

# DDA4210 Advanced Machine Learning

## Lecture 08 Generative Models

Jicong Fan

School of Data Science, CUHK-Shenzhen

March 15, 2023

# Overview

- 1 Introduction
- 2 Variational AutoEncoder (VAE)
- 3 Adversarial Generative Networks (GANs)
- 4 Diffusion Models

## 1 Introduction

2 Variational AutoEncoder (VAE)

3 Adversarial Generative Networks (GANs)

4 Diffusion Models

# Generative Models

- Generative models: generate new data instances with similar distribution as the training data
  - Learn a probability distribution  $p(\mathbf{x})$  from  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$
  - Then sample from  $p(\mathbf{x})$  to generate new data instances
- Deep Generative Models (DGMs) are formed through the combination of generative models and deep neural networks.
- DGMs achieved SOTA performances in many real cases (e.g., image generation, text generation, ChatGPT, etc)

# Generative Models

- Generative models: generate new data instances with similar distribution as the training data
  - Learn a probability distribution  $p(\mathbf{x})$  from  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$
  - Then sample from  $p(\mathbf{x})$  to generate new data instances
- Deep Generative Models (DGMs) are formed through the combination of generative models and deep neural networks.
- DGMs achieved SOTA performances in many real cases (e.g., image generation, text generation, ChatGPT, etc)
- Types of Deep Generative Models
  - Variational AutoEncoder (VAE)
  - Generative Adversarial Networks (GANs)
  - Diffusion Models
  - ...

# Generation instances via deep generative models



NVAE(2020)



DDPM(2020)



StyleGAN(2018)

- Vahdat Arash and Jan Kautz. "NVAE: A Deep Hierarchical Variational Autoencoder". NeurIPS 2020.
- Ho Jonathan et al. "Denoising Diffusion Probabilistic Models", NeurIPS 2020.
- Karras Tero et al. "A Style-Based Generator Architecture for Generative Adversarial Networks". CVPR 2021.

1 Introduction

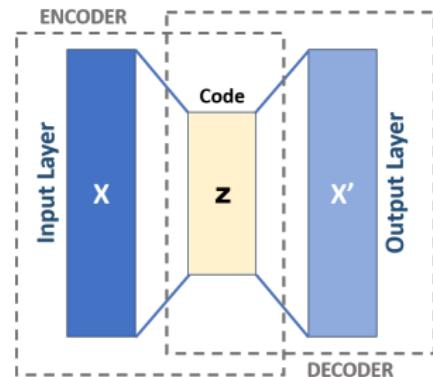
2 Variational AutoEncoder (VAE)

3 Adversarial Generative Networks (GANs)

4 Diffusion Models

# AutoEncoder (AE)

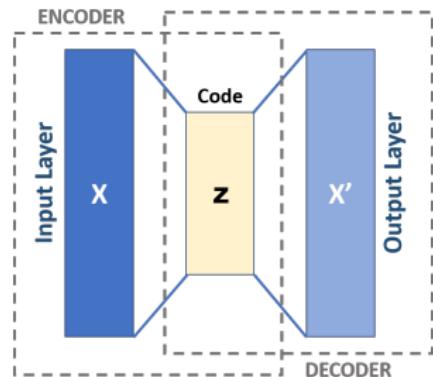
- AutoEncoder (AE) is a type of neural network designed to learn an approximate identity transformation using an unsupervised way and then to reconstruct high-dimensional data and consists of an encoder network  $f_\phi$  and a decoder network  $g_\theta$ , parameterized by  $\phi, \theta$  respectively.
- The middle layer of AE usually has a narrow bottleneck to compress original high-dimensional data to low-dimensional representations.



- Encoder network  $f_\phi : \mathbf{x} \rightarrow \mathbf{z}$
- Decoder network  $g_\theta : \mathbf{z} \rightarrow \mathbf{x}'$
- Optimization objective:
$$\min_{\phi, \theta} \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - g_\theta(f_\phi(\mathbf{x}_i))\| + \mathcal{R}(\phi, \theta)$$

# AutoEncoder (AE)

- AutoEncoder (AE) is a type of neural network designed to learn an approximate identity transformation using an unsupervised way and then to reconstruct high-dimensional data and consists of an encoder network  $f_\phi$  and a decoder network  $g_\theta$ , parameterized by  $\phi, \theta$  respectively.
- The middle layer of AE usually has a narrow bottleneck to compress original high-dimensional data to low-dimensional representations.



- Encoder network  $f_\phi : \mathbf{x} \rightarrow \mathbf{z}$
- Decoder network  $g_\theta : \mathbf{z} \rightarrow \mathbf{x}'$
- Optimization objective:  
$$\min_{\phi, \theta} \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - g_\theta(f_\phi(\mathbf{x}_i))\| + \mathcal{R}(\phi, \theta)$$
- Can we generate new data using AE?

Image from Wikipedia.

# Variational AutoEncoder (VAE): Motivation

- VAE aims to transform  $\mathbf{x}$  into a prior distribution  $p_z$  (rather than a fixed vector  $\mathbf{z}$ ) using encoder  $f_\phi$  and then to reconstruct  $\mathbf{x}$  using decoder  $g_\theta$ .
- Given training data  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  and a prior distribution  $p_z$ . Assume we have trained  $f_\phi$  and  $g_\theta$  of VAE successfully. In order to generate a new sample that looks like a real data point  $\mathbf{x}_i$ , we need the following steps:
  - First, sample a  $\mathbf{z}_i$  from the prior distribution  $p_z$ .
  - Then, a new sample can be generated via the decoder, i.e.,  $g_\theta(\mathbf{z}_i)$ .

# Variational AutoEncoder (VAE): Motivation

- VAE aims to transform  $\mathbf{x}$  into a prior distribution  $p_z$  (rather than a fixed vector  $\mathbf{z}$ ) using encoder  $f_\phi$  and then to reconstruct  $\mathbf{x}$  using decoder  $g_\theta$ .
- Given training data  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$  and a prior distribution  $p_z$ . Assume we have trained  $f_\phi$  and  $g_\theta$  of VAE successfully. In order to generate a new sample that looks like a real data point  $\mathbf{x}_i$ , we need the following steps:
  - First, sample a  $\mathbf{z}_i$  from the prior distribution  $p_z$ .
  - Then, a new sample can be generated via the decoder, i.e.,  $g_\theta(\mathbf{z}_i)$ .
- How to obtain the decoder  $g_\theta$ ? Maximize the probability of generating real data samples (maximum likelihood):

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^n \log p(\mathbf{x}_i | \theta)$$

For simplicity,  $p(\mathbf{x}_i | \theta)$  abbreviates as  $p_\theta(\mathbf{x}_i)$ .

# Variational AutoEncoder (VAE): Maximum Likelihood

- Compute the marginal likelihood

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

# Variational AutoEncoder (VAE): Maximum Likelihood

- Compute the marginal likelihood

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

- Prior  $p_{\theta}(\mathbf{z})$ , e.g.  $\mathcal{N}(\mathbf{0}, \mathbf{I})$
- Likelihood  $p_{\theta}(\mathbf{x}|\mathbf{z})$
- But it is impossible to integrate over all  $\mathbf{z}$ .

- How about using Bayes' theorem?

$$p_{\theta}(\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z}) p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{z}|\mathbf{x})}$$

- $p_{\theta}(\mathbf{z}|\mathbf{x})$  cannot be computed.

# Variational AutoEncoder (VAE): Maximum Likelihood

- Compute the marginal likelihood

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

- Prior  $p_{\theta}(\mathbf{z})$ , e.g.  $\mathcal{N}(\mathbf{0}, \mathbf{I})$
- Likelihood  $p_{\theta}(\mathbf{x}|\mathbf{z})$
- But it is impossible to integrate over all  $\mathbf{z}$ .

- How about using Bayes' theorem?

$$p_{\theta}(\mathbf{x}) = \frac{p_{\theta}(\mathbf{x}|\mathbf{z}) p_{\theta}(\mathbf{z})}{p_{\theta}(\mathbf{z}|\mathbf{x})}$$

- $p_{\theta}(\mathbf{z}|\mathbf{x})$  cannot be computed.

- Solution: Train another neural network (encoder)  $f_{\phi}$  that learns

$$q_{\phi}(\mathbf{z}|\mathbf{x}) \approx p_{\theta}(\mathbf{z}|\mathbf{x})$$

# Variational AutoEncoder (VAE): Maximum Likelihood

- Decompose the log-likelihood:

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})} = \log \frac{p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})q_\phi(\mathbf{z}|\mathbf{x})} \\ &= \log p_\theta(\mathbf{x}|\mathbf{z}) - \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\end{aligned}$$

- Take expectation:

$$\begin{aligned}\log p_\theta(\mathbf{x}) &= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x})] = \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}) d\mathbf{z} \\ &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} d\mathbf{z} \\ &\quad + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\ &= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \\ &\quad + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))\end{aligned}$$

# Variational AutoEncoder (VAE): Evidence Lower Bound

- We have got

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

- Because KL-divergence is always non-negative, we obtain

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \triangleq \mathcal{L}_{\phi,\theta}(\mathbf{x})$$

# Variational AutoEncoder (VAE): Evidence Lower Bound

- We have got

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

- Because KL-divergence is always non-negative, we obtain

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \triangleq \mathcal{L}_{\phi,\theta}(\mathbf{x})$$

- $\mathcal{L}_{\phi,\theta}(\mathbf{x})$  is a **lower bound** (called evidence lower bound, ELBO) of  $\log p_\theta(\mathbf{x})$  and

$$\log p_\theta(\mathbf{x}) = \mathcal{L}_{\phi,\theta}(\mathbf{x}) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

ELBO is also known as the variational lower bound.

# Variational AutoEncoder (VAE): Evidence Lower Bound

- We have got

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

- Because KL-divergence is always non-negative, we obtain

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) \triangleq \mathcal{L}_{\phi,\theta}(\mathbf{x})$$

- $\mathcal{L}_{\phi,\theta}(\mathbf{x})$  is a **lower bound** (called evidence lower bound, ELBO) of  $\log p_\theta(\mathbf{x})$  and

$$\log p_\theta(\mathbf{x}) = \mathcal{L}_{\phi,\theta}(\mathbf{x}) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x}))$$

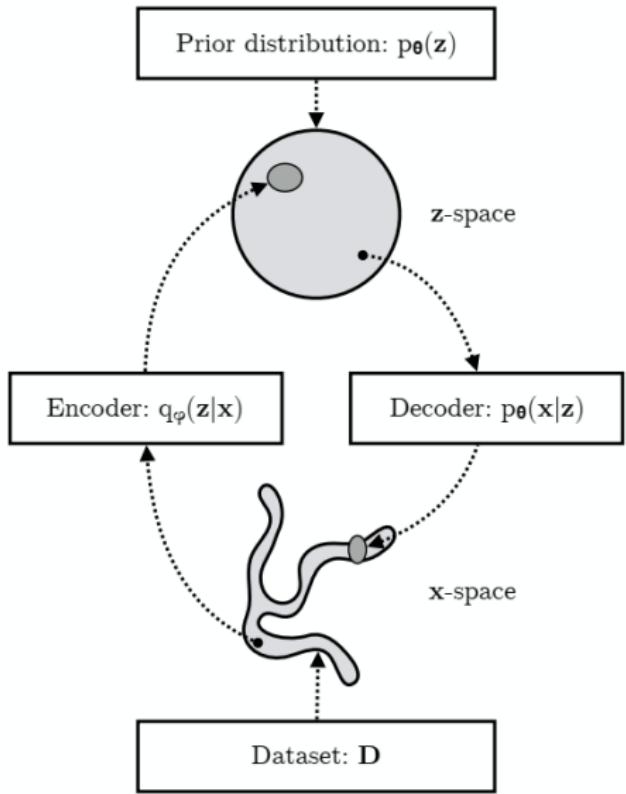
ELBO is also known as the variational lower bound.

- VAE maximizes ELBO, i.e.,

$$\phi^*, \theta^* = \underset{\phi, \theta}{\operatorname{argmax}} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

When  $\mathcal{L}_{\phi,\theta}(\mathbf{x}) = \log p_\theta(\mathbf{x})$ , it holds that  $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x}) = 0$ .

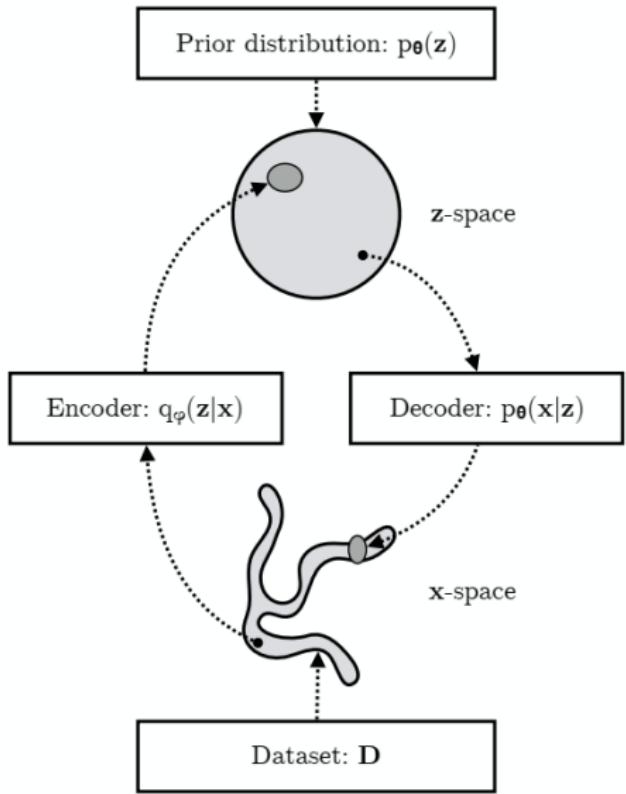
# Variational Auto-Encoder (VAE): Optimization



$$\max_{\phi, \theta} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

- maximize  $\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]$ : reconstruct  $\mathbf{x}$
- minimize  $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$ : approximate prior

# Variational Auto-Encoder (VAE): Optimization

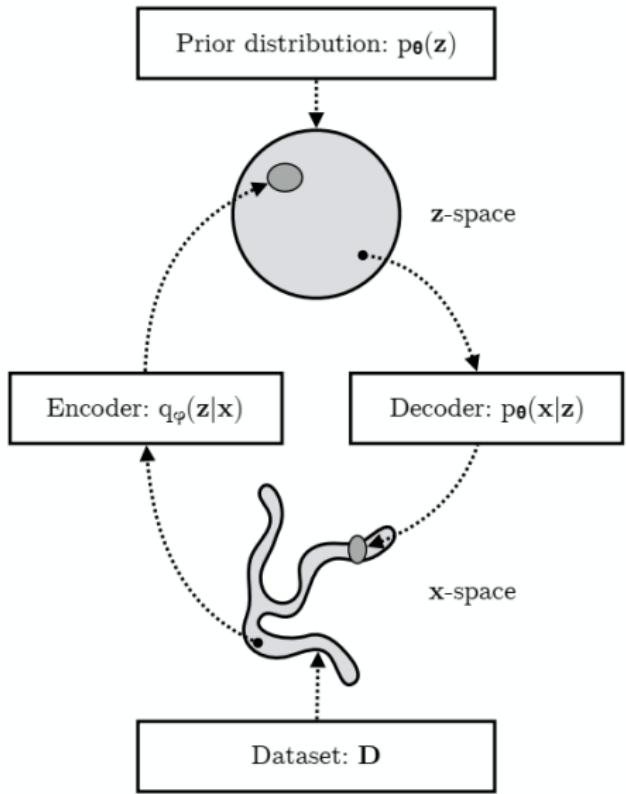


$$\max_{\phi, \theta} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

- maximize  $\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]$ : reconstruct  $\mathbf{x}$
- minimize  $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$ : approximate prior

Suppose  $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ . Then let  $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \text{diag}(\sigma_\phi^2(\mathbf{x})))$ .

# Variational Auto-Encoder (VAE): Optimization



$$\max_{\phi, \theta} \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

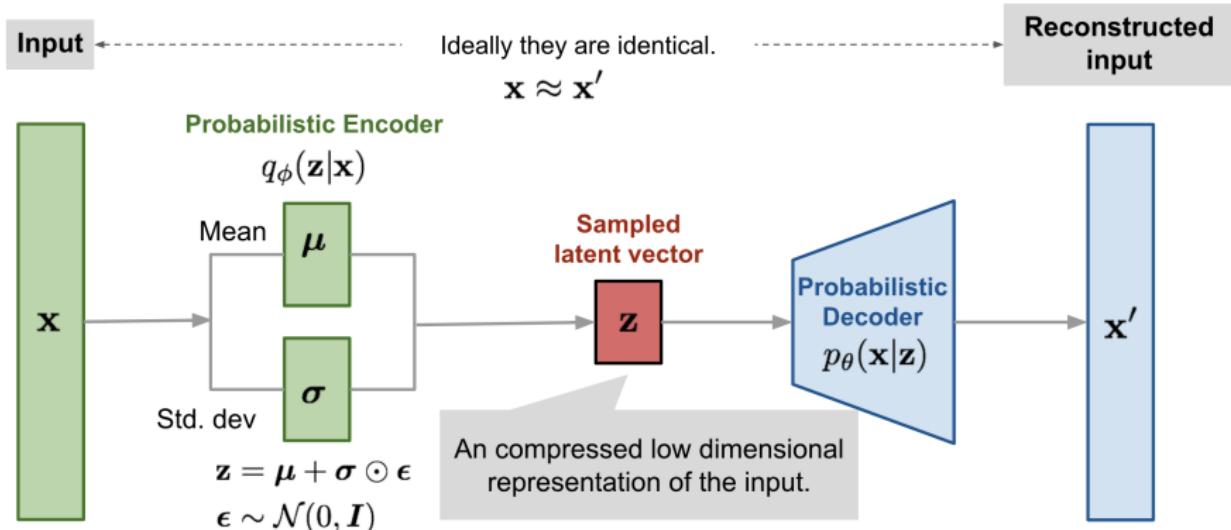
- maximize  $\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})]$ : reconstruct  $\mathbf{x}$
- minimize  $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$ : approximate prior

Suppose  $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ . Then let  $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi(\mathbf{x}), \text{diag}(\sigma_\phi^2(\mathbf{x})))$ .

The expectation term in the loss function requires sampling  $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$ , which is a stochastic process. Therefore we cannot backpropagate the gradient.

Image from Kingma and Welling. An Introduction to Variational Autoencoders.2019.

# VAE: Reparameterization Trick



$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu_\phi(\mathbf{x}), \text{diag}(\sigma_\phi^2(\mathbf{x})))$$

$\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \odot \epsilon$ , where  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  *Reparameterization Trick*

Image from <https://lilianweng.github.io/posts/2018-08-12-vae/>

# Variational AutoEncoder (VAE): Details about Loss

- $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ : consider one element of  $\mathbf{z}$

$$\begin{aligned}& - D_{KL}(q_\phi(z|x) \| p(z)) \\&= \int \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right) \log \left( \frac{\frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(z-\mu_p)^2}{2\sigma_p^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right)} \right) dz \\&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \frac{1}{2} \\&= \frac{1}{2} \left[ 1 + \log\left(\frac{\sigma_q^2}{\sigma_p^2}\right) - \sigma_q^2 - \mu_q^2 \right]\end{aligned}$$

# Variational AutoEncoder (VAE): Details about Loss

- $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))$ : consider one element of  $\mathbf{z}$

$$\begin{aligned}& - D_{KL}(q_\phi(z|x) \| p(z)) \\&= \int \frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right) \log \left( \frac{\frac{1}{\sqrt{2\pi\sigma_p^2}} \exp\left(-\frac{(z-\mu_p)^2}{2\sigma_p^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(-\frac{(z-\mu_q)^2}{2\sigma_q^2}\right)} \right) dz \\&= \log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} + \frac{1}{2} \\&= \frac{1}{2} \left[ 1 + \log\left(\frac{\sigma_q^2}{\sigma_p^2}\right) - \sigma_q^2 - \mu_q^2 \right]\end{aligned}$$

- $\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ :  $p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; g_\theta(\mathbf{z}), \Sigma_{\mathbf{x}})$

$$\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] \propto \|\mathbf{x} - g_\theta(\mathbf{z})\|^2$$

# Variational AutoEncoder (VAE)

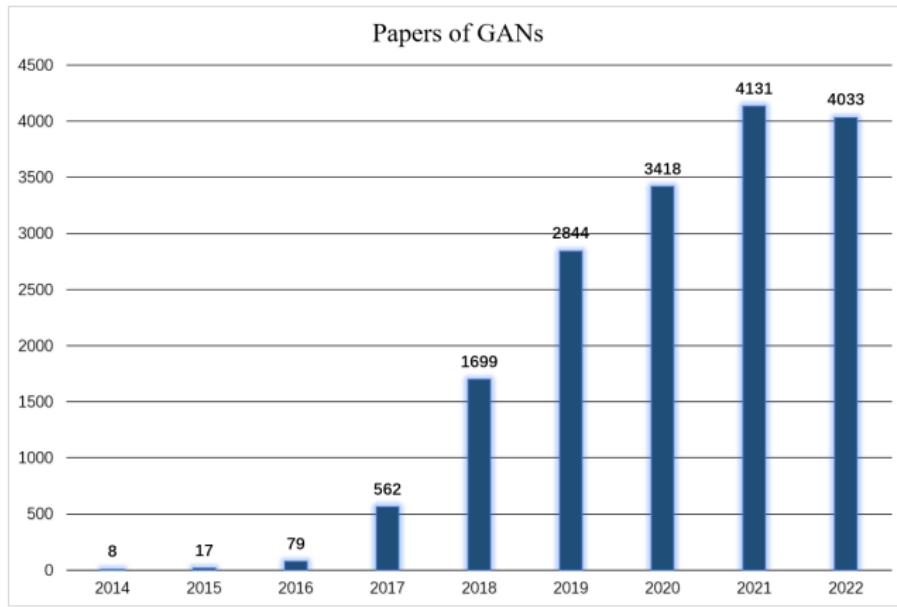
## References

- [Kingma and Welling, 2013] Auto-Encoding Variational Bayes
- [Higgins et al., 2020] beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework
- [Oord et al., 2017] VQ-VAE: Neural Discrete Representation Learning
- [Razavi et al., 2019] Generating Diverse High-Fidelity Images with VQ-VAE-2
- <https://lilianweng.github.io/posts/2018-08-12-vae/>
- [https://en.wikipedia.org/wiki/Variational\\_autoencoder](https://en.wikipedia.org/wiki/Variational_autoencoder)
- <https://en.wikipedia.org/wiki/Autoencoder>

- 1 Introduction
- 2 Variational AutoEncoder (VAE)
- 3 Adversarial Generative Networks (GANs)
- 4 Diffusion Models

# Generative Adversarial Networks (GANs)

- Generative Adversarial Networks is a kind of well-known and popular generative model designed by Ian J. Goodfellow and his colleagues in June 2014.



<https://www.aminer.cn/search/pub?q=generative%20adversarial%20networks&t=b>

# Generative Adversarial Networks

Inspired by game theory, GAN estimates generator via an adversarial process, in which we simultaneously train two neural networks

- A generator  $G$  that is trained to capture the real data distribution so that the generated samples can be as real as possible.
- A discriminator  $D$  that estimates the probability that a sample came from the training data rather than the generator  $G$ .
- Adversarial process: training  $D$  to maximize the probability of assigning the correct label to both training examples and samples from  $G$  and simultaneously training  $G$  to maximize the probability of  $D$  making a mistake.

# Generative Adversarial Networks

Training steps:

- 1 Fix parameters of generator  $G$ , train discriminator  $D$
- 2 Fix parameters of discriminator  $D$ , train generator  $G$
- 3 Repeat step 1,2

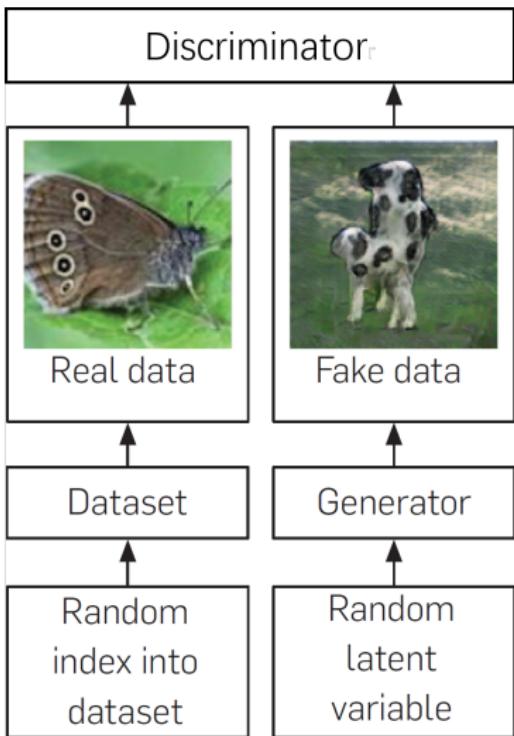


Image from Generative Adversarial Networks (<https://dl.acm.org/doi/10.1145/3422622>)

# Generative Adversarial Networks

## Model architecture of GAN

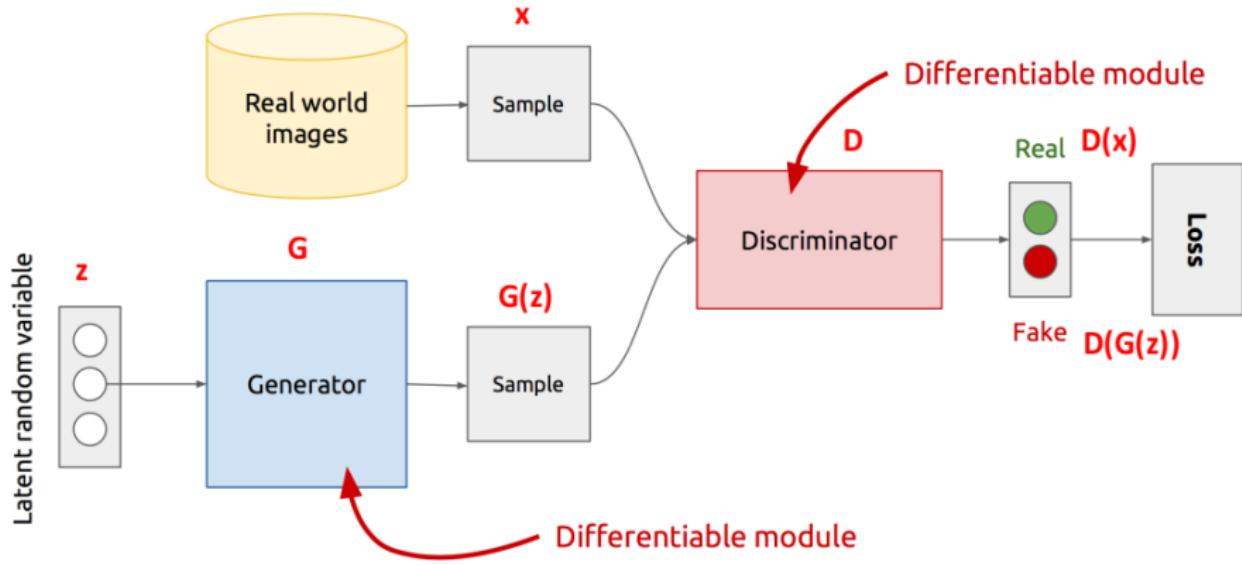


Image from [https://www.cs.toronto.edu/~rgrosse/courses/csc321\\_2018/slides/lec19.pdf](https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf)

# Generative Adversarial Networks

## Train the discriminator

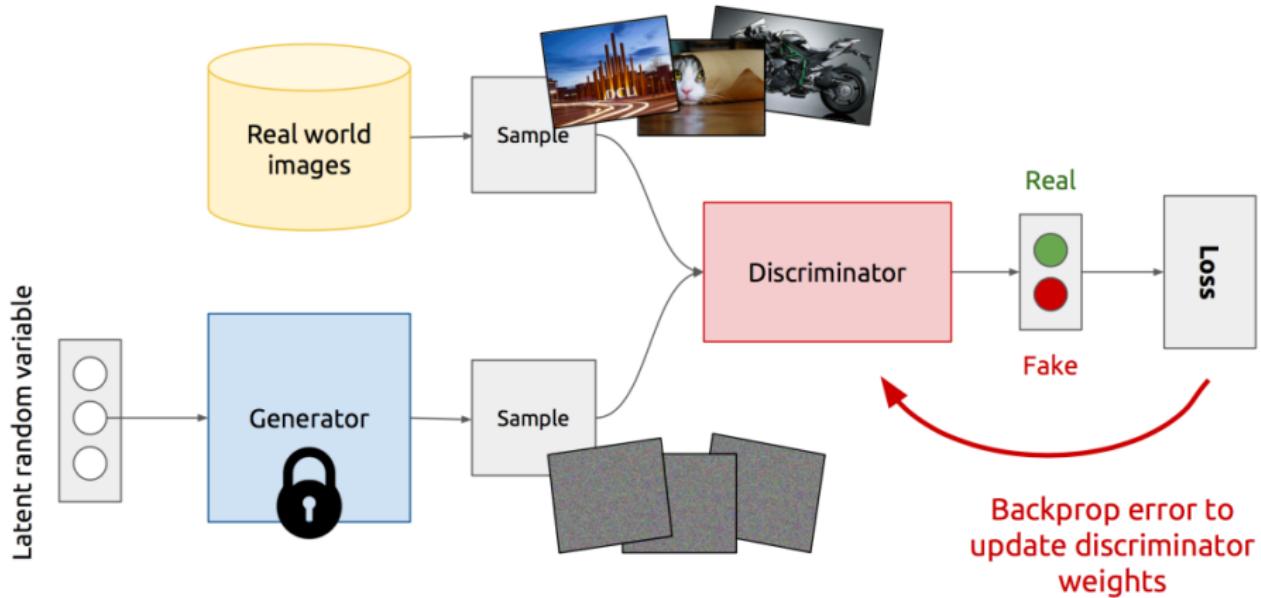


Image from [https://www.cs.toronto.edu/~rgrosse/courses/csc321\\_2018/slides/lec19.pdf](https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf)

# Generative Adversarial Networks

## Train the generator

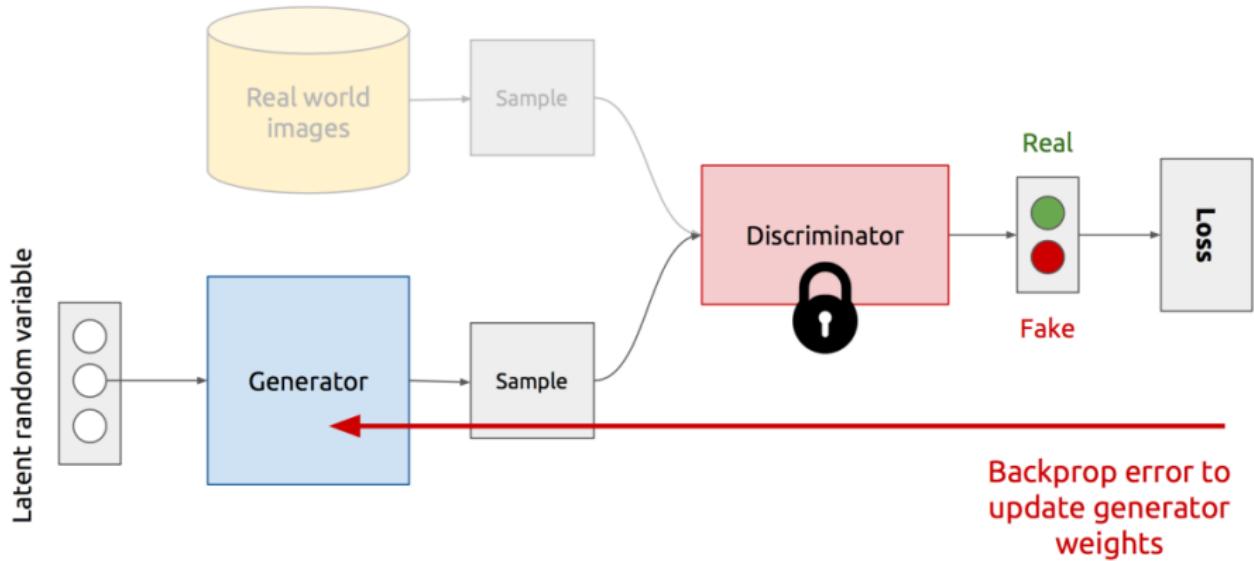


Image from [https://www.cs.toronto.edu/~rgrosse/courses/csc321\\_2018/slides/lec19.pdf](https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf)

# Generative Adversarial Networks

- Notations

- $p_r$ : data distribution over real samples  $\mathbf{x}$
- $p_g$ : the generator's distribution over data  $\mathbf{x}$
- $p_z$ : a prior on input noise variable  $\mathbf{z}$

# Generative Adversarial Networks

- Notations
  - $p_r$ : data distribution over real samples  $\mathbf{x}$
  - $p_g$ : the generator's distribution over data  $\mathbf{x}$
  - $p_z$ : a prior on input noise variable  $\mathbf{z}$
- Ensure the discriminator  $D$ 's decisions over real data are accurate by

$$\text{maximize}_D \mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} [\log D(\mathbf{x})]$$

# Generative Adversarial Networks

- Notations
  - $p_r$ : data distribution over real samples  $\mathbf{x}$
  - $p_g$ : the generator's distribution over data  $\mathbf{x}$
  - $p_z$ : a prior on input noise variable  $\mathbf{z}$
- Ensure the discriminator  $D$ 's decisions over real data are accurate by

$$\text{maximize}_D \mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} [\log D(\mathbf{x})]$$

- Given a fake sample  $G(\mathbf{z})$ ,  $\mathbf{z} \sim p_z(\mathbf{z})$ , the discriminator is expected to output a probability,  $D(G(\mathbf{z}))$ , close to zero by

$$\text{maximize}_D \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

- On the other hand, the generator is trained to increase the chances of  $D$  producing a high probability for generated samples, thus

$$\text{minimize}_G \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

# Generative Adversarial Networks

- Therefore,  $D$  and  $G$  play the following two-player minimax game with loss function  $\mathcal{L}(G, D)$  :

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

# Pseudo Code of GAN

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator,  $k$ , is a hyperparameter. We used  $k = 1$ , the least expensive option, in our experiments.

Discriminator  
updates

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

**end for**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Image from [https://www.cs.toronto.edu/~rgrosse/courses/csc321\\_2018/slides/lec19.pdf](https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf)

# Advantages and Disadvantages of GAN

- Advantages

- Sampling (or generation) is intuitive and straightforward.
- Compared to VAE, the training of GAN doesn't involve MLE.
- Compared to VAE, the generated samples of GAN are more realistic.

# Advantages and Disadvantages of GAN

- Advantages

- Sampling (or generation) is intuitive and straightforward.
- Compared to VAE, the training of GAN doesn't involve MLE.
- Compared to VAE, the generated samples of GAN are more realistic.

- Disadvantages

- Probability distribution is implicit
  - Not straightforward to compute  $p(\mathbf{x})$
  - Thus only good for generating new samples
- The training is hard
  - No convergence guarantee
  - May encounter mode collapse

# A brief history of GANs

- [Goodfellow et al., 2014]: Generative Adversarial Networks (GAN)
- [Mirza et al. 2014]: Conditional GAN
- [Radford et al. 2015]: Deep Convolutional GAN
- [Ming-Yu Liu et al., 2016]: Coupled GAN
- [Karras et al. 2017]: Progressive Growing of GANs
- [Arjovsky et al. 2017]: Wasserstein GAN
- [Zhu et al. 2017]: CycleGAN
- [Han Zhang et al. 2018]: Self-Attention GAN
- [Brock et al. 2018]: Large-scale GAN training (BigGAN)
- [Karras et al. 2018]: A style-based generator architecture for GAN (StyleGAN)

# Progress of GANs on image generation

- human face



2014 (GAN)



2015 (DCGAN)



2016 (CoGAN)



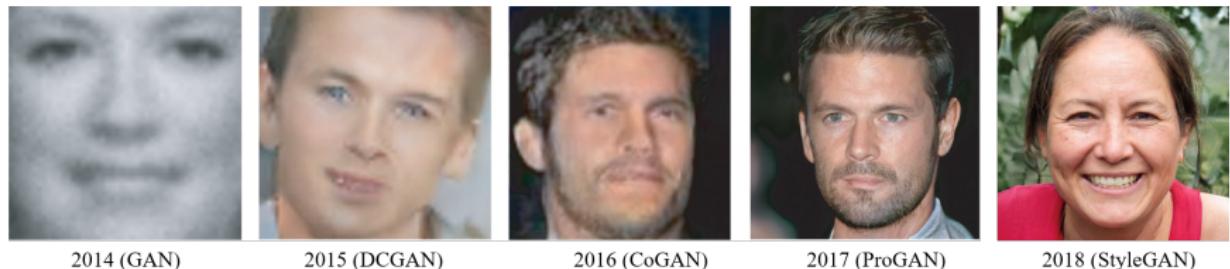
2017 (ProGAN)



2018 (StyleGAN)

# Progress of GANs on image generation

- human face



- other objects



- 1 Introduction
- 2 Variational AutoEncoder (VAE)
- 3 Adversarial Generative Networks (GANs)
- 4 Diffusion Models

# Diffusion Models: Overview

- Diffusion models, also known as diffusion probabilistic models, are a class of latent variable models introduced in 2015 with inspiration from non-equilibrium thermodynamics.
- Overview of different types of generative models

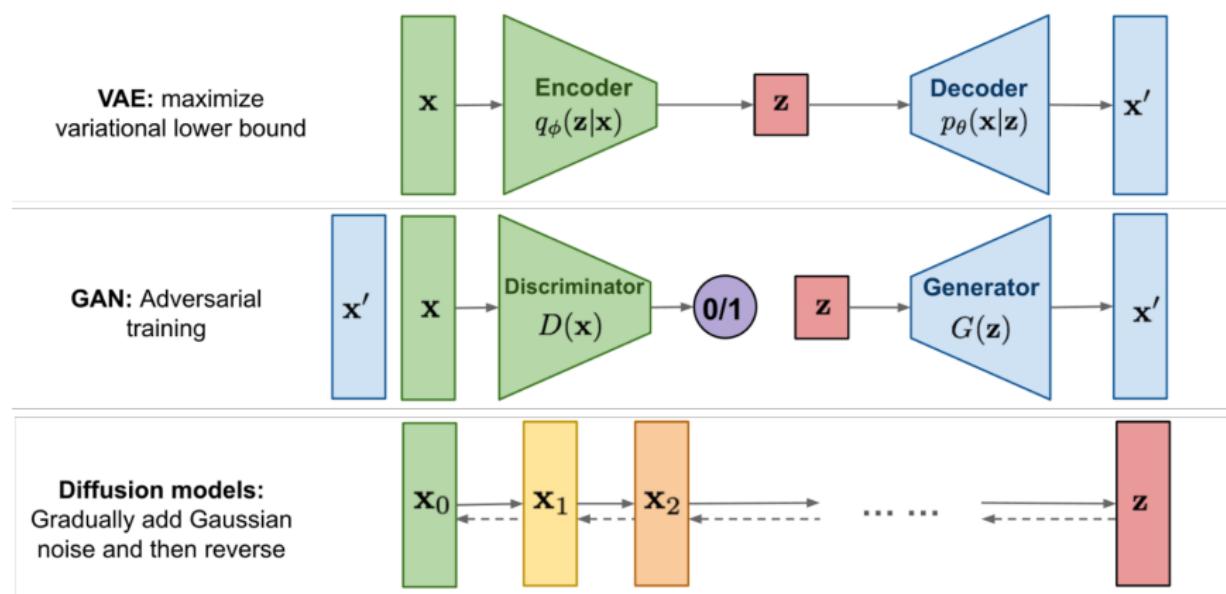


Image from <https://lilianweng.github.io/posts/2021-07-11-diffusion-models>

# Diffusion Models: Forward Diffusion Process

- Given a data point sampled from a real data distribution  $\mathbf{x}_0 \sim q(\mathbf{x})$ , a forward diffusion process adds small noise (e.g. Gaussian noise) to the sample in  $T$  steps slowly, which produces a sequence of noisy samples  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$ :

$$\mathbf{x}_t = \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \sqrt{\beta_t} \boldsymbol{\epsilon}_{t-1}$$

The step sizes are controlled by a variance schedule  $\{\beta_t \in (0, 1)\}_{t=1}^T$ .

- When  $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we have

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

# Diffusion Models: Forward Diffusion Process

- Given a data point sampled from a real data distribution  $\mathbf{x}_0 \sim q(\mathbf{x})$ , a forward diffusion process adds small noise (e.g. Gaussian noise) to the sample in  $T$  steps slowly, which produces a sequence of noisy samples  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$ :

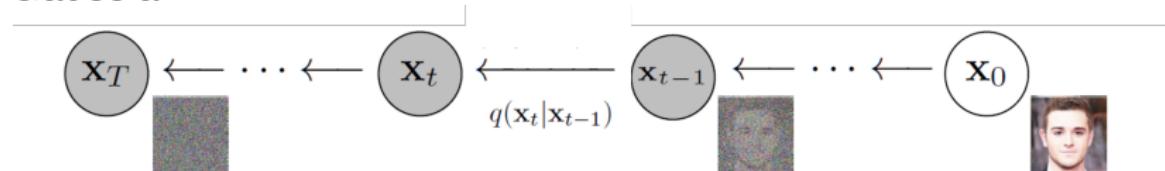
$$\mathbf{x}_t = \sqrt{1 - \beta_t} \mathbf{x}_{t-1} + \sqrt{\beta_t} \boldsymbol{\epsilon}_{t-1}$$

The step sizes are controlled by a variance schedule  $\{\beta_t \in (0, 1)\}_{t=1}^T$ .

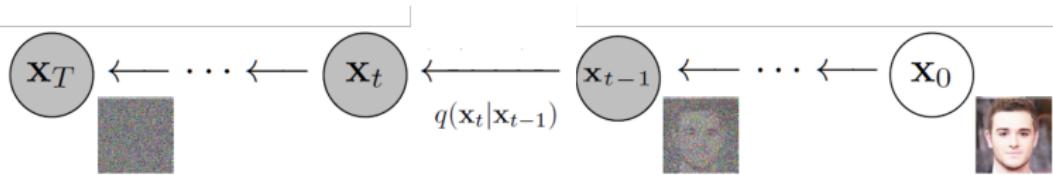
- When  $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , we have

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

- The sample  $\mathbf{x}_0$  gradually loses its distinguishable features as  $t$  becomes larger. Eventually when  $T \rightarrow \infty$ ,  $\mathbf{x}_T$  becomes isotropic Gaussian.



# Diffusion Models: Forward Diffusion Process



$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

We can sample  $\mathbf{x}_t$  at any arbitrary time step  $t$  in a closed form. Let  $\alpha_t = 1 - \beta_t$  and  $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ , then

$$\begin{aligned}\mathbf{x}_t &= \sqrt{\alpha_t} \mathbf{x}_{t-1} + \sqrt{1 - \alpha_t} \epsilon_{t-1} \\ &= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \bar{\epsilon}_{t-2} \\ &= \dots \\ &= \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon\end{aligned}$$

\*  $\bar{\epsilon}_{t-2}$  merged  $\epsilon_{t-1}$  and  $\epsilon_{t-2}$ .  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ . It follows that

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}).$$

# Diffusion Models: Reverse Diffusion Process

- If the diffusion process can be reversed, using  $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ , we can create a true sample from a Gaussian noise input  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .
- If  $\beta_t$  is small enough,  $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$  will also be Gaussian.

# Diffusion Models: Reverse Diffusion Process

- If the diffusion process can be reversed, using  $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ , we can create a true sample from a Gaussian noise input  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ .
- If  $\beta_t$  is small enough,  $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$  will also be Gaussian.
- We learn a model  $p_\theta$  to conduct the reverse diffusion process:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$$

$\mu_\theta$  and  $\Sigma_\theta$  are the outputs of a neural network parameterized by  $\theta$ .  
The inputs are  $\mathbf{x}_t$  and  $t$ .

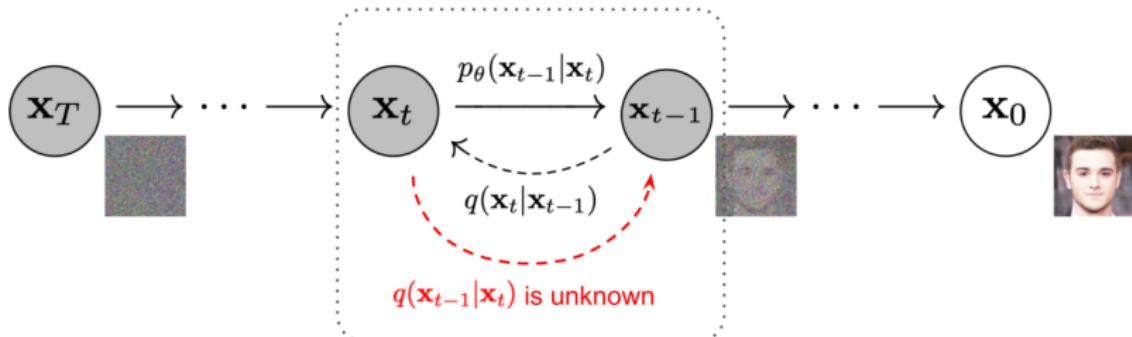
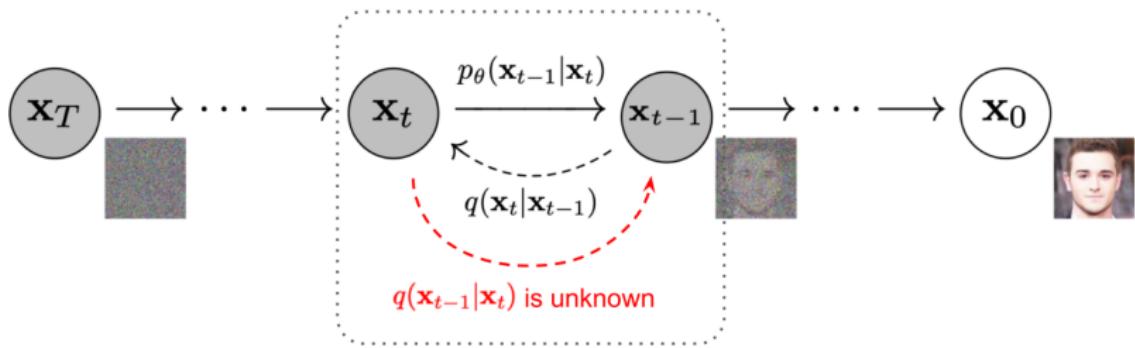


Image from the <https://lilianweng.github.io/posts/2021-07-11-diffusion-models>

# Diffusion Models: Reverse Diffusion Process



The reverse conditional probability is tractable when conditioned on  $\mathbf{x}_0$ :

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$$

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t, \quad \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}\beta_t}}{1 - \bar{\alpha}_t} \mathbf{x}_0$$

The derivation is a little complex and hence omitted. See [Sohl-Dickstein et al. 2015].

# Diffusion Models: Training

Minimize the variational bound on negative log-likelihood:

$$\begin{aligned}\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] &\leq \mathbb{E}_q \left[ -\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \\&= \mathbb{E}_q \left[ -\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\&= \mathbb{E}_q \left[ \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \| p(\mathbf{x}_T))}_{L_T} \right. \\&\quad \left. + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]\end{aligned}$$

# Diffusion Models: Training

Minimize the variational bound on negative log-likelihood:

$$\begin{aligned}\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] &\leq \mathbb{E}_q \left[ -\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \\&= \mathbb{E}_q \left[ -\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\&= \mathbb{E}_q \left[ \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \| p(\mathbf{x}_T))}_{L_T} \right. \\&\quad \left. + \sum_{t>1} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right] \\L_{t-1} &= \mathbb{E}_q \left[ \frac{1}{2\sigma_t^2} \|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2 \right] + C \\&= \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t(\mathbf{x}_0, \epsilon) - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) - \mu_\theta(\mathbf{x}_t(\mathbf{x}_0, \epsilon), t) \right\|^2 \right] + C\end{aligned}$$

For derivations, refer to [Sohl-Dickstein et al. 2015] and [Ho et al. 2020].

# Diffusion Models: Training

## Reparameterization

$$\begin{aligned} L_{t-1} &= \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t(\mathbf{x}_0, \epsilon) - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) - \mu_\theta(\mathbf{x}_t(\mathbf{x}_0, \epsilon), t) \right\|^2 \right] + C \\ &= \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \left\| \epsilon - \epsilon_\theta \left( \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2 \right] + C \end{aligned}$$

# Diffusion Models: Training

## Reparameterization

$$\begin{aligned} L_{t-1} &= \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t(\mathbf{x}_0, \epsilon) - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) - \mu_\theta(\mathbf{x}_t(\mathbf{x}_0, \epsilon), t) \right\|^2 \right] + C \\ &= \mathbb{E}_{\mathbf{x}_0, \epsilon} \left[ \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \left\| \epsilon - \epsilon_\theta \left( \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2 \right] + C \end{aligned}$$

A simplified objective [Ho et al. 2020] that ignores the weighting term and the final optimization objective is:

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[ \left\| \epsilon - \epsilon_\theta \left( \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2 \right] \quad (1)$$

## Optimization: SGD

# Diffusion Models: Training and Sampling

---

## Algorithm 1 Training

---

- 1: **repeat**
- 2:    $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
- 3:    $t \sim \text{Uniform}(\{1, \dots, T\})$
- 4:    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 5:   Take gradient descent step on  
      
$$\nabla_{\theta} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t)\|^2$$
- 6: **until** converged

---

---

## Algorithm 2 Sampling

---

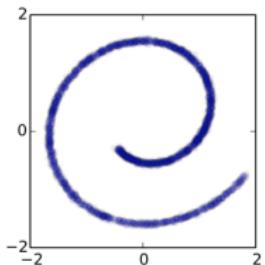
- 1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 2: **for**  $t = T, \dots, 1$  **do**
- 3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  if  $t > 1$ , else  $\mathbf{z} = \mathbf{0}$
- 4:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}} \left( \mathbf{x}_t - \frac{1 - \bar{\alpha}_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
- 5: **end for**
- 6: **return**  $\mathbf{x}_0$

---

Image from Ho et al. 2020.

# Diffusion Models: Examples

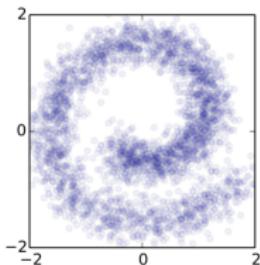
$t = 0$



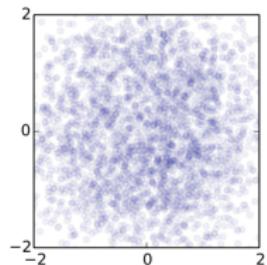
The forward trajectory

$q(\mathbf{x}_{0:T})$

$t = \frac{T}{2}$



$t = T$



The reverse trajectory

$p_{\theta}(\mathbf{x}_{0:T})$

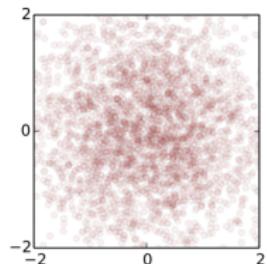
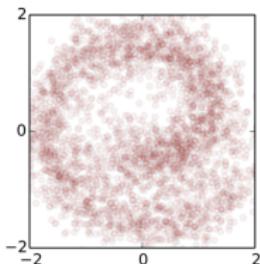
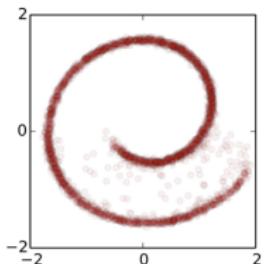
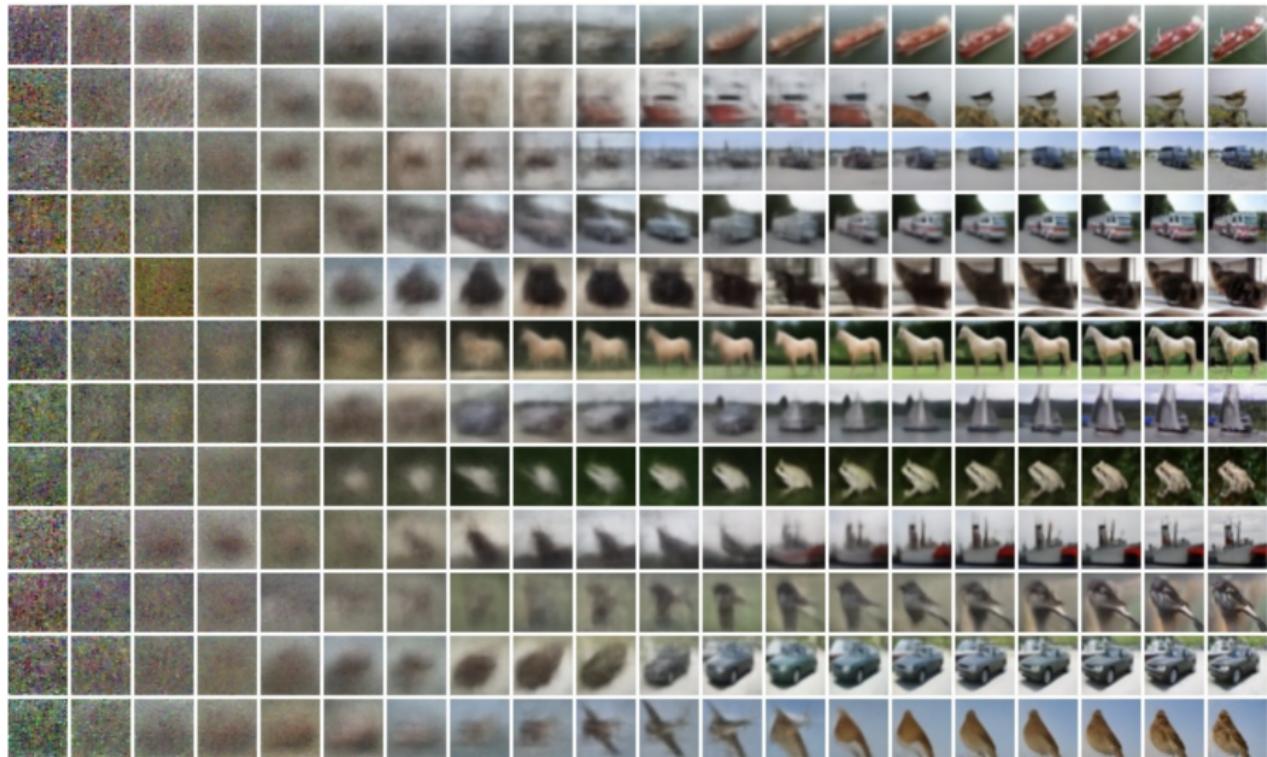


Image from Sohl-Dickstein et al. 2015.

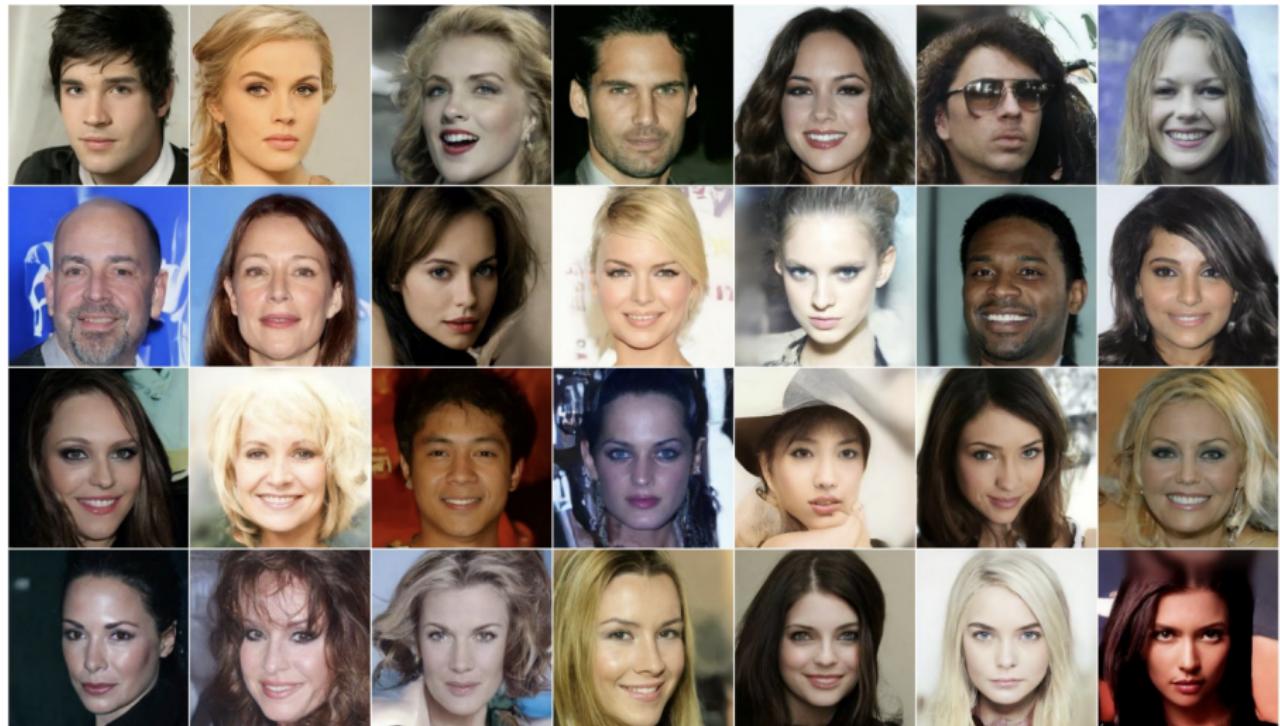
# Diffusion Models: Examples

CIFAR10 progressive generation [Ho et al. 2020]



# Diffusion Models: Examples

CelebA-HQ  $256 \times 256$  generated samples [Ho et al. 2020]



# Diffusion Models: Advantages and Disadvantages

- Advantages

- The quality of generated samples are often higher than VAE and GAN.
- Probability distribution is explicit.

# Diffusion Models: Advantages and Disadvantages

- Advantages
  - The quality of generated samples are often higher than VAE and GAN.
  - Probability distribution is explicit.
- Disadvantages
  - The training process is time-consuming.
  - It is very slow to generate a sample from DDPM since  $T$  is often very large, i.e. 1000.

## References

- [Sohi-Dickstein et al. 2015] Deep Unsupervised Learning using Nonequilibrium Thermodynamics
- [Ho et al. 2020] Denoising Diffusion Probabilistic Nodels
- [Nichol et al. 2021] Improved Denoising Diffusion Probabilistic Models
- [Song et al. 2020 ] Denoising Diffusion Implicit Models
- <https://lilianweng.github.io/posts/2021-07-11-diffusion-models/>

# Learning Outcomes

- Understand the main ideas of VAE, GAN, and diffusion model
- Understand the derivation of the objective function of VAE
- Know the advantages and disadvantages of VAE, GAN, and diffusion model
- Be able to use at least one of VAE, GAN, and diffusion model to generate realistic data samples.