CSE7850/CX4803 Machine Learning in Computational Biology

Lecture 12: Generative AI

Yunan Luo

Acknowledgements: Some of the slides were adapted from
https://github.com/r-isachenko/2022-2023-DGM-AIMasters-course, https://cvpr2022-tutorial-diffusion-models.github.io/

Responses from the entry survey

Fall 2024 CSE7850/CX4803 First survey

[Optional] Are there any special topics you hope to learn in this course?

Your answer

Some of the responses:

generative models for biology

LLMs and applying it to the healthcare field

GAN

Neural networks, PCA, dimensionality reduction

Applying MINLP

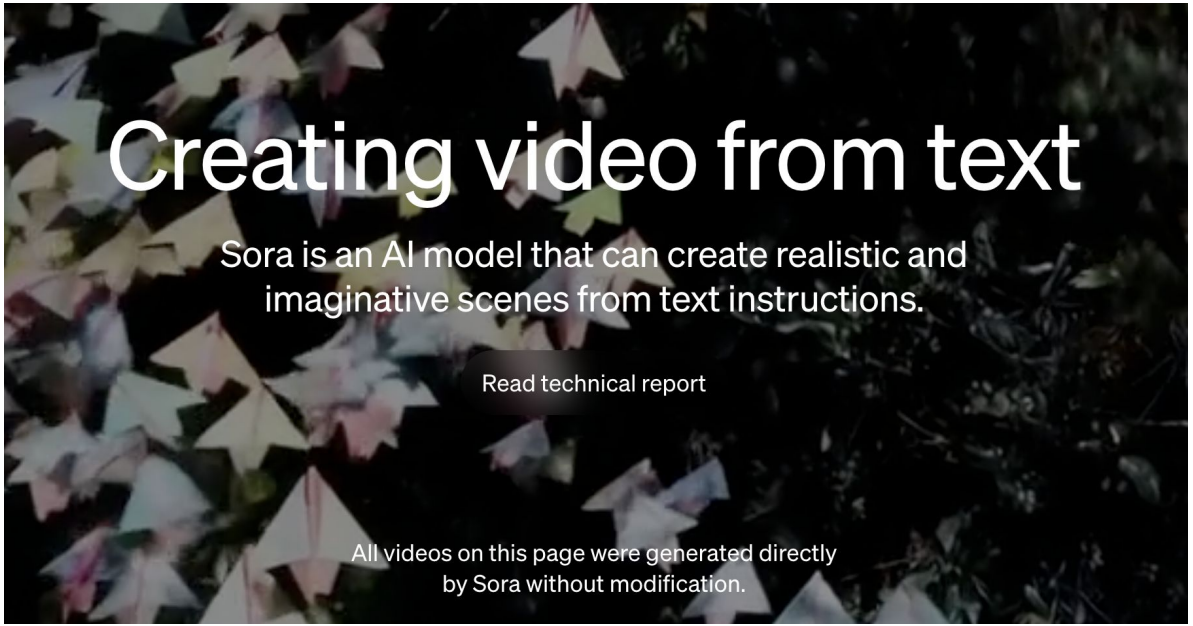
I hope to learn more about protein language models and methods of using diffusion/flow matching for novel structure generation

I am confuse with Classification in my research now, so I more interested this topic.

Diffusion models, optimal transport in the context of comp bio

Generative ML models!

Last week

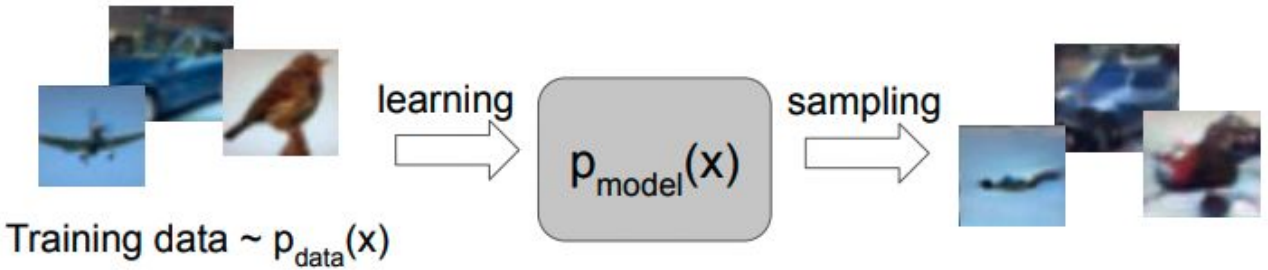


https://openai.com/sora

Sora: text-to-video generative model (OpenAI)

Generative Modeling

Given training data, generate new samples from same distribution



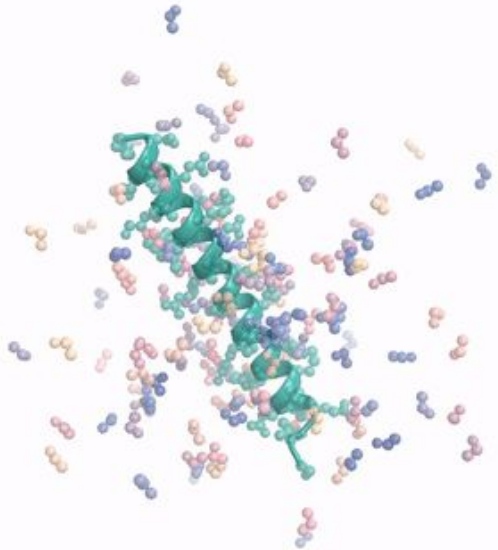
Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sampling new x from $p_{model}(x)$**

Slides credits: Stanford CS231N, 2022

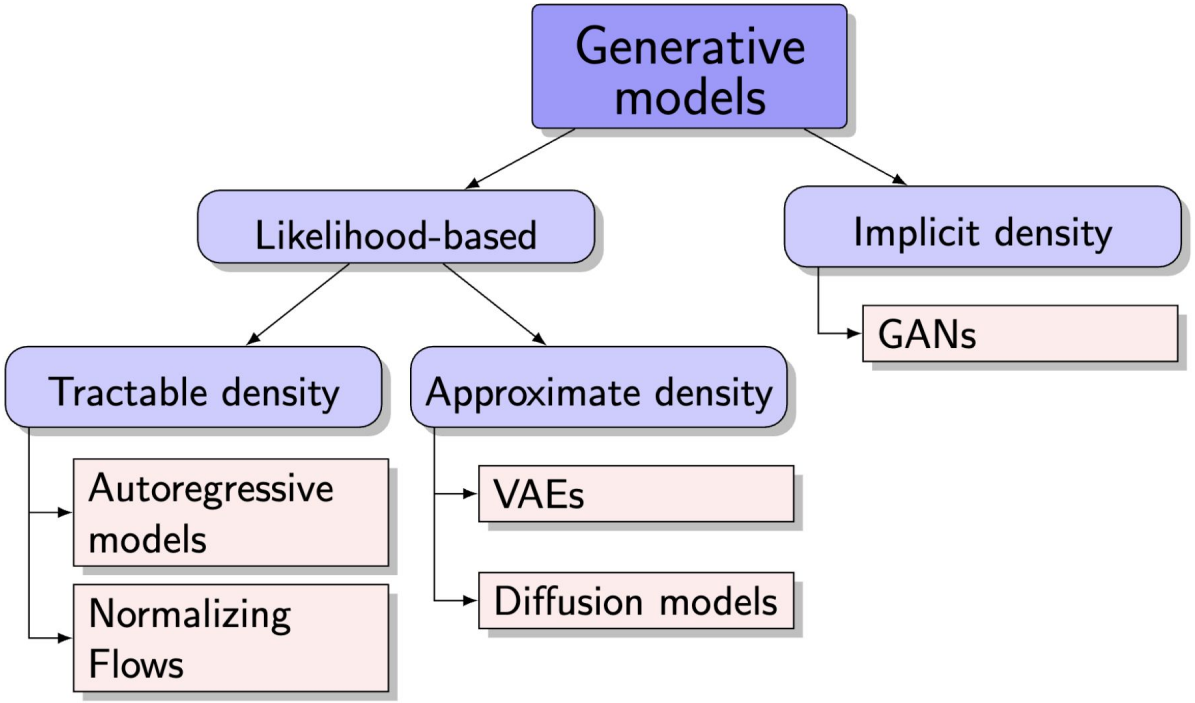
Examples: ChatGPT (2022)

Example: Protein Design (2022)



https://www.ipd.uw.edu/2022/12/a-diffusion-model-for-protein-design/

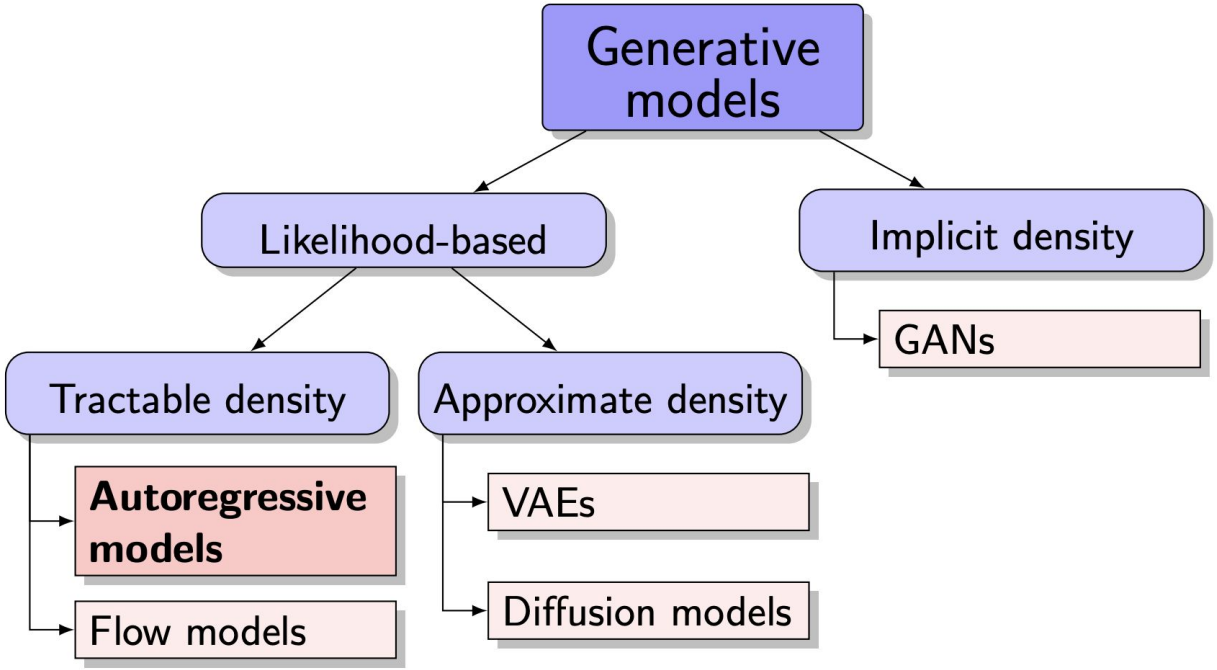
Generative models zoo



Outline

- Autoregressive models
- VAE
- GAN
- Diffusion models

Generative models zoo



Problem statement

We are given i.i.d. samples $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, where $\mathbf{x}_i \in \mathcal{X}$ (e.g. $\mathcal{X} = \mathbb{R}^m$) comes from unknown distribution $\pi(\mathbf{x})$.

Goal

We would like to learn a distribution $\pi(\mathbf{x})$ for

- evaluating $\pi(\mathbf{x})$ for new samples (how likely to get object $\mathbf{x}$?);
- sampling from $\pi(\mathbf{x})$ (to get new objects $\mathbf{x} \sim \pi(\mathbf{x})$).

Instead of searching true $\pi(\mathbf{x})$ over all probability distributions, learn function approximation $p(\mathbf{x}|\boldsymbol{\theta}) \approx \pi(\mathbf{x})$.

Autoregressive models

MLE problem (Maximum likelihood estimation)

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

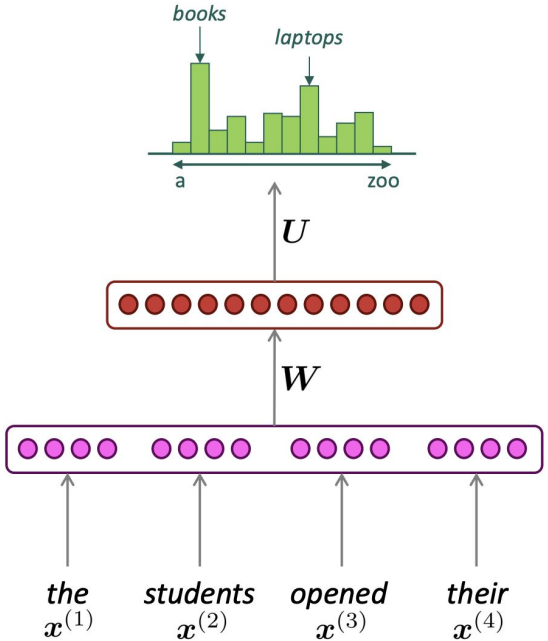
- We would like to solve the problem using gradient-based optimization.
- We have to efficiently compute $\log p(\mathbf{x}|\theta)$ and $\frac{\partial \log p(\mathbf{x}|\theta)}{\partial \theta}$.

Likelihood as product of conditionals

Let $\mathbf{x} = (x_1, \ldots, x_m)$, $\mathbf{x}_{1:j} = (x_1, \ldots, x_j)$. Then

$$p(\mathbf{x}|\theta) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1}, \theta); \quad \log p(\mathbf{x}|\theta) = \sum_{j=1}^{m} \log p(x_j|\mathbf{x}_{1:j-1}, \theta).$$

Example: $p(x_1, x_2, x_3) = p(x_2) \cdot p(x_1|x_2) \cdot p(x_3|x_1, x_2)$.



Autoregressive models

MLE problem (Maximum likelihood estimation)

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

- We would like to solve the problem using gradient-based optimization.
- We have to efficiently compute $\log p(\mathbf{x}|\theta)$ and $\frac{\partial \log p(\mathbf{x}|\theta)}{\partial \theta}$.

challenges

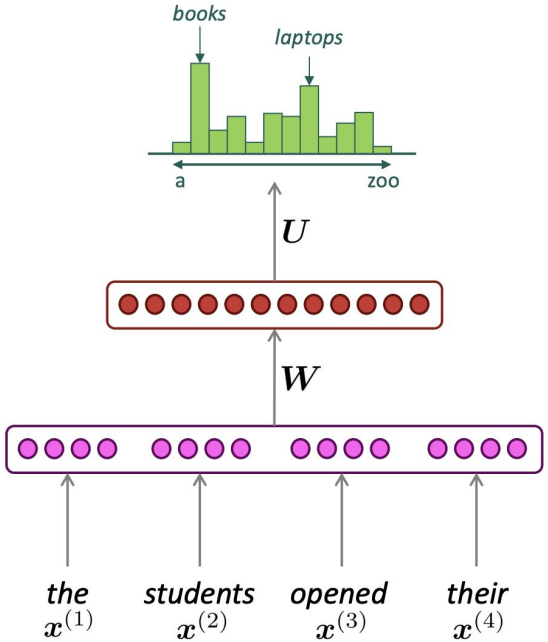
Likelihood as product of conditionals

Let $\mathbf{x} = (x_1, \ldots, x_m)$, $\mathbf{x}_{1:j} = (x_1, \ldots, x_j)$. Then

$$p(\mathbf{x}|\theta) = \prod_{j=1}^{m} p(x_j|\mathbf{x}_{1:j-1}, \theta); \quad \log p(\mathbf{x}|\theta) = \sum_{j=1}^{m} \log p(x_j|\mathbf{x}_{1:j-1}, \theta).$$

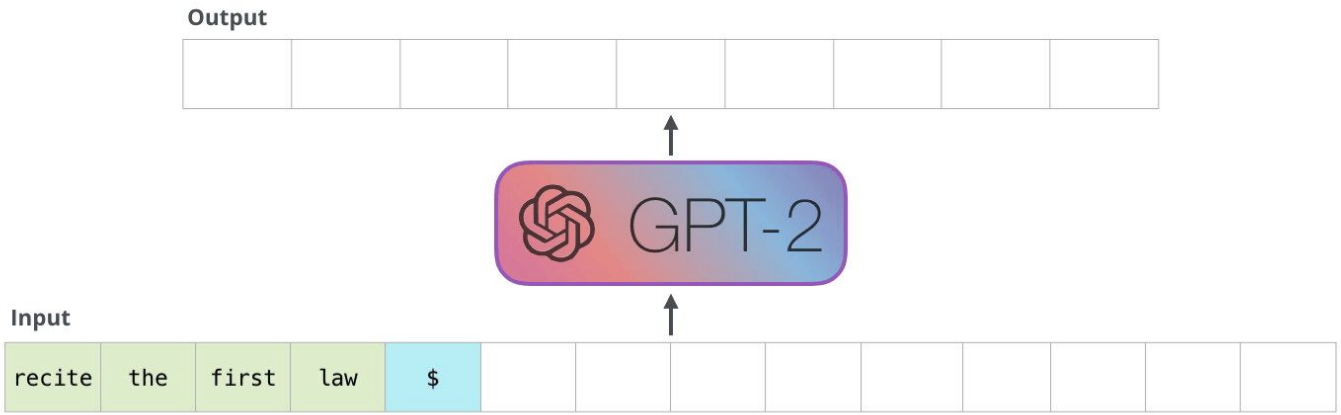
solution

Example: $p(x_1, x_2, x_3) = p(x_2) \cdot p(x_1|x_2) \cdot p(x_3|x_1, x_2)$.

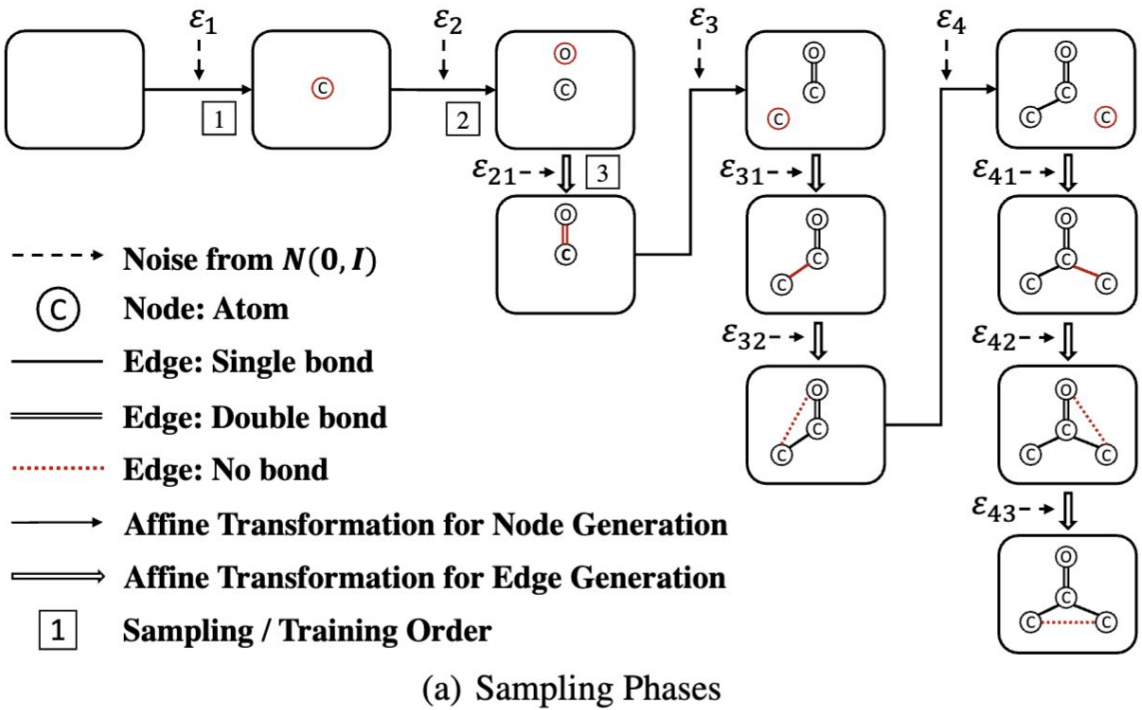


How to generate?

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



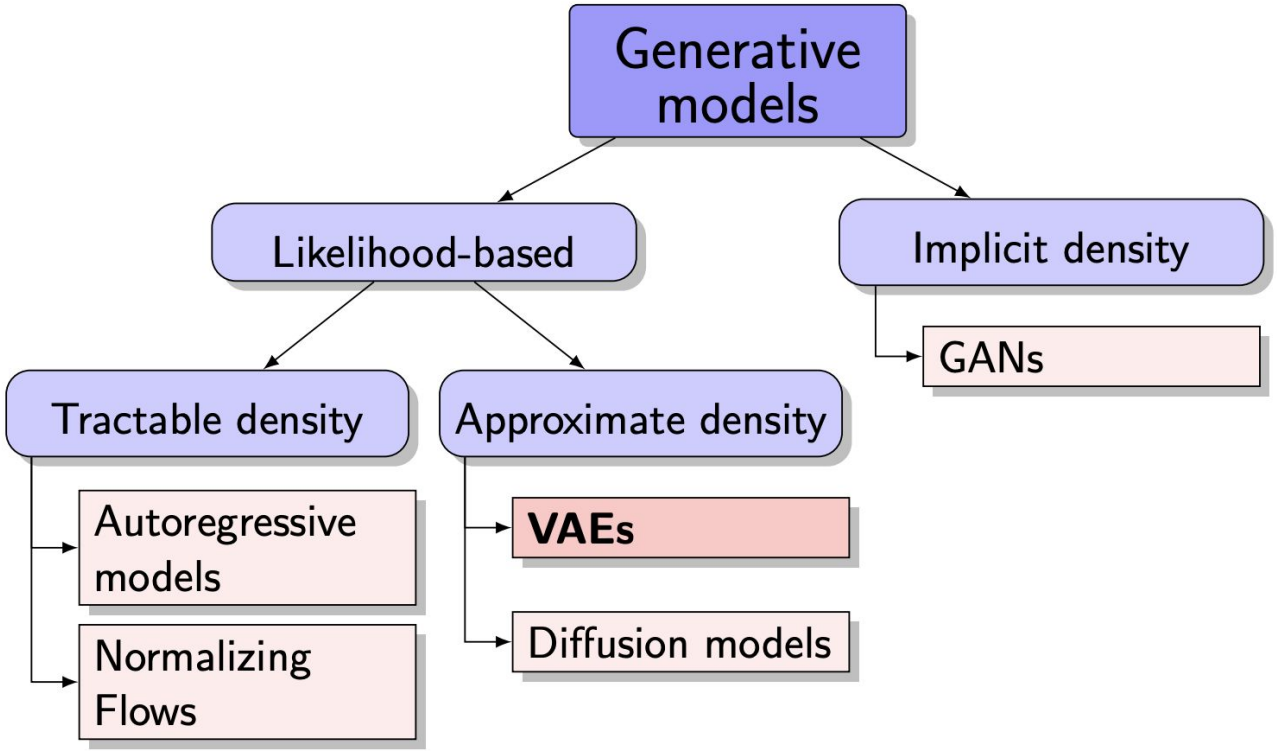
Autoregressive model for Graphs



(a) Sampling Phases

Shi, Chence et al. “GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation.” *ArXiv* abs/2001.09382 (2020): n. pag.

Generative models zoo



Latent variable models

MLE problem

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

The distribution $p(\mathbf{x}|\theta)$ could be very complex and intractable (as well as real distribution $\pi(\mathbf{x})$).

Extended probabilistic model

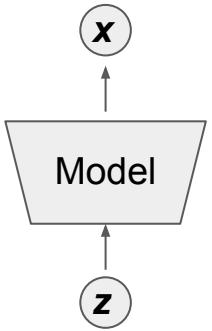
Introduce latent variable $\mathbf{z}$ for each sample $\mathbf{x}$

$$p(\mathbf{x}, \mathbf{z}|\theta) = p(\mathbf{x}|\mathbf{z}, \theta)p(\mathbf{z}); \quad \log p(\mathbf{x}, \mathbf{z}|\theta) = \log p(\mathbf{x}|\mathbf{z}, \theta) + \log p(\mathbf{z}).$$

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

Motivation

The distributions $p(\mathbf{x}|\mathbf{z}, \theta)$ and $p(\mathbf{z})$ could be quite simple.



Intuition: consider ***x*** as an image, ***z*** as latent factors to generate ***x***: attributes, orientation, etc

Latent variable models

MLE problem

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

The distribution $p(\mathbf{x}|\theta)$ could be very complex and intractable (as well as real distribution $\pi(\mathbf{x})$).

challenges

Extended probabilistic model

Introduce latent variable $\mathbf{z}$ for each sample $\mathbf{x}$

$$p(\mathbf{x}, \mathbf{z}|\theta) = p(\mathbf{x}|\mathbf{z}, \theta)p(\mathbf{z}); \quad \log p(\mathbf{x}, \mathbf{z}|\theta) = \log p(\mathbf{x}|\mathbf{z}, \theta) + \log p(\mathbf{z}).$$

solution

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

Motivation

The distributions $p(\mathbf{x}|\mathbf{z}, \theta)$ and $p(\mathbf{z})$ could be quite simple.

Intuition: consider ***x*** as an image, ***z*** as latent factors to generate ***x***: attributes, orientation, etc

Maximum likelihood estimation for latent variable models

MLE for extended problem

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) =$$

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

However, $\mathbf{Z}$ is unknown.

MLE for original problem

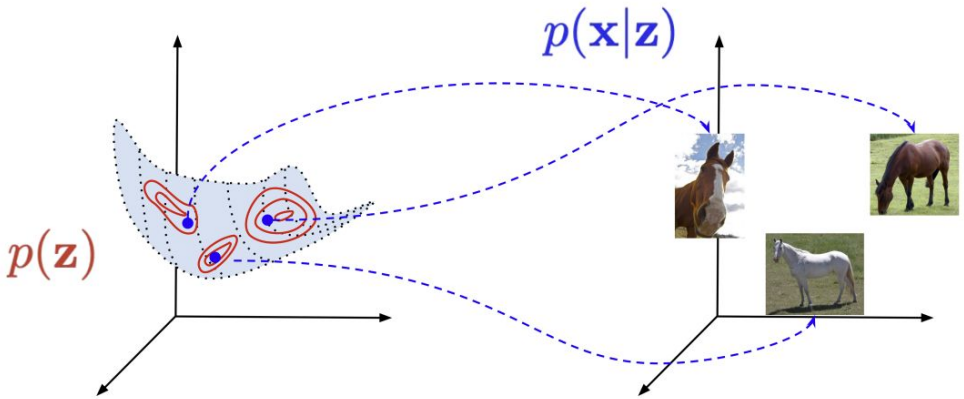
"Average" all possible $\boldsymbol{z}$!

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

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log \int p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) d\mathbf{z}_i =$$

$$= \arg\max_{\boldsymbol{\theta}} \log \sum_{i=1}^{n} \int p(\mathbf{x}_i|\mathbf{z}_i, \boldsymbol{\theta}) p(\mathbf{z}_i) d\mathbf{z}_i.$$

Naive approach



Monte-Carlo estimation

$$p(\mathbf{x}|\boldsymbol{\theta}) = \int p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta})p(\mathbf{z})d\mathbf{z} = \mathbb{E}_{p(\mathbf{z})}p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}) \approx \frac{1}{K}\sum_{k=1}^{K} p(\mathbf{x}|\mathbf{z}_k,\boldsymbol{\theta}),$$

where each $\mathbf{z}_k \sim p(\mathbf{z})$.

Challenge: to cover the space properly, the number of samples grows exponentially with respect to dimensionality of $\mathbf{z}$.

Maximum likelihood

$$p(\mathbf{x}) = \int p(\mathbf{z})\, p(\mathbf{x} \mid \mathbf{z})\, d\mathbf{z}$$

Intractable to compute for every z

Gaussian prior

Neural network

Maximizing a lower bound

$$\log p(\mathbf{x}) = \log \int p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

$$= \log \int q(\mathbf{z})\, \frac{p(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

Auxiliary distribution **q(z)**

Maximizing a lower bound

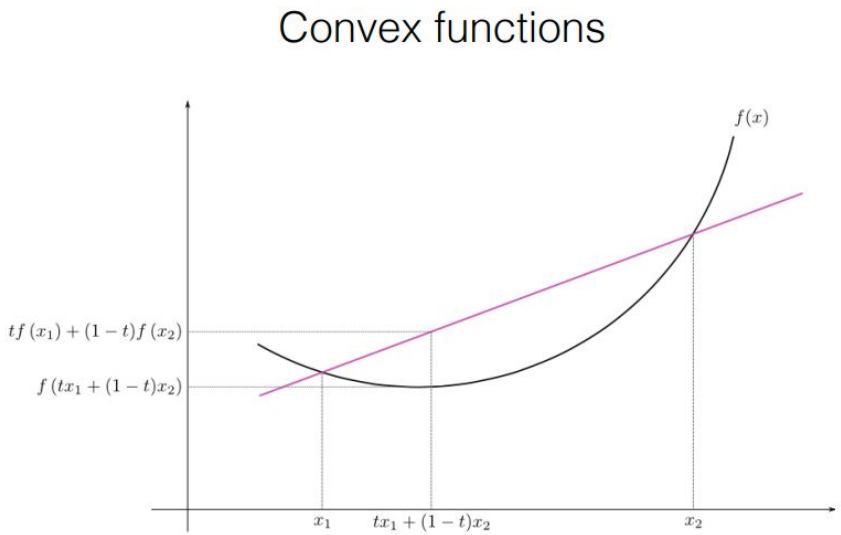
$$\log p(\mathbf{x}) = \log \int p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

$$= \log \int q(\mathbf{z})\, \frac{p(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

$$\geq \int q(\mathbf{z}) \log \left[\frac{p(\mathbf{z})}{q(\mathbf{z})}\, p(\mathbf{x}|\mathbf{z}) \right] \mathrm{d}\mathbf{z}$$

Auxiliary distribution **q(z)**

Jensen's inequality



$$tf(x_1) + (1-t)f(x_2) \geq f(tx_1 + (1-t)x_2)$$

Generalized version $$\int_x f(x)p(x) \geq f(\int_x xp(x))$$

Slides credit: CS598JP, UIUC, 2020

Maximizing a lower bound

$$\log p(\mathbf{x}) = \log \int p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

$$= \log \int q(\mathbf{z})\, \frac{p(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

Auxiliary distribution **q(z)**

$$\geq \int q(\mathbf{z}) \log \left[\frac{p(\mathbf{z})}{q(\mathbf{z})}\, p(\mathbf{x}|\mathbf{z}) \right] \mathrm{d}\mathbf{z}$$

Jensen's inequality

Maximizing a lower bound

$$\log p(\mathbf{x}) = \log \int p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

$$= \log \int q(\mathbf{z})\, \frac{p(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z}$$

Auxiliary distribution **q(z)**

$$\geq \int q(\mathbf{z}) \log \left[\frac{p(\mathbf{z})}{q(\mathbf{z})}\, p(\mathbf{x}|\mathbf{z}) \right] \mathrm{d}\mathbf{z}$$

Jensen's inequality

$$= \mathbb{E}_q \left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})} \right] + \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z}) \right]$$

Maximizing a lower bound

$$\log p(\mathbf{x}) \geq \mathbb{E}_q \left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})} \right] + \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z}) \right]$$

- Reconstruction term
  - Encourages the model to reconstruct the input
- If we parameterize p(x|z) as Gaussian

$$\begin{aligned} \log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; G_{\boldsymbol{\theta}}(\mathbf{z}), \eta \mathbf{I}) \\ &= \log \left[\frac{1}{(2\pi\eta)^{D/2}} \exp\left(-\frac{1}{2\eta} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2 \right) \right] \\ &= -\frac{1}{2\eta} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2 + \text{const} \end{aligned}$$

Adapted from slides of Roger Grosse and Jimmy Ba

Maximizing a lower bound

$$\log p(\mathbf{x}) \geq \mathbb{E}_q \left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})} \right] + \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z}) \right]$$

- Can be written as $-\mathrm{D}_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z}))$. KL term.
- D_{KL} is the Kullback-Leibler (KL) divergence $\mathrm{D}_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z})) \triangleq \mathbb{E}_q \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z})} \right]$
  - Widely used to measure the distance between two probability distributions
- Typically, $p(\mathbf{z}) = N(\mathbf{0}, \mathbf{1})$
  - The KL term encourages q(z) to be close to the standard normal distribution $N(\mathbf{0}, \mathbf{1})$.

Adapted from slides of Roger Grosse and Jimmy Ba

Maximizing a lower bound

Variational lower bound

$$\log p(\mathbf{x}) \geq \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z})\right] - \mathrm{D}_{\mathrm{KL}}(q\|p)$$

The role of each of the two terms:

The reconstruction term

$$\mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] = -\frac{1}{2\sigma^2}\mathbb{E}_q[\|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2] + \text{const}$$

is minimized when q is a point mass on

$$\mathbf{z}_* = \arg\min_{\mathbf{z}} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2.$$

But a point mass would have infinite KL divergence.

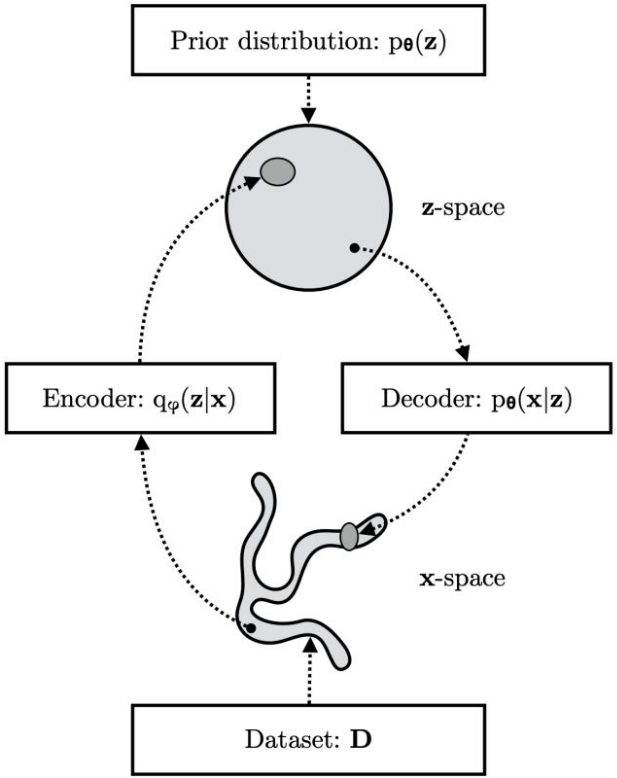
Adapted from slides of Roger Grosse and Jimmy Ba

Variational autoencoder (VAE)

- VAE learns stochastic mapping between x-space, from complicated distribution $\pi(x)$, and a latent z-space, with simple distribution.

- The stochastic *encoder* $q(z|x, \varphi)$ (inference model), approximates the true but intractable posterior $p(z|x, \theta)$ of the generative model.

- The generative model learns a joint distribution $p(x, z|\theta) = p(z)p(x|z, \theta)$, with a prior distribution $p(z)$, and a stochastic *decoder* $p(x|z, \theta)$.

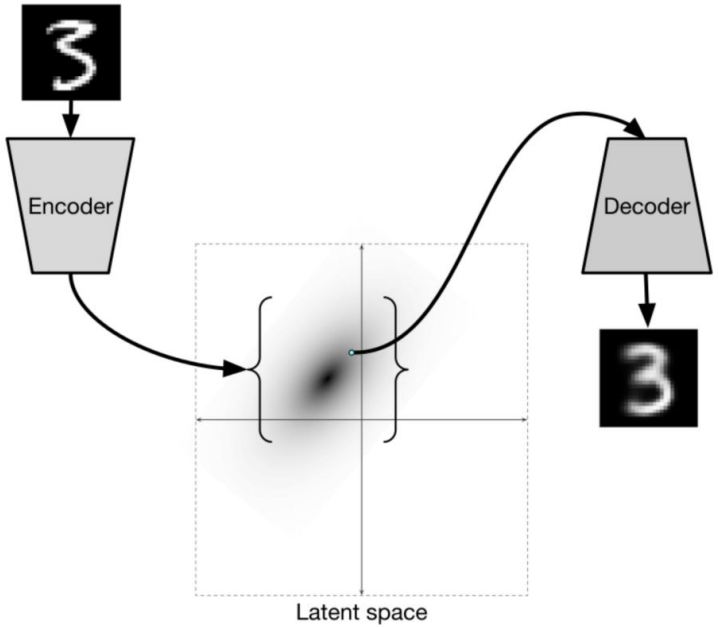


Variational autoencoder (VAE)

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

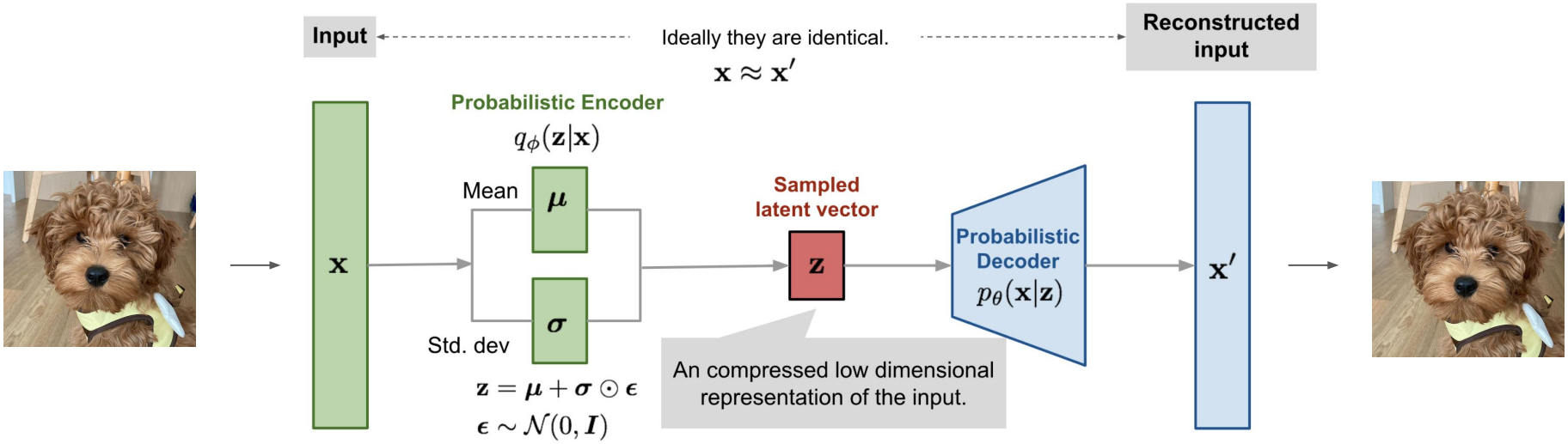
decoder

encoder



Variational autoencoder (VAE)

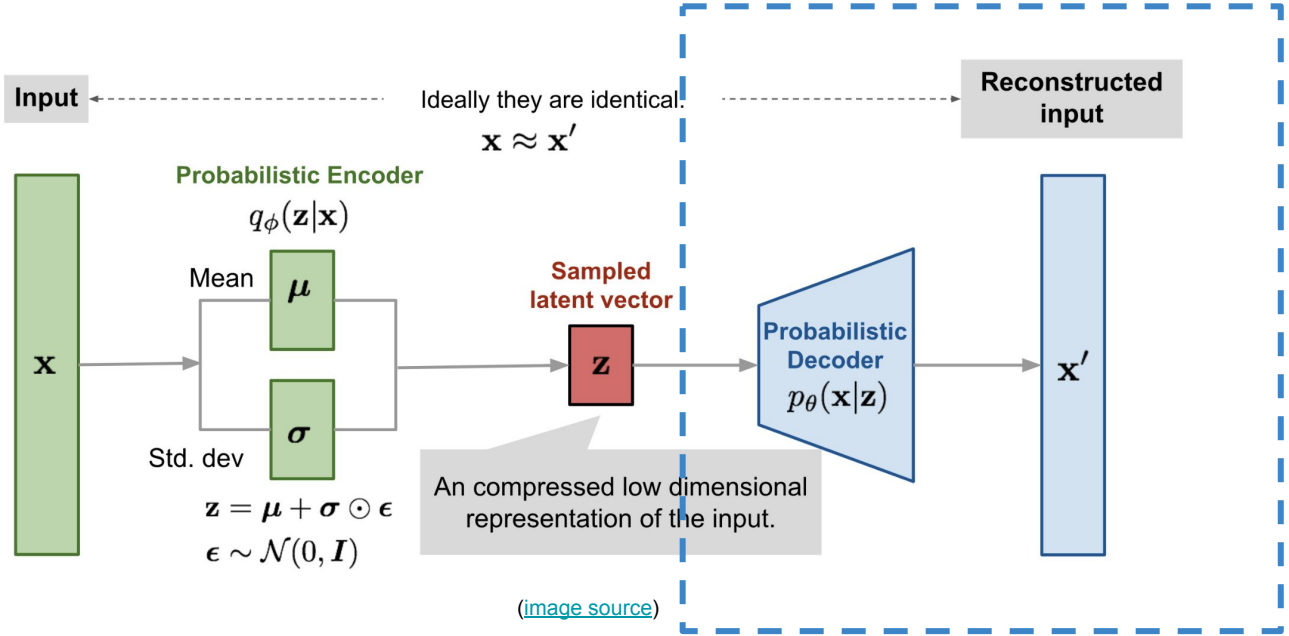
- Encoder $q(\mathbf{z}|\mathbf{x}, \phi) = \text{NN}_e(\mathbf{x}, \phi)$ outputs $\boldsymbol{\mu}_\phi(\mathbf{x})$ and $\boldsymbol{\sigma}_\phi(\mathbf{x})$.
- Decoder $p(\mathbf{x}|\mathbf{z}, \boldsymbol{\theta}) = \text{NN}_d(\mathbf{z}, \boldsymbol{\theta})$ outputs parameters of the sample distribution.



(image source)

How to generate?

- Use the decoder to generate new data

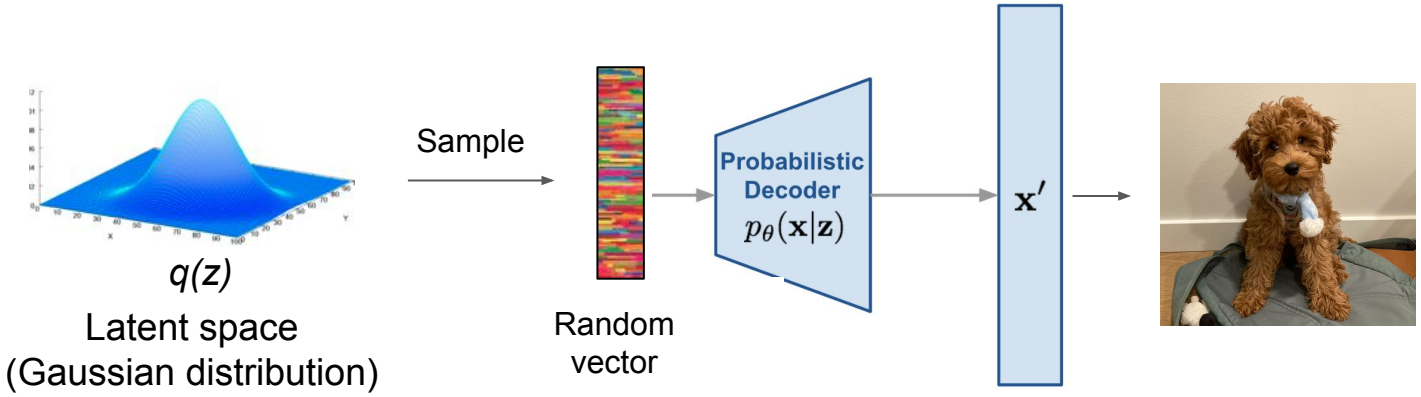


(image source)

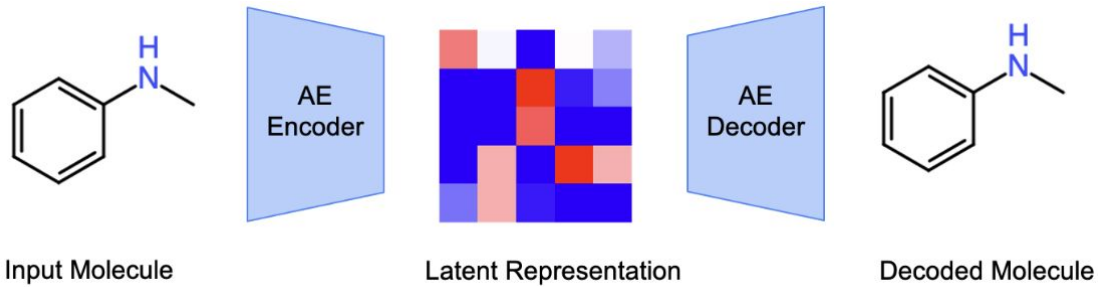
How to generate?

- Sample a random Gaussian vector from the latent space
- Decode it to an image using the trained decoder

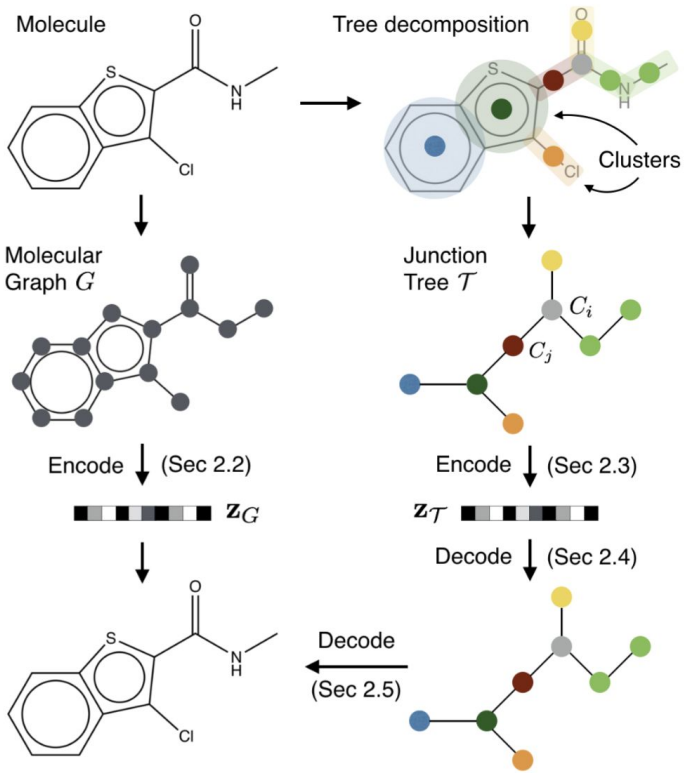
$$\max \; \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z})\right] - \mathrm{D}_{\mathrm{KL}}(q\|p)$$



VAE for Graphs



Junction Tree Variational Autoencoder



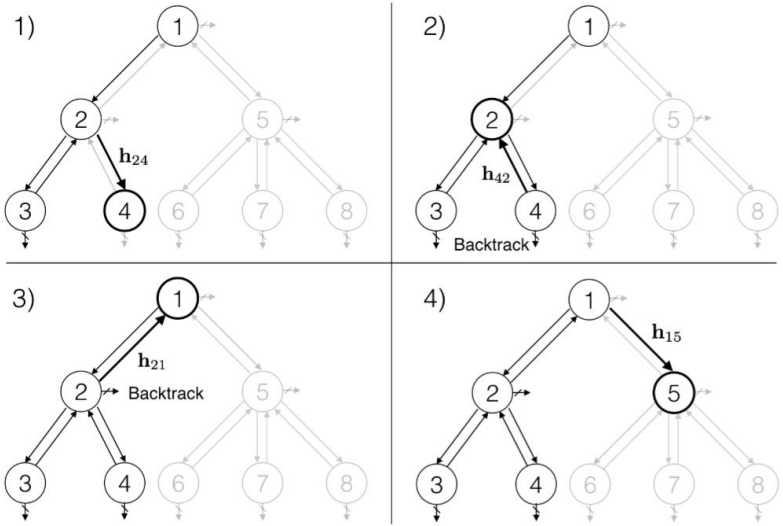
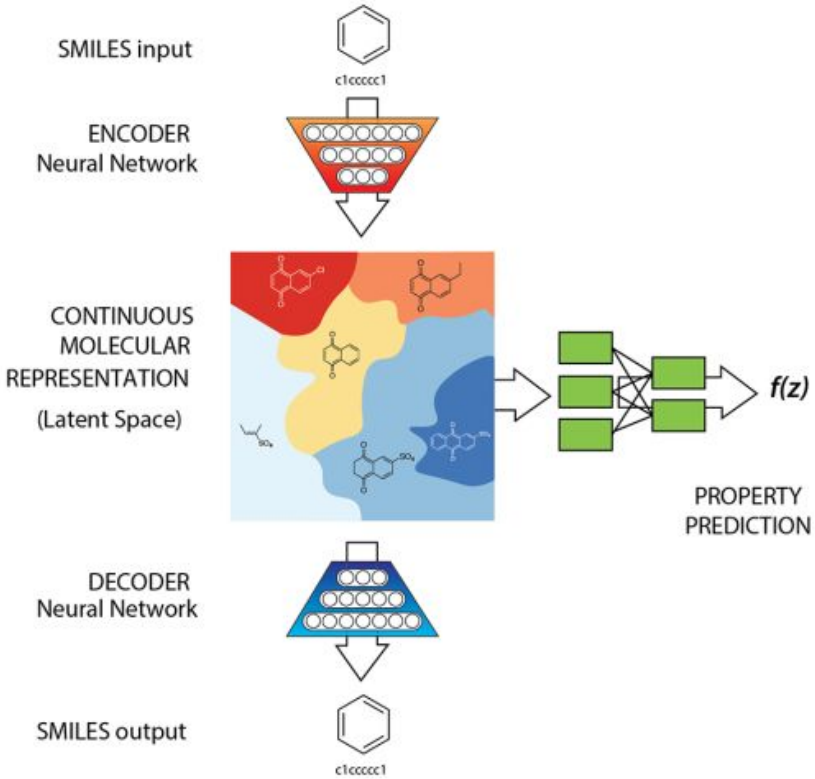


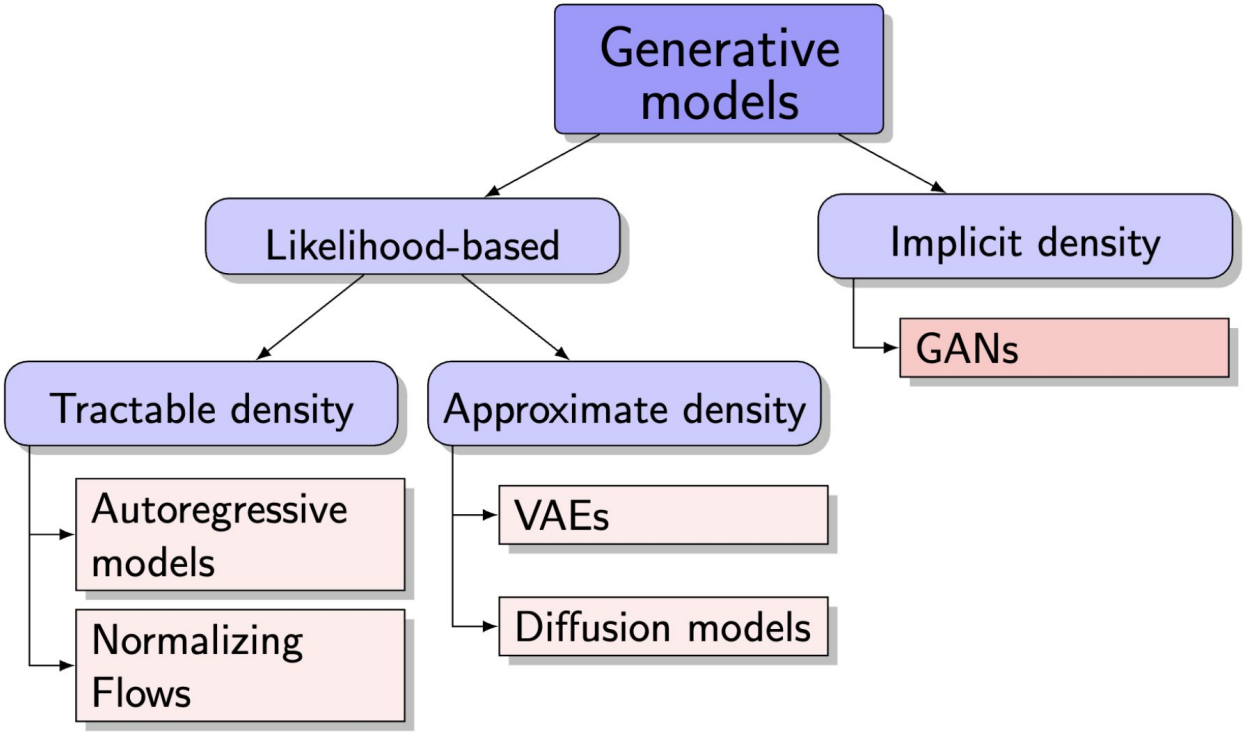
Figure 4. Illustration of the tree decoding process. Nodes are labeled in the order in which they are generated. 1) Node 2 expands child node 4 and predicts its label with message $\mathbf{h}_{24}$. 2) As node 4 is a leaf node, decoder backtracks and computes message $\mathbf{h}_{42}$. 3) Decoder continues to backtrack as node 2 has no more children. 4) Node 1 expands node 5 and predicts its label.

Jin, Wengong et al. "Junction Tree Variational Autoencoder for Molecular Graph Generation." *ArXiv* abs/1802.04364 (2018): n. pag.

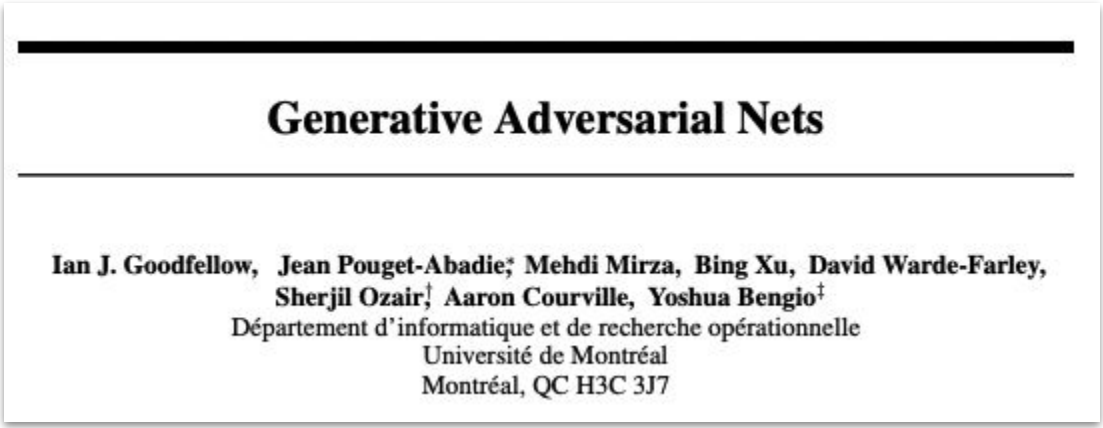
Optimizing molecular property using VAE



https://pubs.acs.org/doi/10.1021/acscentsci.7b00572


Generative models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive models
Normalizing Flows
VAEs
Diffusion models

Generative Adversarial network (GAN)



Generative Adversarial Nets

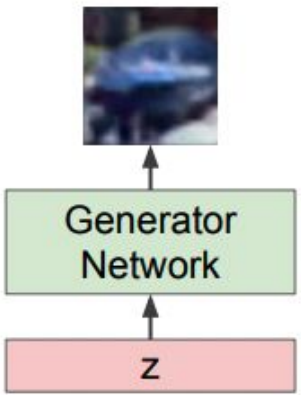
Ian J. Goodfellow, Jean Pouget-Abadie[*], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[†], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

GAN

- **Problem**: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- **Solution**: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Output: Sample from training distribution



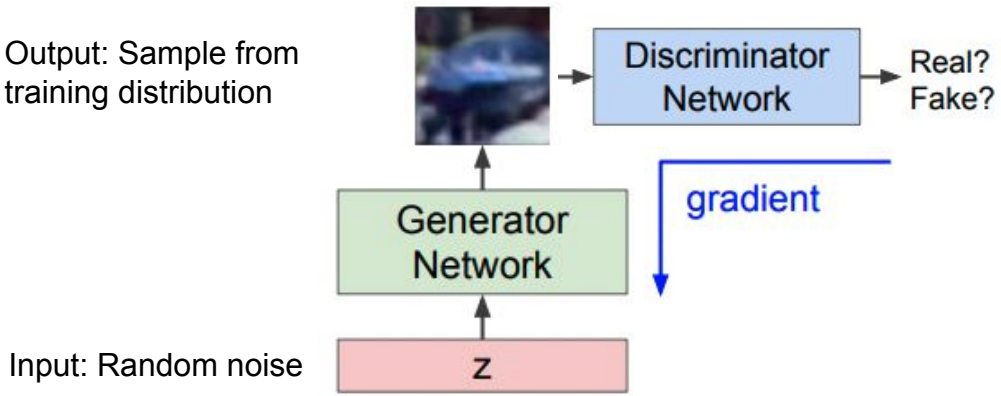
Input: Random noise

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Objective: generated images should look "real"

GAN

- **Problem**: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- **Solution**: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.



But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

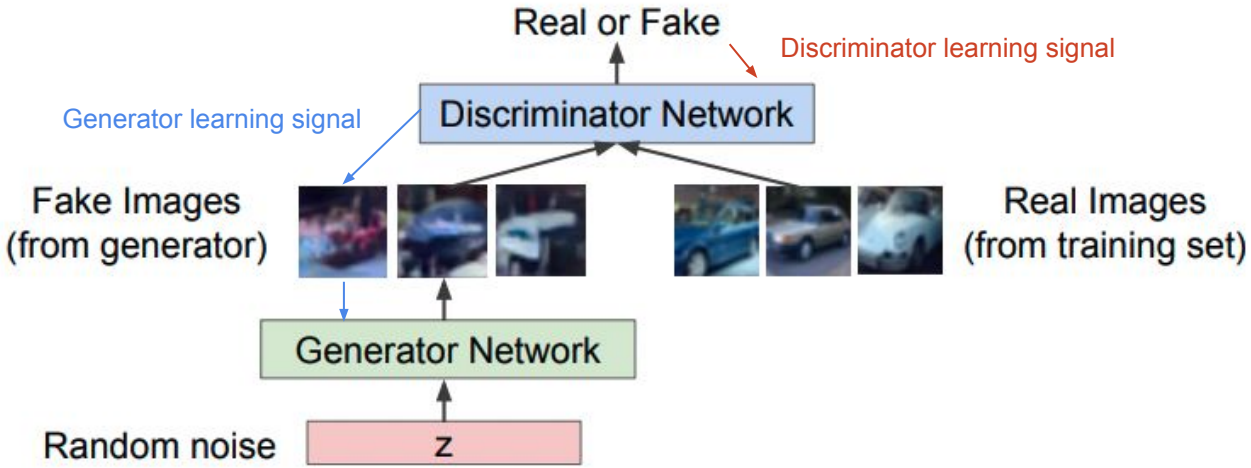
Objective: generated images should look "real"

Solution: Use a discriminator network to tell whether the generate image is within data distribution ("real") or not

Adapted from Stanford CS231N, 2022

Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images



Adapted from Stanford CS231N, 2022

Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

- **Discriminator** (θ_d) wants to maximize objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)

Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images
Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

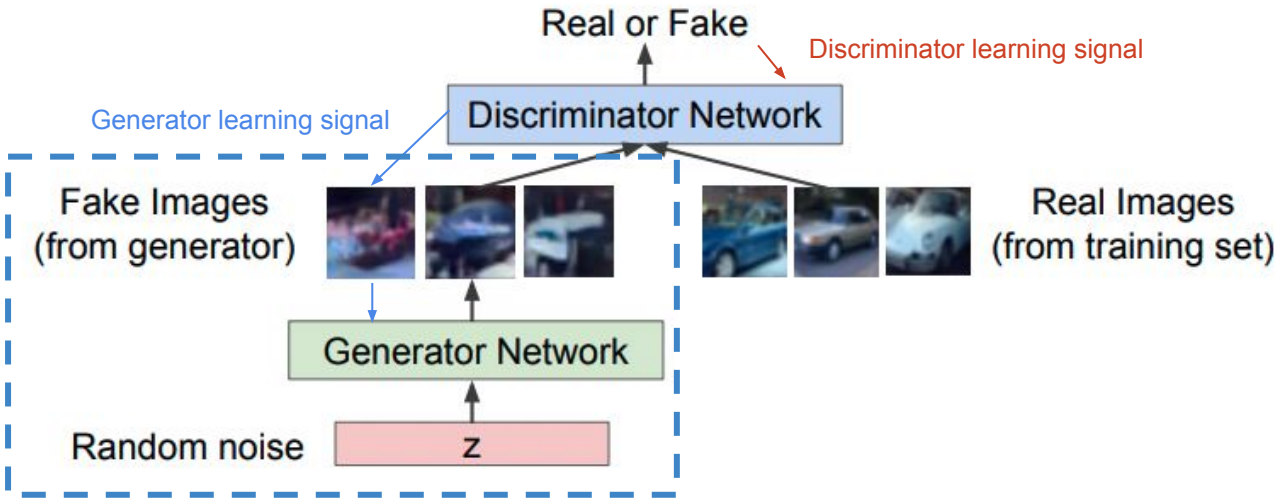
- **Discriminator** (θ_d) wants to maximize objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)

- **Generator** (θ_g) wants to minimize objective such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

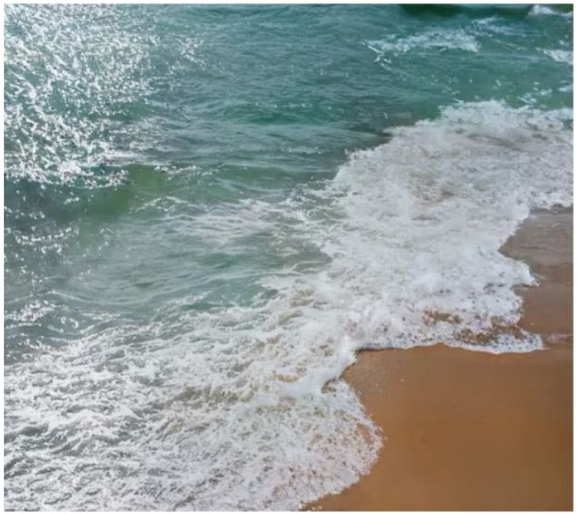
Adapted from Stanford CS231N, 2022

How to generate?

- Use the generator to generate new data



The quality of GAN-generated images is improving

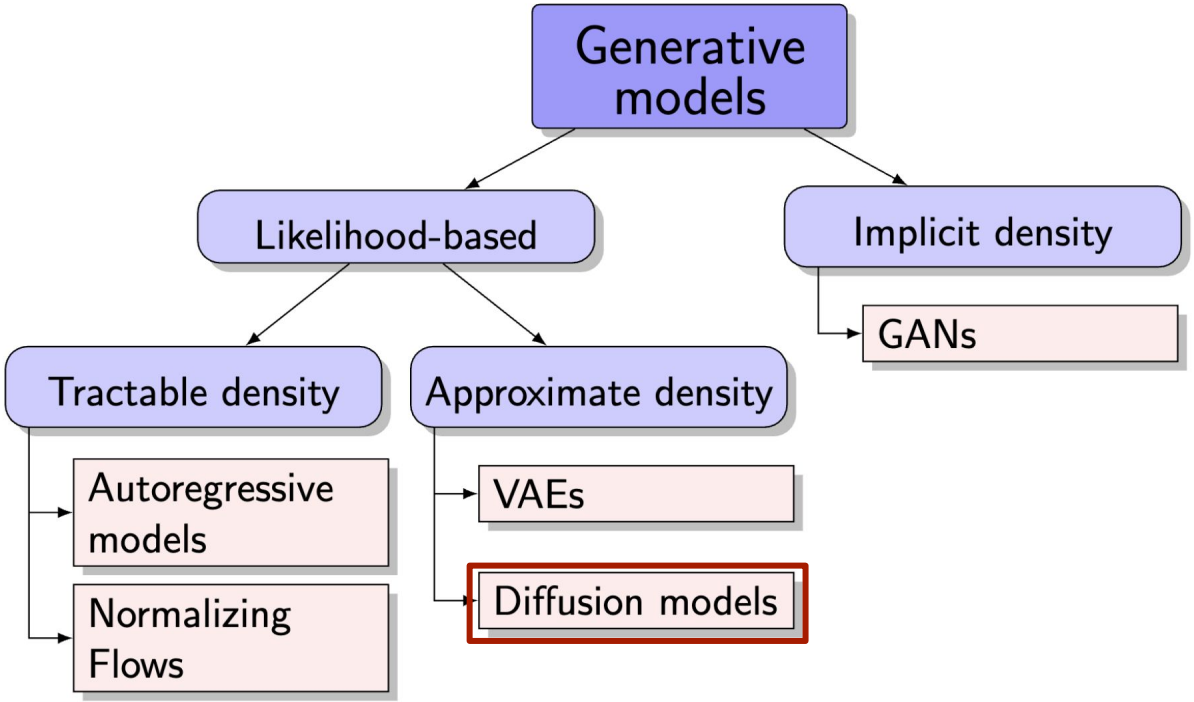


https://arxiv.org/abs/2106.12423

GAN in 2023?

GigaGAN: Kang et al., CVPR 2023

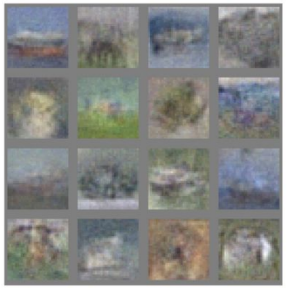
https://mingukkang.github.io/GigaGAN/


Generative models
Likelihood-based
Implicit density
Tractable density
Approximate density
GANs
Autoregressive models
Normalizing Flows
VAEs
Diffusion models

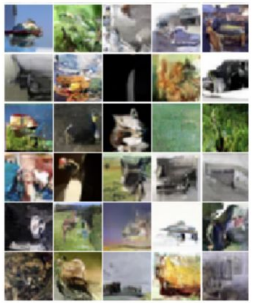
Where we came from

VAEs, 2013

GANs, 2014



PixelCNN, 2016



BigGAN, 2019

Imagen, 2022

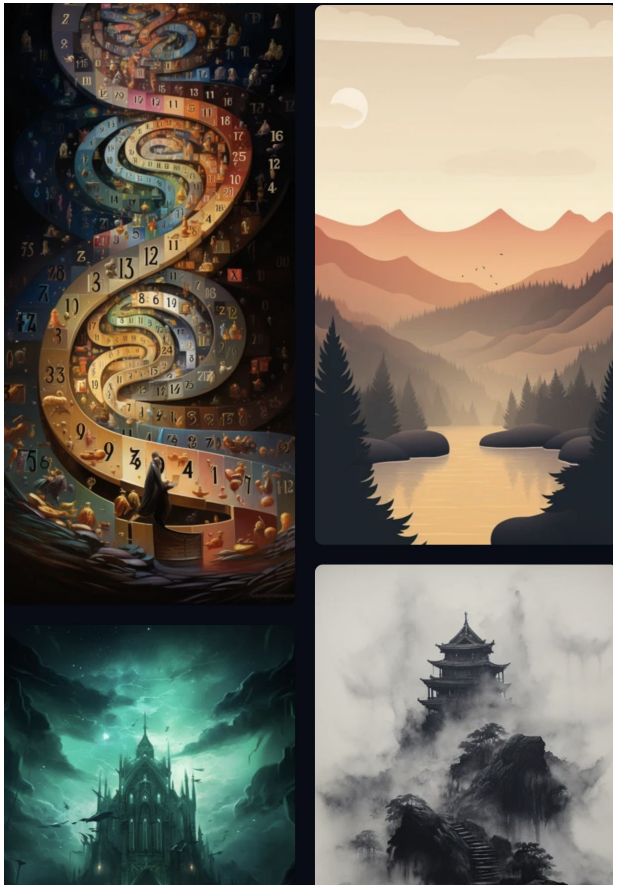
DALLE-3, 2023/08

Sora, 2024/02

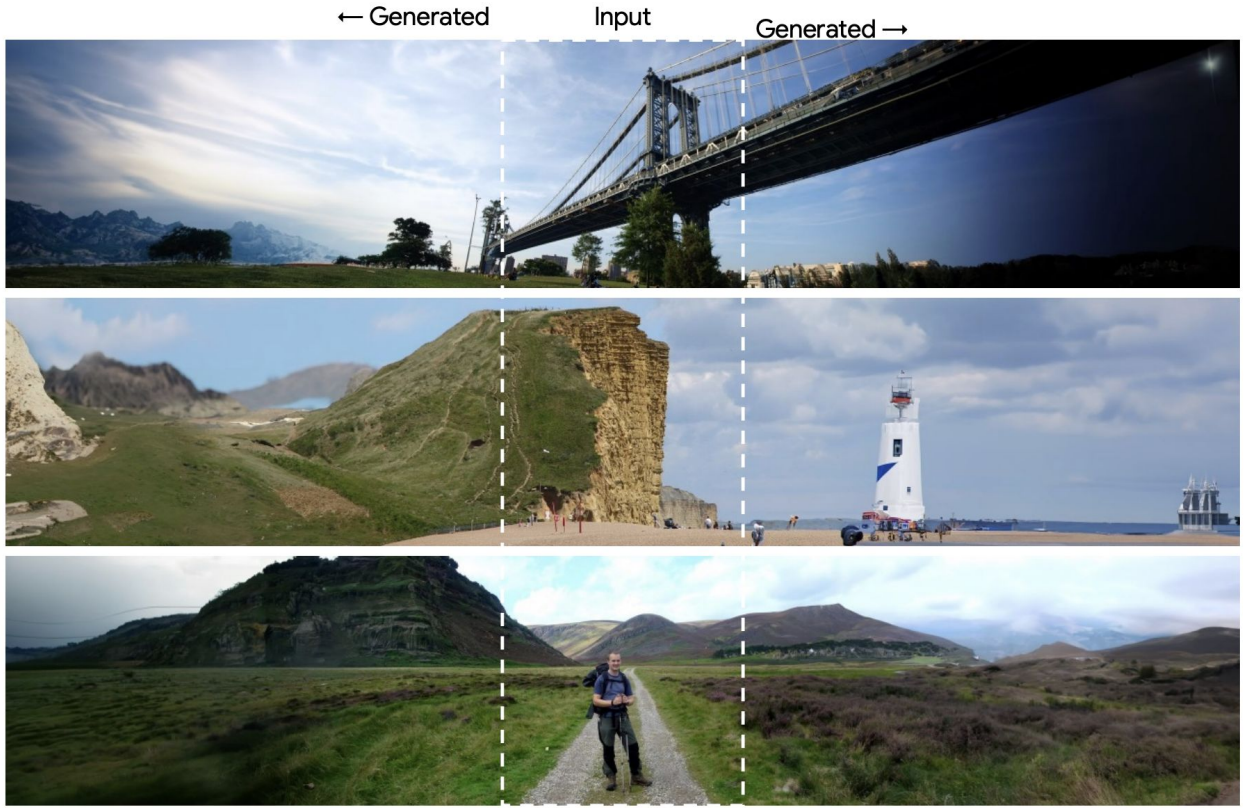
AI Art

Midjourney

https://www.midjourney.com/showcase/recent/



Applications: Outfilling



Palette: Image-to-Image Diffusion Models, 2022

Sora: text-to-image

Scaling transformers for video generation

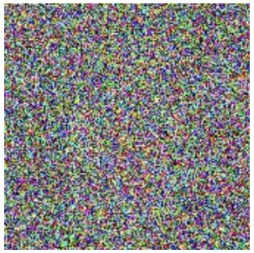
Sora is a diffusion model[21,22,23,24,25]; given input noisy patches (and conditioning information like text prompts), it's trained to predict the original "clean" patches. Importantly, Sora is a diffusion *transformer*.[26] Transformers have demonstrated remarkable scaling properties across a variety of domains, including language modeling,[13,14] computer vision,[15,16,17,18] and image generation.[27,28,29]

Denoising Diffusion Models

Learning to generate by denoising

Image

?



Noise

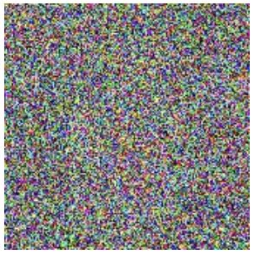
(e.g., sampled from Gaussian)

Denoising Diffusion Models

Learning to generate by denoising

Image

How to get this noisy image as training data?



Noise

(e.g., sampled from Gaussian)

Denoising Diffusion Models

Learning to generate by denoising

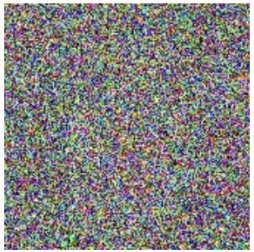
x_0

Image

Just add random Gaussian noise!

x_1

Noisy image



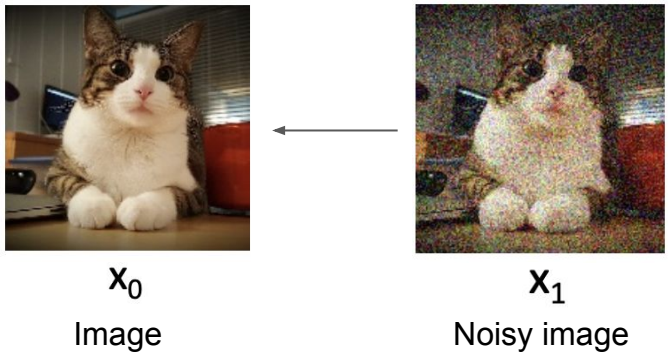
Noise

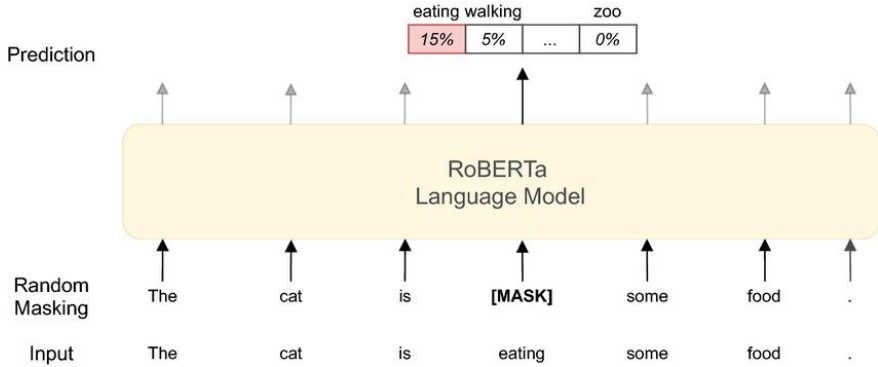
(e.g., sampled from Gaussian)

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

Similar to …

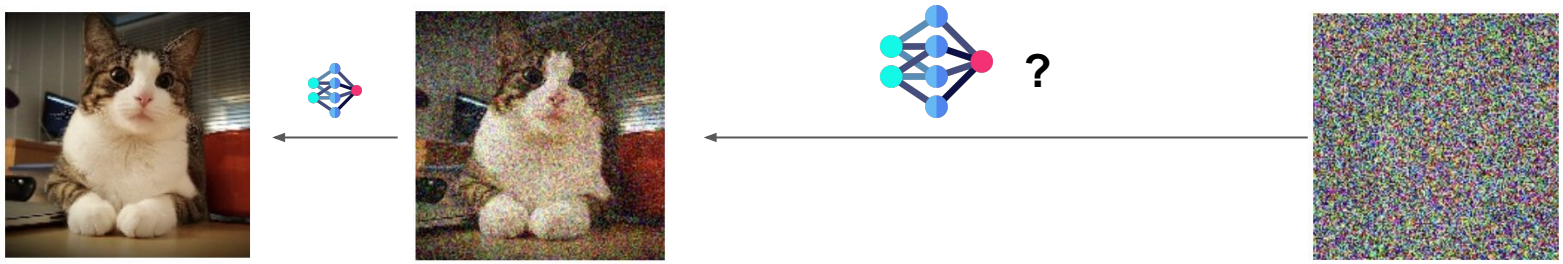
Learning to generate by denoising





Denoising Diffusion Models

Learning to generate by denoising



Denoising Diffusion Models

Learning to generate by denoising

x_0

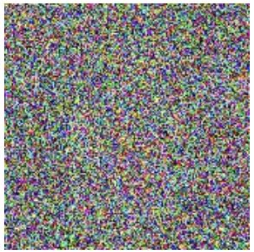
Image

x_1

Noisy image

x_2

Noisy++ image



Noise

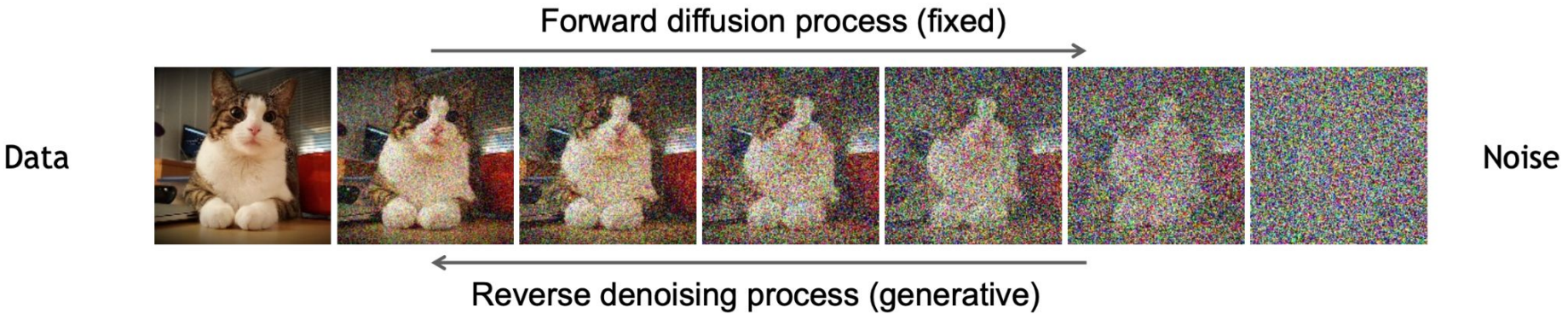
(e.g., sampled from Gaussian)

Denoising Diffusion Models

Learning to generate by denoising

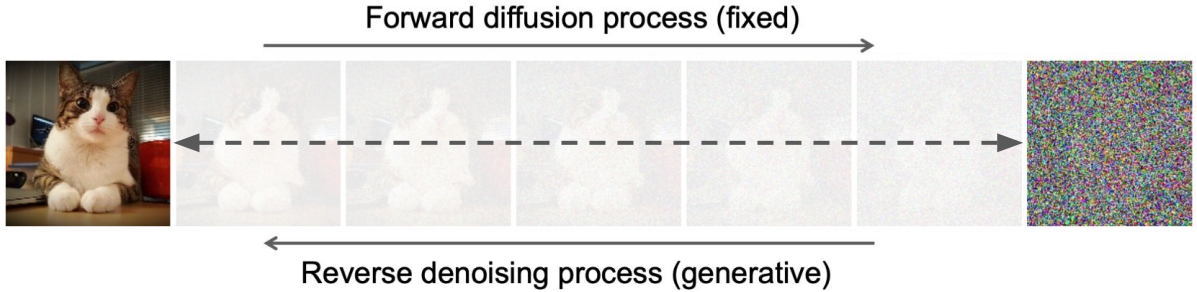
Denoising diffusion models consist of two processes:

- **Forward diffusion** process that gradually adds noise to input
- **Reverse denoising** process that learns to generate data by denoising

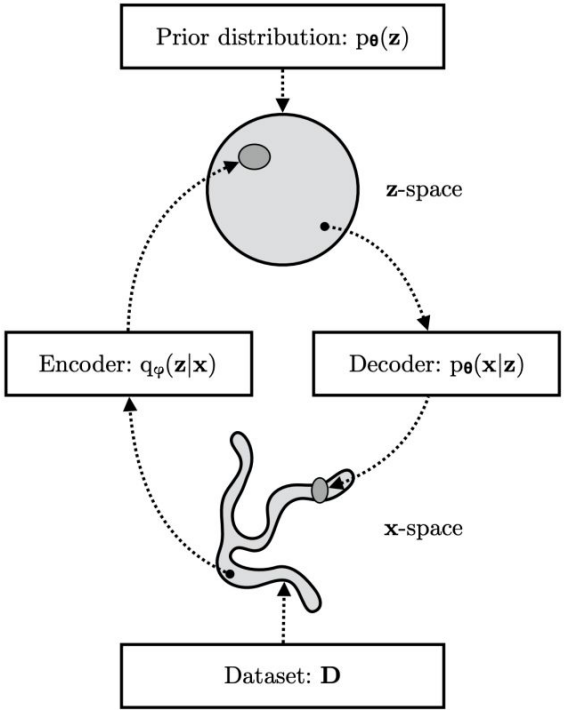


Mapping real data point to noise (or latent sample)

Diffusion model

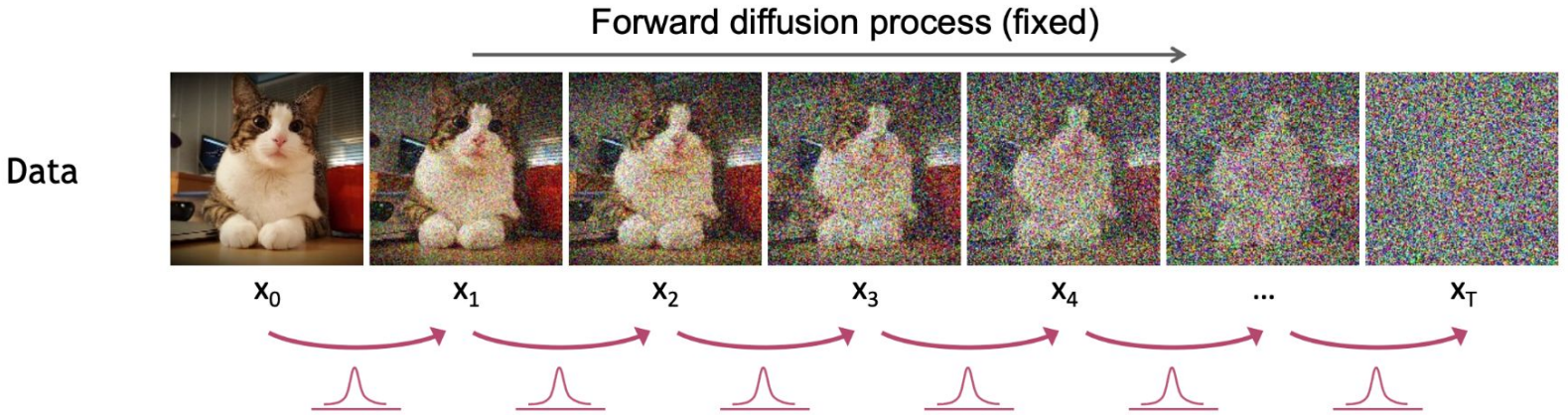


VAE



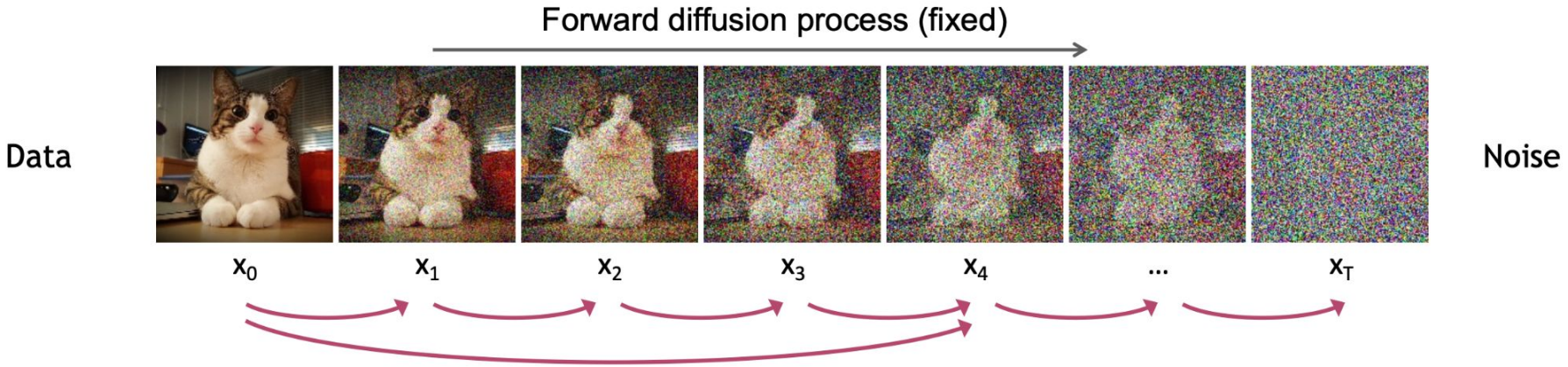
Forward Diffusion Process

The formal definition of the forward process in T steps:



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$

Diffusion Kernel



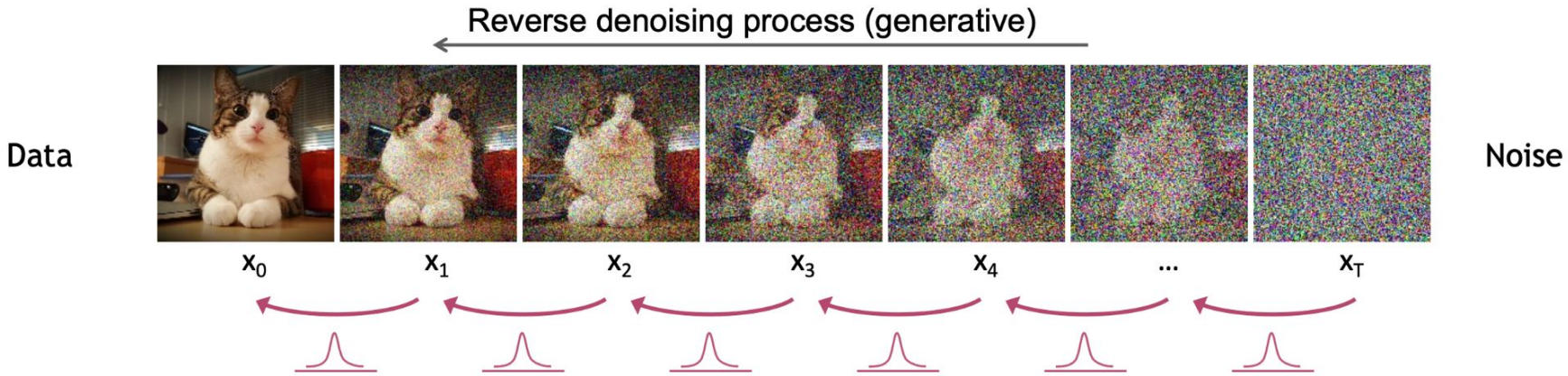
Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ ➡ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

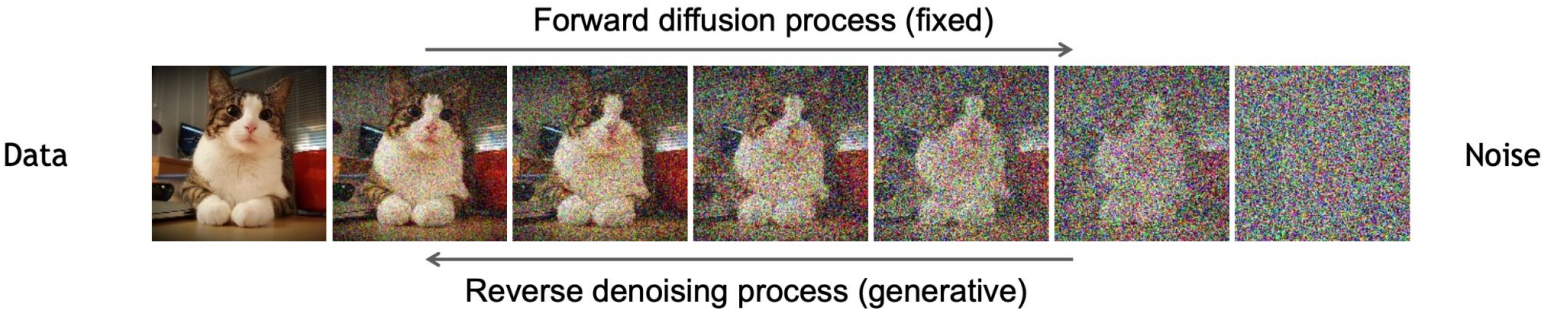
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}, \sigma_t^2\mathbf{I})$$

Trainable network
(U-net, Denoising Autoencoder)

➡

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

Training a denoising diffusion model



Forward pass (clean → noise): $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \implies \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

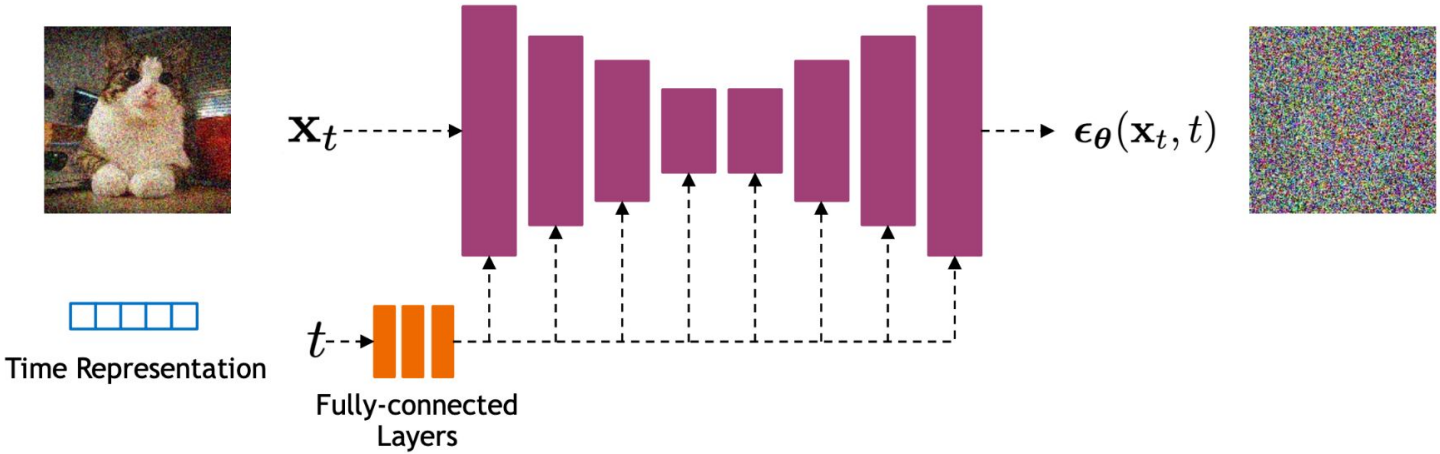
Reverse pass (noise → clean): $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$

Loss function (Ho et al. NeurIPS 2020):

$$\mathbb{E}_{\mathbf{x}_0\sim q(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I}),t\sim\mathcal{U}(1,T)}\left[||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon}_{\mathbf{x}_t}, t)||^2\right]$$

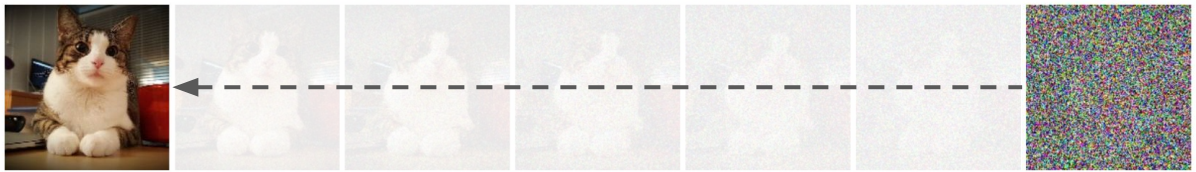
Implementation Considerations

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

How to generate?



(Sampled from Gaussian)

Case study: Text-to-image diffusion model

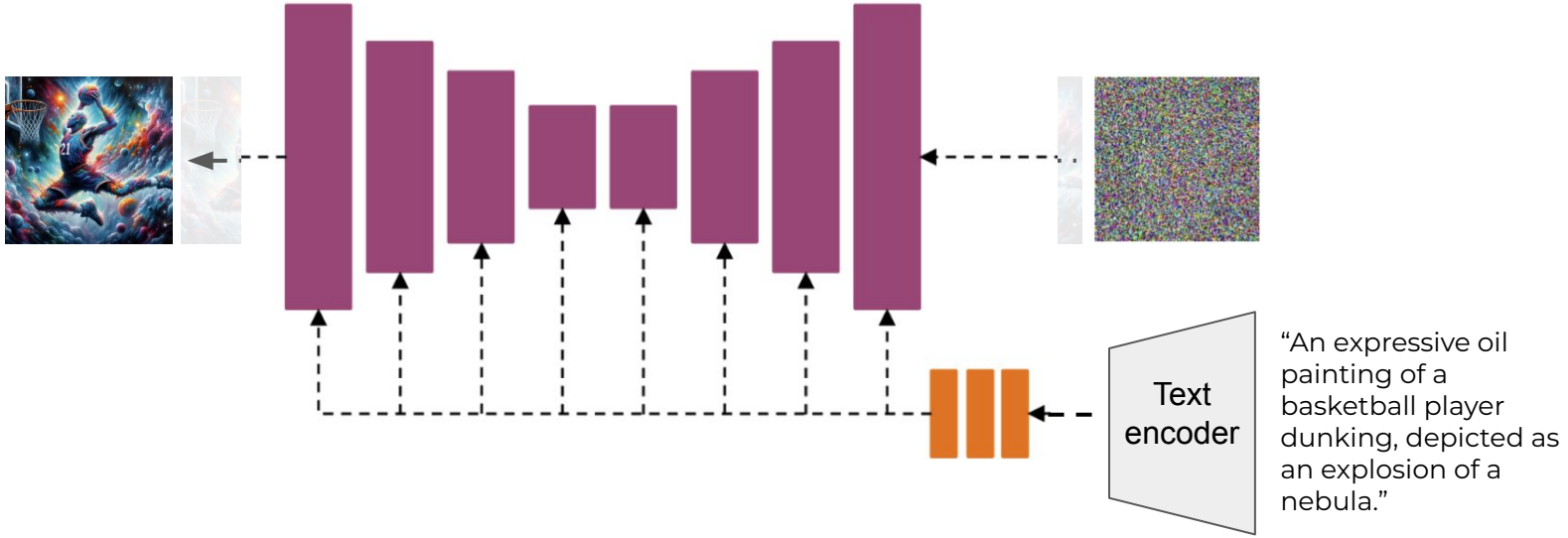
Input

Output

"An expressive oil painting of a basketball player dunking, depicted as an explosion of a nebula."

Text-to-image Diffusion Model

Text-to-image diffusion model



Sora: text-to-video

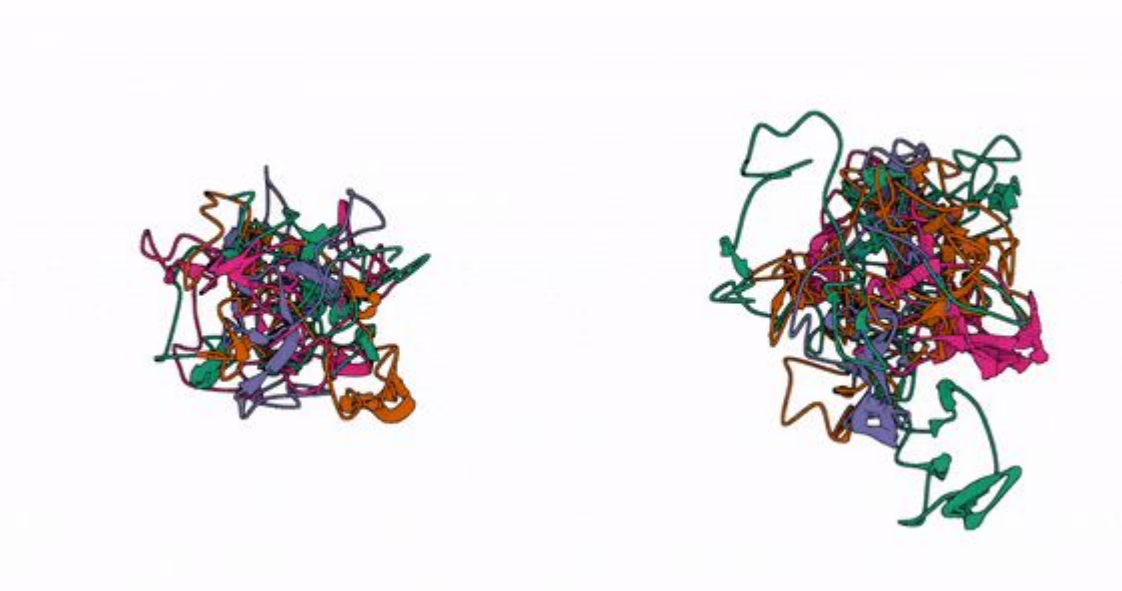
Prompt: A litter of golden retriever puppies playing in the snow. Their heads pop out of the snow, covered in.

Prompt: A close up view of a glass sphere that has a zen garden within it. There is a small dwarf in the sphere who is raking the zen garden and creating patterns in the sand.

See example videos at: https://openai.com/sora

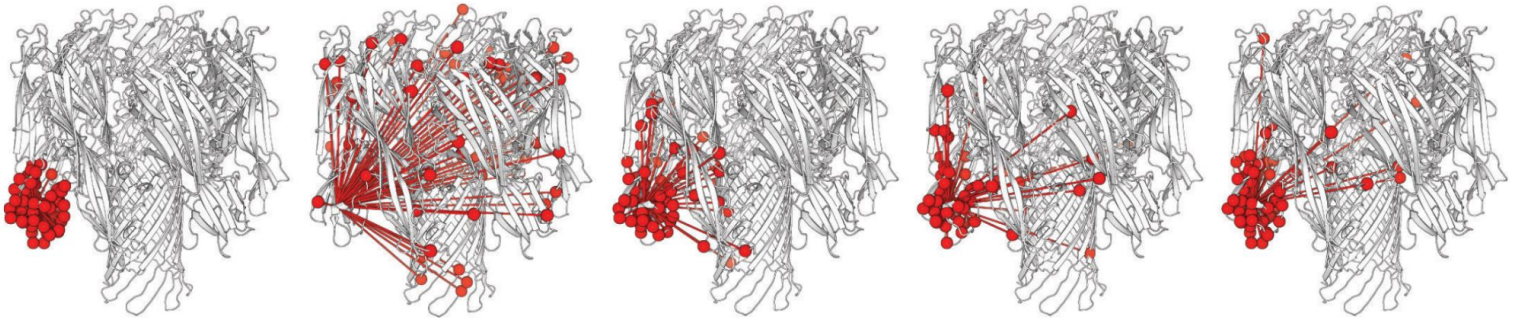
Sora is not just a video generator -- it’s a world simulator!

Diffusion model for proteins



Ingraham, John et al. “Illuminating protein space with a programmable generative model.” *bioRxiv* (2022): n. pag.

Diffusion model for proteins



Edge propensity

$p(\mathcal{E}_{ij} \in \mathcal{G}(\mathbf{x}) | D_{ij}(\mathbf{x})) \propto$ constant $\exp\left(-\frac{D_{ij}(\mathbf{x})}{l}\right)$ $\frac{1}{D_{ij}(\mathbf{x})^3}$

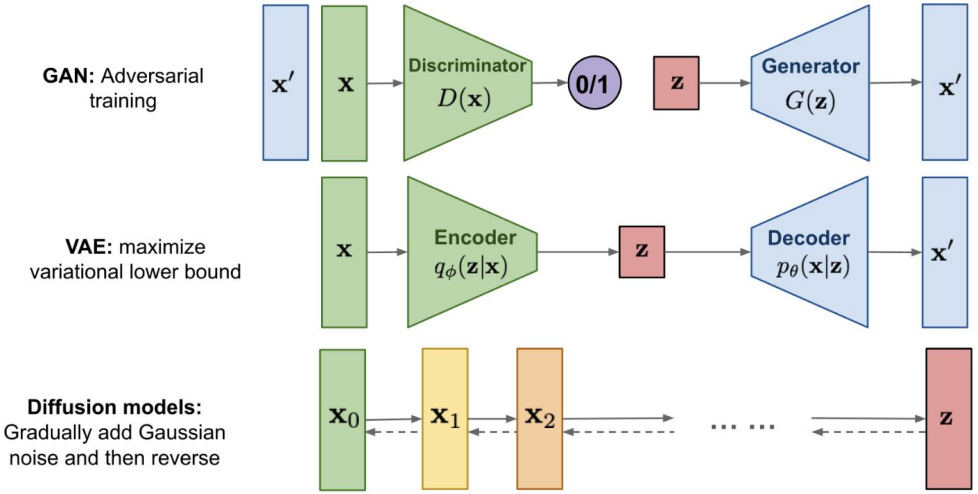
Marginal distance propensity (uniform grid)

$p(D_{ij}(\mathbf{x}) | \mathcal{E}_{ij} \in \mathcal{G}(\mathbf{x})) \propto$ $D_{ij}(\mathbf{x})^2$ $D_{ij}(\mathbf{x})^2 \exp\left(-\frac{D_{ij}(\mathbf{x})}{l}\right)$ $\frac{1}{D_{ij}(\mathbf{x})}$

Ingraham, John et al. “Illuminating protein space with a programmable generative model.” *bioRxiv* (2022): n. pag.

Conclusion

- Autoregressive models
- VAE
- GAN
- Diffusion models



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/