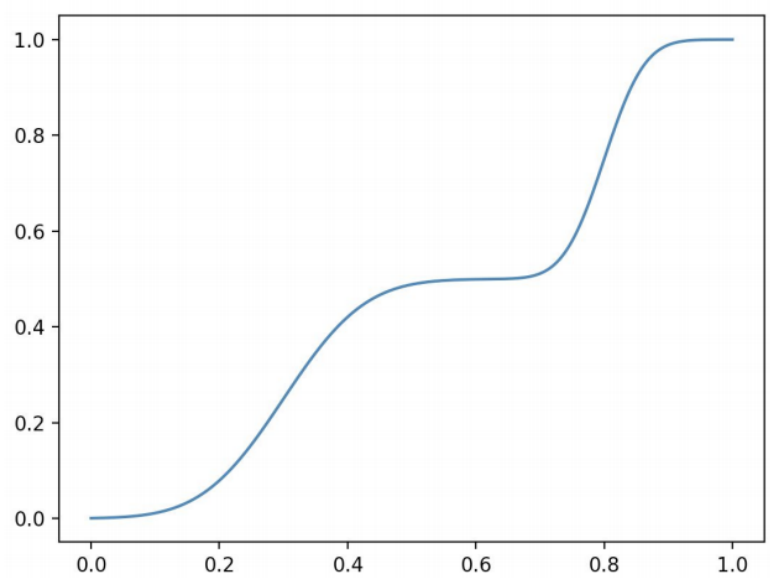




*image credit: https://sites.google.com/view/berkeley-cs294-158-sp19/home*

# Flows intuition

- Let $z \sim p(z)$ is a random variable with base distribution $p(z) = U[0, 1]$.
- Let $x \sim p(x)$ is a random variable with complex distribution $p(x)$ and cdf $F(x)$.
- Then noise variable $z$ can be transformed to $x$ using inverse cdf $F^{-1}$ ($x = F^{-1}(z)$).

How to transform random variable $z$ which has a distribution different from uniform to $x$?

- Let $z \sim p(z)$ is a random variable with base distribution $p(z)$ and cdf $G(z)$.
- Then $z_0 = G(z)$ has base distribution $p(z_0) = U[0, 1]$.
- Let $x \sim p(x)$ is a random variable with complex distribution $p(x)$ and cdf $F(x)$.
- Then noise variable $z$ can be transformed to $x$ using cdf $G$ and inverse cdf $F^{-1}$ ($x = F^{-1}(z_0) = F^{-1}(G(z))$).

# Outline

# Residual Flows

## Matrix determinant lemma

$$\det\left(\mathbf{I}_m + \mathbf{V}\mathbf{W}^T\right) = \det(\mathbf{I}_d + \mathbf{W}^T\mathbf{V}), \quad \text{where } \mathbf{V}, \mathbf{W} \in \mathbb{R}^{m\times d}.$$

## Planar flow

$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) = \mathbf{z} + \mathbf{v}\,\sigma(\mathbf{w}^T\mathbf{z} + b).$$

Here $\boldsymbol{\theta} = \{\mathbf{v}, \mathbf{w}, b\}$, $\sigma(\cdot)$ is a smooth element-wise non-linearity.

$$\left|\det\left(\frac{\partial g(\mathbf{z}, \boldsymbol{\theta})}{\partial \mathbf{z}}\right)\right| = \left|\det\left(\mathbf{I} + \sigma'(\mathbf{w}^T\mathbf{z} + b)\mathbf{v}\mathbf{w}^T\right)\right| = \left|1 + \sigma'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}^T\mathbf{v}\right|$$

The transformation is invertible, for example, if

$$\sigma = \tanh; \quad \sigma'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}^T\mathbf{v} \geq -1.$$

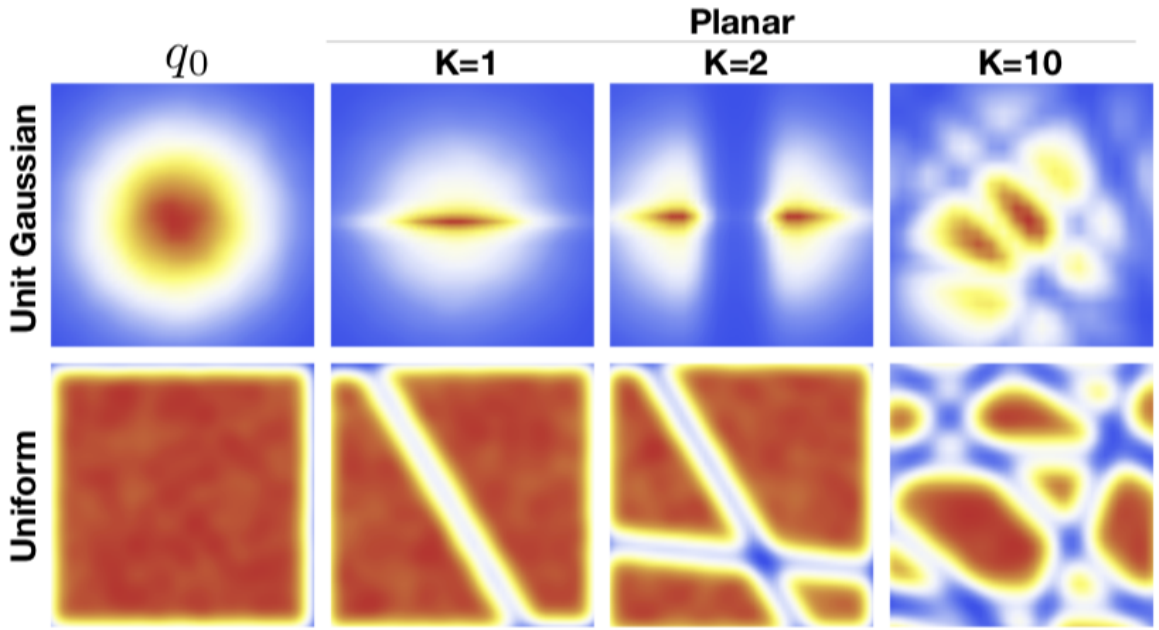
---

*Rezende D. J., Mohamed S. Variational Inference with Normalizing Flows, 2015*

# Residual Flows

## Expressiveness of planar flows

$$\mathbf{z}_K = g_1 \circ \cdots \circ g_K(\mathbf{z}); \quad g_k = g(\mathbf{z}_k, \theta_k) = \mathbf{z}_k + \mathbf{v}_k \, \sigma(\mathbf{w}_k^T \mathbf{z}_k + b_k).$$




## Sylvester flow: planar flow extension

$$g(\mathbf{z}, \theta) = \mathbf{z} + \mathbf{V} \, \sigma(\mathbf{W}^T \mathbf{z} + \mathbf{b}).$$

---

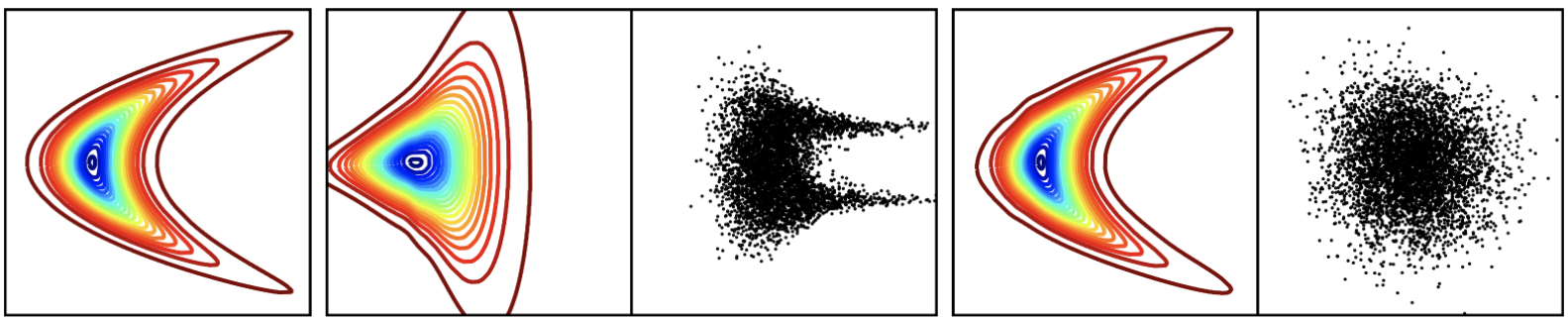
*Rezende D. J., Mohamed S. Variational Inference with Normalizing Flows, 2015*
*Berg R. et al. Sylvester normalizing flows for variational inference, 2018*

# Masked autoregressive flow (MAF)

## Gaussian autoregressive model

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}) = \prod_{j=1}^{m} \mathcal{N}\left(x_j|\mu_j(\mathbf{x}_{1:j-1}), \sigma_j^2(\mathbf{x}_{1:j-1})\right).$$

We could use MADE for the conditionals. Samples from the base distribution could be an indicator of how good the flow was fitted.



(a) Target density (b) MADE with Gaussian conditionals (c) MAF with 5 layers

MAF is just a stacked MADE model with different ordering.

- Parallel density estimation.
- Sequential sampling.

*Papamakarios G., Pavlakou T., Murray I. Masked Autoregressive Flow for Density Estimation, 2017*

# Outline

# Autoregressive flows

$$x_j = \tau(z_j, c(\mathbf{z}_{1:j-1})) \quad \Leftrightarrow \quad z_j = \tau^{-1}(x_j, c(\mathbf{z}_{1:j-1}))$$

- $\tau(\cdot, \cdot)$ – coupling law (invertible by first argument, differentiable).
- $c(\cdot)$ – coupling function (do not need to be invertible, could be neural network).

## Coupling law $\tau(\cdot, \cdot)$

- $\tau(x, c) = x + c$ – additive;
- $\tau(x, c) = x \odot c_1 + c_2$ – affine.

What is the Jacobian for the additive/affine coupling law?

## Jacobian

$$\det\left(\frac{\partial \mathbf{x}}{\partial \mathbf{z}}\right) = \prod_{j=1}^{m} \frac{\partial x_j}{\partial z_j} = \prod_{j=1}^{m} \frac{\partial \tau(z_j, c(\mathbf{z}_{1:j-1}))}{\partial z_j}$$

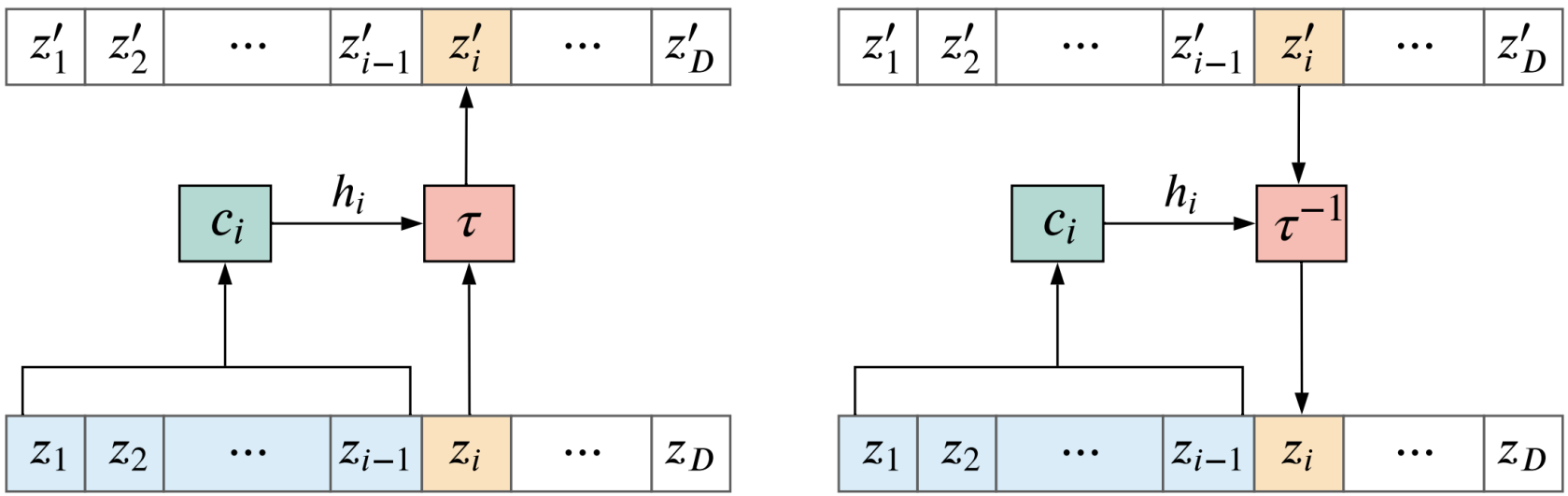
---

*Papamakarios G. et al. Normalizing flows for probabilistic modeling and inference, 2019*

# Autoregressive flows

## Forward and inverse transforms

$$x_j = \tau(z_j, c(\mathbf{z}_{1:j-1})) \quad \Leftrightarrow \quad z_j = \tau^{-1}(x_j, c(\mathbf{z}_{1:j-1}))$$




- Forward transform is **not sequential**.
- Inverse transform is **sequential**.

---

*Papamakarios G. et al. Normalizing flows for probabilistic modeling and inference, 2019*

# Outline

# Inverse gaussian autoregressive flow (IAF)

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

## Gaussian autoregressive flow

$$x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}) = (z_j - \tilde{\mu}_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\tilde{\sigma}_j(\mathbf{x}_{1:j-1})}$$

$$z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})} = \tilde{\sigma}_j(\mathbf{x}_{1:j-1}) \cdot x_j + \tilde{\mu}_j(\mathbf{x}_{1:j-1}).$$

Let just swap $\mathbf{z}$ and $\mathbf{x}$.

## Inverse gaussian autoregressive flow

$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) \quad \Rightarrow \quad x_j = \tilde{\sigma}_j(\mathbf{z}_{1:j-1}) \cdot z_j + \tilde{\mu}_j(\mathbf{z}_{1:j-1})$$

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) \quad \Rightarrow \quad z_j = (x_j - \tilde{\mu}_j(\mathbf{z}_{1:j-1})) \cdot \frac{1}{\tilde{\sigma}_j(\mathbf{z}_{1:j-1})}.$$

---

*Kingma D. P. et al. Improving Variational Inference with Inverse Autoregressive Flow, 2016*

# Inverse gaussian autoregressive flow (IAF)

## Gaussian autoregressive flow: $g(\mathbf{z}, \boldsymbol{\theta})$

$$x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}).$$

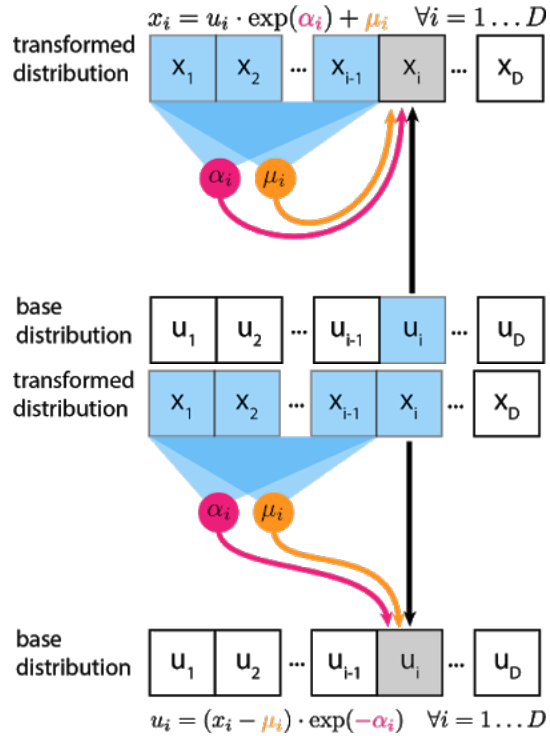
## Inverse transform: $f(\mathbf{x}, \boldsymbol{\theta})$

$$z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})};$$

$$z_j = \tilde{\sigma}_j(\mathbf{x}_{1:j-1}) \cdot x_j + \tilde{\mu}_j(\mathbf{x}_{1:j-1}).$$

## Inverse gaussian autoregressive flow: $g(\mathbf{z}, \boldsymbol{\theta})$

$$x_j = \tilde{\sigma}_j(\mathbf{z}_{1:j-1}) \cdot z_j + \tilde{\mu}_j(\mathbf{z}_{1:j-1}).$$



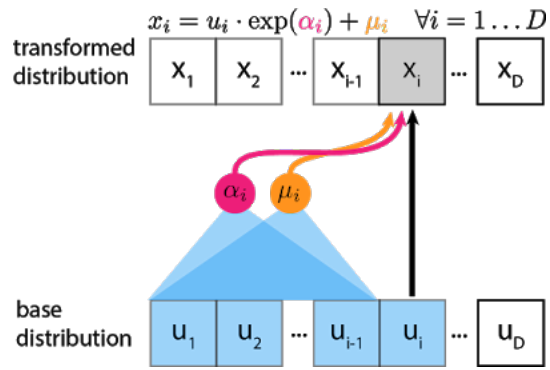




*image credit: https://blog.evjang.com/2018/01/nf2.html*

# Inverse gaussian autoregressive flow (IAF)

## Inverse gaussian autoregressive flow

$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) \quad \Rightarrow \quad x_j = \tilde{\sigma}_j(\mathbf{z}_{1:j-1}) \cdot z_j + \tilde{\mu}_j(\mathbf{z}_{1:j-1})$$

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) \quad \Rightarrow \quad z_j = (x_j - \tilde{\mu}_j(\mathbf{z}_{1:j-1})) \cdot \frac{1}{\tilde{\sigma}_j(\mathbf{z}_{1:j-1})}.$$

## Reverse KL for NF

$$KL(p||\pi) = \mathbb{E}_{p(\mathbf{z})}\left[\log p(\mathbf{z}) - \log|\det(\mathbf{J}_g)| - \log \pi(g(\mathbf{z}, \boldsymbol{\theta}))\right]$$

- We need to be able to compute $g(\mathbf{z}, \boldsymbol{\theta})$ and its Jacobian.
- We need to be able to sample from the density $p(\mathbf{z})$ (do not need to evaluate it) and to evaluate(!) $\pi(\mathbf{x})$.
- We don't need to think about computing the function $f(\mathbf{x}, \boldsymbol{\theta})$.

# Gaussian autoregressive NF

## Gaussian AR NF

$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) \quad \Rightarrow \quad x_j = \sigma_j(\mathbf{x}_{1:j-1}) \cdot z_j + \mu_j(\mathbf{x}_{1:j-1}).$$

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) \quad \Rightarrow \quad z_j = (x_j - \mu_j(\mathbf{x}_{1:j-1})) \cdot \frac{1}{\sigma_j(\mathbf{x}_{1:j-1})}.$$

- Sampling is sequential, density estimation is parallel.
- Forward KL is a natural loss.

## Inverse gaussian AR NF

$$\mathbf{x} = g(\mathbf{z}, \boldsymbol{\theta}) \quad \Rightarrow \quad x_j = \tilde{\sigma}_j(\mathbf{z}_{1:j-1}) \cdot z_j + \tilde{\mu}_j(\mathbf{z}_{1:j-1})$$

$$\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta}) \quad \Rightarrow \quad z_j = (x_j - \tilde{\mu}_j(\mathbf{z}_{1:j-1})) \cdot \frac{1}{\tilde{\sigma}_j(\mathbf{z}_{1:j-1})}.$$

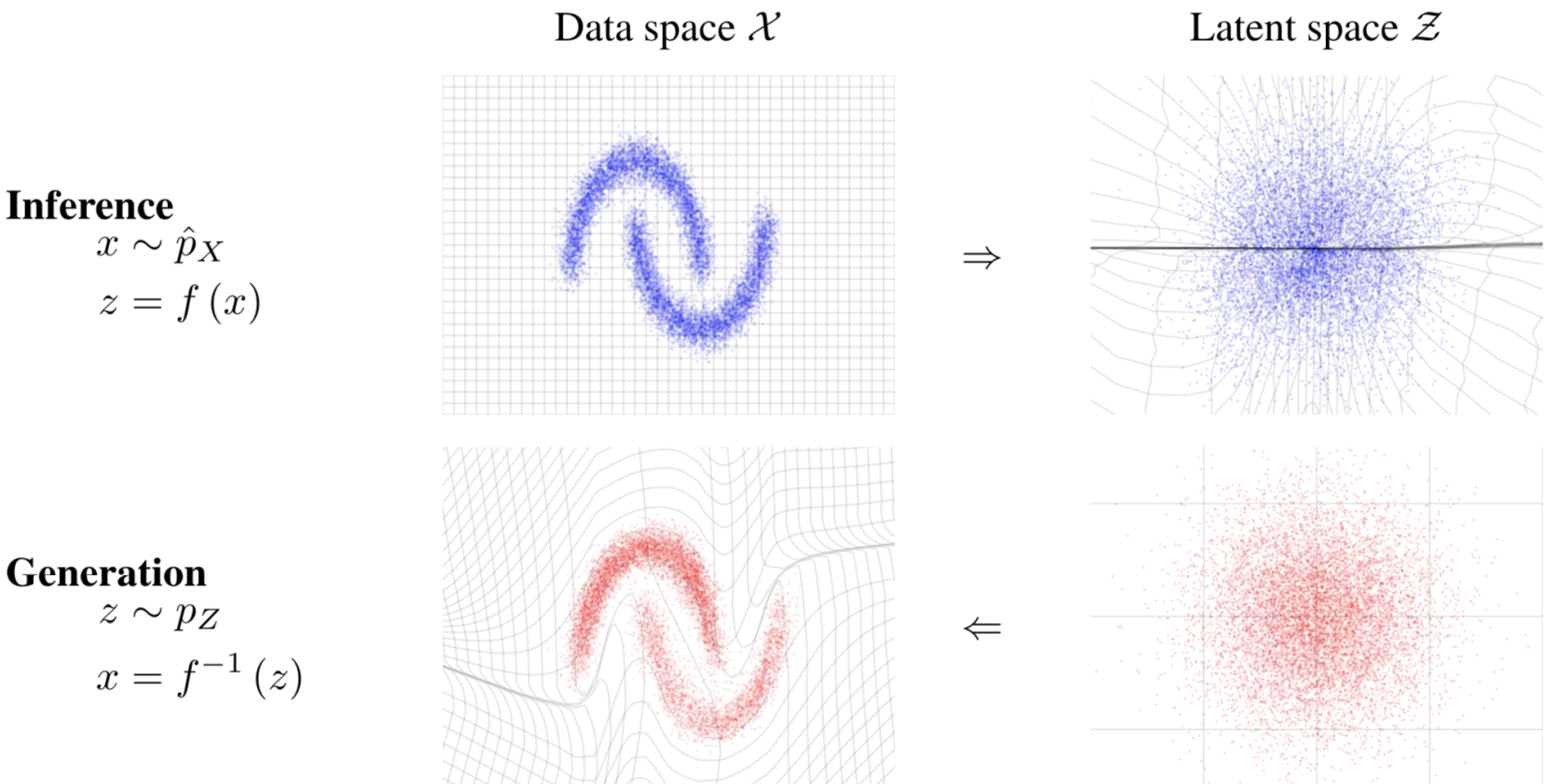
- Sampling is parallel, density estimation is sequential.
- Reverse KL is a natural loss.

---

*Kingma D. P. et al. Improving Variational Inference with Inverse Autoregressive Flow, 2016*

# Autoregressive flows

Gaussian AR NF and inverse gaussian AR NF are mutually interchangeable.




*Dinh L., Sohl-Dickstein J., Bengio S. Density estimation using Real NVP, 2016*

# Outline

# MAF/IAF pros and cons

## MAF

- Sampling is slow.
- Likelihood evaluation is fast.

## IAF

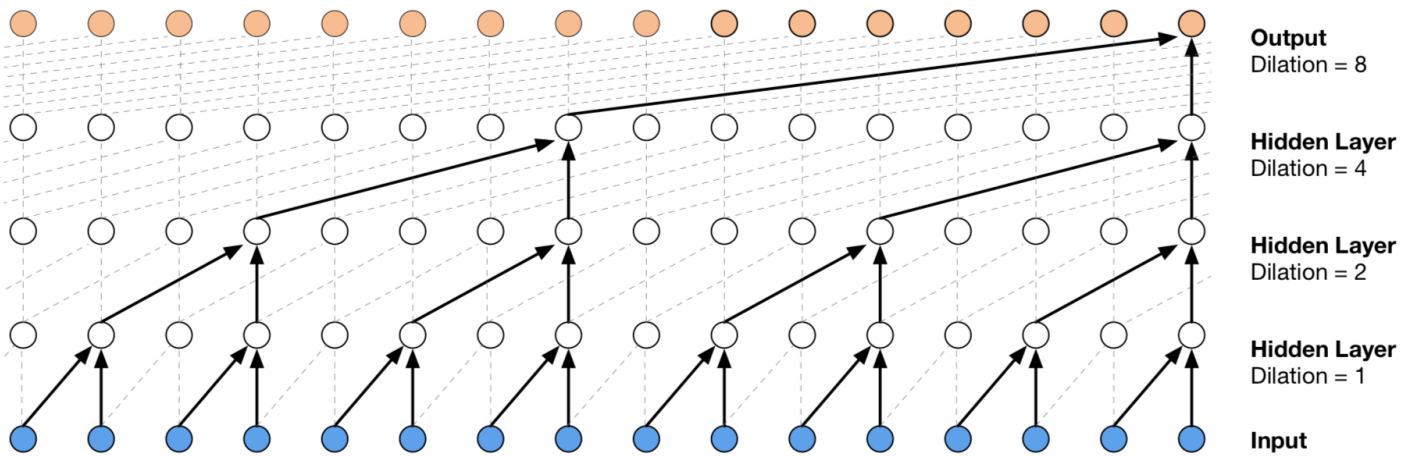
- Sampling is fast.
- Likelihood evaluation is slow.

How to take the best of both worlds?

## WaveNet

Autoregressive model with caused dilated convolutions for raw audio waveforms generation.

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{t=1}^{T} p(x_t|\mathbf{x}_{1:t-1}, \boldsymbol{\theta}).$$




*Oord A. et al. Wavenet: A generative model for raw audio, 2016*

# AF vs IAF vs RealNVP

## MADE/AF

$$\mathbf{x} = \boldsymbol{\sigma}(\mathbf{x}) \odot \mathbf{z} + \boldsymbol{\mu}(\mathbf{x}).$$

Estimating the density $p(\mathbf{x}|\boldsymbol{\theta})$ - 1 pass, sampling - $m$ passes.

## IAF

$$\mathbf{x} = \tilde{\boldsymbol{\sigma}}(\mathbf{z}) \odot \mathbf{z} + \tilde{\boldsymbol{\mu}}(\mathbf{z}).$$

Estimating the density $p(\mathbf{x}|\boldsymbol{\theta})$ - $m$ passes, sampling - 1 pass.

## RealNVP

$$\begin{cases} \mathbf{x}_{1:d} & = \mathbf{z}_{1:d}; \\ \mathbf{x}_{d:m} & = \mathbf{z}_{d:m} \odot c_1(\mathbf{z}_{1:d}, \boldsymbol{\theta}) + c_2(\mathbf{z}_{1:d}, \boldsymbol{\theta}). \end{cases}$$

---

*Papamakarios G., Pavlakou T., Murray I. Masked Autoregressive Flow for Density Estimation, 2017*

# AF vs IAF vs RealNVP

## RealNVP

$$\begin{cases} \mathbf{x}_{1:d} &= \mathbf{z}_{1:d}; \\ \mathbf{x}_{d:m} &= \mathbf{z}_{d:m} \odot c_1(\mathbf{z}_{1:d}, \boldsymbol{\theta}) + c_2(\mathbf{z}_{1:d}, \boldsymbol{\theta}). \end{cases}$$

- Calculating the density $p(\mathbf{x}|\boldsymbol{\theta})$ - 1 pass.
- Sampling - 1 pass.

RealNVP is a special case of AF and IAF:

## AF

$$\begin{cases} \mu_j = 0, \sigma_j = 1, \; j = 1, \ldots, d; \\ \mu_j, \sigma_j - \text{functions of } \mathbf{x}_{1:d}, \; j = d+1, \ldots, m. \end{cases}$$

## IAF

$$\begin{cases} \tilde{\mu}_j = 0, \tilde{\sigma}_j = 1, \; j = 1, \ldots, d; \\ \tilde{\mu}_j, \tilde{\sigma}_j - \text{functions of } \mathbf{z}_{1:d}, \; j = d+1, \ldots, m. \end{cases}$$

---

*Papamakarios G., Pavlakou T., Murray I. Masked Autoregressive Flow for Density Estimation, 2017*

# Linear flows

## RealNVP

$$\begin{cases} \mathbf{z}_{1:d} = \mathbf{x}_{1:d}; \\ \mathbf{z}_{d:m} = \tau(\mathbf{x}_{d:m}, c(\mathbf{x}_{1:d})); \end{cases} \quad \Leftrightarrow \quad \begin{cases} \mathbf{x}_{1:d} = \mathbf{z}_{1:d}; \\ \mathbf{x}_{d:m} = \tau^{-1}(\mathbf{z}_{d:m}, c(\mathbf{z}_{1:d})). \end{cases}$$

- First step is a **split** operator which decouples a variable into 2 subparts: $\mathbf{x}_1$ and $\mathbf{x}_2$ (usualy channel-wise).
- We should **permute** components between different layers.

$$\mathbf{z} = \mathbf{W}\mathbf{x}, \quad \mathbf{W} \in \mathbb{R}^{m \times m}$$

In general, we need $O(m^3)$ to invert matrix.

## Invertibility

- Diagonal matrix $O(m)$.
- Triangular matrix $O(m^2)$.
- It is impossible to parametrize all invertible matrices.

# Parallel WaveNet

- 24kHz instead of 16kHz using increased dilated convolution filter size from 2 to 3.
- 16-bit signals with mixture of logistics instead of 8-bit signal with 256-way categorical distribution.

## Probability density distillation

1. Train usual WaveNet (MAF) via MLE (teacher network).
2. Train IAF WaveNet (student network), which attempts to match the probability of its own samples under the distribution learned by the teacher.

## Student objective

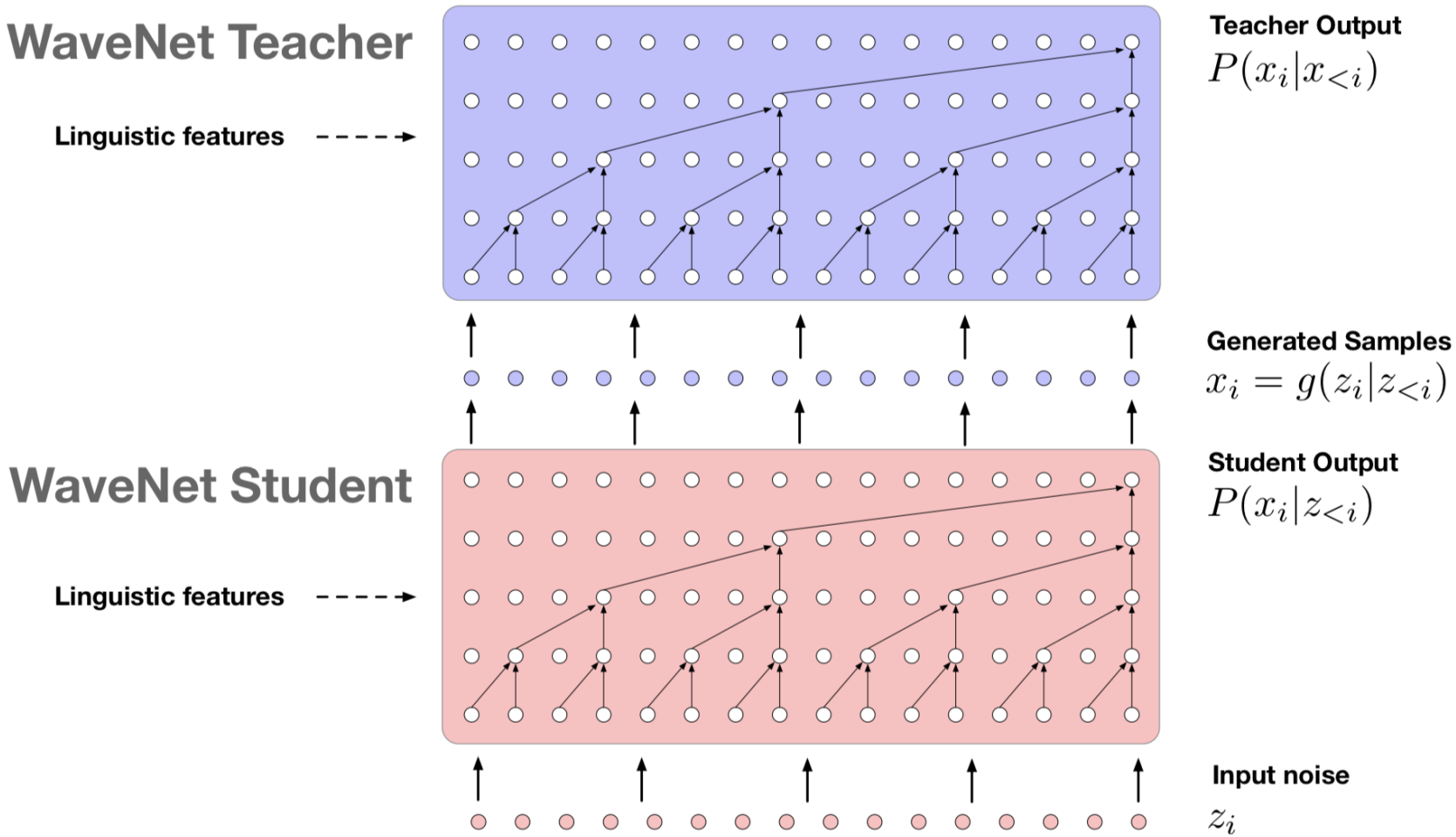
$$KL(p_s||p_t) = H(p_s, p_t) - H(p_s).$$

More than 1000x speed-up relative to original WaveNet!

---

*Oord A. et al. Parallel WaveNet: Fast High-Fidelity Speech Synthesis, 2017*

# Parallel WaveNet

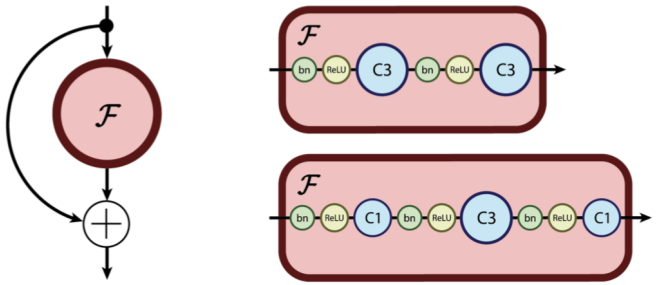



*Oord A. et al. Parallel WaveNet: Fast High-Fidelity Speech Synthesis, 2017*

# Outline

# RevNets, 2017

- 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).

---

*Gomez A. N. et al. The Reversible Residual Network: Backpropagation Without Storing Activations, 2017*

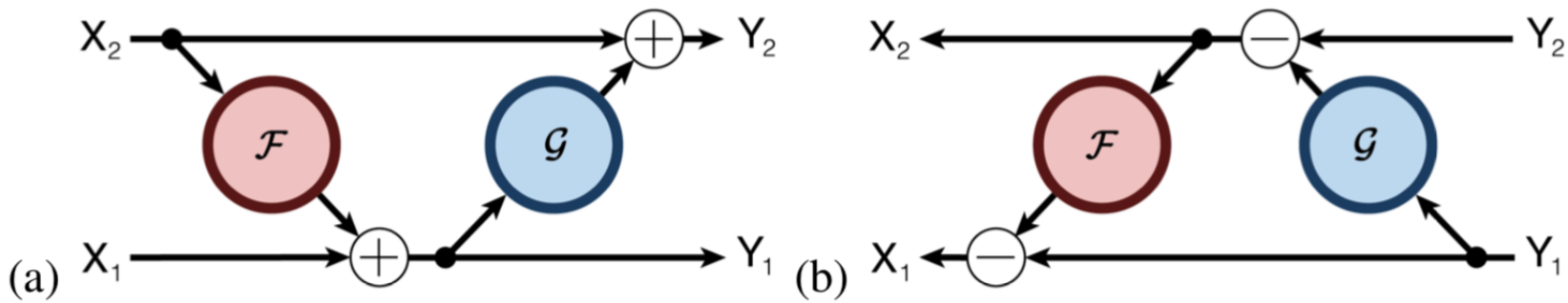
# RevNets, 2017

## NICE

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

## RevNet

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




*Gomez A. N. et al. The Reversible Residual Network: Backpropagation Without Storing Activations, 2017*

# RevNets, 2017

| Architecture | CIFAR-10 [15] | | CIFAR-100 [15] | |
|---|---|---|---|---|
| | ResNet | RevNet | ResNet | RevNet |
| 32 (38) | **7.14%** | 7.24% | 29.95% | **28.96%** |
| 110 | **5.74%** | 5.76% | 26.44% | **25.40%** |
| 164 | 5.24% | **5.17%** | **23.37%** | 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.

*Gomez A. N. et al. The Reversible Residual Network: Backpropagation Without Storing Activations, 2017*

# i-RevNet, 2018

## 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.

---

*Jacobsen J. H., Smeulders A., Oyallon E. i-RevNet: Deep Invertible Networks, 2018*

# i-RevNet, 2018

## Hypothesis

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

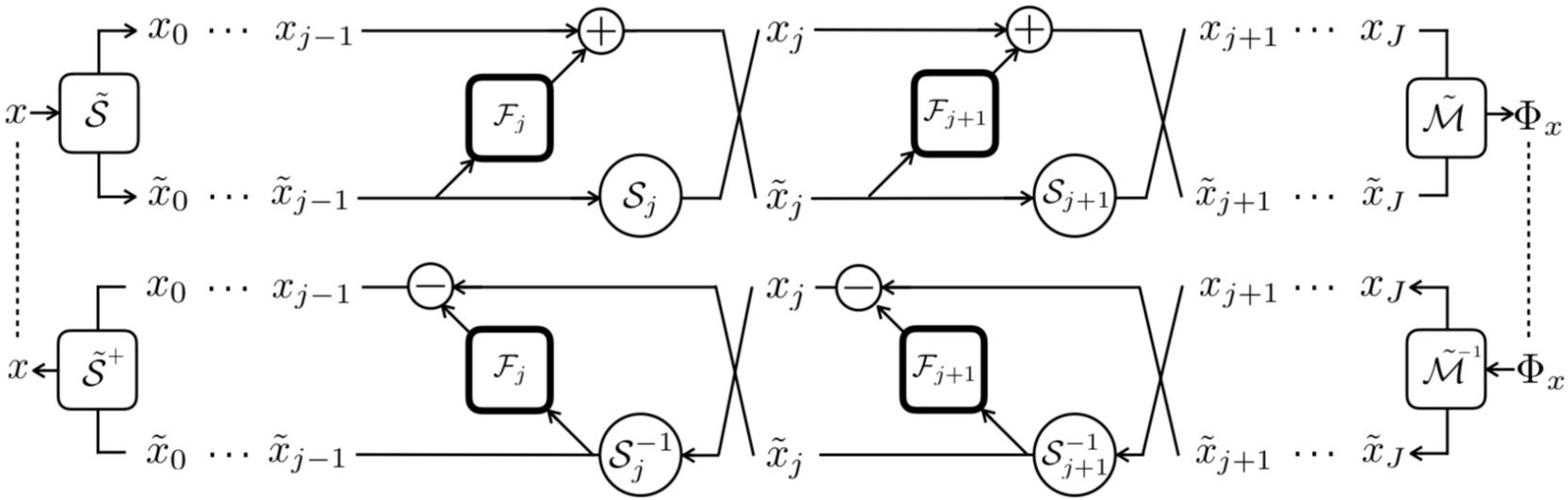
## Idea

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.

---

*Jacobsen J. H., Smeulders A., Oyallon E. i-RevNet: Deep Invertible Networks, 2018*

# i-RevNet, 2018




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

*Jacobsen J. H., Smeulders A., Oyallon E. i-RevNet: Deep Invertible Networks, 2018*

# Outline

# Discrete data vs continuous model

Let our data $\mathbf{y}$ comes from discrete distribution $\Pi(\mathbf{y})$ and we have continuous model $p(\mathbf{x}|\boldsymbol{\theta}) = \text{NN}(\mathbf{x}, \boldsymbol{\theta})$.

- Images (and not only images) are discrete data, pixels lie in the integer domain ($\{0, 255\}$).
- By fitting a continuous density model $p(\mathbf{x}|\boldsymbol{\theta})$ to discrete data $\Pi(\mathbf{y})$, one can produce a degenerate solution with all probability mass on discrete values.

## Discrete model

- Use **discrete** model (e.x. $P(\mathbf{y}|\boldsymbol{\theta}) = \text{Cat}(\boldsymbol{\pi}(\boldsymbol{\theta}))$).
- Minimize any suitable divergence measure $D(\Pi, P)$.
- NF works only with continuous data $\mathbf{x}$ (there are discrete NF, see papers below).
- If pixel value is not presented in the train data, it won't be predicted.

---

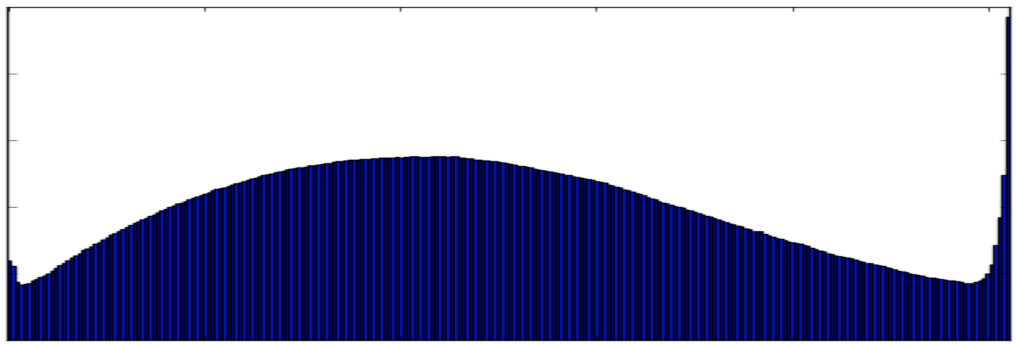
*Hoogeboom E. et al. Integer discrete flows and lossless compression, 2019*
*Tran D. et al. Discrete flows: Invertible generative models of discrete data, 2019*

# Discrete data vs continuous model

## Continuous model

- Use **continuous** model (e.x. $p(\mathbf{x}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}), \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\mathbf{x}))$), but
  - **discretize** model (make the model outputs discrete): transform $p(\mathbf{x}|\boldsymbol{\theta})$ to $P(\mathbf{y}|\boldsymbol{\theta})$;
  - **dequantize** data (make the data continuous): transform $\Pi(\mathbf{y})$ to $\pi(\mathbf{x})$.
- Continuous distribution knows numerical relationships.

## CIFAR-10 pixel values distribution



*Salimans T. et al. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications, 2017*

# Discretization of continuous distribution

## Model discretization through CDF

$$F(\mathbf{x}|\boldsymbol{\theta}) = \int_{-\infty}^{\mathbf{x}} p(\mathbf{x}'|\boldsymbol{\theta})d\mathbf{x}'; \quad P(\mathbf{y}|\boldsymbol{\theta}) = F(\mathbf{y}+0.5|\boldsymbol{\theta}) - F(\mathbf{y}-0.5|\boldsymbol{\theta})$$

## Mixture of logistic distributions

$$p(x|\mu, s) = \frac{\exp^{-(x-\mu)/s}}{s(1+\exp^{-(x-\mu)/s})^2}; \quad p(x|\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}) = \sum_{k=1}^{K} \pi_k p(x|\mu_k, s_k).$$

## PixelCNN++

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}); \quad p(x_j|\mathbf{x}_{1:j-1}, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p(x|\mu_k, s_k).$$

Here, $\pi_k = \pi_{k,\boldsymbol{\theta}}(\mathbf{x}_{1:j-1})$, $\mu_k = \mu_{k,\boldsymbol{\theta}}(\mathbf{x}_{1:j-1})$, $s_k = s_{k,\boldsymbol{\theta}}(\mathbf{x}_{1:j-1})$.

For the pixel edge cases of 0, replace $y - 0.5$ by $-\infty$, and for 255 replace $y + 0.5$ by $+\infty$.

---

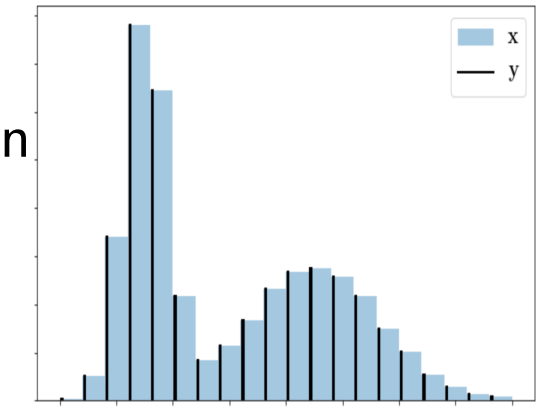
*Salimans T. et al. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications, 2017*

# Uniform dequantization

Let dequantize discrete distribution $\Pi(\mathbf{y})$ to continuous distribution $\pi(\mathbf{x})$ in the following way: $\mathbf{x} = \mathbf{y} + \mathbf{u}$, where $\mathbf{u} \sim U[0, 1]$.

## Theorem

Fitting continuous model $p(\mathbf{x}|\boldsymbol{\theta})$ on uniformly dequantized data is equivalent to maximization of a lower bound on log-likelihood for a discrete model:




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

## Proof

$$\mathbb{E}_\pi \log p(\mathbf{x}|\boldsymbol{\theta}) = \int \pi(\mathbf{x}) \log p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} = \sum \Pi(\mathbf{y}) \int_{U[0,1]} \log p(\mathbf{y}+\mathbf{u}|\boldsymbol{\theta}) d\mathbf{u} \leq$$

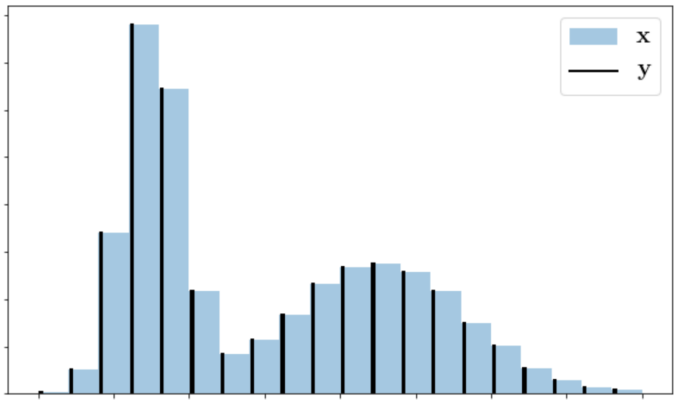
$$\leq \sum \Pi(\mathbf{y}) \log \int_{U[0,1]} p(\mathbf{y} + \mathbf{u}|\boldsymbol{\theta}) d\mathbf{u} =$$

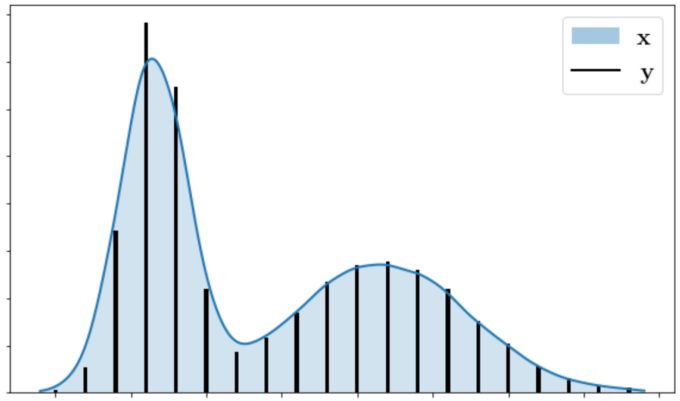
$$= \sum \Pi(\mathbf{y}) \log P(\mathbf{y}|\boldsymbol{\theta}) = \mathbb{E}_\Pi \log P(\mathbf{y}|\boldsymbol{\theta}).$$

---

*Theis L., Oord A., Bethge M. A note on the evaluation of generative models. 2015*

# Variational dequantization




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

Introduce variational dequantization noise distribution $q(\mathbf{u}|\mathbf{y})$, which tells what kind of noise we have to add to our discrete data. Treat it as an approximate posterior as in VAE model.

# Variational dequantization

## Variational lower bound

$$\log P(\mathbf{y}|\boldsymbol{\theta}) = \left[\log \int q(\mathbf{u}|\mathbf{y}) \frac{p(\mathbf{y}+\mathbf{u}|\boldsymbol{\theta})}{q(\mathbf{u}|\mathbf{y})} d\mathbf{u}\right] \geq$$

$$\geq \int q(\mathbf{u}|\mathbf{y}) \log \frac{p(\mathbf{y}+\mathbf{u}|\boldsymbol{\theta})}{q(\mathbf{u}|\mathbf{y})} d\mathbf{u} = \mathcal{L}(q, \boldsymbol{\theta}).$$

Uniform dequantization is a special case of variational dequantization ($q(\mathbf{u}|\mathbf{y}) = U[0,1]$).

## Flow++: flow-based variational dequantization

Let $\mathbf{u} = g_{\boldsymbol{\lambda}}(\boldsymbol{\epsilon}, \mathbf{y})$ is a flow model with base distribution $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$:

$$q(\mathbf{u}|\mathbf{y}) = p(f_{\boldsymbol{\lambda}}(\mathbf{u}, \mathbf{y})) \cdot \left|\det \frac{\partial f_{\boldsymbol{\lambda}}(\mathbf{u}, \mathbf{y})}{\partial \mathbf{u}}\right|.$$

$$\log P(\mathbf{y}|\boldsymbol{\theta}) \geq \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\theta}) = \int p(\boldsymbol{\epsilon}) \log \left( \frac{p(\mathbf{y} + g_{\boldsymbol{\lambda}}(\boldsymbol{\epsilon}, \mathbf{y})|\boldsymbol{\theta})}{p(\boldsymbol{\epsilon}) \cdot |\det \mathbf{J}_g|^{-1}} \right) d\boldsymbol{\epsilon}.$$

---

*Ho J. et al. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design, 2019*

# Flow++

## Flow-based variational dequantization

$$\log P(\mathbf{x}|\boldsymbol{\theta}) \geq \int p(\boldsymbol{\epsilon}) \log \left( \frac{p(\mathbf{x} + g(\boldsymbol{\epsilon}, \mathbf{x}, \boldsymbol{\lambda}))}{p(\boldsymbol{\epsilon}) \cdot |\det \mathbf{J}_g|^{-1}} \right) d\boldsymbol{\epsilon}.$$

*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 | – | – |
| | **Flow++ (ours)** | **3.08** | **3.86** | **3.69** |
| 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 |

*Ho J. et al. Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design, 2019*

# Outline

# Continuous Normalizing Flows

## Forward transform + log-density

$$\begin{bmatrix} \mathbf{x} \\ \log p(\mathbf{x}|\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \mathbf{z} \\ \log p(\mathbf{z}) \end{bmatrix} + \int_{t_0}^{t_1} \begin{bmatrix} f(\mathbf{z}(t), \boldsymbol{\theta}) \\ -\text{trace}\left(\frac{\partial f(\mathbf{z}(t), \boldsymbol{\theta})}{\partial \mathbf{z}(t)}\right) \end{bmatrix} dt.$$

- Discrete-in-time normalizing flows need invertible $f$. It costs $O(d^3)$ to get determinant of Jacobian.
- Continuous-in-time flows require only smoothness of $f$. It costs $O(d^2)$ to get trace of Jacobian.

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

## Hutchinson's trace estimator

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

## FFJORD density estimation

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

---

*Grathwohl W. et al. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018*

# FFJORD

| | Method | 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 | ✓ | ✓ | ✗ |
| | **FFJORD** | ✓ | ✓ | ✓ |

## Density estimation (forward KL)

| | 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* |

## Flows for variational inference (reverse KL)

| | MNIST | Omniglot | Frey Faces | Caltech Silhouettes |
|---|---|---|---|---|
| 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 | $\mathbf{82.82 \pm .01}$ | $\mathbf{98.33 \pm .09}$ | $\mathbf{4.39 \pm .01}$ | $\mathbf{104.03 \pm .43}$ |

*Grathwohl W. et al. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models, 2018*

# Outline

# ELBO interpretations

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})).$$

$$\mathcal{L}(q, \boldsymbol{\theta}) = \int q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi}) \log \frac{p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} d\mathbf{z}.$$

- Evidence minus posterior KL

$$\mathcal{L}(q, \boldsymbol{\theta}) = \log p(\mathbf{x}|\boldsymbol{\theta}) - KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})).$$

- Average negative energy plus entropy

$$\begin{aligned}\mathcal{L}(q, \boldsymbol{\theta}) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} \left[\log p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) - \log q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})\right] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} \log p(\mathbf{x}, \mathbf{z}|\boldsymbol{\theta}) + \mathbb{H}\left[q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})\right].\end{aligned}$$

- Average reconstruction minus KL to prior

$$\begin{aligned}\mathcal{L}(q, \boldsymbol{\theta}) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} \left[\log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) + \log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})\right] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z})).\end{aligned}$$

# ELBO surgery, 2016

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

## ELBO interpretations

- Evidence minus posterior KL

$$\mathcal{L}(q, \theta) = \log p(\mathbf{X}|\theta) - KL(q(\mathbf{Z}|\mathbf{X})||p(\mathbf{Z}|\mathbf{X}, \theta)).$$

- Average negative energy plus entropy

$$\mathcal{L}(q, \theta) = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X})} p(\mathbf{X}, \mathbf{Z}|\theta) + \mathbb{H}\left[q(\mathbf{Z}|\mathbf{X})\right].$$

- Average term-by-term reconstruction minus KL to prior

$$\mathcal{L}(q, \theta) = \frac{1}{n} \sum_{i=1}^{n} \left[\mathbb{E}_{q(\mathbf{z}_i|\mathbf{x}_i)} \log p(\mathbf{x}_i|\mathbf{z}_i, \theta) - KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i))\right].$$

---

http://approximateinference.org/accepted/HoffmanJohnson2016.pdf

# ELBO surgery, 2016

$$\mathcal{L}(q, \theta) = \frac{1}{n} \sum_{i=1}^{n} \left[ \mathbb{E}_{q(\mathbf{z}_i|\mathbf{x}_i)} \log p(\mathbf{x}_i|\mathbf{z}_i, \theta) - KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i)) \right].$$

## Theorem

$$\frac{1}{n} \sum_{i=1}^{n} KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i)) = KL(q(\mathbf{z})||p(\mathbf{z})) + \mathbb{I}_{q(i,\mathbf{z})}[i, \mathbf{z}],$$

where $i$ is treated as random variable:

$$q(i, \mathbf{z}) = q(i)q(\mathbf{z}|i); \quad p(i, \mathbf{z}) = p(i)p(\mathbf{z}); \quad q(i) = p(i) = \frac{1}{n}; \quad q(\mathbf{z}|i) = q(\mathbf{z}|\mathbf{x}_i).$$

$$q(\mathbf{z}) = \sum_{i=1}^{n} q(i, \mathbf{z}) = \frac{1}{n} \sum_{i=1}^{n} q(\mathbf{z}|\mathbf{x}_i); \quad \mathbb{I}_{q(i,\mathbf{z})}[i, \mathbf{z}] = \mathbb{E}_{q(i,\mathbf{z})} \log \frac{q(i, \mathbf{z})}{q(i)q(\mathbf{z})}.$$

---

http://approximateinference.org/accepted/HoffmanJohnson2016.pdf

# ELBO surgery, 2016

## Theorem

$$\frac{1}{n}\sum_{i=1}^{n} KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i)) = KL(q(\mathbf{z})||p(\mathbf{z})) + \mathbb{I}_{q(i,\mathbf{z})}[i,\mathbf{z}].$$

## Proof

$$\frac{1}{n}\sum_{i=1}^{n} KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i)) = \sum_{i=1}^{n}\int q(i)q(\mathbf{z}|i)\log\frac{q(\mathbf{z}|i)}{p(\mathbf{z})}d\mathbf{z} =$$

$$= \sum_{i=1}^{n}\int q(i,\mathbf{z})\log\frac{q(i,\mathbf{z})}{p(\mathbf{z})p(i)}d\mathbf{z} = \int\sum_{i=1}^{n} q(i,\mathbf{z})\log\frac{q(\mathbf{z})q(i|\mathbf{z})}{p(\mathbf{z})p(i)}d\mathbf{z} =$$

$$= \int q(\mathbf{z})\log\frac{q(\mathbf{z})}{p(\mathbf{z})}d\mathbf{z} + \int\sum_{i=1}^{n} q(i|\mathbf{z})q(\mathbf{z})\log\frac{q(i|\mathbf{z})}{p(i)}d\mathbf{z} =$$

$$= KL(q(\mathbf{z})||p(\mathbf{z})) - \mathbb{E}_{q(\mathbf{z})}\mathbb{H}\left[q(i|\mathbf{z})\right] + \log n.$$

---

http://approximateinference.org/accepted/HoffmanJohnson2016.pdf

# ELBO surgery, 2016

## Theorem

$$\frac{1}{n}\sum_{i=1}^{n} KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i)) = KL(q(\mathbf{z})||p(\mathbf{z})) + \mathbb{I}_{q(i,\mathbf{z})}[i,\mathbf{z}].$$

## Proof (continued)

$$\frac{1}{n}\sum_{i=1}^{n} KL(q(\mathbf{z}_i|\mathbf{x}_i)||p(\mathbf{z}_i)) = KL(q(\mathbf{z})||p(\mathbf{z})) - \mathbb{E}_{q(\mathbf{z})}\mathbb{H}\left[q(i|\mathbf{z})\right] + \log n$$

$$\mathbb{I}_{q(i,\mathbf{z})}[i,\mathbf{z}] = \mathbb{E}_{q(i,\mathbf{z})} \log \frac{q(i,\mathbf{z})}{q(i)q(\mathbf{z})} = \mathbb{E}_{q(\mathbf{z})}\mathbb{E}_{q(i|\mathbf{z})} \log \frac{q(i|\mathbf{z})q(\mathbf{z})}{q(i)q(\mathbf{z})} =$$

$$= \mathbb{E}_{q(\mathbf{z})}\mathbb{E}_{q(i|\mathbf{z})} \log \frac{q(i|\mathbf{z})}{q(i)} = -\mathbb{E}_{q(\mathbf{z})}\mathbb{H}\left[q(i|\mathbf{z})\right] + \log n.$$

---

http://approximateinference.org/accepted/HoffmanJohnson2016.pdf

# Outline

# Learnable VAE prior

## Optimal prior

$$KL(q_{\text{agg}}(\mathbf{z})||p(\mathbf{z})) = 0 \quad \Leftrightarrow \quad p(\mathbf{z}) = q_{\text{agg}}(\mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n} q(\mathbf{z}|\mathbf{x}_i).$$

## Mixture of Gaussians

$$p(\mathbf{z}|\boldsymbol{\lambda}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2), \quad \boldsymbol{\lambda} = \{w_k, \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k\}_{k=1}^{K}.$$

## Variational Mixture of posteriors (VampPrior)

$$p(\mathbf{z}|\boldsymbol{\lambda}) = \frac{1}{K}\sum_{k=1}^{K} q(\mathbf{z}|\mathbf{u}_k),$$

where $\boldsymbol{\lambda} = \{\mathbf{u}_1, \ldots, \mathbf{u}_K\}$ are trainable pseudo-inputs.

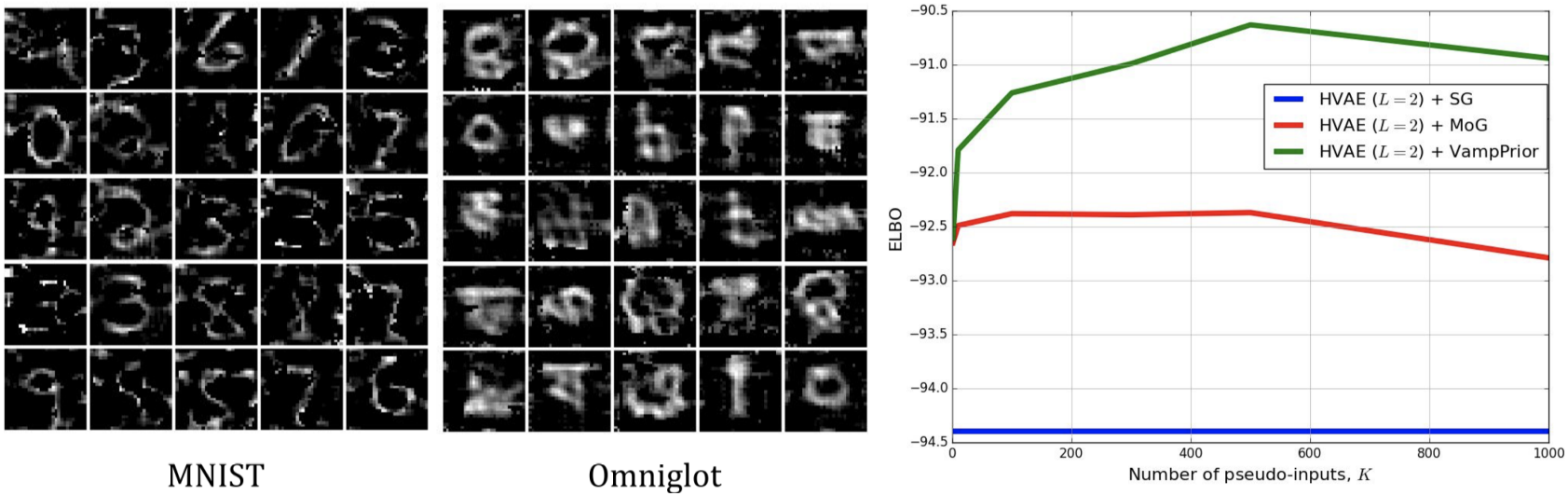
- Multimodal $\Rightarrow$ prevents over-regularization;.
- $K \ll n \Rightarrow$ prevents from potential overfitting + less expensive to train.

*Tomczak J. M., Welling M. VAE with a VampPrior, 2017*

# VampPrior

- Do we really need the multimodal prior?
- Is it beneficial to couple the prior with the variational posterior or the MoG prior is enough?

## Results




**Top row:** generated images by PixelHVAE + VampPrior for chosen pseudo-input in the left top corner.

**Bottom row:** pseudo-inputs for different datasets.

*Tomczak J. M., Welling M. VAE with a VampPrior, 2017*

# Outline

# Normalizing Flows in VAE posterior

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \left[\log p(\mathbf{x}|\mathbf{z}, \theta) + \log p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}, \phi)\right]$$

Let apply NF to VAE posterior!
Assume $q(\mathbf{z}|\mathbf{x}, \phi)$ (VAE encoder) is a base distribution for a flow model.

## Flow model in latent space

$$\log q(\mathbf{z}^*|\mathbf{x}, \phi, \boldsymbol{\lambda}) = \log q(\mathbf{z}|\mathbf{x}, \phi) + \log \left|\det \left(\frac{d\mathbf{z}}{d\mathbf{z}^*}\right)\right|$$

$$\mathbf{z}^* = f(\mathbf{z}, \boldsymbol{\lambda}) = g^{-1}(\mathbf{z}, \boldsymbol{\lambda})$$

- Encoder outputs base distribution $q(\mathbf{z}|\mathbf{x}, \phi)$.
- Flow model $\mathbf{z}^* = f(\mathbf{z}, \boldsymbol{\lambda})$ transforms the base distribution $q(\mathbf{z}|\mathbf{x}, \phi)$ to the distribution $q(\mathbf{z}^*|\mathbf{x}, \phi, \boldsymbol{\lambda})$.
- Distribution $q(\mathbf{z}^*|\mathbf{x}, \phi, \boldsymbol{\lambda})$ is used as a variational distribution for ELBO maximization.
- Here $\phi$ – encoder parameters, $\boldsymbol{\lambda}$ – flow parameters.

---

*Rezende D. J., Mohamed S. Variational Inference with Normalizing Flows, 2015*

# Normalizing Flows in VAE posterior

## ELBO with flow-based VAE posterior

$$\mathcal{L}(\phi, \theta, \lambda) = \mathbb{E}_{q(\mathbf{z}^*|\mathbf{x},\phi,\lambda)} \log p(\mathbf{x}|\mathbf{z}^*, \theta) - KL(q(\mathbf{z}^*|\mathbf{x}, \phi, \lambda)||p(\mathbf{z}^*)) =$$
$$= \mathbb{E}_{q(\mathbf{z}^*|\mathbf{x},\phi,\lambda)} \left[\log p(\mathbf{x}|\mathbf{z}^*, \theta) + \log p(\mathbf{z}^*) - \log q(\mathbf{z}^*|\mathbf{x}, \phi, \lambda)\right] =$$
$$= \mathbb{E}_{q(\mathbf{z}^*|\mathbf{x},\phi,\lambda)} \Bigg[ \log p(\mathbf{x}|\mathbf{z}^*, \theta) + \log p(\mathbf{z}^*) -$$
$$- \Big(\log q(g(\mathbf{z}^*, \lambda)|\mathbf{x}, \phi) + \log |\det (\mathbf{J}_g)|\Big)\Bigg].$$

KL term in ELBO is **reverse** KL divergence with respect to $\lambda$.

- RealNVP with coupling layers.
- Inverse autoregressive flow (slow $f(\mathbf{z}, \lambda)$, fast $g(\mathbf{z}^*, \lambda)$).
- Is it OK to use AF for VAE posterior?

*Rezende D. J., Mohamed S. Variational Inference with Normalizing Flows, 2015*

# Normalizing Flows in VAE posterior

## Theorem (flow KL duality, Lecture 5)

$$KL(\pi(\mathbf{x})||p(\mathbf{x}|\boldsymbol{\theta})) = KL(p(\mathbf{z}|\boldsymbol{\theta})||p(\mathbf{z})).$$

## ELBO with flow-based VAE posterior

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\lambda}) &= \mathbb{E}_{q(\mathbf{z}^*|\mathbf{x},\boldsymbol{\phi},\boldsymbol{\lambda})} \log p(\mathbf{x}|\mathbf{z}^*, \boldsymbol{\theta}) - KL(q(\mathbf{z}^*|\mathbf{x}, \boldsymbol{\phi}, \boldsymbol{\lambda})||p(\mathbf{z}^*)) \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log p(\mathbf{x}|f(\mathbf{z}, \boldsymbol{\lambda}), \boldsymbol{\theta}) - KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}|\boldsymbol{\lambda})).\end{aligned}$$

(Here we use Flow KL duality theorem and LOTUS trick.)

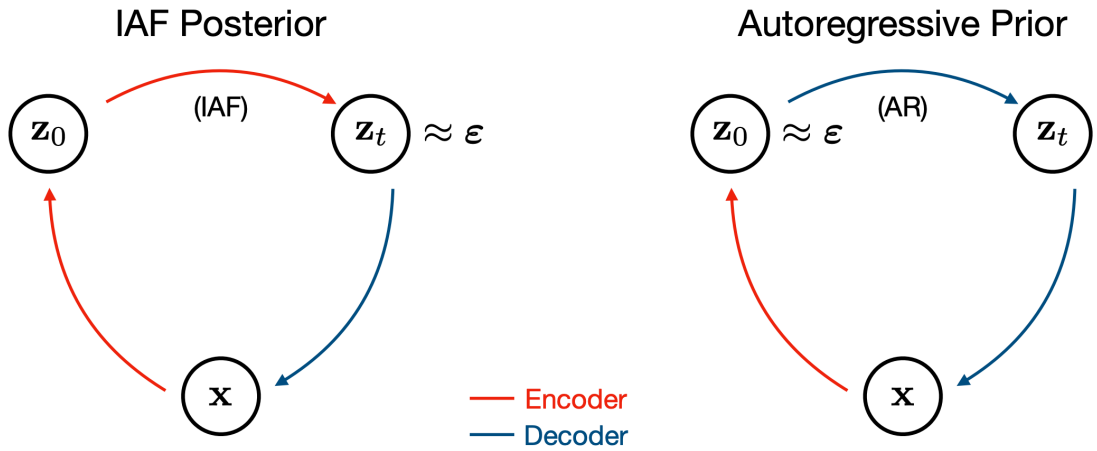
## ELBO with flow-based VAE prior

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\boldsymbol{\phi})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - KL(q(\mathbf{z}|\mathbf{x}, \boldsymbol{\phi})||p(\mathbf{z}|\boldsymbol{\lambda}))$$

---

*Chen X. et al. Variational Lossy Autoencoder, 2016*

# Flows-based VAE prior vs posterior




- Flow-based posterior decoder path: $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$.
- Flow-based prior decoder path: $\mathbf{z}^* \sim p(\mathbf{z}^*)$, $\mathbf{z} = f(\mathbf{z}^*, \boldsymbol{\lambda})$, $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta})$.

*image credit: https://courses.cs.washington.edu/courses/cse599i/20au*

# Outline

# InfoGAN

## GAN objective

$$\min_G \max_D V(G, D)$$

$$V(G, D) = \mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z})))$$

Latent vector $\mathbf{z}$ is not imposed to be disentangled.
InfoGAN decomposes input vector:

- $\mathbf{z}$ – incompressible noise;
- $\mathbf{c}$ – structured latent code $p(\mathbf{c}) = \prod_{j=1}^{d} p(c_j)$.

## Information-theoretic regularization

$$\max I(\mathbf{c}, G(\mathbf{z}, \mathbf{c}))$$

Information in the latent code $\mathbf{c}$ should not be lost in the generation process.

*Chen X. et al. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, 2016*

# InfoGAN

## Objective

$$\min_G \max_D V(G, D) - \lambda I(\mathbf{c}, G(\mathbf{z}, \mathbf{c}))$$

## Variational Information Maximization

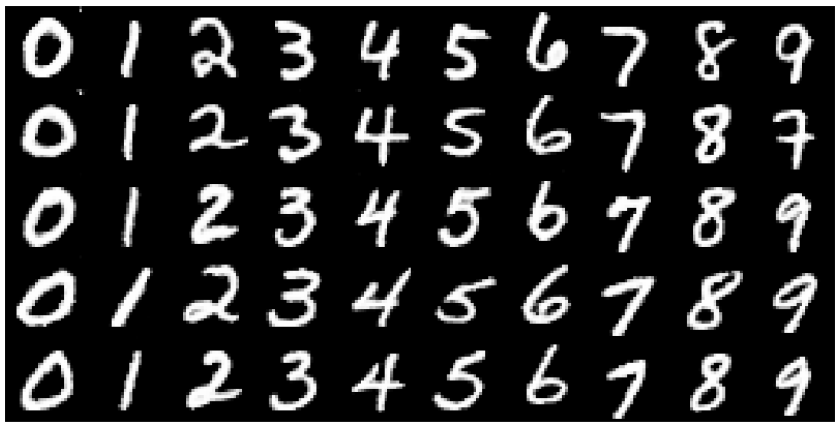
$$\begin{aligned} I(\mathbf{c}, G(\mathbf{z}, \mathbf{c})) &= H(\mathbf{c}) - H(\mathbf{c}|G(\mathbf{z}, \mathbf{c})) = \\ &= H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim G(\mathbf{z},\mathbf{c})}\left[\mathbb{E}_{\mathbf{c}'\sim p(\mathbf{c}|\mathbf{x})} \log p(\mathbf{c}'|\mathbf{x})\right] = \\ &= H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim G(\mathbf{z},\mathbf{c})} KL(p(\mathbf{c}'|\mathbf{x})||q(\mathbf{z}'|\mathbf{x})) + \\ &\quad + \mathbb{E}_{\mathbf{x}\sim G(\mathbf{z},\mathbf{c})}\mathbb{E}_{\mathbf{c}'\sim p(\mathbf{c}|\mathbf{x})} \log q(\mathbf{c}'|\mathbf{x}) \geq \\ &\geq H(\mathbf{c}) + \mathbb{E}_{\mathbf{x}\sim G(\mathbf{z},\mathbf{c})}\mathbb{E}_{\mathbf{c}'\sim p(\mathbf{c}|\mathbf{x})} \log q(\mathbf{c}'|\mathbf{x}) = \\ &\quad H(\mathbf{c}) + \mathbb{E}_{\mathbf{c}\sim p(\mathbf{c})}\mathbb{E}_{\mathbf{x}\sim G(\mathbf{z},\mathbf{c})} \log q(\mathbf{c}|\mathbf{x}) \end{aligned}$$

---

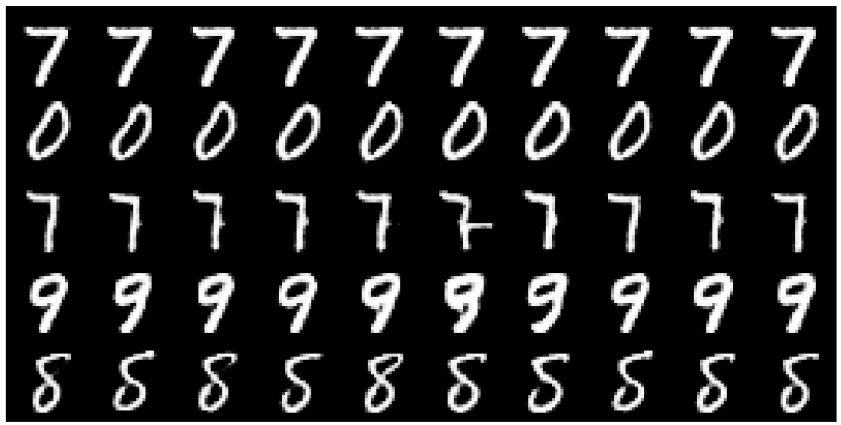
*Chen X. et al. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, 2016*

# InfoGAN

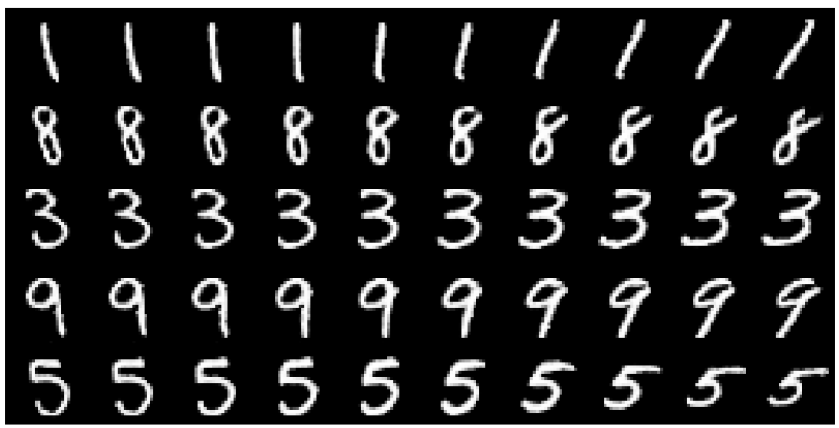
## Latent codes on MNIST



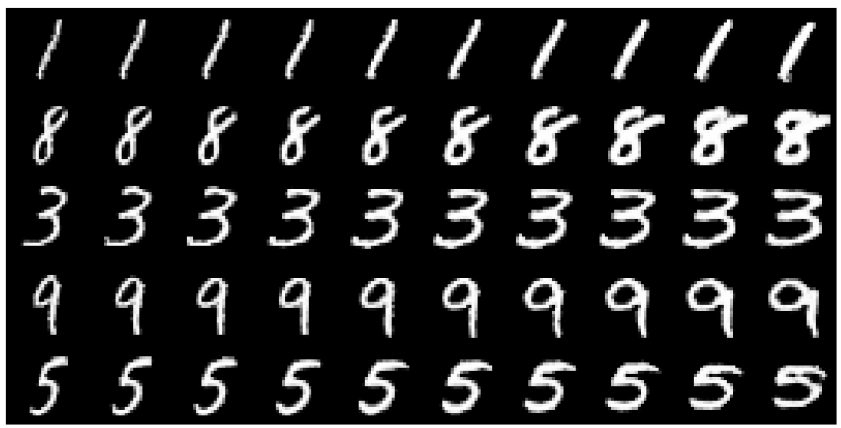
(a) Varying $c_1$ on InfoGAN (Digit type)



(b) Varying $c_1$ on regular GAN (No clear meaning)



(c) Varying $c_2$ from $-2$ to 2 on InfoGAN (Rotation)



(d) Varying $c_3$ from $-2$ to 2 on InfoGAN (Width)

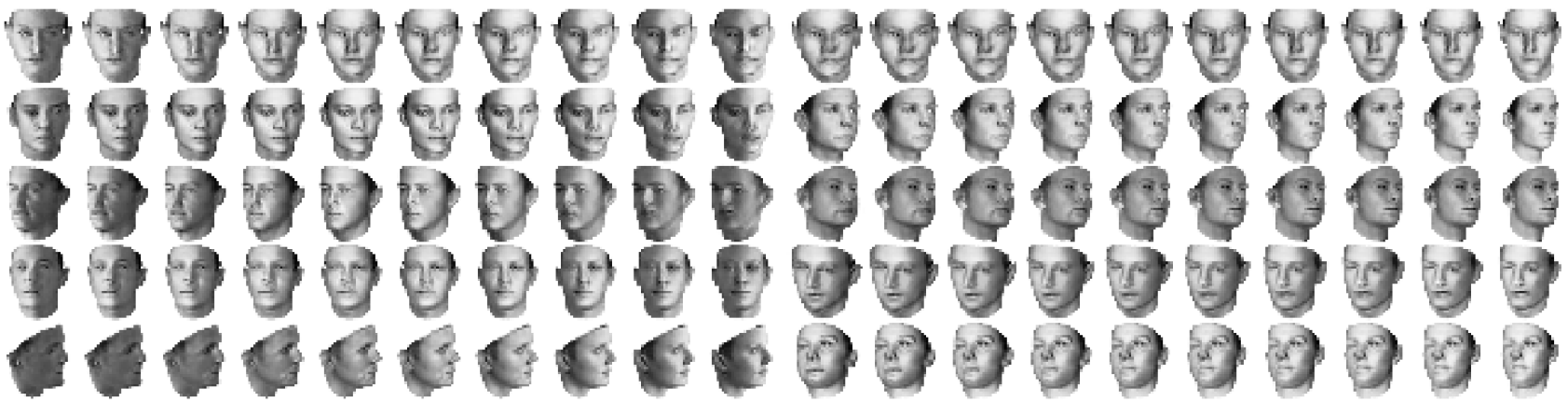
---

*Chen X. et al. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, 2016*

# InfoGAN

## Latent codes on 3D Faces

(a) Azimuth (pose) (b) Elevation



(c) Lighting (d) Wide or Narrow

---

*Chen X. et al. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, 2016*

# Disentangled representations

**Representation learning** is looking for an interpretable representation of the independent data generative factors.

## Disentanglement informal definition

Every single latent unit are sensitive to changes in a single generative factor, while being invariant to changes in other factors.

## ELBO objective

$$\mathcal{L}(q, \boldsymbol{\theta}, \beta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - \beta \cdot KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).$$

What do we get at $\beta = 1$?

## Constrained optimization

$$\max_{q, \boldsymbol{\theta}} \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}), \quad \text{subject to } KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) < \epsilon.$$

**Note:** It leads to poorer reconstructions and a loss of high frequency details.

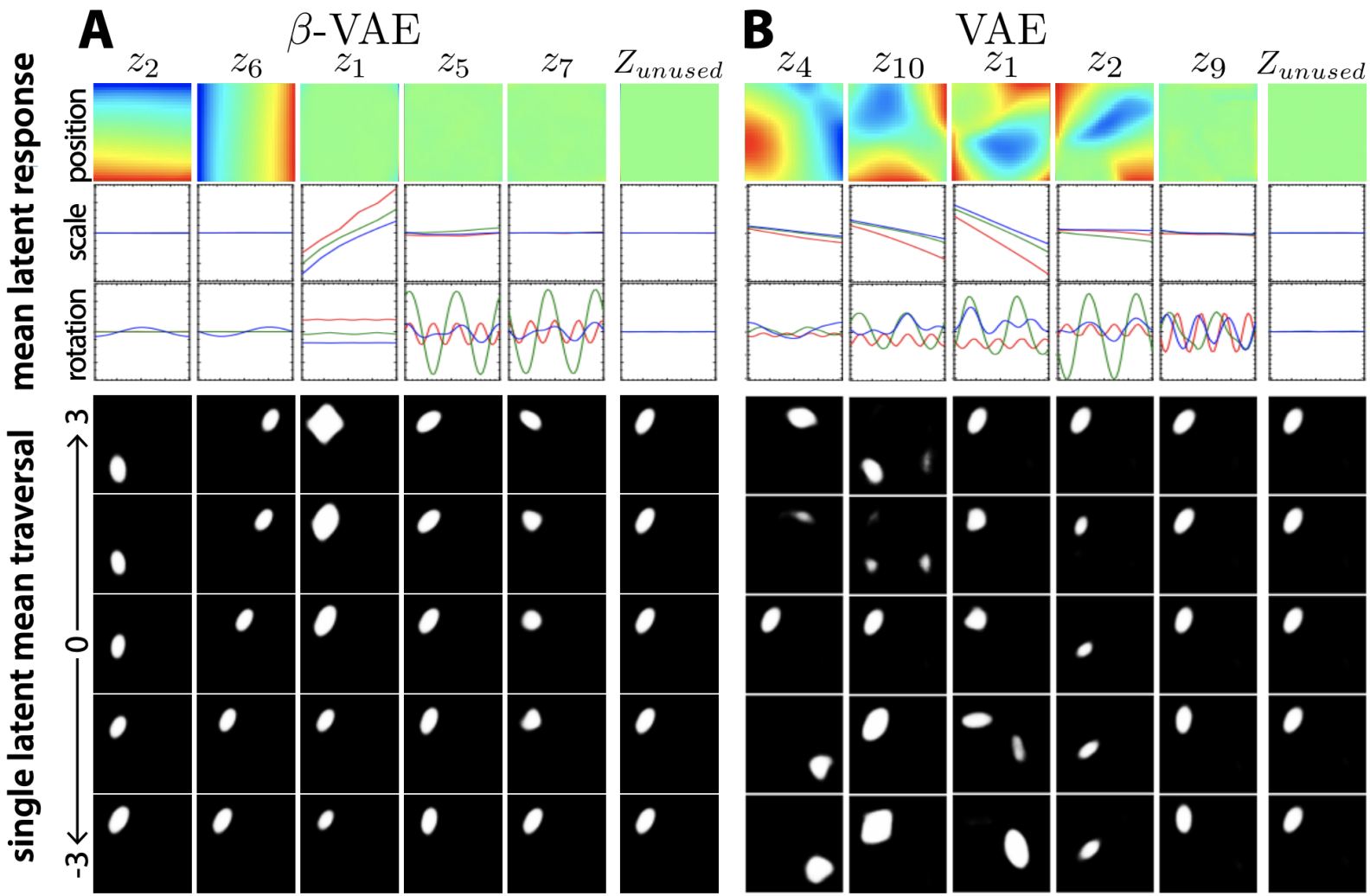
---

*Higgins I. et al. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, 2017*

# $\beta$-VAE samples


*Higgins I. et al. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, 2017*

# $\beta$-VAE analysis




*Higgins I. et al. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, 2017*

# $\beta$-VAE

## ELBO

$$\mathcal{L}(q, \boldsymbol{\theta}, \beta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - \beta \cdot KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).$$

## ELBO surgery

$$\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_i(q, \boldsymbol{\theta}, \beta) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{q(\mathbf{z}|\mathbf{x}_i)} \log p(\mathbf{x}_i|\mathbf{z}, \boldsymbol{\theta})}_{\text{Reconstruction loss}} - \underbrace{\beta \cdot \mathbb{I}_q[\mathbf{x}, \mathbf{z}]}_{\text{MI}} - \underbrace{\beta \cdot KL(q_{\text{agg}}(\mathbf{z})||p(\mathbf{z}))}_{\text{Marginal KL}}$$

## Minimization of MI

- It is not necessary and not desirable for disentanglement.
- It hurts reconstruction.

---

*Burgess C. P. et al. Understanding disentangling in $\beta$-VAE, 2018*

# $\beta$-VAE

## Disentangling metric

1. Generate two sets of objects

$$\mathbf{x}_{li} \sim \text{Sim}(\mathbf{v}_{li}, \mathbf{w}_{li}); \quad \mathbf{x}_{lj} \sim \text{Sim}(\mathbf{v}_{lj}, \mathbf{w}_{lj}); \quad y_{ij} \sim U[1, d].$$

$$\mathbf{v}_{li} \sim p(\mathbf{v}); \quad \mathbf{v}_{lj} \sim p(\mathbf{v})\,([v_{li}]_y = [v_{lj}]_y); \quad \mathbf{w}_{li}, \mathbf{w}_{lj} \sim p(\mathbf{w}).$$

2. Find representations

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mu(\mathbf{x})|\sigma^2(\mathbf{x})\right); \quad \mathbf{z}_{li} = \mu(\mathbf{x}_{li}); \quad \mathbf{z}_{lj} = \mu(\mathbf{x}_{lj}).$$

3. Use accuracy of classifier $p(y|\mathbf{z}_{\text{diff}})$ with a low VC-dimension as metric of disentanglement

$$\mathbf{z}_{\text{diff}} = \frac{1}{L}\sum_{l=1}^{L} |\mathbf{z}_{li} - \mathbf{z}_{lj}|.$$

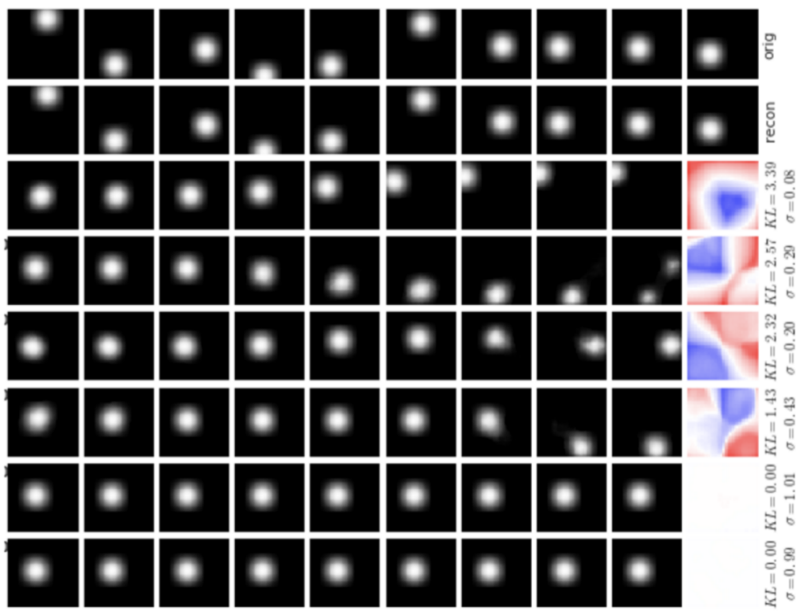
---

*Higgins I. et al. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework, 2017*

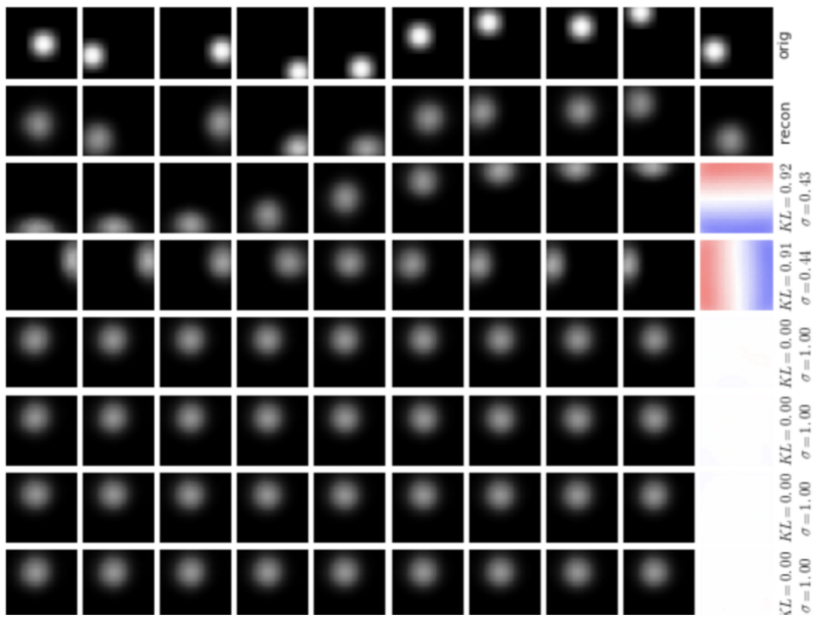
# $\beta$-VAE

- **Top row:** original images.
- **Second row:** the corresponding reconstructions.
- **Remaining rows:** latent traversals ordered by KL divergence with the prior.
- **Heatmaps:** latent activations for each 2D position.

β = 1




β = 150

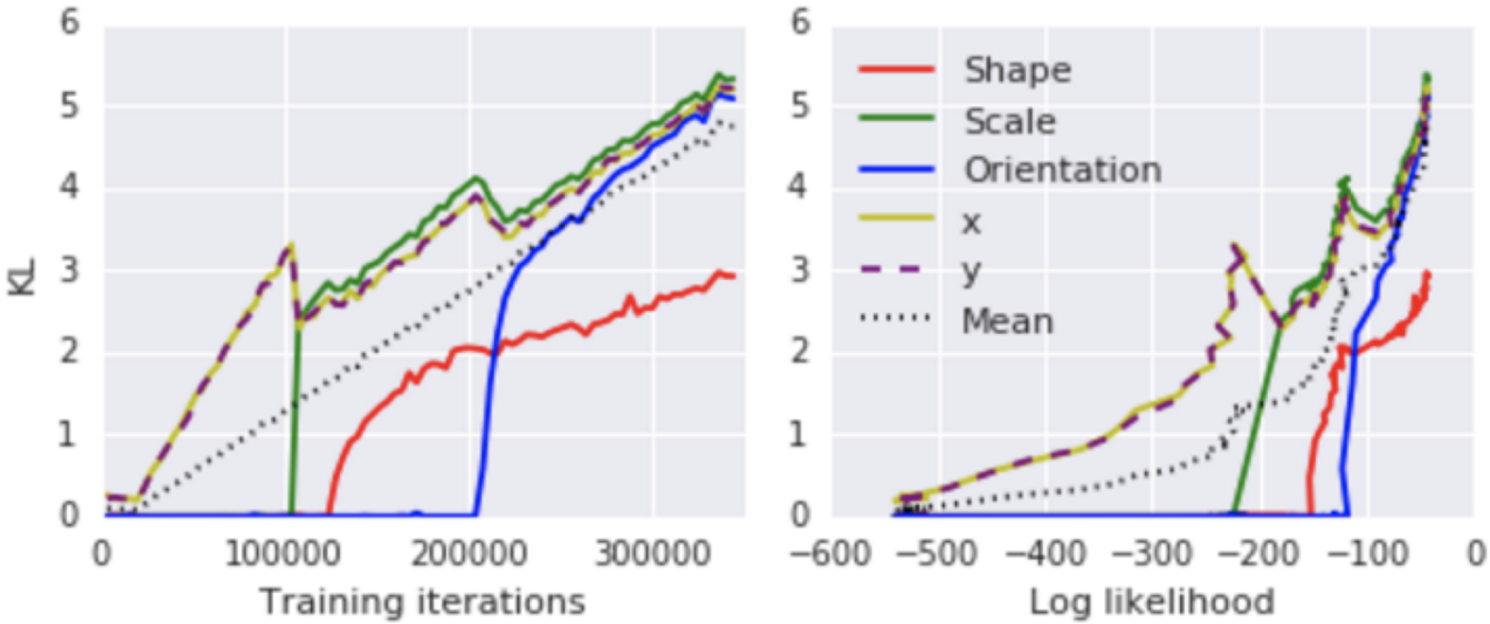



*Burgess C. P. et al. Understanding disentangling in $\beta$-VAE, 2018*

# $\beta$-VAE

## Controlled encoding capacity

$$\mathcal{L}(q, \boldsymbol{\theta}, \beta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - |KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) - C|.$$



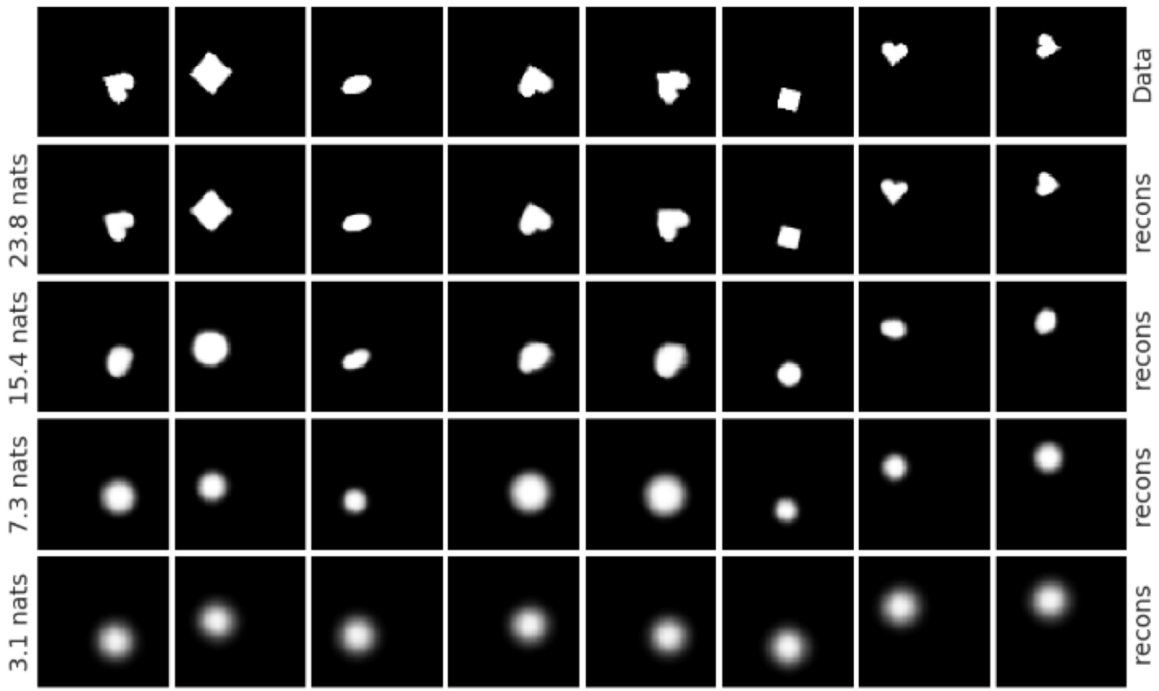

The early capacity is allocated to positional latents only, followed by a scale latent, then shape and orientation latents.

*Burgess C. P. et al. Understanding disentangling in $\beta$-VAE, 2018*

# $\beta$-VAE

## Controlled encoding capacity

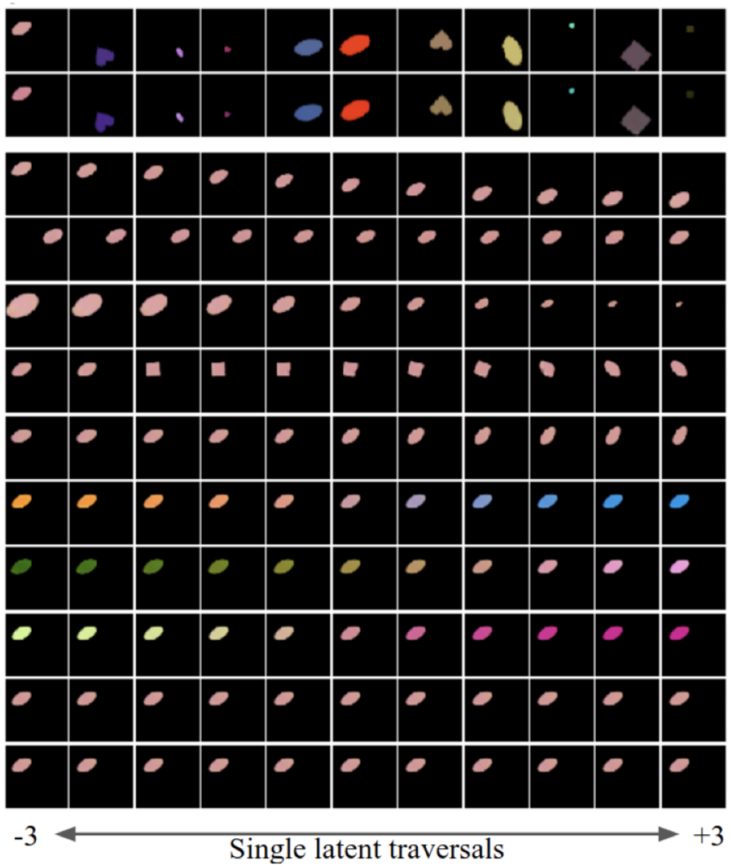
$$\mathcal{L}(q, \boldsymbol{\theta}, \beta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) - |KL(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) - C|.$$



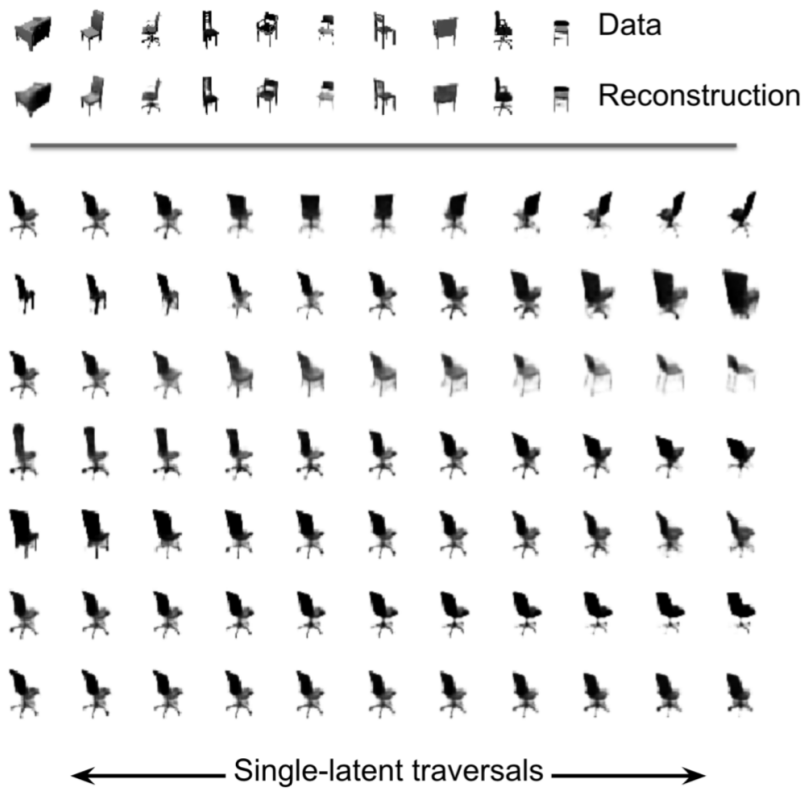

*Burgess C. P. et al. Understanding disentangling in $\beta$-VAE, 2018*

# $\beta$-VAE

(a) Coloured dSprites




(b) 3D Chairs




*Burgess C. P. et al. Understanding disentangling in $\beta$-VAE, 2018*

# DIP-VAE: disentangled posterior

## Disentangled aggregated variational posterior

$$q_{\text{agg}}(\mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n} q(\mathbf{z}|\mathbf{x}) = \prod_{j=1}^{d} q_{\text{agg}}(z_j)$$

## DIP-VAE objective

$$\mathcal{L}_{\text{DIP}}(q, \theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_i(q, \theta) - \lambda \cdot KL(q_{\text{agg}}(\mathbf{z})||p(\mathbf{z})) =$$

$$= \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left[\mathbb{E}_{q(\mathbf{z}|\mathbf{x}_i)} \log p(\mathbf{x}_i|\mathbf{z}, \theta)\right]}_{\text{Reconstruction loss}} \underbrace{- \mathbb{I}_q[\mathbf{x}, \mathbf{z}]}_{\text{MI}} \underbrace{-(1+\lambda) \cdot KL(q_{\text{agg}}(\mathbf{z})||p(\mathbf{z}))}_{\text{Marginal KL}}$$

Marginal KL term is intractable. $\Rightarrow$ Let match the moments of $q_{\text{agg}}(\mathbf{z})$ and $p(\mathbf{z})$:

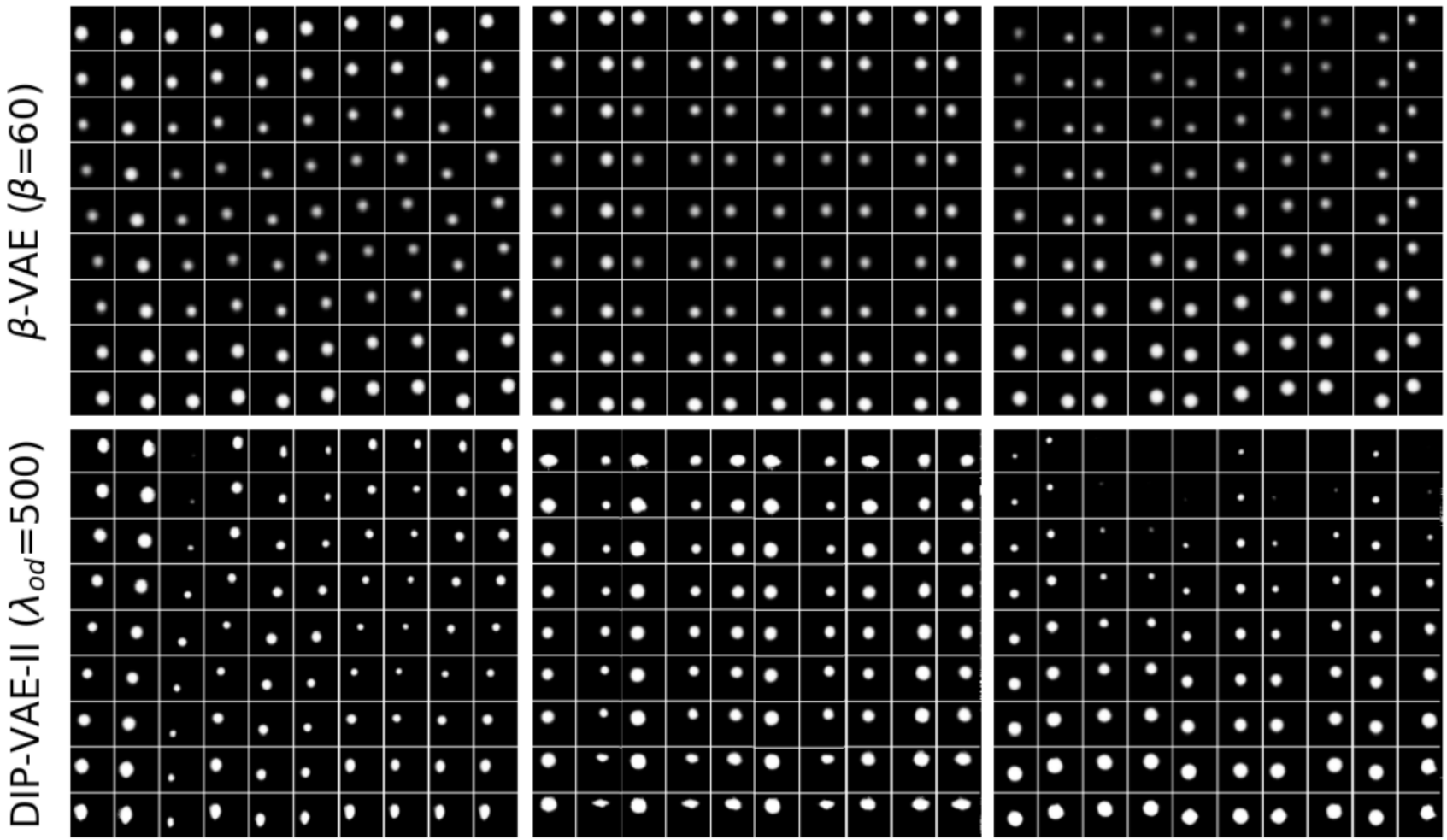
$$\text{cov}_{q_{\text{agg}}(\mathbf{z})}(\mathbf{z}) = \mathbb{E}_{q_{\text{agg}}(\mathbf{z})}\left[(\mathbf{z} - \mathbb{E}_{q_{\text{agg}}(\mathbf{z})}(\mathbf{z}))(\mathbf{z} - \mathbb{E}_{q_{\text{agg}}(\mathbf{z})}(\mathbf{z}))^T\right].$$

---

*Kumar A., Sattigeri P., Balakrishnan A. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations, 2017*

# DIP-VAE: analysis

Reconstructions become better.




*Kumar A., Sattigeri P., Balakrishnan A. Variational Inference of Disentangled Latent Concepts from Unlabeled Observations, 2017*

# FactorVAE

## Disentangled aggregated variational posterior

$$q(\mathbf{z}) = \frac{1}{n}\sum_{i=1}^{n} q(\mathbf{z}|\mathbf{x}) = \prod_{j=1}^{d} q(z_j)$$

## Total correlation regularizer

$$\min KL(q(\mathbf{z})||\prod_{j=1}^{d} q(z_j))$$

## FactorVAE objective

$$\min_{\phi,\boldsymbol{\theta}} \mathcal{L}(\phi,\boldsymbol{\theta}) - \gamma \cdot KL(q(\mathbf{z})||\prod_{j=1}^{d} q(z_j))$$

- The last term is intractable.
- FactorVAE uses density ratio trick for estimation.

---

*Kim H., Mnih A. Disentangling by Factorising, 2018*

# FactorVAE

Consider two distributions $q_1(\mathbf{x})$, $q_2(\mathbf{x})$ and probilistic model

$$p(\mathbf{x}|y) = \begin{cases} q_1(\mathbf{x}), & \text{if } y = 1, \\ q_2(\mathbf{x}), & \text{if } y = 0, \end{cases} \qquad y \sim \text{Bern}(0.5).$$

## Density ratio trick

$$\frac{q_1(\mathbf{x})}{q_2(\mathbf{x})} = \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} = \frac{p(y=1|\mathbf{x})p(\mathbf{x})}{p(y=1)} \Big/ \frac{p(y=0|\mathbf{x})p(\mathbf{x})}{p(y=0)} =$$

$$= \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} = \frac{p(y=1|\mathbf{x})}{1 - p(y=1|\mathbf{x})} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}$$

Here $D(\mathbf{x})$ could be treated as a discriminator a model the output of which is a probability that $\mathbf{x}$ is a sample from $q_1(\mathbf{x})$ rather than from $q_2(\mathbf{x})$.

---

*Kim H., Mnih A. Disentangling by Factorising, 2018*

# FactorVAE

## FactorVAE objective

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \text{ELBO}(\boldsymbol{\theta},\boldsymbol{\phi}) - \gamma \cdot KL(q(\mathbf{z})||\prod_{j=1}^{d} q(z_j))$$

## Total correlation regularizer

$$KL(q(\mathbf{z})||\prod_{j=1}^{d} q(z_j)) = KL(q(\mathbf{z})||\bar{q}(\mathbf{z})) =$$

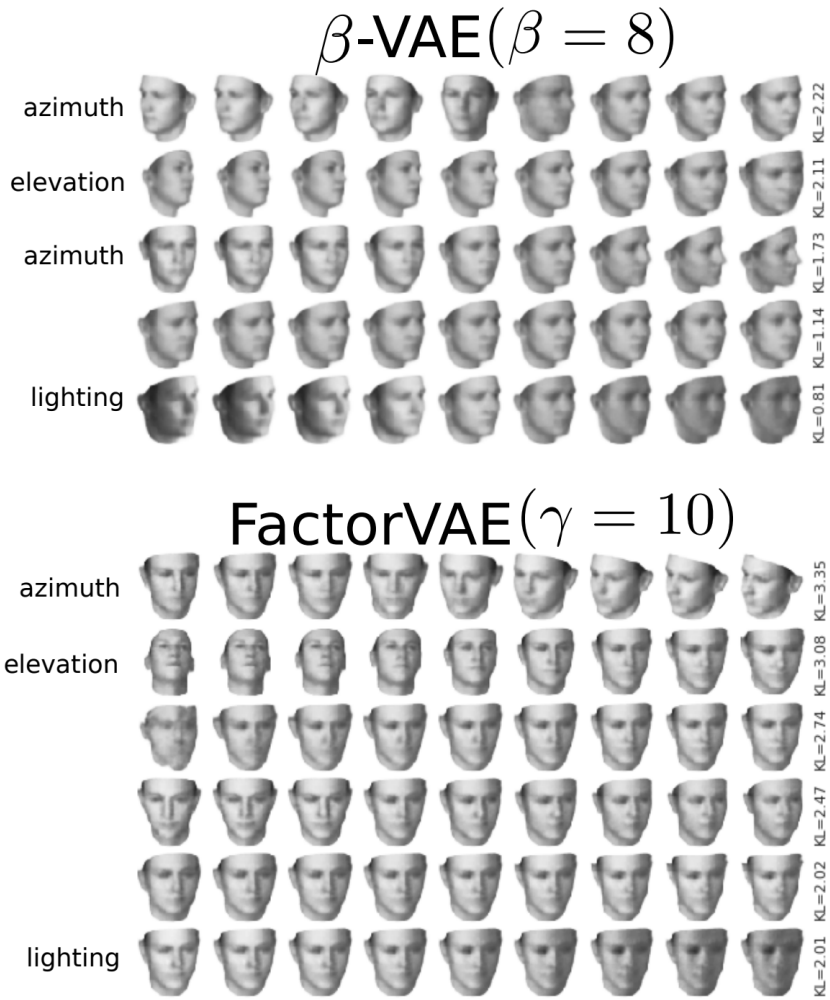
$$= \mathbb{E}_{q(\mathbf{z})} \log \frac{q(\mathbf{z})}{\bar{q}(\mathbf{z})} \approx \mathbb{E}_{q(\mathbf{z})} \log \frac{D(\mathbf{z})}{1 - D(\mathbf{z})}$$

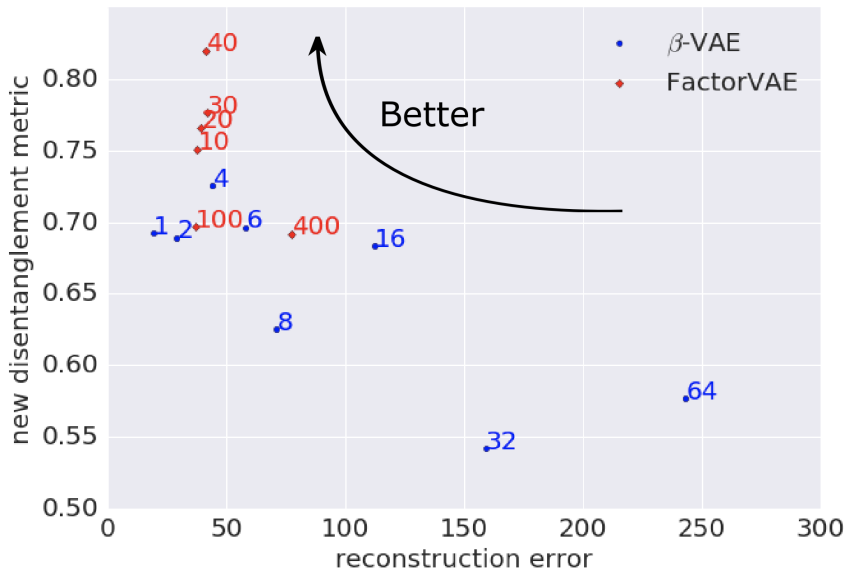
VAE and GAN are trained simultaneously.

---

*Kim H., Mnih A. Disentangling by Factorising, 2018*

# FactorVAE




*Kim H., Mnih A. Disentangling by Factorising, 2018*

# Challenging disentanglement assumptions

## Theorem

Let $\mathbf{z}$ has density $p(\mathbf{z}) = \prod_{i=1}^{d} p(z_i)$. Then, there exists an **infinite** family of bijective functions $f : \text{supp}(\mathbf{z}) \to \text{supp}(\mathbf{z})$:

- $\frac{\partial f_i(\mathbf{z})}{\partial z_j} \neq 0$ for all $i$ and $j$ ($\mathbf{z}$ and $f(\mathbf{z})$ are completely entangled);
- $P(\mathbf{z} \leq \mathbf{u}) = P(f(\mathbf{z}) \leq \mathbf{u})$ for all $\mathbf{u} \in \text{supp}(\mathbf{z})$.

Consider a generative model with disentangled representation $\mathbf{z}$.

- $\exists\ \hat{\mathbf{z}} = f(\mathbf{z})$ where $\hat{\mathbf{z}}$ is completely entangled with respect to $\mathbf{z}$.
- The disentanglement method cannot distinguish between the two equivalent generative models:

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} = \int p(\mathbf{x}|\hat{\mathbf{z}})p(\hat{\mathbf{z}})d\hat{\mathbf{z}}.$$

Theorem claims that unsupervised disentanglement learning is impossible for arbitrary generative models with a factorized prior.

---

*Locatello F. et al. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations, 2018*

# Challenging Disentanglement Assumptions

## Proof (1)

1. Consider the function $g : \text{supp}(\mathbf{z}) \to [0,1]^d$:

$$g_i(\mathbf{u}) = P(z_i \leq u_i), \quad i = 1, \ldots, d.$$

   - $g$ is bijective (since $p(\mathbf{z}) = \prod_{i=1}^{d} p(z_i)$).
   - $\frac{\partial g_i(\mathbf{u})}{\partial u_i} \neq 0$, for all $i$ and $\frac{\partial g_i(\mathbf{u})}{\partial u_j} = 0$ for all $i \neq j$.
   - $g(\mathbf{z})$ is an independent $d$-dimensional uniform distribution.

2. Consider $h : (0,1]^d \to \mathbb{R}^d$

$$h_i(\mathbf{u}) = \psi^{-1}(u_i), \quad i = 1, \ldots, d.$$

   Here $\psi$ denotes the CDF of a standard normal distribution.

   - $h$ is bijective.
   - $\frac{\partial h_i(\mathbf{u})}{\partial u_i} \neq 0$, for all $i$ and $\frac{\partial h_i(\mathbf{u})}{\partial u_j} = 0$ for all $i \neq j$.
   - $h(g(\mathbf{z}))$ is a $d$-dimensional standard normal distribution.

---

*Locatello F. et al. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations, 2018*

# Challenging Disentanglement Assumptions

## Proof (2)

Let $\mathbf{A} \in \mathbb{R}^{d \times d}$ be an arbitrary orthogonal matrix with $A_{ij} \neq 0$ for all $i, j$. The family of such matrices is infinite.

- $\mathbf{A}$ is orthogonal, it is invertible and thus defines a bijective linear operator.
- $\mathbf{A}h(g(\mathbf{z})) \in \mathbb{R}^d$ is hence an independent, multivariate standard normal distribution.
- $h^{-1}(\mathbf{A}h(g(\mathbf{z}))) \in \mathbb{R}^d$ is an independent $d$-dimensional uniform distribution.

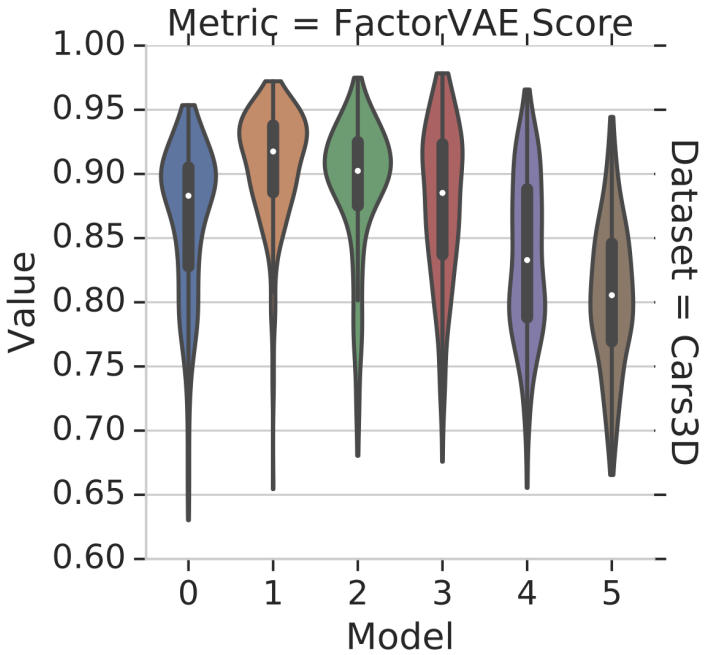
Define $f : \text{supp}(\mathbf{z}) \to \text{supp}(\mathbf{z})$:

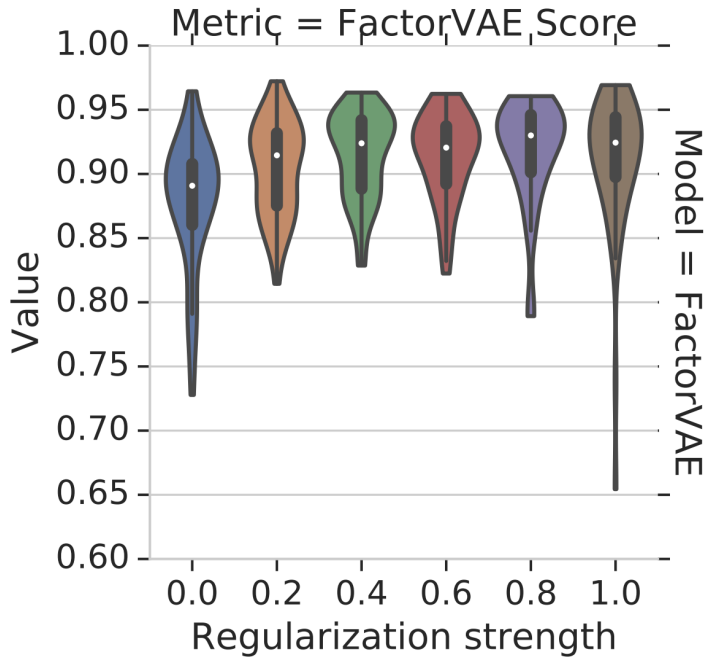
$$f(\mathbf{u}) = g^{-1}(h^{-1}(\mathbf{A}h(g(\mathbf{z})))).$$

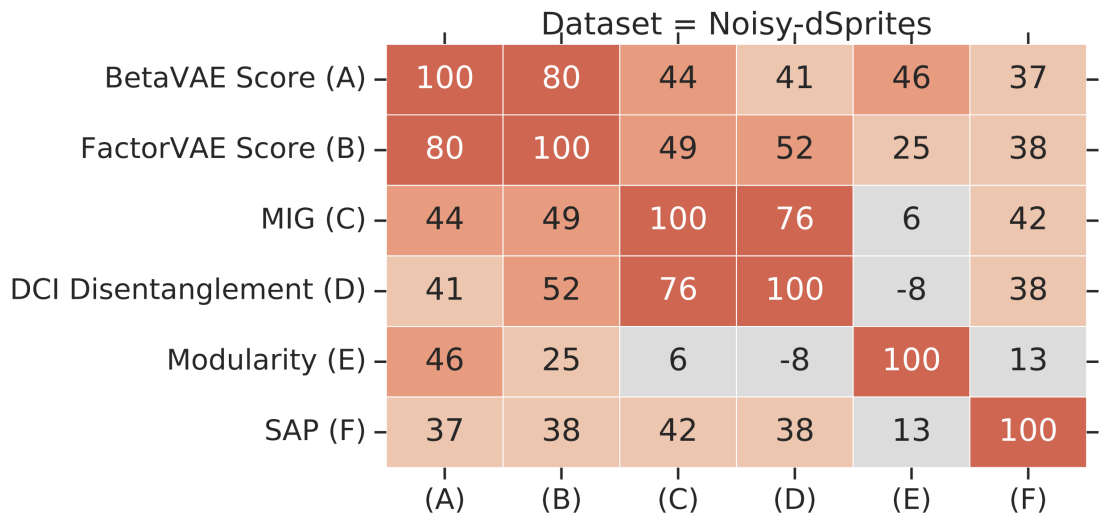
By definition $f(\mathbf{z})$ has the same marginal distribution as $\mathbf{z}$:
$P(\mathbf{z} \leq \mathbf{u}) = P(f(\mathbf{z}) \leq \mathbf{u})$ and $\frac{\partial f_i(\mathbf{z})}{\partial z_j} \neq 0$.

*Locatello F. et al. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations, 2018*

# Challenging disentanglement assumptions




*Locatello F. et al. Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations, 2018*

# Outline

# Likelihood-free learning

- Likelihood is not a perfect quality measure for generative model.
- Likelihood could be intractable.

## Where did we start

We would like to approximate true data distribution $\pi(\mathbf{x})$. Instead of searching true $\pi(\mathbf{x})$ over all probability distributions, learn function approximation $p(\mathbf{x}|\boldsymbol{\theta}) \approx \pi(\mathbf{x})$.

Imagine we have two sets of samples

- $\mathcal{S}_1 = \{\mathbf{x}_i\}_{i=1}^{n_1} \sim \pi(\mathbf{x})$ – real samples;
- $\mathcal{S}_2 = \{\mathbf{x}_i\}_{i=1}^{n_2} \sim p(\mathbf{x}|\boldsymbol{\theta})$ – generated (or fake) samples.

## Two sample test

$$H_0 : \pi(\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta}), \quad H_1 : \pi(\mathbf{x}) \neq p(\mathbf{x}|\boldsymbol{\theta})$$

Define test statistic $T(\mathcal{S}_1, \mathcal{S}_2)$. The test statistic is likelihood free. If $T(\mathcal{S}_1, \mathcal{S}_2) < \alpha$, then accept $H_0$, else reject it.

# Likelihood-free learning

## Two sample test

$$H_0 : \pi(\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta}), \quad H_1 : \pi(\mathbf{x}) \neq p(\mathbf{x}|\boldsymbol{\theta})$$

## Desired behaviour

- $p(\mathbf{x}|\boldsymbol{\theta})$ minimizes the value of test statistic $T(\mathcal{S}_1, \mathcal{S}_2)$.
- It is hard to find an appropriate test statistic in high dimensions. $T(\mathcal{S}_1, \mathcal{S}_2)$ could be learnable.

## Generative adversarial network (GAN) objective

- **Generator:** generative model $\mathbf{x} = G(\mathbf{z})$, which makes generated sample more realistic. Here $\mathbf{z} \sim p(\mathbf{z})$, $\mathbf{x} \sim p(\mathbf{x}|\boldsymbol{\theta})$.
- **Discriminator:** a classifier $D(\mathbf{x}) \in [0, 1]$, which distinguishes real samples from generated samples.

$$\min_G \max_D \left[ \mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} \log(1 - D(\mathbf{x})) \right]$$

---

*Goodfellow I. J. et al. Generative Adversarial Networks, 2014*

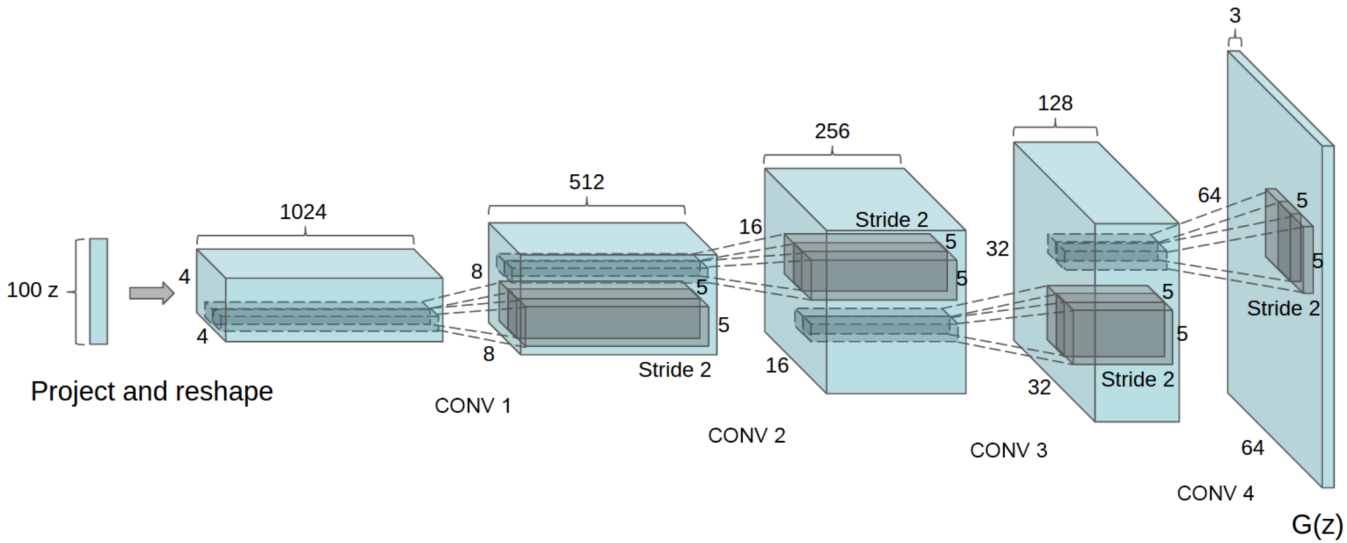
# Outline

# Vanilla GAN results


*Goodfellow I. J. et al. Generative Adversarial Networks, 2014*

# Deep Convolutional GAN

## Architecture

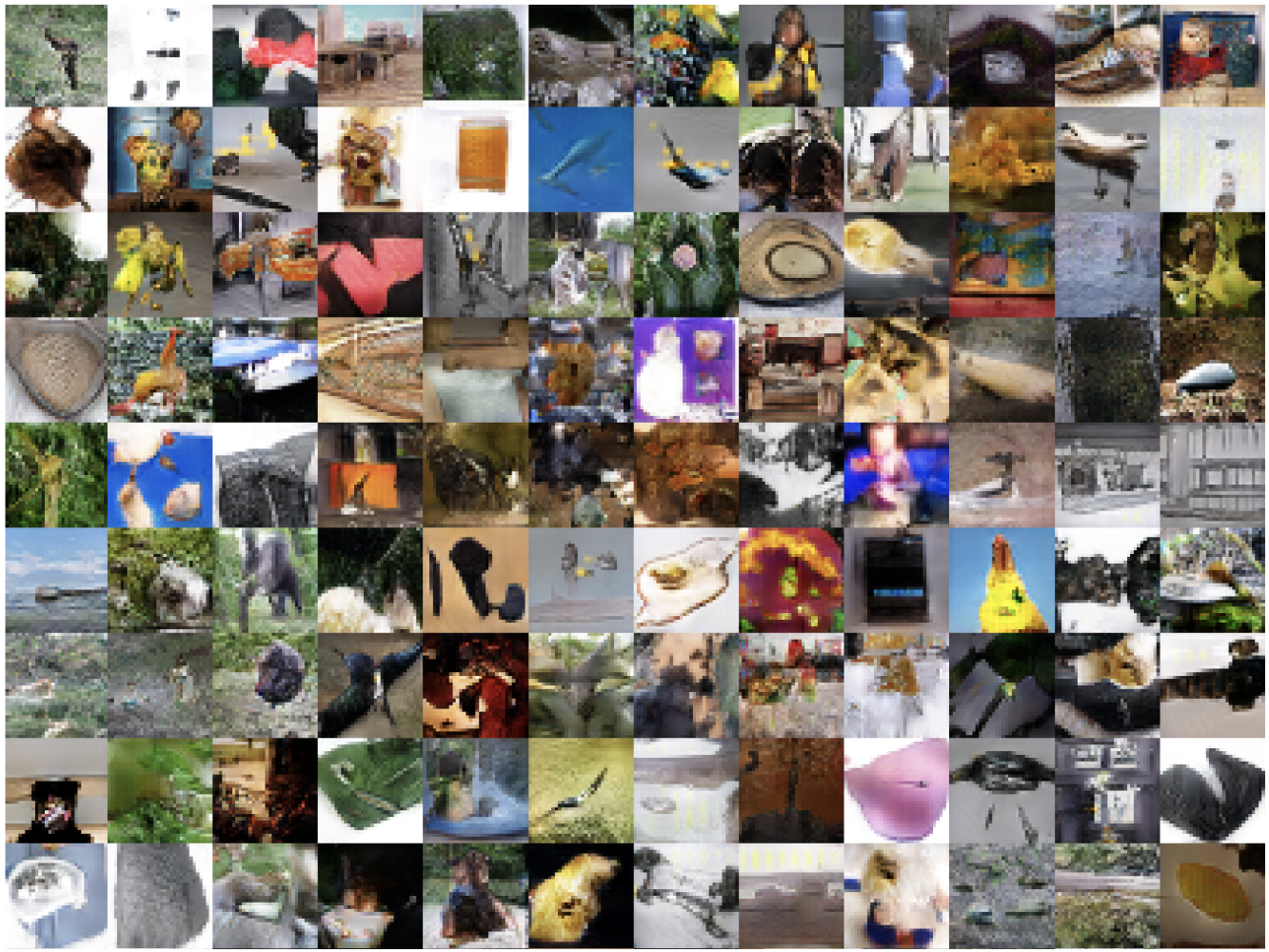



- Mean-pooling instead of max-pooling.
- Transposed convolutions in the generator for upsampling.
- Downsample with strided convolutions and average pooling.
- ReLU for generator, Leaky-ReLU (0.2) for discriminator.
- Output nonlinearity: tanh for Generator, sigmoid for discriminator.
- Batch Normalization used to prevent mode collapse (not applied at the output of $G$ and input of $D$).
- Adam: small LR = 2e-4; small momentum: 0.5, batch-size: 128.

*Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015*

# Deep Convolutional GAN

## ImageNet samples



*Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015*

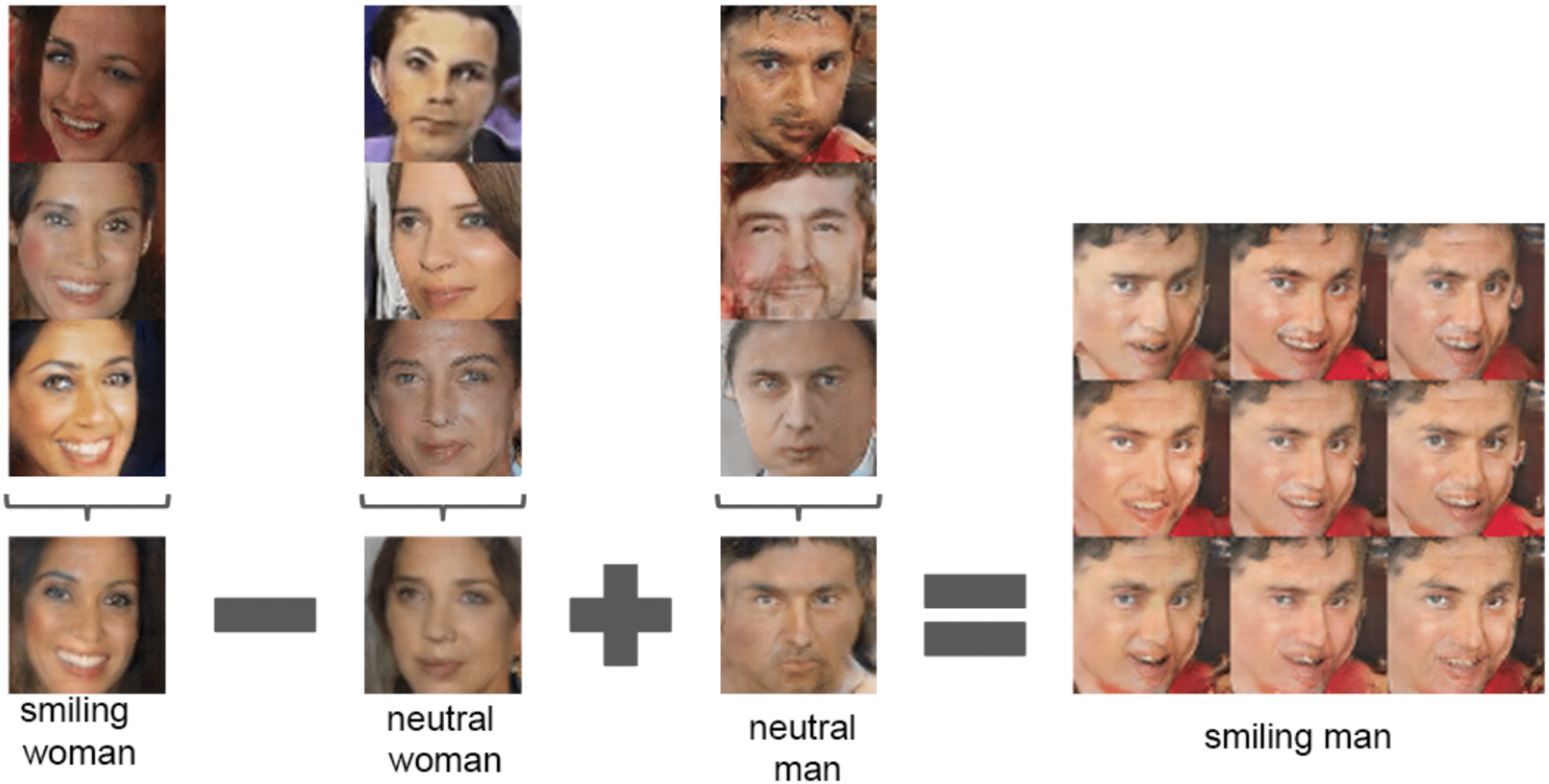
# Deep Convolutional GAN

## Smooth interpolations

*Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015*

# Deep Convolutional GAN

## Vector arithmetic




*Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015*

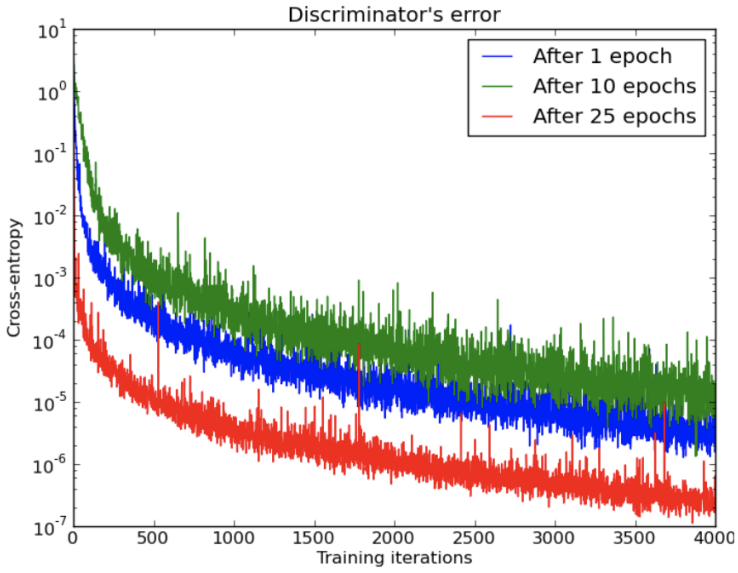
# Outline

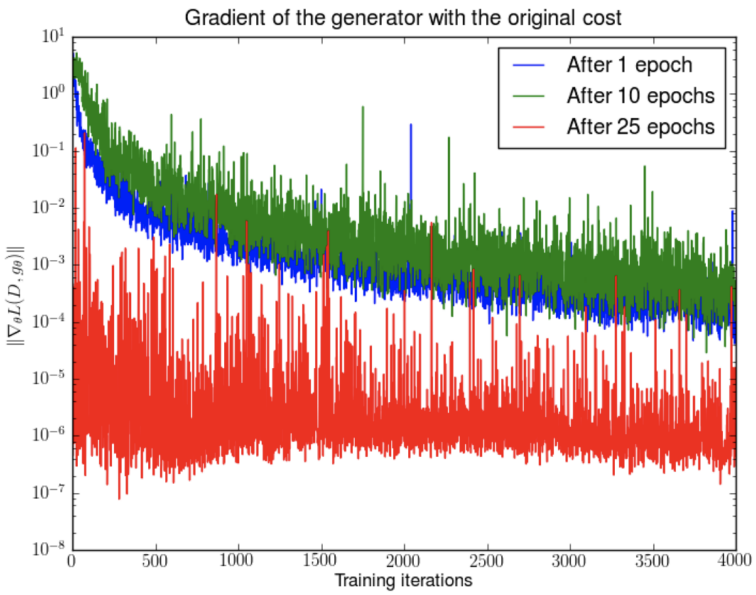
# Vanishing gradients

## Objective

$$\min_{\theta} \max_{\phi} \left[ \mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}, \phi) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z}, \theta), \phi)) \right]$$

Early in learning, $G$ is poor, $D$ can reject samples with high confidence. In this case, $\log(1 - D(G(\mathbf{z}, \theta), \phi))$ saturates.




*Arjovsky M., Bottou L. Towards Principled Methods for Training Generative Adversarial Networks, 2017*

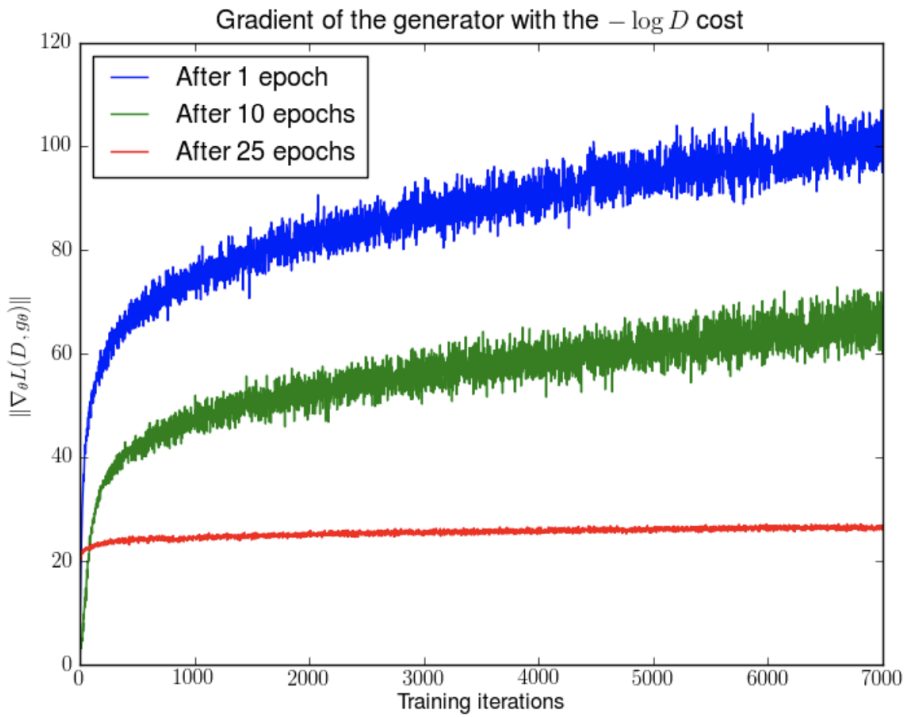
# Vanishing gradients

## Objective

$$\min_{\theta} \max_{\phi} \left[ \mathbb{E}_{\pi(\mathbf{x})} \log D(\mathbf{x}, \phi) + \mathbb{E}_{p(\mathbf{z})} \log(1 - D(G(\mathbf{z}, \theta), \phi)) \right]$$

## Non-saturating GAN

- Maximize $\log D(G(z))$ instead of minimizing $\log(1 - D(G(\mathbf{z})))$.
- Gradients are getting much stronger, but the training is unstable (with increasing mean and variance).




*Arjovsky M., Bottou L. Towards Principled Methods for Training Generative Adversarial Networks, 2017*

# Outline

# Improved techniques for training GANs

- Feature matching

$$\mathcal{L}_G = \|\mathbb{E}_{\pi(\mathbf{x})}\mathbf{d}(\mathbf{x}) - \mathbb{E}_{p(\mathbf{z})}\mathbf{d}(G(\mathbf{z}))\|_2^2$$

Here $\mathbf{d}(\mathbf{x})$ – intermediate layer of discriminator. Matching the learned discriminator statistics instead of the output of the discriminator. Helps to avoid the vanishing gradients for sufficiently good discriminator.

- Historical averaging adds extra loss term for generator and discriminator losses

$$\|\boldsymbol{\theta} - \frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\theta}_t\|_2^2.$$

Here $\boldsymbol{\theta}_t$ – value of parameters at the previous step $t$. It allows to stabilize training procedure.

---

*Salimans T. et al. Improved Techniques for Training GANs, 2016*

# Improved techniques for training GANs

- One-sided label smoothing. Instead of using one-hot labels in classification, use $(1 - \alpha)$ for real data (the generated samples are not smoothed).

$$D^*(\mathbf{x}) = \frac{(1-\alpha)\pi(\mathbf{x})}{\pi(\mathbf{x}) + p(\mathbf{x}|\boldsymbol{\theta})}$$

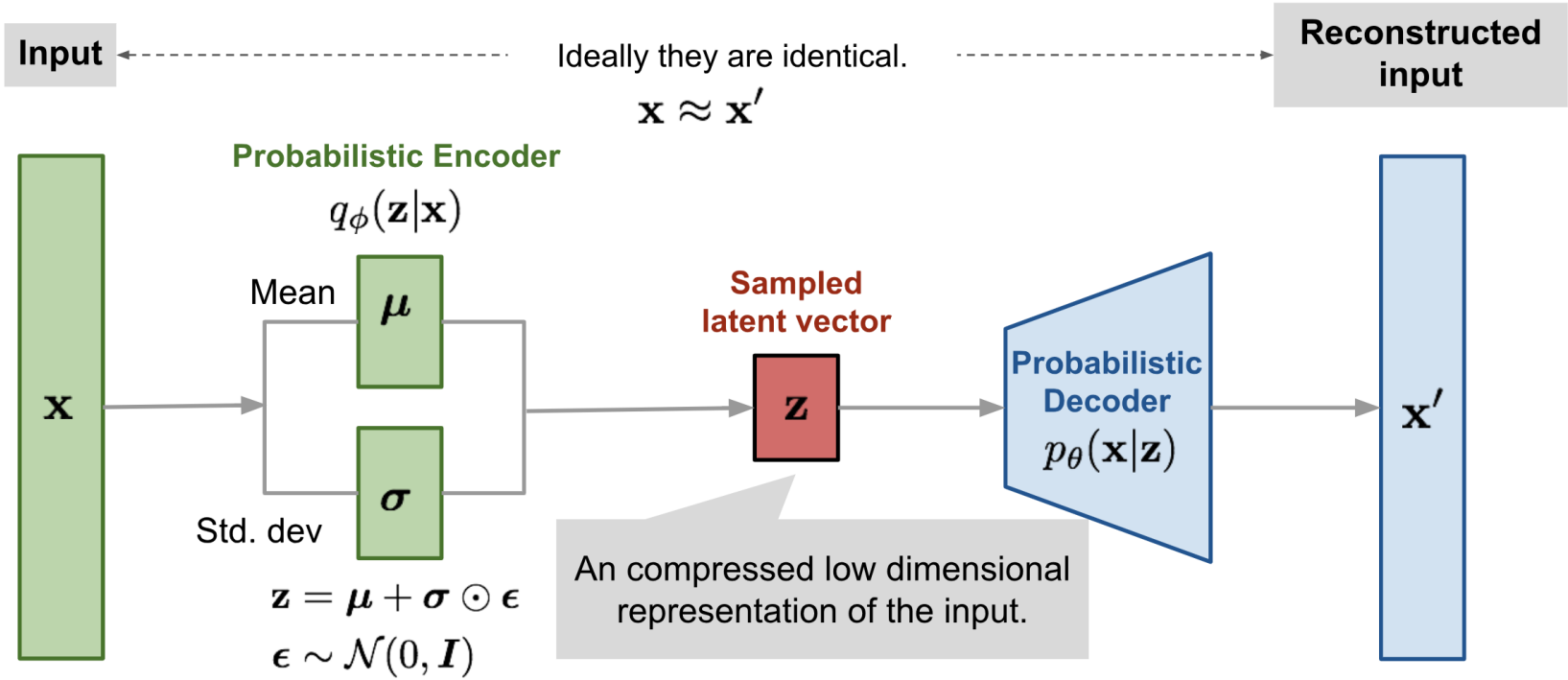
- Virtual batch normalization. BatchNorm makes samples within minibatch are highly correlated.

Use reference fixed batch to compute the normalization statistics. To avoid overfitting construct batch with the reference batch and the current sample.

*Salimans T. et al. Improved Techniques for Training GANs, 2016*

# Outline

# VAE recap




- Encoder $q(\mathbf{z}|\mathbf{x}, \phi) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\sigma}_\phi(\mathbf{x}))$.
- Variational posterior $q(\mathbf{z}|\mathbf{x}, \phi)$ originally approximates the true posterior $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$.
- Which methods are you already familiar with to make the posterior is more flexible?

---

*image credit:*
*https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html*

# Adversarial Variational Bayes

## ELBO objective

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

What is the problem to make the variational posterior model an implicit model?

- The first term is the reconstruction loss that needs only samples from $q(\mathbf{z}|\mathbf{x}, \phi)$ to evaluate.
- Reparametrization trick allows to get gradients of the reconstruction loss

$$\nabla_\phi \int q(\mathbf{z}|\mathbf{x}, \phi) f(\mathbf{z}) d\mathbf{z} = \nabla_\phi \int r(\epsilon) f(\mathbf{z}) d\epsilon$$

$$= \int r(\epsilon) \nabla_\phi f(g_\phi(\mathbf{x}, \epsilon)) d\epsilon \approx \nabla_\phi f(g_\phi(\mathbf{x}, \epsilon^*)),$$

where $\epsilon^* \sim r(\epsilon), \quad \mathbf{z} = g_\phi(\mathbf{x}, \epsilon), \quad \mathbf{z} \sim q(\mathbf{z}|\mathbf{x}, \phi)$.

---

*Mescheder L., Nowozin S., Geiger A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, 2017*

# Adversarial Variational Bayes

## ELBO objective

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

What is the problem to make the variational posterior model an implicit model?

- The second term requires the explicit value of $q(\mathbf{z}|\mathbf{x}, \phi)$.
- We could join second and third terms:

$$KL = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log \frac{q(\mathbf{z}|\mathbf{x}, \phi)}{p(\mathbf{z})} = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log \frac{q(\mathbf{z}|\mathbf{x}, \phi)\pi(\mathbf{x})}{p(\mathbf{z})\pi(\mathbf{x})}.$$

- We have to estimate density ratio

$$r(\mathbf{x}, \mathbf{z}) = \frac{q_1(\mathbf{x}, \mathbf{z})}{q_2(\mathbf{x}, \mathbf{z})} = \frac{q(\mathbf{z}|\mathbf{x}, \phi)\pi(\mathbf{x})}{p(\mathbf{z})\pi(\mathbf{x})}.$$

---

*Mescheder L., Nowozin S., Geiger A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, 2017*

# Density ratio trick

Consider two distributions $q_1(\mathbf{x})$, $q_2(\mathbf{x})$ and probabilistic model

$$p(\mathbf{x}|y) = \begin{cases} q_1(\mathbf{x}), & \text{if } y = 1, \\ q_2(\mathbf{x}), & \text{if } y = 0, \end{cases} \qquad y \sim \text{Bern}(0.5).$$

## Density ratio

$$\frac{q_1(\mathbf{x})}{q_2(\mathbf{x})} = \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} = \frac{p(y=1|\mathbf{x})p(\mathbf{x})}{p(y=1)} \Big/ \frac{p(y=0|\mathbf{x})p(\mathbf{x})}{p(y=0)} =$$

$$= \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} = \frac{p(y=1|\mathbf{x})}{1 - p(y=1|\mathbf{x})} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})}$$

Here $D(\mathbf{x})$ is a discriminator model the output of which is a probability that $\mathbf{x}$ is a sample from $q_1(\mathbf{x})$ rather than from $q_2(\mathbf{x})$.

$$\max_D \left[\mathbb{E}_{q_1(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{q_2(\mathbf{x})} \log(1 - D(\mathbf{x}))\right]$$

---

*Mescheder L., Nowozin S., Geiger A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, 2017*

# Density ratio trick

$$r(\mathbf{x}, \mathbf{z}) = \frac{q_1(\mathbf{x}, \mathbf{z})}{q_2(\mathbf{x}, \mathbf{z})} = \frac{q(\mathbf{z}|\mathbf{x}, \phi)\pi(\mathbf{x})}{p(\mathbf{z})\pi(\mathbf{x})}.$$

## Density ratio

$$\frac{q_1(\mathbf{x}, \mathbf{z})}{q_2(\mathbf{x}, \mathbf{z})} = \frac{p(y = 1|\mathbf{x}, \mathbf{z})}{1 - p(y = 1|\mathbf{x}, \mathbf{z})} = \frac{D(\mathbf{x}, \mathbf{z})}{1 - D(\mathbf{x}, \mathbf{z})}$$

## Adversarial Variational Bayes

$$\max_D \left[\mathbb{E}_{\pi(\mathbf{x})}\mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \log D(\mathbf{x}, \mathbf{z}) + \mathbb{E}_{\pi(\mathbf{x})}\mathbb{E}_{p(\mathbf{z})} \log(1 - D(\mathbf{x}, \mathbf{z}))\right]$$

Monte-Carlo estimation for KL divergence:

$$KL(q(\mathbf{z}|\mathbf{x}, \phi)||p(\mathbf{z})) \approx \frac{D(\mathbf{x}, \mathbf{z})}{1 - D(\mathbf{x}, \mathbf{z})}.$$

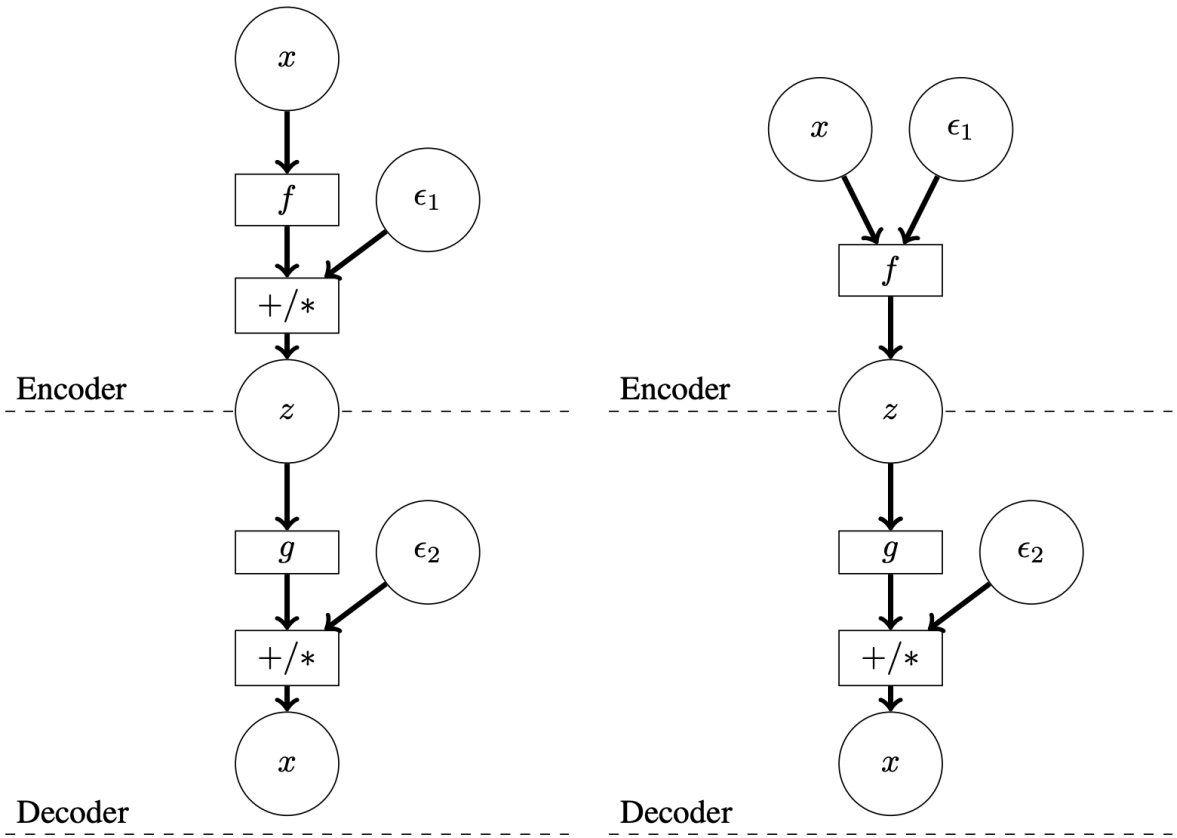
---

*Mescheder L., Nowozin S., Geiger A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, 2017*

# Adversarial Variational Bayes

## ELBO objective

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x},\phi)} \left[ \log p(\mathbf{x}|\mathbf{z}, \theta) - \log \frac{q(\mathbf{z}|\mathbf{x}, \phi)}{p(\mathbf{z})} \right] \to \max_{\phi,\theta}.$$

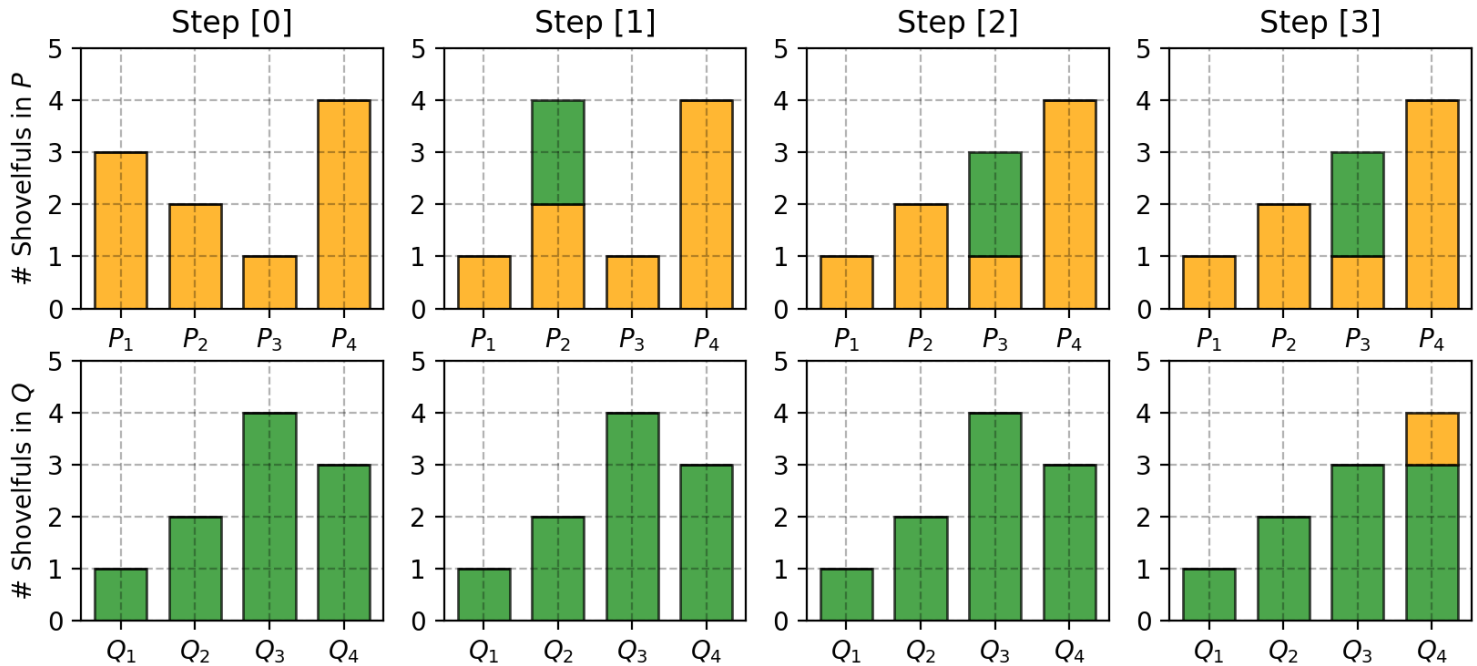



*Mescheder L., Nowozin S., Geiger A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks, 2017*

# Outline

# Wasserstein distance (discrete)

A.k.a. **Earth Mover's distance**. The minimum cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution.




$$W(P, Q) = 2(\text{step } 1) + 2(\text{step } 2) + 1(\text{step } 3) = 5$$

*Weng L. From GAN to WGAN, 2019*

# Wasserstein GAN

**Algorithm 1** WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

**Require:** : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size. $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.

**Require:** : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.

**while** $\theta$ has not converged **do**
  **for** $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  **end for**
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
**end while**

*Arjovsky M., Chintala S., Bottou L. Wasserstein GAN, 2017*

# Wasserstein GAN with Gradient Penalty

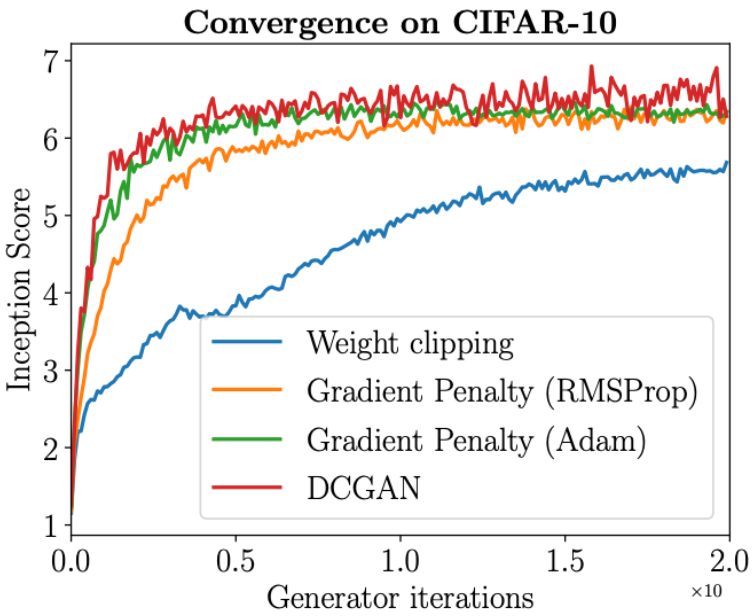
**Algorithm 1** WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

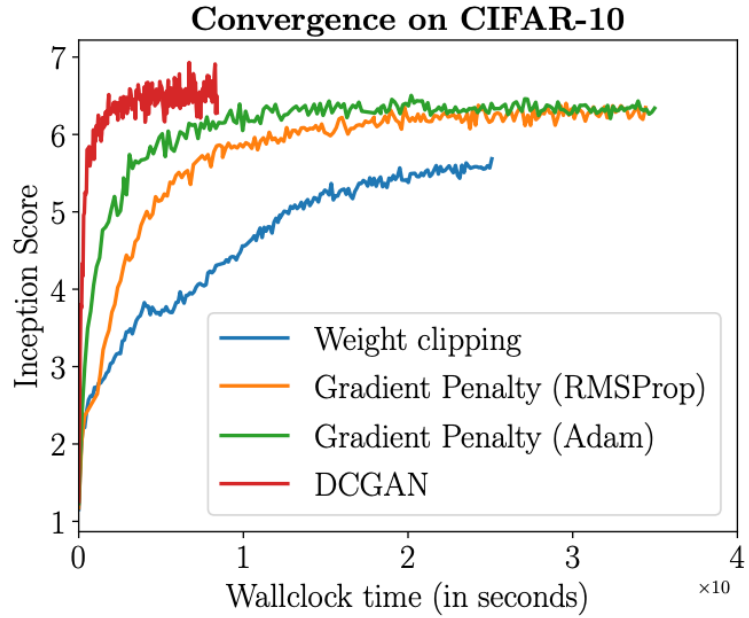
**Require:** The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
**Require:** initial critic parameters $w_0$, initial generator parameters $\theta_0$.

**while** $\theta$ has not converged **do**
  **for** $t = 1, ..., n_{\text{critic}}$ **do**
    **for** $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    **end for**
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  **end for**
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
**end while**

*Gulrajani I. et al. Improved Training of Wasserstein GANs, 2017*

# Wasserstein GAN with Gradient Penalty



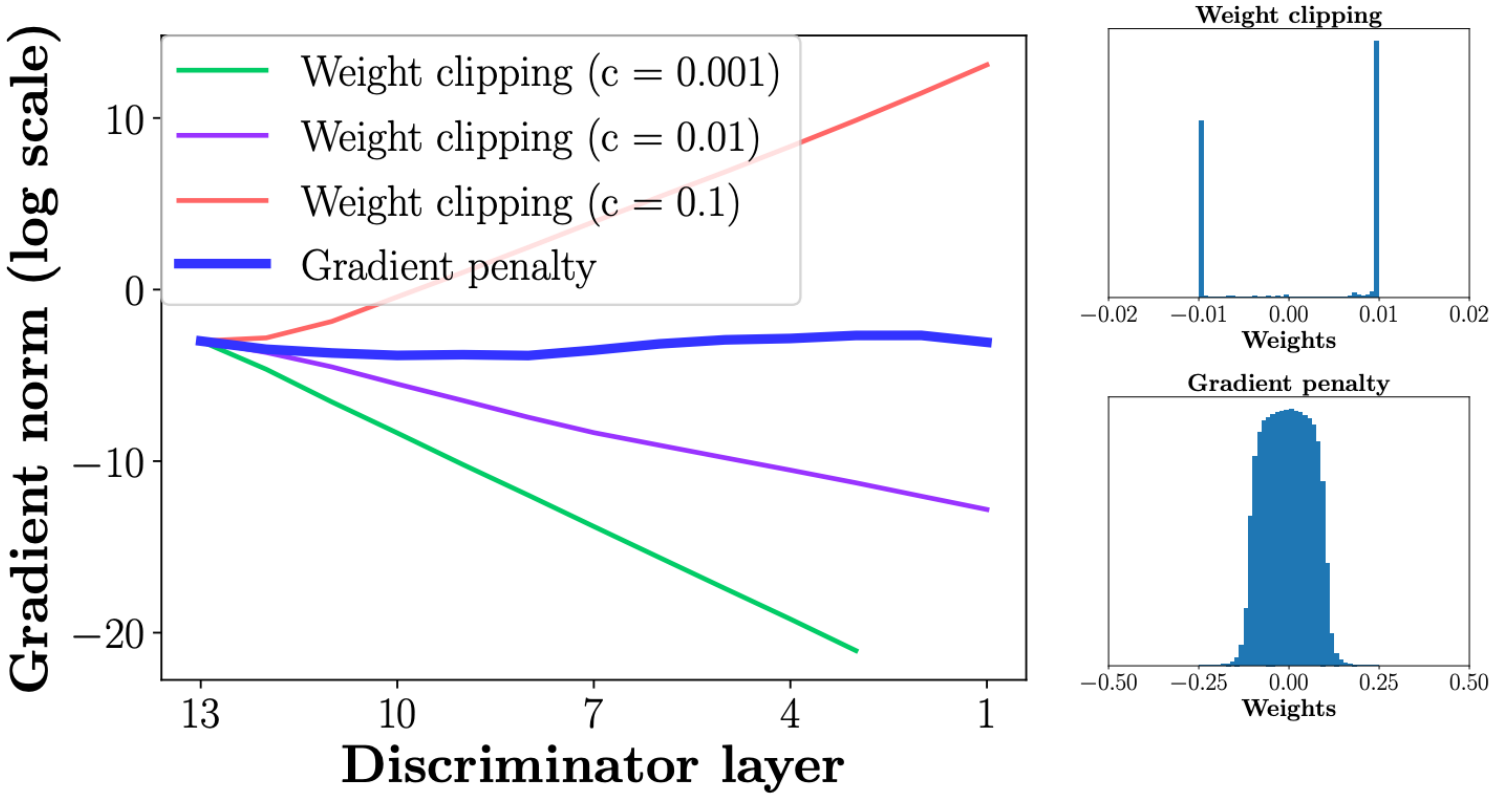

## WGAN-GP convergence

| Min. score | Only GAN | Only WGAN-GP | Both succeeded | Both failed |
|---|---|---|---|---|
| 1.0 | 0 | 8 | 192 | 0 |
| 3.0 | 1 | 88 | 110 | 1 |
| 5.0 | 0 | 147 | 42 | 11 |
| 7.0 | 1 | 104 | 5 | 90 |
| 9.0 | 0 | 0 | 0 | 200 |

*Gulrajani I. et al. Improved Training of Wasserstein GANs, 2017*

# Outline

# Wasserstein GAN with Gradient Penalty




## Weight clipping analysis

- The gradients either grow or decay exponentially.
- Gradient penalty makes the gradients more stable.

---

*Gulrajani I. et al. Improved Training of Wasserstein GANs, 2017*

# Wasserstein GAN with Gradient Penalty

## Theorem

Let $\pi(\mathbf{x})$ and $p(\mathbf{x})$ be two distribution in $\mathcal{X}$, a compact metric space. Let $\gamma$ be the optimal transportation plan between $\pi(\mathbf{x})$ and $p(\mathbf{x})$. Then

1. there is 1-Lipschitz function $f^*$ which is the optimal solution of
$$\max_{\|f\|_L \leq 1} \left[\mathbb{E}_{\pi(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{p(\mathbf{x})} f(\mathbf{x})\right].$$
2. if $f^*$ is differentiable, $\gamma(\mathbf{y} = \mathbf{z}) = 0$ and $\hat{\mathbf{x}}_t = t\mathbf{y} + (1-t)\mathbf{z}$ with $\mathbf{y} \sim \pi(\mathbf{x})$, $\mathbf{z} \sim p(\mathbf{x}|\boldsymbol{\theta})$, $t \in [0, 1]$ it holds that
$$\mathbb{P}_{(\mathbf{y},\mathbf{z})\sim\gamma}\left[\nabla f^*(\hat{\mathbf{x}}_t) = \frac{\mathbf{z} - \hat{\mathbf{x}}_t}{\|\mathbf{z} - \hat{\mathbf{x}}_t\|}\right] = 1.$$

## Corollary

$f^*$ has gradient norm 1 almost everywhere under $\pi(\mathbf{x})$ and $p(\mathbf{x})$.

---

*Gulrajani I. et al. Improved Training of Wasserstein GANs, 2017*

# Wasserstein GAN with Gradient Penalty

A differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere.

## Gradient penalty

$$W(\pi||p) = \underbrace{\mathbb{E}_{\pi(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{p(\mathbf{x})} f(\mathbf{x})}_{\text{original critic loss}} + \lambda \underbrace{\mathbb{E}_{U[0,1]} \left[ (\|\nabla f(\hat{\mathbf{x}})\|_2 - 1)^2 \right]}_{\text{gradient penalty}},$$

- Samples $\hat{\mathbf{x}}_t = t \cdot \mathbf{y} + (1-t) \cdot \mathbf{z}$ with $t \in [0, 1]$ are uniformly sampled along straight lines between pairs of points: $\mathbf{y} \sim \pi(\mathbf{x})$ and $\mathbf{z} \sim p(\mathbf{x}|\boldsymbol{\theta})$.
- Enforcing the unit gradient norm constraint everywhere is intractable, it turns out to be sifficient to enforce it only along these straight lines.

---

*Gulrajani I. et al. Improved Training of Wasserstein GANs, 2017*

# Outline

# Spectral Normalization GAN

## Definition

$\|\mathbf{A}\|_2$ is a *spectral norm* of matrix $\mathbf{A}$:

$$\|\mathbf{A}\|_2 = \max_{\mathbf{h}\neq 0} \frac{\|\mathbf{A}\mathbf{h}\|_2}{\|\mathbf{h}\|_2} = \max_{\|\mathbf{h}\|_2 \leq 1} \|\mathbf{A}\mathbf{h}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^T\mathbf{A})},$$

where $\lambda_{\max}(\mathbf{A}^T\mathbf{A})$ is the largest eigenvalue value of $\mathbf{A}^T\mathbf{A}$.

## Statement 1

if $\mathbf{g}$ is a K-Lipschitz vector function then

$$\|\mathbf{g}\|_L \leq K = \sup_{\mathbf{x}} \|\nabla \mathbf{g}(\mathbf{x})\|_2.$$

## Statement 2

Lipschitz norm of superposition is bounded above by product of Lipschitz norms

$$\|\mathbf{g}_1 \circ \mathbf{g}_2\|_L \leq \|\mathbf{g}_1\|_L \cdot \|\mathbf{g}_2\|_L$$

---

*Miyato T. et al. Spectral Normalization for Generative Adversarial Networks, 2018*

# Spectral Normalization GAN

Let consider the critic $f_\phi(\mathbf{x})$ of the following form:

$$f_\phi(\mathbf{x}) = \mathbf{W}_{K+1}\sigma_K(\mathbf{W}_K\sigma_{K-1}(\dots\sigma_1(\mathbf{W}_1\mathbf{x})\dots)).$$

This feedforward network is a superposition of simple functions.

- $\sigma_k$ is a pointwise nonlinearities. We assume that $\|\sigma_k\|_L = 1$ (it holds for ReLU).
- $\mathbf{g}(\mathbf{x}) = \mathbf{W}^T\mathbf{x}$ is a linear transformation ($\nabla\mathbf{g}(\mathbf{x}) = \mathbf{W}$).

$$\|\mathbf{g}\|_L \leq \sup_{\mathbf{x}} \|\nabla\mathbf{g}(\mathbf{x})\|_2 = \|\mathbf{W}\|_2.$$

## Critic spectral norm

$$\|f\|_L \leq \|\mathbf{W}_{K+1}\|_2 \cdot \prod_{k=1}^{K} \|\sigma_k\|_L \cdot \|\mathbf{W}_k\|_2 = \prod_{k=1}^{K+1} \|\mathbf{W}_k\|_2.$$

If we replace the weights in the critic $f_\phi(\mathbf{x})$ by $\mathbf{W}_k^{SN} = \mathbf{W}_k/\|\mathbf{W}_k\|_2$, we will get $\|f\|_L \leq 1$.

---

*Miyato T. et al. Spectral Normalization for Generative Adversarial Networks, 2018*

# Spectral Normalization GAN

How to compute $\|\mathbf{W}\|_2 = \sqrt{\lambda_{\max}(\mathbf{W}^T\mathbf{W})}$?
We are not able to apply SVD at each iteration.

## Power iteration (PI) method

- $\mathbf{u}_0$ – random vector.
- for $m = 0, \ldots, M-1$: ($M$ is a fixed number of steps)

$$\mathbf{v}_{m+1} = \frac{\mathbf{W}^T\mathbf{u}_m}{\|\mathbf{W}^T\mathbf{u}_m\|}, \quad \mathbf{u}_{m+1} = \frac{\mathbf{W}\mathbf{v}_{m+1}}{\|\mathbf{W}\mathbf{v}_{m+1}\|}.$$

- approximate the spectral norm

$$\|\mathbf{W}\|_2 = \sqrt{\lambda_{\max}(\mathbf{W}^T\mathbf{W})} \approx \mathbf{u}_M^T\mathbf{W}\mathbf{v}_M.$$

## SNGAN gradient update

- Apply PI method to get approximation of spectral norm.
- Normalize weights $\mathbf{W}_k^{SN} = \mathbf{W}_k / \|\mathbf{W}_k\|_2$.
- Apply gradient rule to $\mathbf{W}$.

---

*Miyato T. et al. Spectral Normalization for Generative Adversarial Networks, 2018*

# Spectral Normalization GAN

**Algorithm 1** SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer $l$:
  1. Apply power iteration method to a unnormalized weight $W^l$:

$$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2 \tag{20}$$

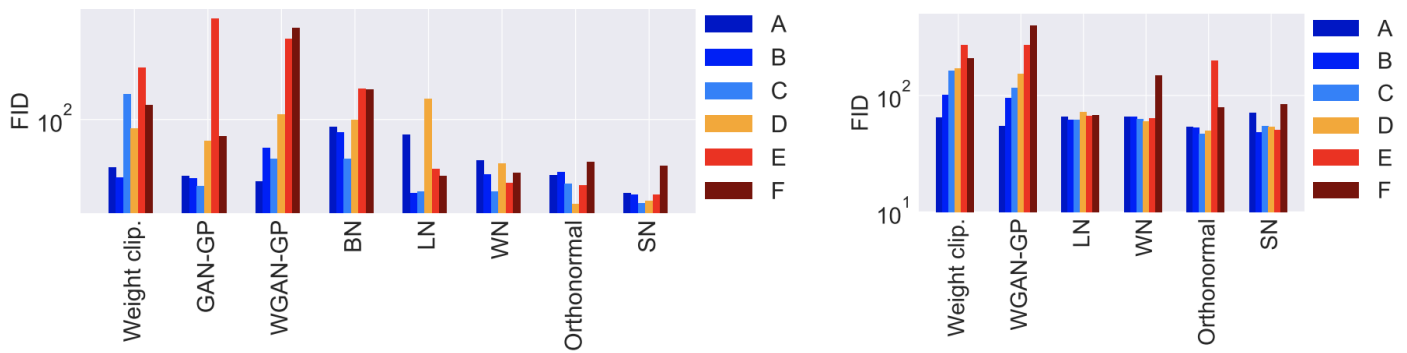
$$\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2 \tag{21}$$

  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l \tag{22}$$

  3. Update $W^l$ with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate $\alpha$:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M) \tag{23}$$




(a) CIFAR-10 (b) STL-10

*Miyato T. et al. Spectral Normalization for Generative Adversarial Networks, 2018*

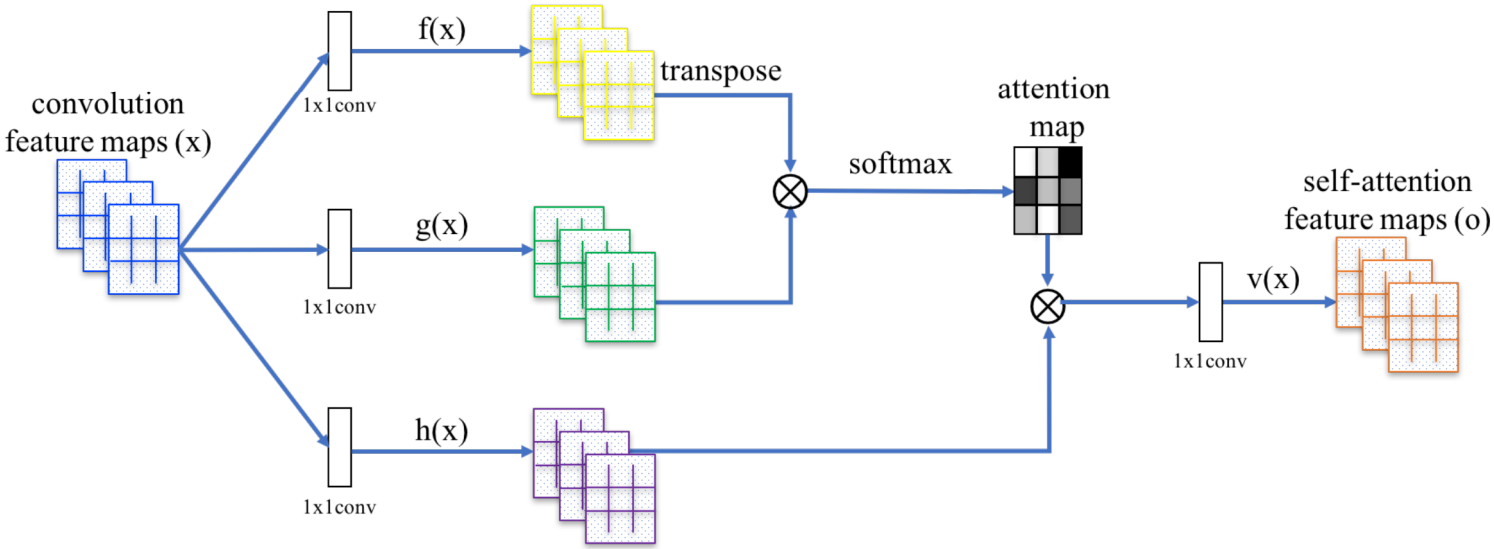
# Outline

# Evolution of GANs


- **Standard GAN** https://arxiv.org/abs/1406.2661
- **DCGAN** https://arxiv.org/abs/1511.06434
- **CoGAN** https://arxiv.org/abs/1606.07536
- **ProGAN** https://arxiv.org/abs/1710.10196
- **StyleGAN** https://arxiv.org/abs/1812.04948

# Self-Attention GAN

Convolutional layers process the information in a local neighborhood $\Rightarrow$ inefficient for modeling long-range dependencies in images.



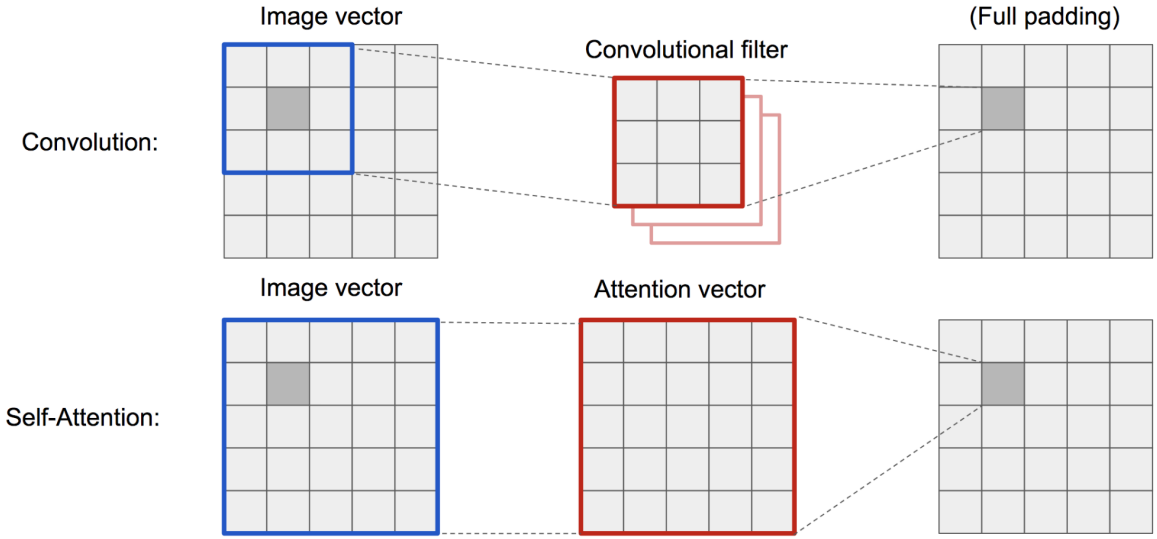

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}_f\mathbf{x}, \quad \mathbf{g}(\mathbf{x}) = \mathbf{W}_g\mathbf{x}, \quad \mathbf{h}(\mathbf{x}) = \mathbf{W}_h\mathbf{x}, \quad \mathbf{v}(\mathbf{x}) = \mathbf{W}_v\mathbf{x}$$

$$s_{ij} = \mathbf{f}(\mathbf{x}_i)^T\mathbf{g}(\mathbf{x}_j), \quad a_{ij} = \frac{\exp s_{ij}}{\sum_{i=1}^{N}\exp s_{ij}}, \quad \mathbf{o}_j = \mathbf{v}\left(\sum_{i=1}^{N} a_{ij}\mathbf{h}(\mathbf{x}_i)\right)$$

*Zhang H. et al. Self-Attention Generative Adversarial Networks, 2018*

# Self-Attention GAN

## Convolution vs Attention




## Visualization of attention maps

*image credit: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html*
*Zhang H. et al. Self-Attention Generative Adversarial Networks, 2018*

# BigGAN

## Batch-size is matter

| Batch | Ch. | Param (M) | Shared | Skip-$z$ | Ortho. | Itr $\times 10^3$ | FID | IS |
|---|---|---|---|---|---|---|---|---|
| 256 | 64 | 81.5 | SA-GAN Baseline | | | 1000 | 18.65 | 52.52 |
| 512 | 64 | 81.5 | ✗ | ✗ | ✗ | 1000 | 15.30 | 58.77($\pm$1.18) |
| 1024 | 64 | 81.5 | ✗ | ✗ | ✗ | 1000 | 14.88 | 63.03($\pm$1.42) |
| 2048 | 64 | 81.5 | ✗ | ✗ | ✗ | 732 | 12.39 | 76.85($\pm$3.83) |
| 2048 | 96 | 173.5 | ✗ | ✗ | ✗ | 295($\pm$18) | 9.54($\pm$0.62) | 92.98($\pm$4.27) |

## Samples (512x512)

*Brock A., Donahue J., Simonyan K. Large Scale GAN Training for High Fidelity Natural Image Synthesis, 2018*

# Progressive Growing GAN

## Problems with HR image generation

- Disjoint manifolds $\Rightarrow$ gradient problem.
- Small minibatch $\Rightarrow$ training instability.

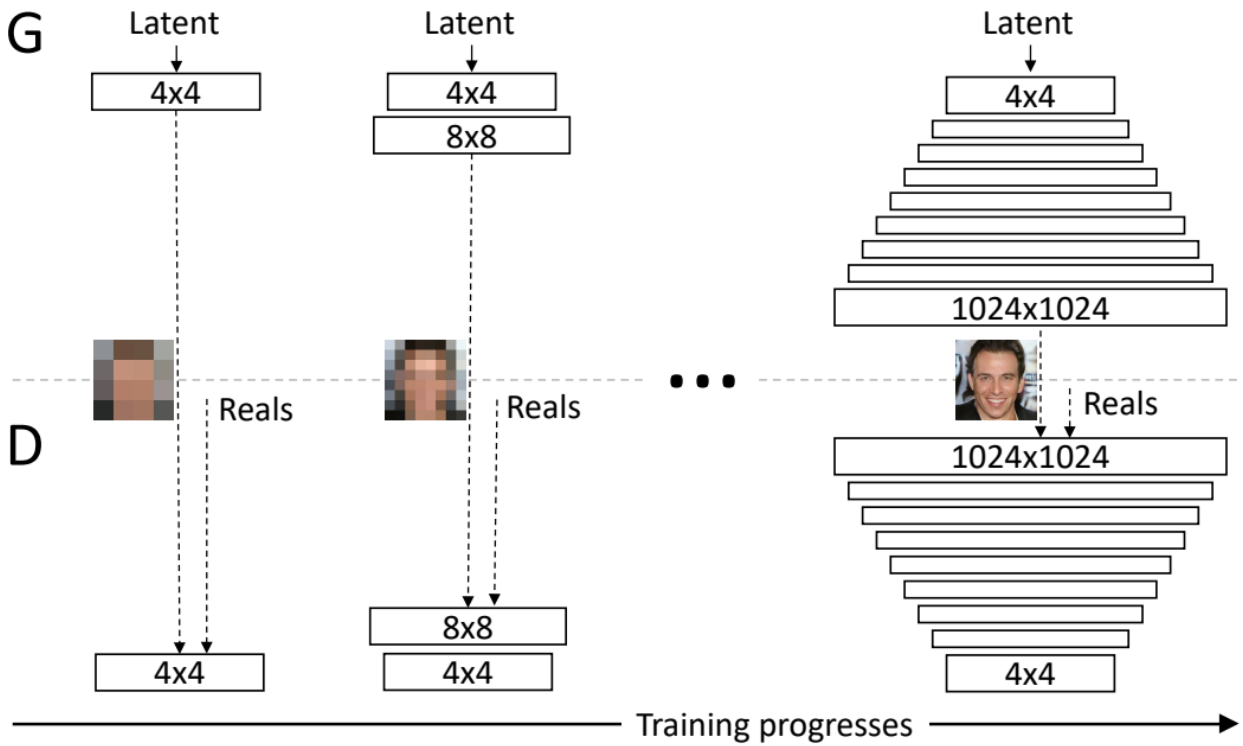
## Samples (1024x1024)

*Karras T. et al. Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017*

# Progressive Growing GAN

Grow both the generator and discriminator progressively, new layers will introduce higher-resolution details as the training progresses.

- Train GAN which generate 4x4 images (2 convs for G and D).
- Add upsampling layers to G, downsampling layers to D.
- Train GAN which generate 8x8 images.
- etc.




*Karras T. et al. Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017*

# StyleGAN

- Generating of HR images is hard.
- Progressive growing greatly simplifies the task.
- The ability to control specific features of the generated image is very limited.

## Face image features

- Coarse (pose, general hair style, face shape). Resolution $4^2 - 8^2$.
- Middle (finer facial features, hair style, eyes open/closed). Resolution $16^2 - 32^2$.
- Fine (color scheme (eye, hair and skin) and micro features). Resolution $64^2 - 1024^2$.

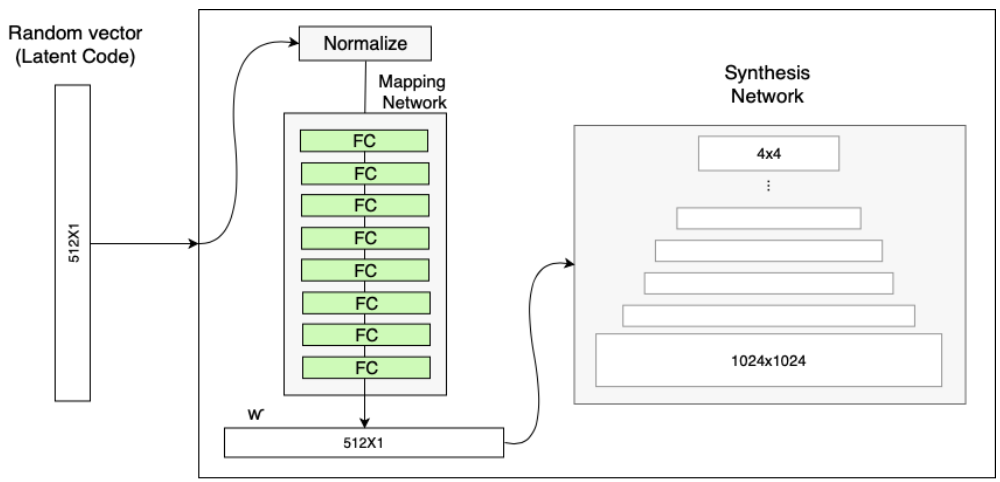
---

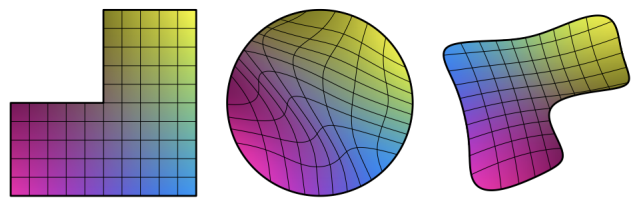
*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN

## Mapping Network

- Generator input is likely to be **disentangled**. Each component of input vector **z** should be responsible for one generative factor.
- Mapping network $f : \mathcal{Z} \to \mathcal{W}$ is used to reduce correlations between components of **z**.



(a) Distribution of features in training set

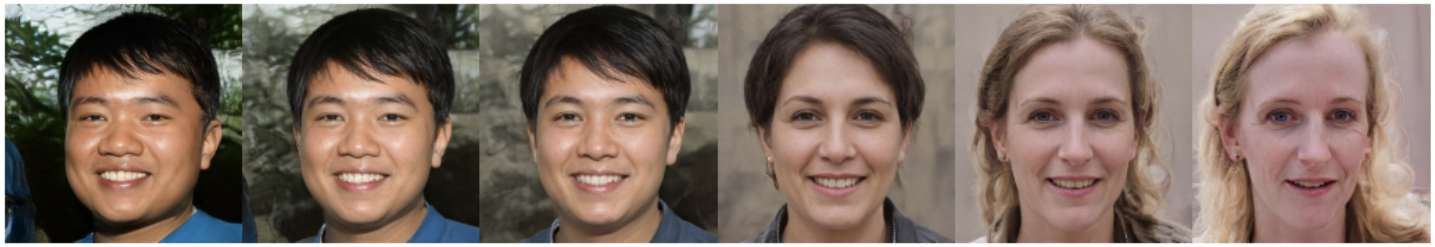
(b) Mapping from $\mathcal{Z}$ to features

(c) Mapping from $\mathcal{W}$ to features

*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN

## Truncation trick



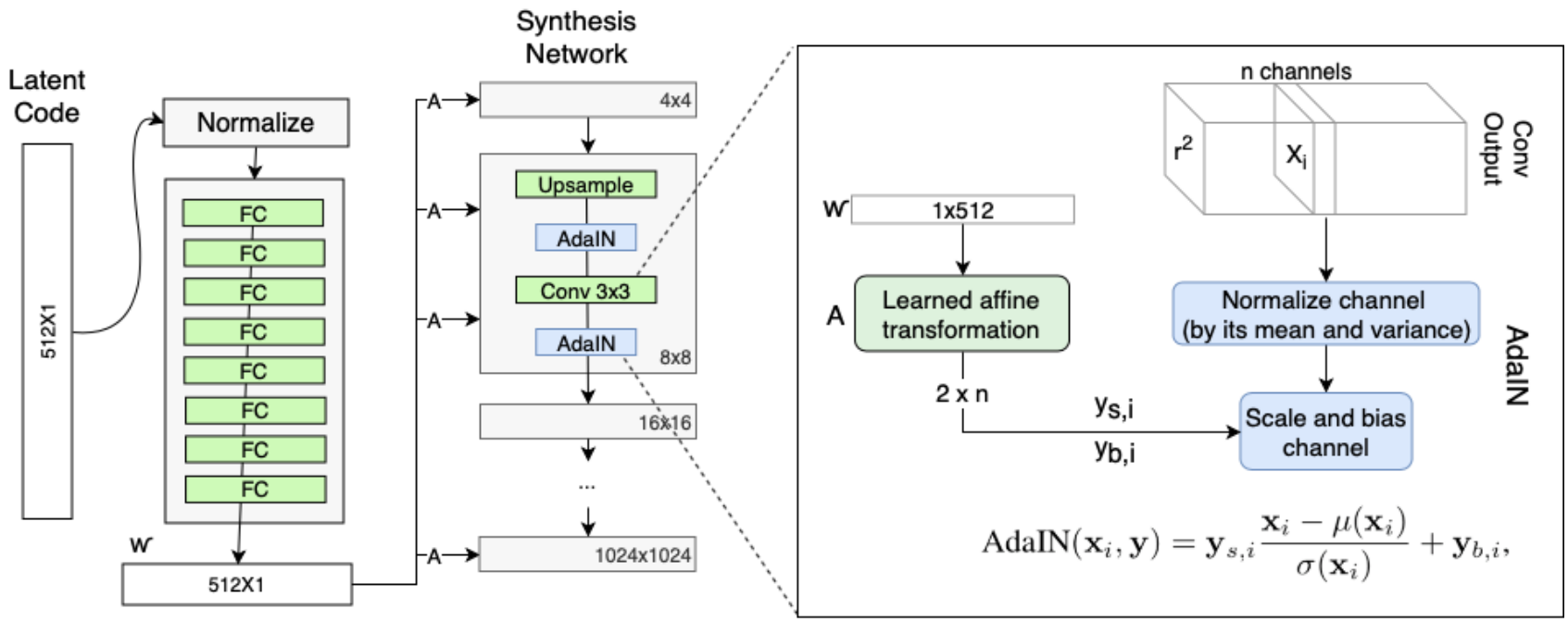

## Samples (1024x1024)

*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN

## Step 2: Style modulation

- Adaptive Instance Normalization transfers the **w** vector to the synthesis Network.
- The module is added to each resolution to define the visual expression of the features.

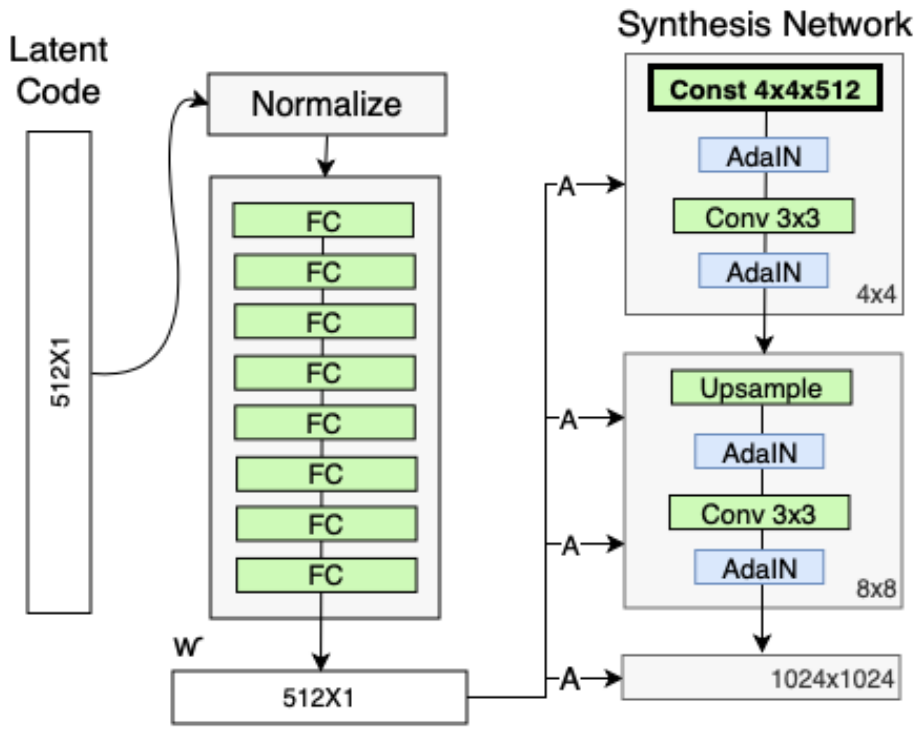



*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN

## Step 3: Remove traditional input

Mapping network provides stochasticity to different stages of the synthesis network. Input of the synthesis network is a trainable vector.

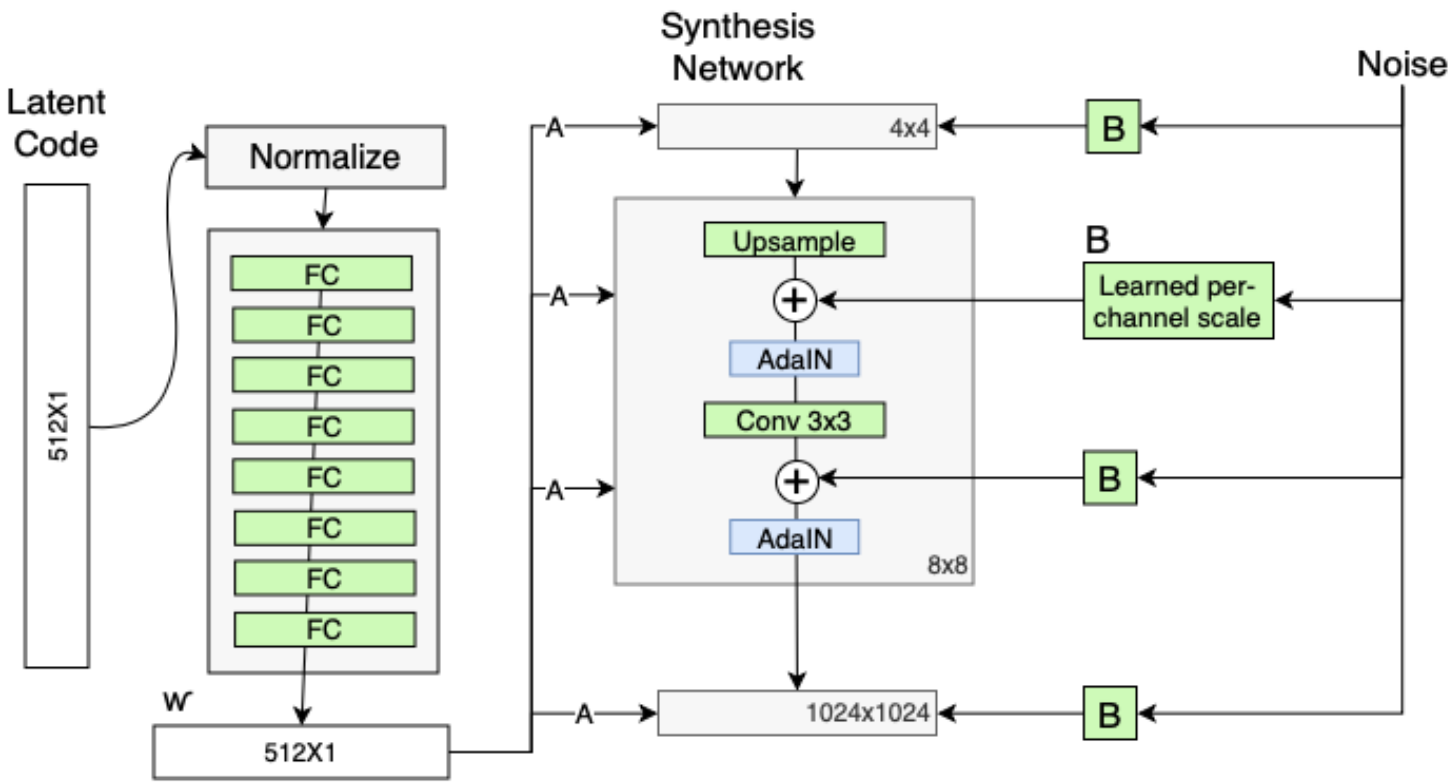



*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN

## Step 4: Stochastic variation

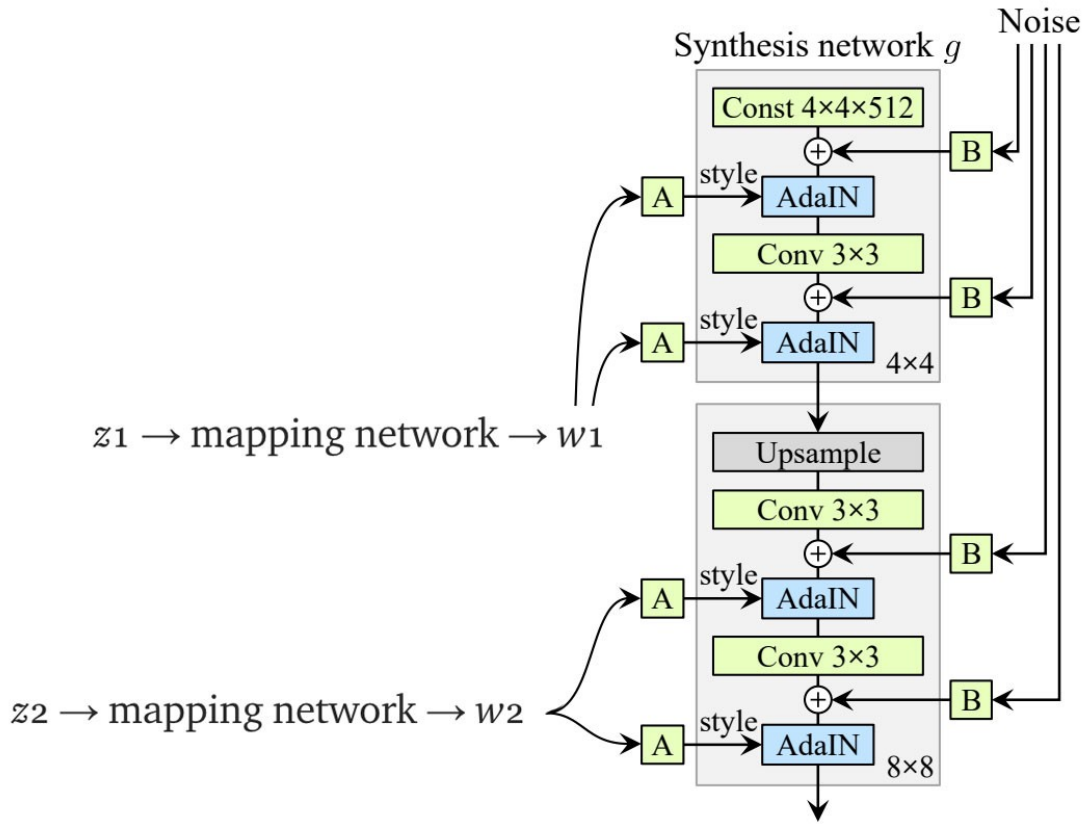
Inject random noise to add small aspects, such as freckles, exact placement of hairs, wrinkles, features which make the image more realistic and increase the variety of outputs.




*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN

## Step 4: Style Mixing

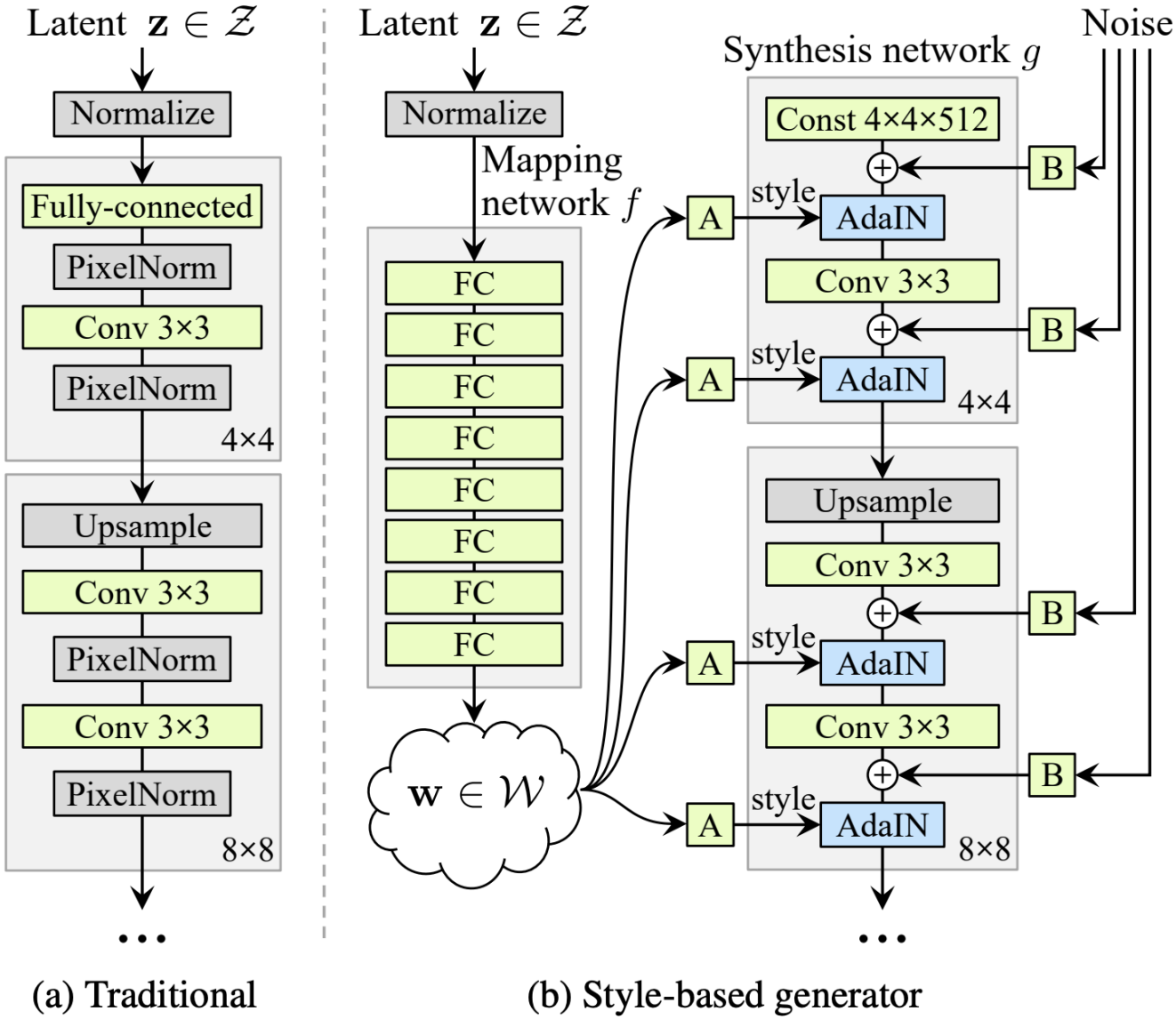



- Makes different levels of synthesis network to be independent.
- Allows to couple diffirent styles.

---

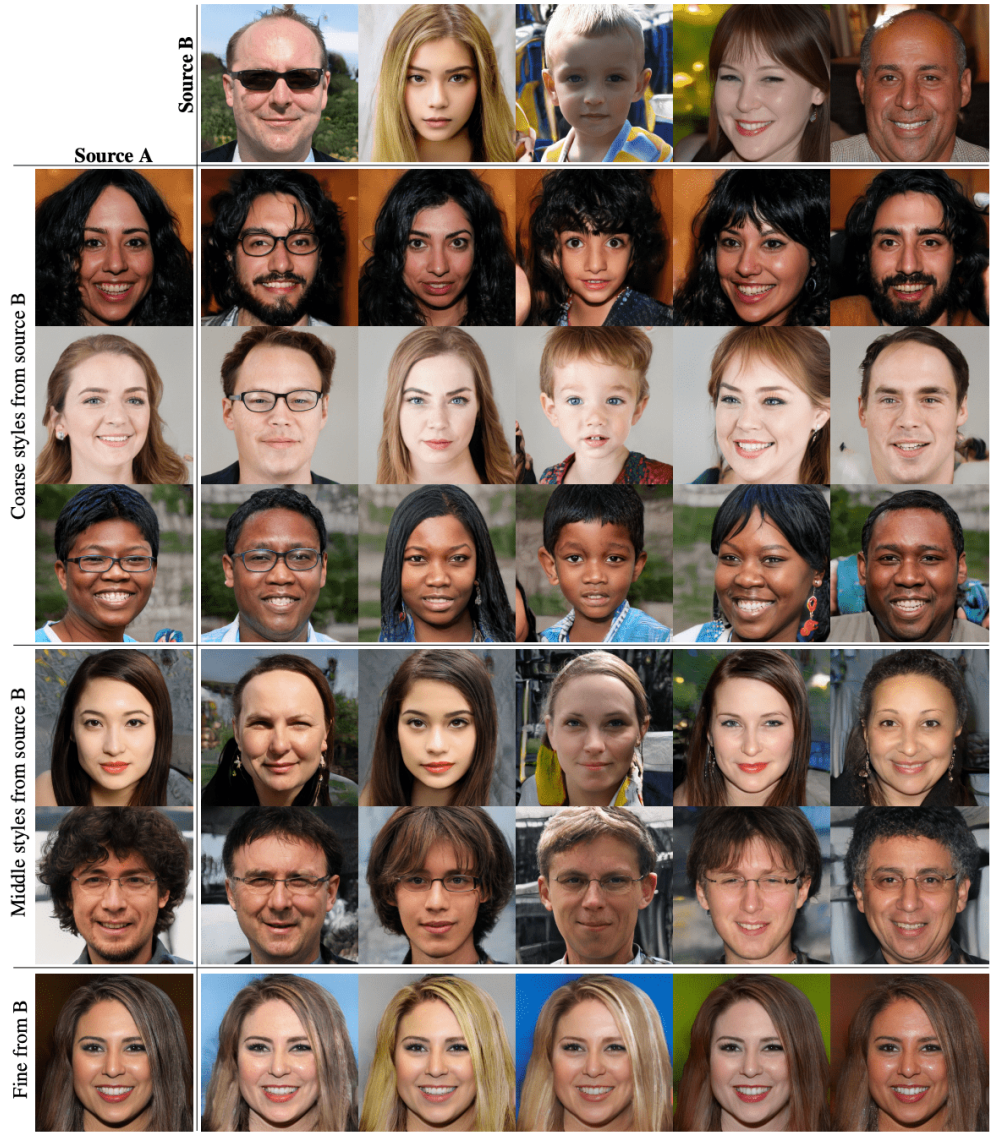
*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN




*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# StyleGAN




*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# Outline

# Evaluation of likelihood-free models

## What do we want from samples?

- **Sharpness.** The conditional distribution $p(y|\mathbf{x})$ should have low entropy (each image $\mathbf{x}$ should have distinctly recognizable object).
- **Diversity.** The marginal distribution $p(y) = \int p(y|\mathbf{x})p(\mathbf{x})d\mathbf{x}$ should have high entropy (there should be as many classes generated as possible).

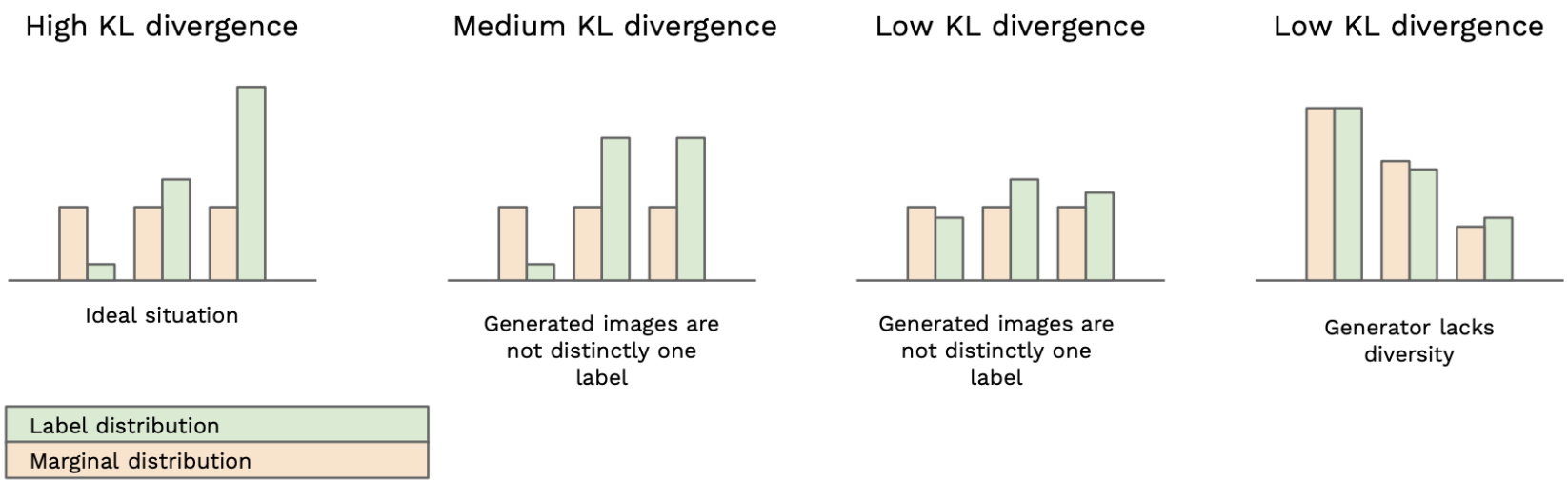



*image credit: https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a*

# Evaluation of likelihood-free models

## What do we want from samples?

- Sharpness $\Rightarrow$ low $H(y|\mathbf{x}) = -\sum_y \int_{\mathbf{x}} p(y, \mathbf{x}) \log p(y|\mathbf{x}) d\mathbf{x}$.
- Diversity $\Rightarrow$ high $H(y) = -\sum_y p(y) \log p(y)$.

## Inception Score

$$
\begin{aligned}
IS &= \exp(H(y) - H(y|\mathbf{x})) \\
&= \exp\left(-\sum_y p(y) \log p(y) + \sum_y \int_{\mathbf{x}} p(y, \mathbf{x}) \log p(y|\mathbf{x}) d\mathbf{x}\right) \\
&= \exp\left(\sum_y \int_{\mathbf{x}} p(y, \mathbf{x}) \log \frac{p(y|\mathbf{x})}{p(y)} d\mathbf{x}\right) \\
&= \exp\left(\mathbb{E}_{\mathbf{x}} \sum_y p(y|\mathbf{x}) \log \frac{p(y|\mathbf{x})}{p(y)}\right) = \exp\left(\mathbb{E}_{\mathbf{x}} KL(p(y|\mathbf{x})||p(y))\right)
\end{aligned}
$$

*Salimans T. et al. Improved Techniques for Training GANs, 2016*

# Evaluation of likelihood-free models

## Theorem (informal)

If $\pi(\mathbf{x})$ and $p(\mathbf{x}|\boldsymbol{\theta})$ has moment generation functions then

$$\pi(\mathbf{x}) = p(\mathbf{x}|\boldsymbol{\theta}) \Leftrightarrow \mathbb{E}_{\pi}\mathbf{x}^k = \mathbb{E}_{p}\mathbf{x}^k, \quad \forall k \geq 1.$$

This is intractable to calculate all moments.

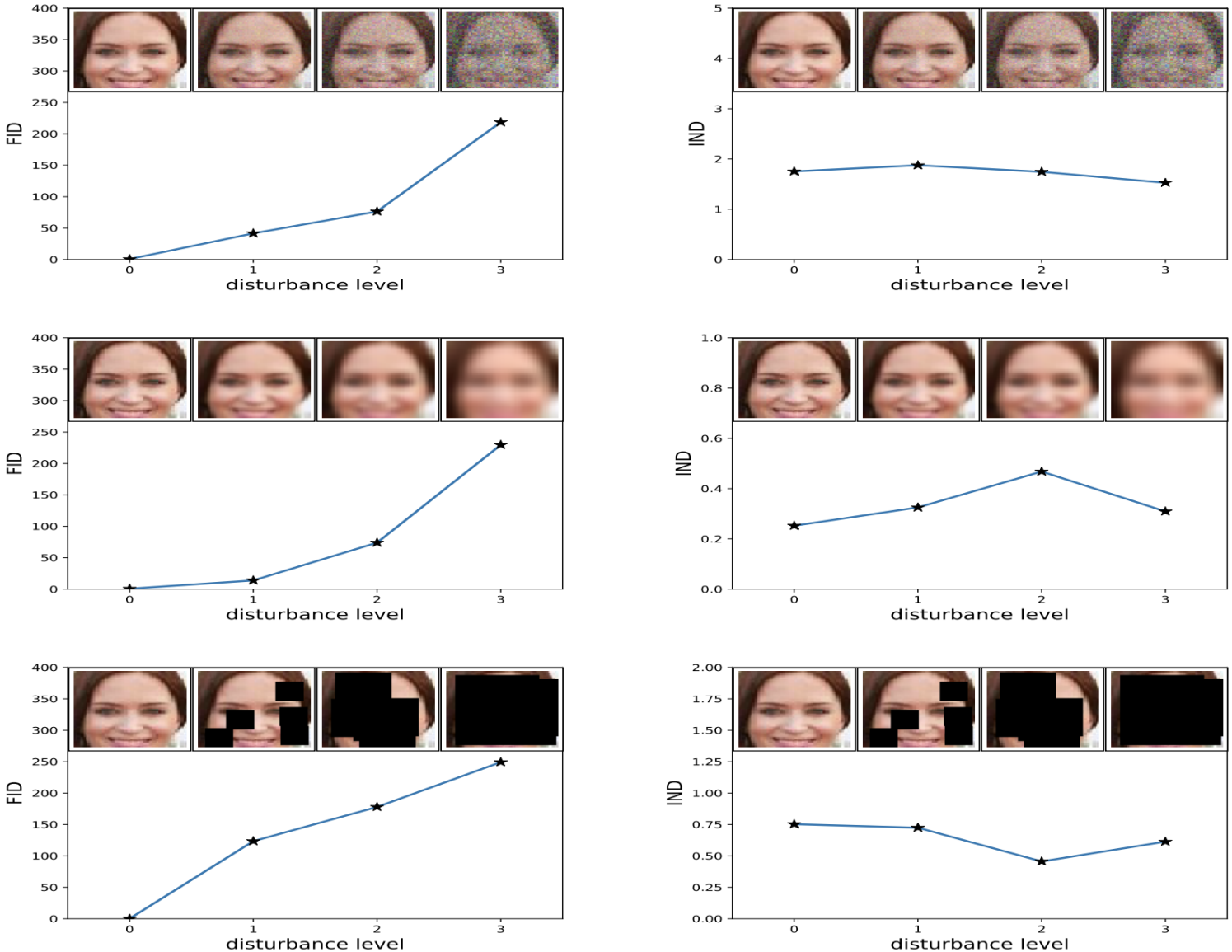
## Frechet Inception Distance

$$FID(\pi, p) = \|\mathbf{m}_{\pi} - \mathbf{m}_{p}\|_2^2 + \mathrm{Tr}\left(\boldsymbol{\Sigma}_{\pi} + \boldsymbol{\Sigma}_{p} - 2\sqrt{\boldsymbol{\Sigma}_{\pi}\boldsymbol{\Sigma}_{p}}\right)$$

- Representations are the outputs of the intermediate layer from the pretrained classification model.
- $\mathbf{m}_{\pi}$, $\boldsymbol{\Sigma}_{\pi}$ are the mean vector and the covariance matrix of feature representations for samples from $\pi(\mathbf{x})$
- $\mathbf{m}_{p}$, $\boldsymbol{\Sigma}_{p}$ are the mean vector and the covariance matrix of feature representations for samples from $p(\mathbf{x}|\boldsymbol{\theta})$.

*Heusel M. et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2017*

# Evaluation of likelihood-free models




*Heusel M. et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2017*

# Limitations

## Inception Score

$$IS = \exp\left(\mathbb{E}_{\mathbf{x}} KL(p(y|\mathbf{x})||p(y))\right)$$

- If generator produces images with a different set of labels from the classifier training set, IS will be low.
- If generator produces one image per class, the IS will be perfect (there is no measure of intra-class diversity).

## Frechet Inception Distance

$$FID = \|\mathbf{m}_\pi - \mathbf{m}_p\|_2^2 + \mathrm{Tr}\left(\mathbf{\Sigma}_\pi + \mathbf{\Sigma}_p - 2\sqrt{\mathbf{\Sigma}_\pi \mathbf{\Sigma}_p}\right)$$

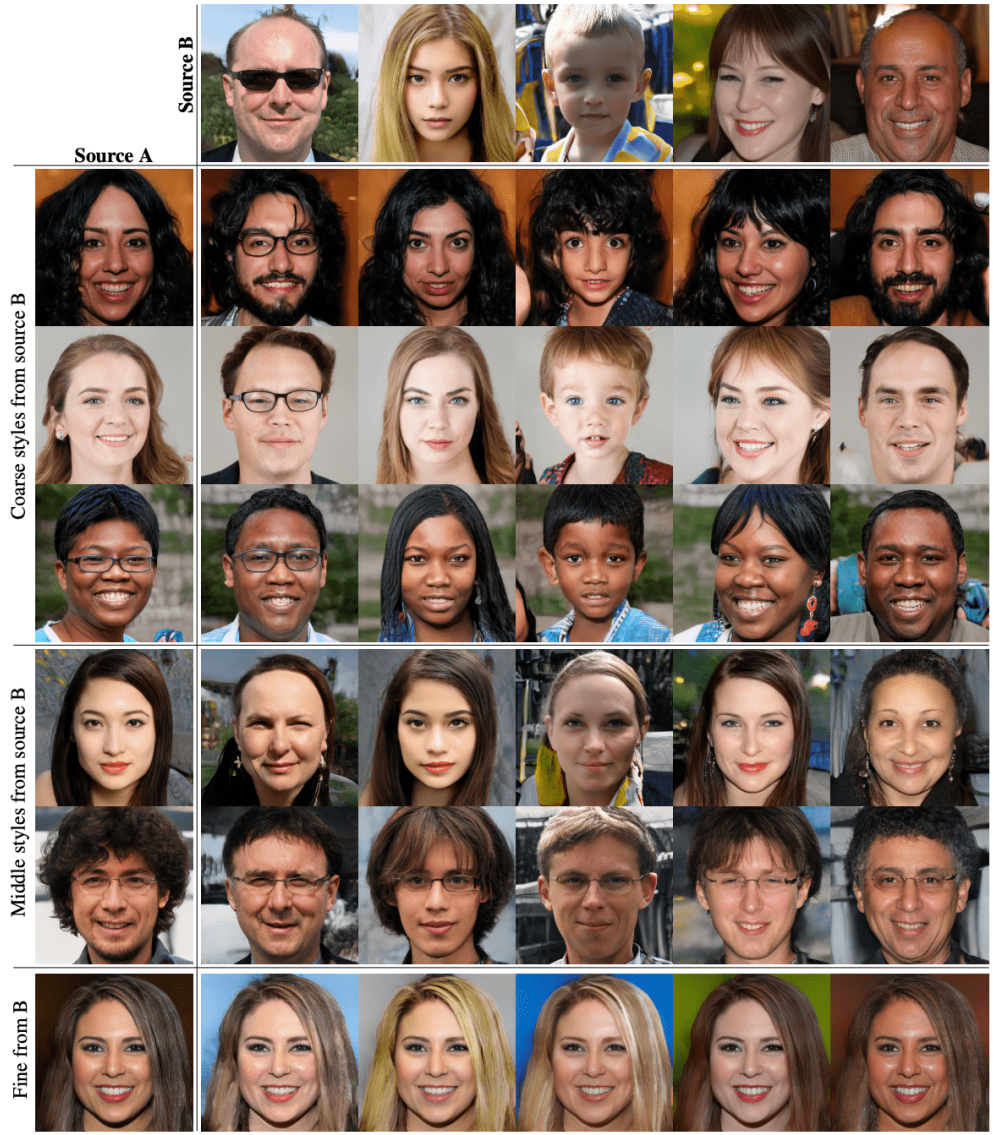
- Needs a large sample size for evaluation.
- Calculation of FID is slow.
- Uses the normality assumption!

Both scores depend on the pretrained classifier $p(y|\mathbf{x})$.

---

*Barratt S., Sharma R. A Note on the Inception Score, 2018*
*Heusel M. et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, 2017*

# StyleGAN




*Karras T., Laine S., Aila T. A Style-Based Generator Architecture for Generative Adversarial Networks, 2018*

# Outline

# Outline

# Vector Quantized VAE

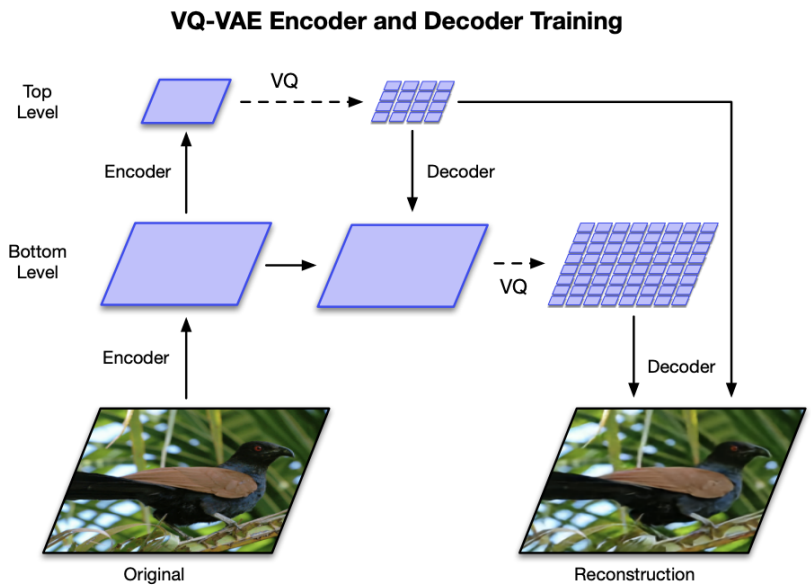
- The prior distribution over the discrete latents $p(\hat{\mathbf{z}})$ is a categorical distribution.
- It could be made autoregressive by depending on other $\hat{\mathbf{z}}$ in the feature map.
- While training the VQ-VAE, the prior is kept constant and uniform.
- After training, fit an autoregressive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

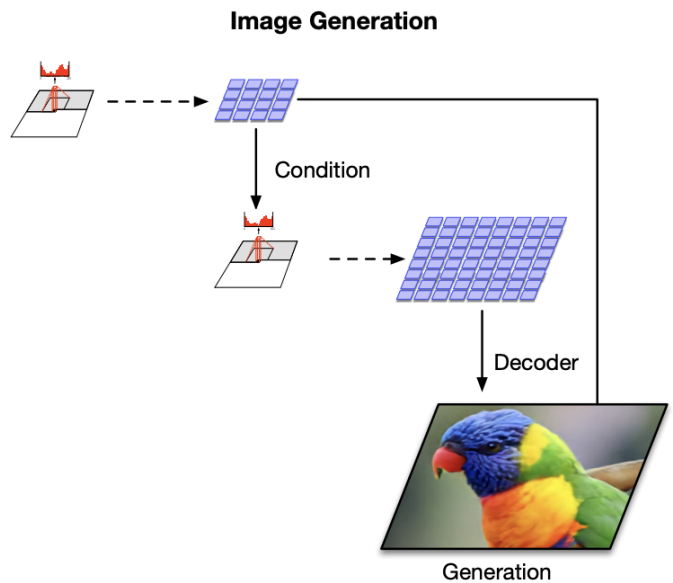
## Samples

*Oord A., Vinyals O., Kavukcuoglu K. Neural Discrete Representation Learning, 2017*

# 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.




*Razavi A., Oord A., Vinyals O. Generating Diverse High-Fidelity Images with VQ-VAE-2, 2019*

# Vector Quantized VAE-2

**Algorithm 1** VQ-VAE training (stage 1)

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

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

▷ quantize with top codebook eq 1

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

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

▷ quantize with bottom codebook eq 1

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

$\hat{\mathbf{x}} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$

▷ Loss according to eq 2

$\theta \leftarrow Update(\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}))$

**Algorithm 2** Prior training (stage 2)

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

**for** $\mathbf{x} \in$ training set **do**

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

  $\mathbf{e}_{bottom} \leftarrow Quantize(E_{bottom}(\mathbf{x}, \mathbf{e}_{top}))$

  $\mathbf{T}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$

  $\mathbf{T}_{bottom} \leftarrow \mathbf{T}_{bottom} \cup \mathbf{e}_{bottom}$

**end for**

$p_{top} = \texttt{TrainPixelCNN}(\mathbf{T}_{top})$

$p_{bottom} = \texttt{TrainCondPixelCNN}(\mathbf{T}_{bottom}, \mathbf{T}_{top})$

▷ Sampling procedure

**while** true **do**

  $\mathbf{e}_{top} \sim p_{top}$

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

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

**end while**

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

*Razavi A., Oord A., Vinyals O. Generating Diverse High-Fidelity Images with VQ-VAE-2, 2019*

# Outline

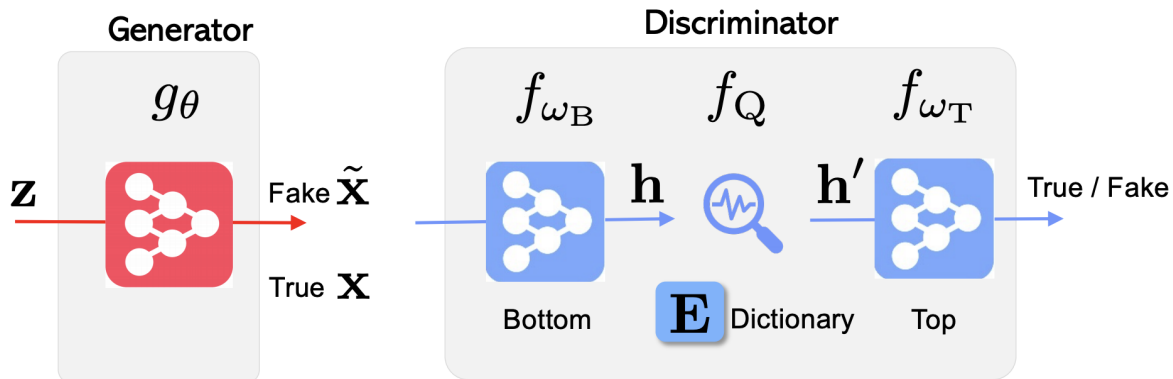
# Feature Quantized GAN

- GAN tries to find Nash equilibrium, minibatch training is unstable. GAN relies heavily on the minibatch statistics.
- Lots of feature matching strategies were proposed to stabilize the training.

## Feature quantized GAN discriminator

$$D(\mathbf{x}) = f_{\mathbf{w}_T} \circ f_{\mathbf{w}_B}(\mathbf{x}) \quad \Rightarrow \quad D(\mathbf{x}) = f_{\mathbf{w}_T} \circ f_Q \circ f_{\mathbf{w}_B}(\mathbf{x}).$$

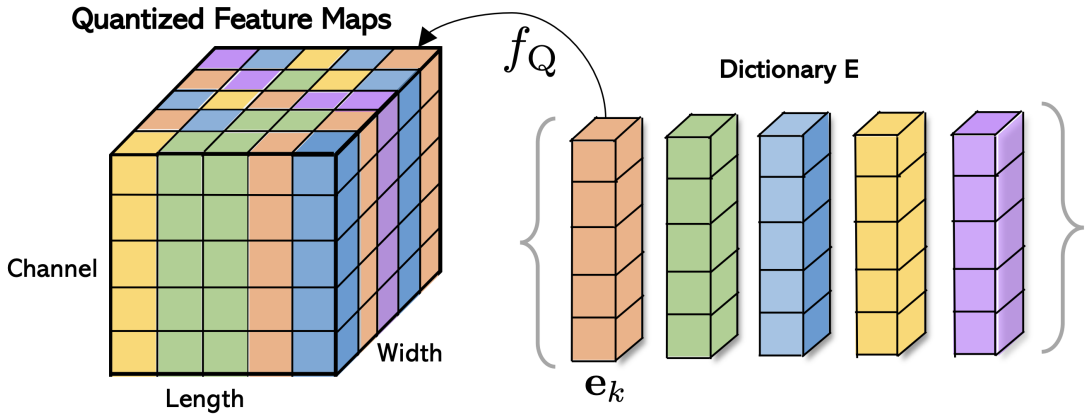
Here $f_Q$ is a vector quantization operation.

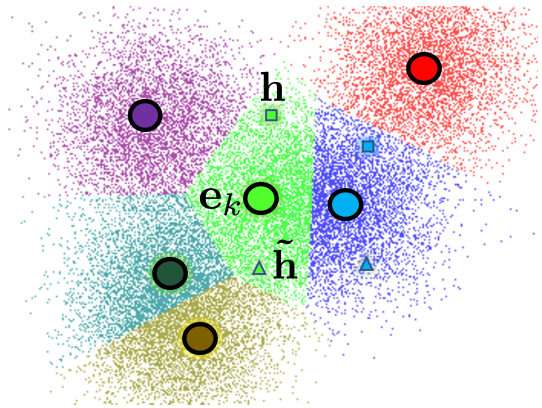



*Zhao Y. et al. Feature Quantization Improves GAN Training, 2020*

# Feature Quantized GAN

## Quantization procedure




## Quantized features


*Zhao Y. et al. Feature Quantization Improves GAN Training, 2020*

# Feature Quantized GAN

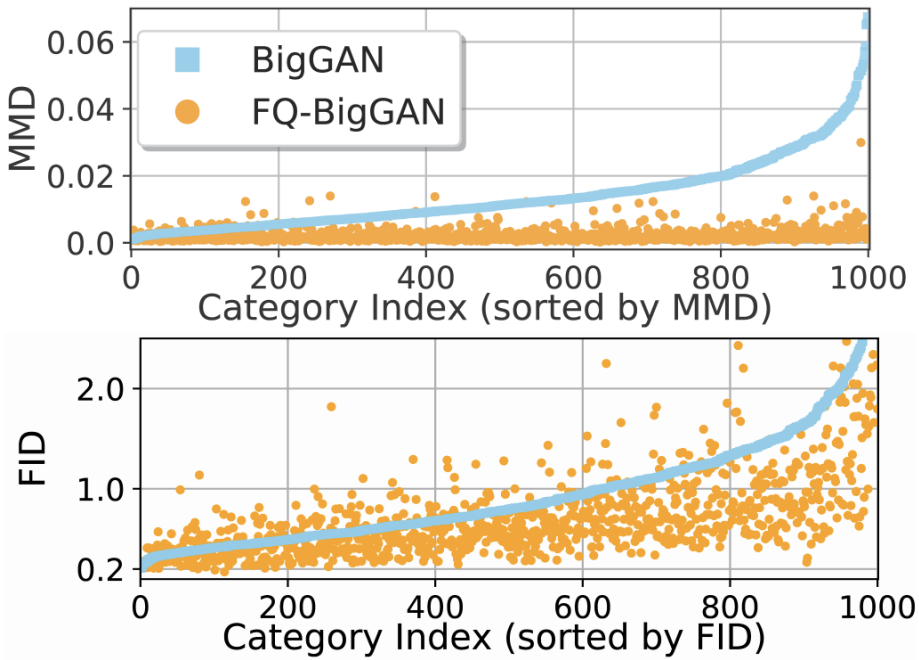
## ImageNet-1000

| Models | | 64 × 64 | 128 × 128 |
|---|---|---|---|
| | | FID* ↓ / IS* ↑ | FID* ↓ / IS* ↑ |
| Half | TAC-GAN | - | 23.75 / 28.86±0.29‡ |
| | BigGAN | 12.75 / 21.84±0.34 | 22.77 / 38.05±0.79‡ |
| | FQ-BigGAN | **12.62 / 21.99**±0.32 | **19.11 / 41.92**±1.15 |
| 256K | BigGAN | 10.55 / 25.43±0.15 | 14.88 / 63.03±1.42† |
| | FQ-BigGAN | **9.67 / 25.96**±0.24 | **13.77** / 54.36±1.07 |

## FFHQ

| Resolution | $32^2$ | $64^2$ | $128^2$ | $1024^2$ |
|---|---|---|---|---|
| StyleGAN | 3.28 | 4.82 | 6.33 | 5.24 |
| **FQ**-StyleGAN | **3.01** | **4.36** | **5.98** | **4.89** |

## Per class metrics for ImageNet




*Zhao Y. et al. Feature Quantization Improves GAN Training, 2020*

# Outline

# Outline

# Implicit score matching

## Theorem

Under some regularity conditions, it holds

$$\frac{1}{2}\mathbb{E}_\pi \left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log \pi(\mathbf{x}) \right\|_2^2 = \mathbb{E}_\pi \left[ \frac{1}{2} \|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2 + \operatorname{tr}\left( \nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}) \right) \right] + \text{const}$$

## Proof (only for 1D)

$$\mathbb{E}_\pi \left\| s(x) - \nabla_x \log \pi(x) \right\|_2^2 = \mathbb{E}_\pi \left[ s(x)^2 + (\nabla_x \log \pi(x))^2 - 2[s(x) \nabla_x \log \pi(x)] \right]$$

$$\mathbb{E}_\pi [s(x) \nabla_x \log \pi(x)] = \int \pi(x) s(x) \nabla_x \log \pi(x) dx = \int \underbrace{\nabla_x \log p(x)}_{g} \underbrace{\nabla_x \pi(x)}_{\nabla f} dx$$

$$= \underbrace{\nabla_x \log p(x)}_{g} \underbrace{\pi(x)}_{f} \Big|_{-\infty}^{+\infty} - \int \underbrace{\nabla_x \left( \nabla_x \log p(x) \right)}_{\nabla g} \underbrace{\pi(x)}_{f} dx$$

$$= -\mathbb{E}_\pi \nabla_x s(x)$$

$$\frac{1}{2}\mathbb{E}_\pi \left\| s(x) - \nabla_x \log \pi(x) \right\|_2^2 = \mathbb{E}_\pi \left[ \frac{1}{2} s(x)^2 + \nabla_x s(x) \right] + \text{const.}$$

---

*Hyvarinen A. Estimation of non-normalized statistical models by score matching, 2005*

# Implicit score matching

## Theorem

$$\frac{1}{2}\mathbb{E}_\pi \left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log \pi(\mathbf{x}) \right\|_2^2 = \mathbb{E}_\pi \left[ \frac{1}{2} \|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2 + \operatorname{tr}\left(\nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\right) \right] + \text{const}$$

Here $\nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\mathbf{x}}^2 \log p(\mathbf{x}|\boldsymbol{\theta})$ is a Hessian matrix.

1. The right hand side is complex due to Hessian matrix – **sliced score matching**.
2. The left hand side is intractable due to unknown $\pi(\mathbf{x})$ – **denoising score matching**.

## Sliced score matching (Hutchinson's trace estimation)

$$\operatorname{tr}\left(\nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})\right) = \mathbb{E}_{p(\boldsymbol{\epsilon})} \left[ \boldsymbol{\epsilon}^T \nabla_{\mathbf{x}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}) \boldsymbol{\epsilon} \right]$$

---

*Song Y. Sliced Score Matching: A Scalable Approach to Density and Score Estimation, 2019*

*Song Y. Generative Modeling by Estimating Gradients of the Data Distribution, blog post, 2021*

# Outline

# Classifier guidance

$$q(\mathbf{y}|\mathbf{x}_{t-1}, \mathbf{x}_t) = \frac{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{y})}{q(\mathbf{x}_{t-1}, \mathbf{x}_t)} =$$

$$= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y})q(\mathbf{y}|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t|\mathbf{x}_{t-1})eq(\mathbf{x}_{t-1})} = q(\mathbf{y}|\mathbf{x}_{t-1}).$$

## Conditional distribution

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}) = \frac{q(\mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{y})}{q(\mathbf{x}_t, \mathbf{y})} =$$

$$= \frac{q(\mathbf{y}|\mathbf{x}_{t-1}, \mathbf{x}_t)q(\mathbf{x}_{t-1}|\mathbf{x}_t)q(\mathbf{x}_t)}{q(\mathbf{y}|\mathbf{x}_t)q(\mathbf{x}_t)} =$$

$$= q(\mathbf{y}|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_t) \cdot \text{const}(\mathbf{x}_{t-1}).$$

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\phi}) = p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi})p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) \cdot \text{const}(\mathbf{x}_{t-1}).$$

- $p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta})$ - our unsupervised diffusion model.
- $p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi})$ - classifier for noised samples $\mathbf{x}_{t-1}$

---

*Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021*

# Classifier guidance

## Conditional distribution

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\phi}) = p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi}) \cdot p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) \cdot \text{const}(\mathbf{x}_{t-1})$$

$$\log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\phi}) = \log p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi}) + \log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) + \text{const}$$

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\mathbf{x}_t, t))$$

$$\log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) = -\frac{\|\mathbf{x}_{t-1} - \boldsymbol{\mu}\|^2}{2\boldsymbol{\sigma}^2} + \text{const}(\mathbf{x}_{t-1})$$

## Taylor expansion

$$\log p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi}) \approx \log p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi})|_{\mathbf{x}_{t-1}=\boldsymbol{\mu}} +$$
$$+ (\mathbf{x}_{t-1} - \boldsymbol{\mu}) \cdot \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi})|_{\mathbf{x}_{t-1}=\boldsymbol{\mu}} =$$
$$= (\mathbf{x}_{t-1} - \boldsymbol{\mu}) \cdot \mathbf{g} + \text{const}(\mathbf{x}_{t-1}),$$

where $\mathbf{g} = \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi})|_{\mathbf{x}_{t-1}=\boldsymbol{\mu}}$.

---

*Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021*

# Classifier guidance

$$\log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\phi}) = (\mathbf{x}_{t-1} - \boldsymbol{\mu}) \cdot \mathbf{g} - \frac{\|\mathbf{x}_{t-1} - \boldsymbol{\mu}\|^2}{2\boldsymbol{\sigma}^2} + \text{const}(\mathbf{x}_{t-1})$$
$$= -\frac{\|\mathbf{x}_{t-1} - \boldsymbol{\mu} - \boldsymbol{\sigma} \odot \mathbf{g}\|^2}{2\boldsymbol{\sigma}^2} + \text{const}(\mathbf{x}_{t-1})$$
$$= \log \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \mathbf{g}, \boldsymbol{\sigma}^2) + \text{const}(\mathbf{x}_{t-1})$$

## Guided sampling

1. Sample $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$.
2. Compute mean of $p(\mathbf{x}_{t-1}|\mathbf{x}_t, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t), \tilde{\beta}_t \mathbf{I})$:

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{\alpha_t(1 - \bar{\alpha}_t)}} \cdot \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t).$$

3. Compute $\mathbf{g} = \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{y}|\mathbf{x}_{t-1}, \boldsymbol{\phi})|_{\mathbf{x}_{t-1}=\boldsymbol{\mu}}$.
4. Get denoised image $\mathbf{x}_{t-1} = (\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \sqrt{\tilde{\beta}_t} \cdot \mathbf{g}) + \sqrt{\tilde{\beta}_t} \cdot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$.

---

*Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021*