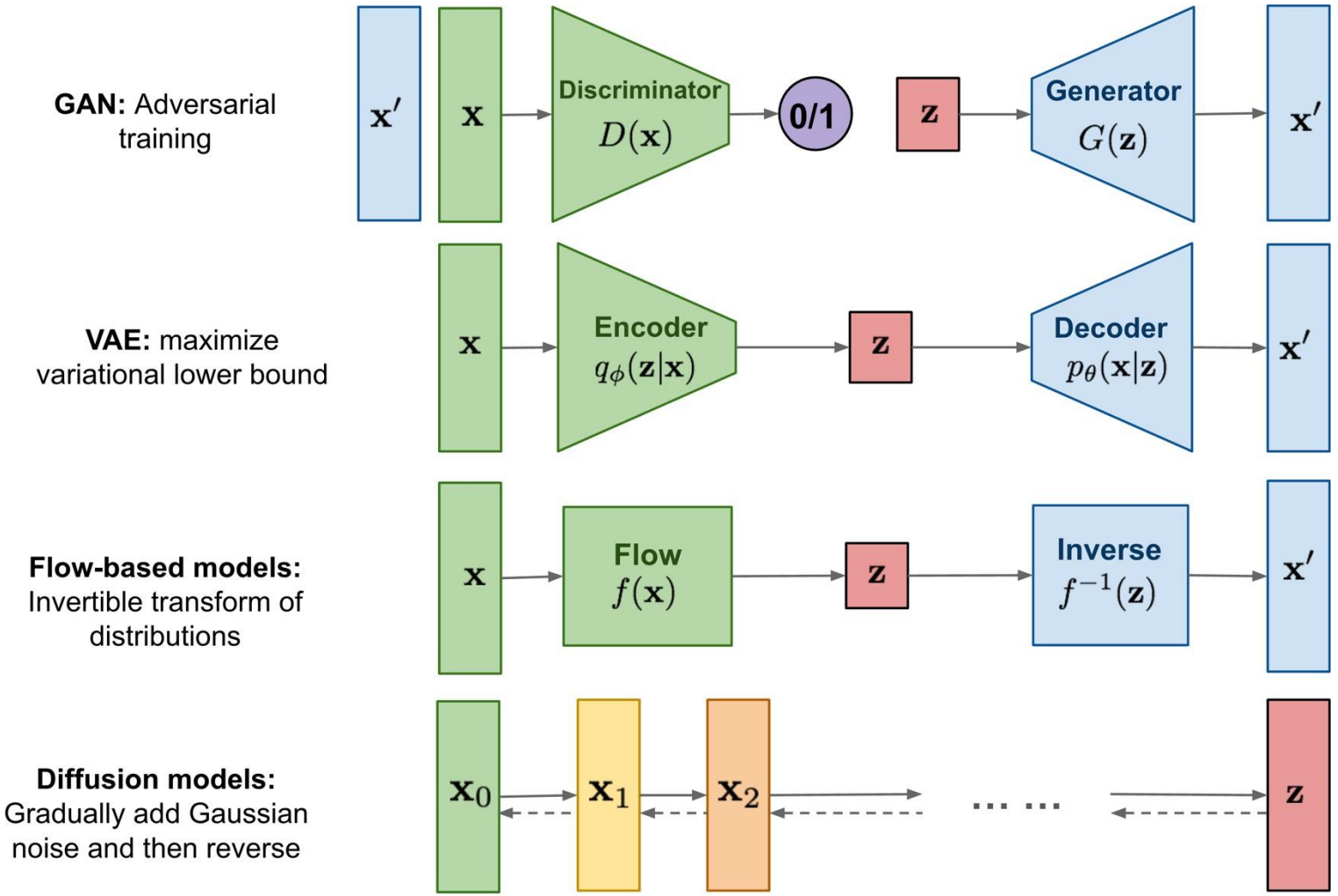
Diffusion Model

CSE 849 Deep Learning
Spring 2025

Zijun Cui

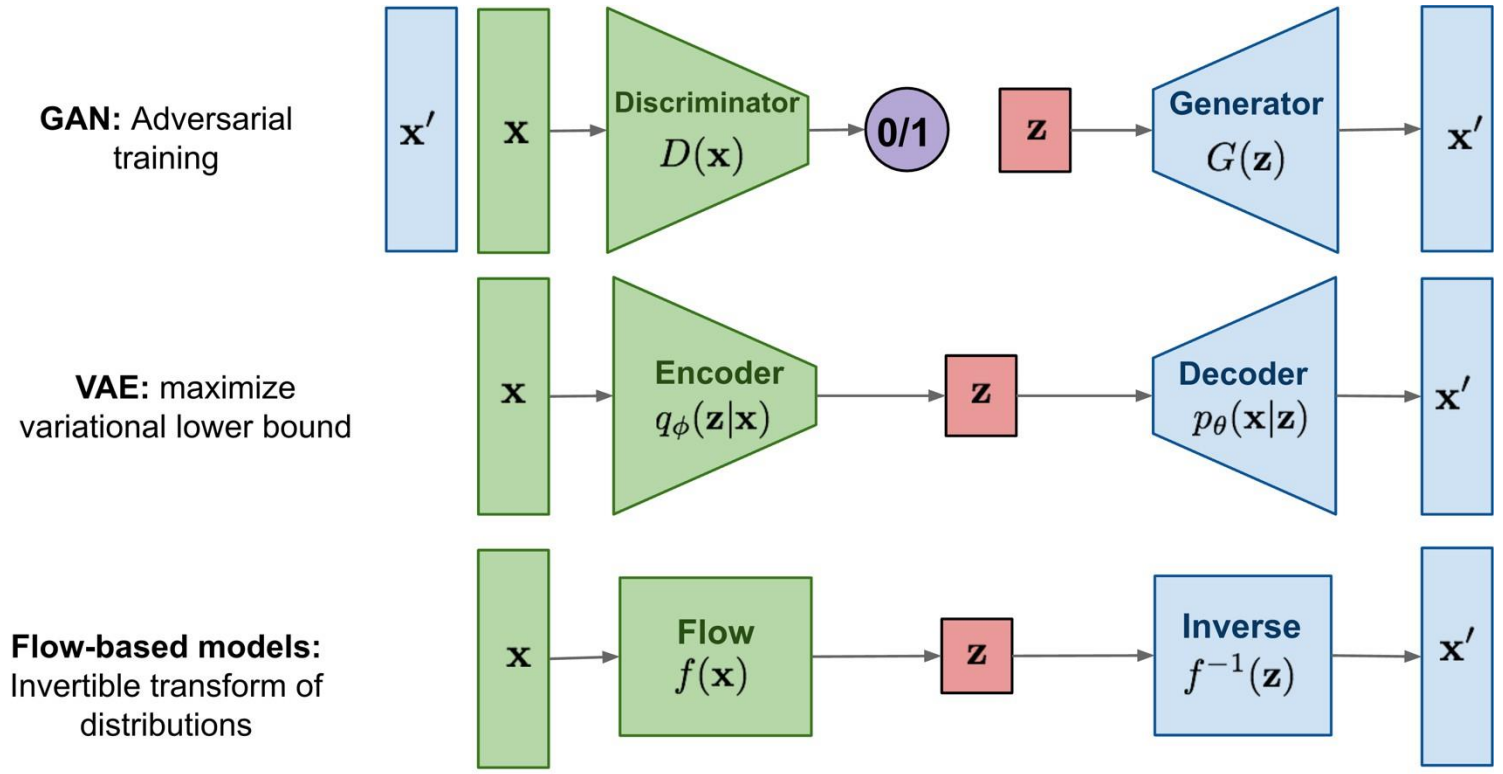
Slides are revised based on Deep Learning CMU by Hao Chen

Generative Models



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

Generative Models



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

2022 / 2023 : The year of generative modeling?

Parti

Pathways Autoregressive Text-to-Image Model

DALL·E 2

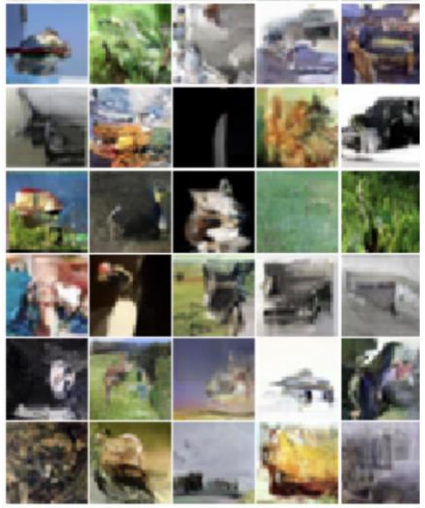
Stable Diffusion

Where we came from

VAEs, 2013

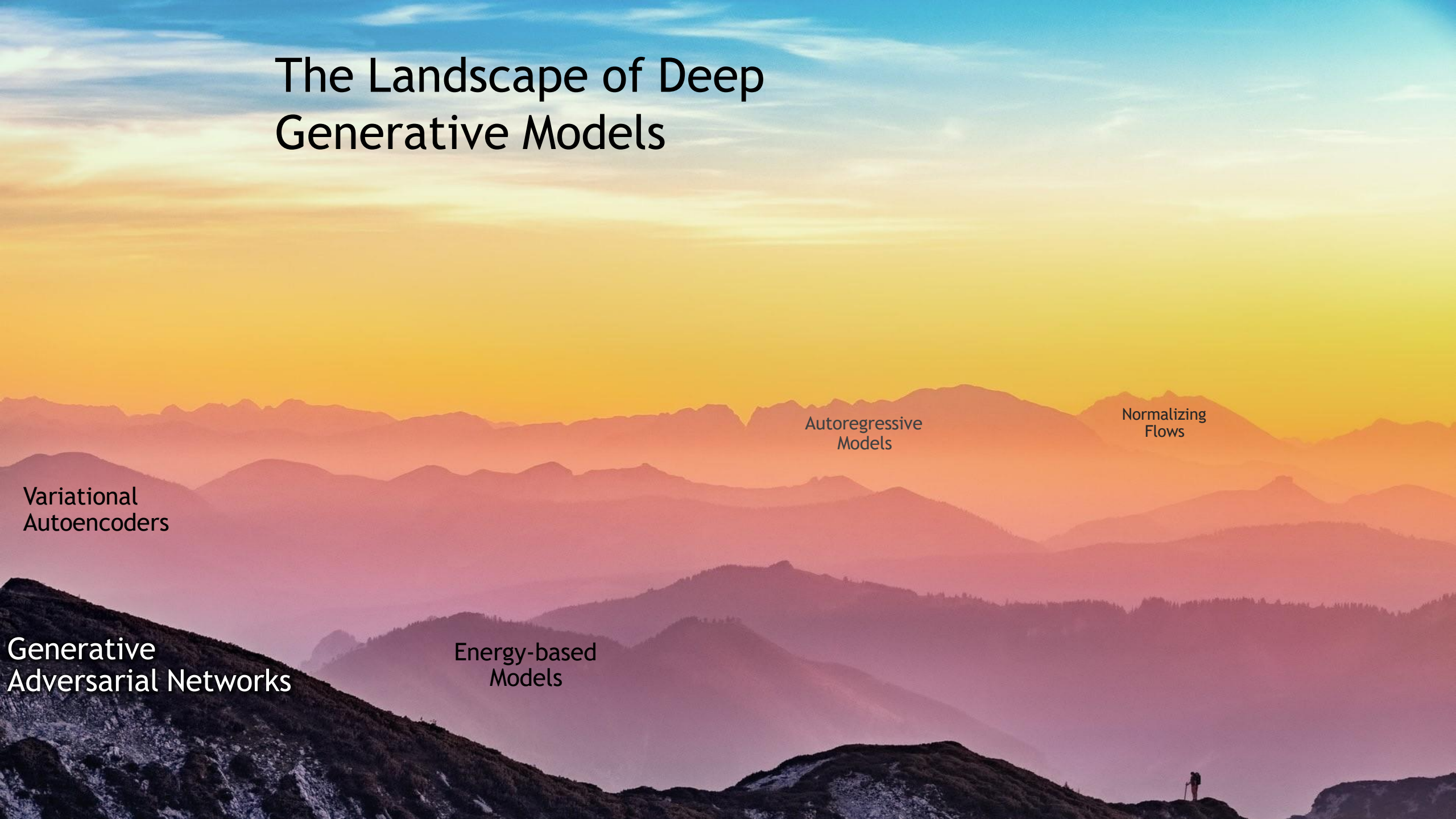
GANs, 2014

PixelCNN, 2016

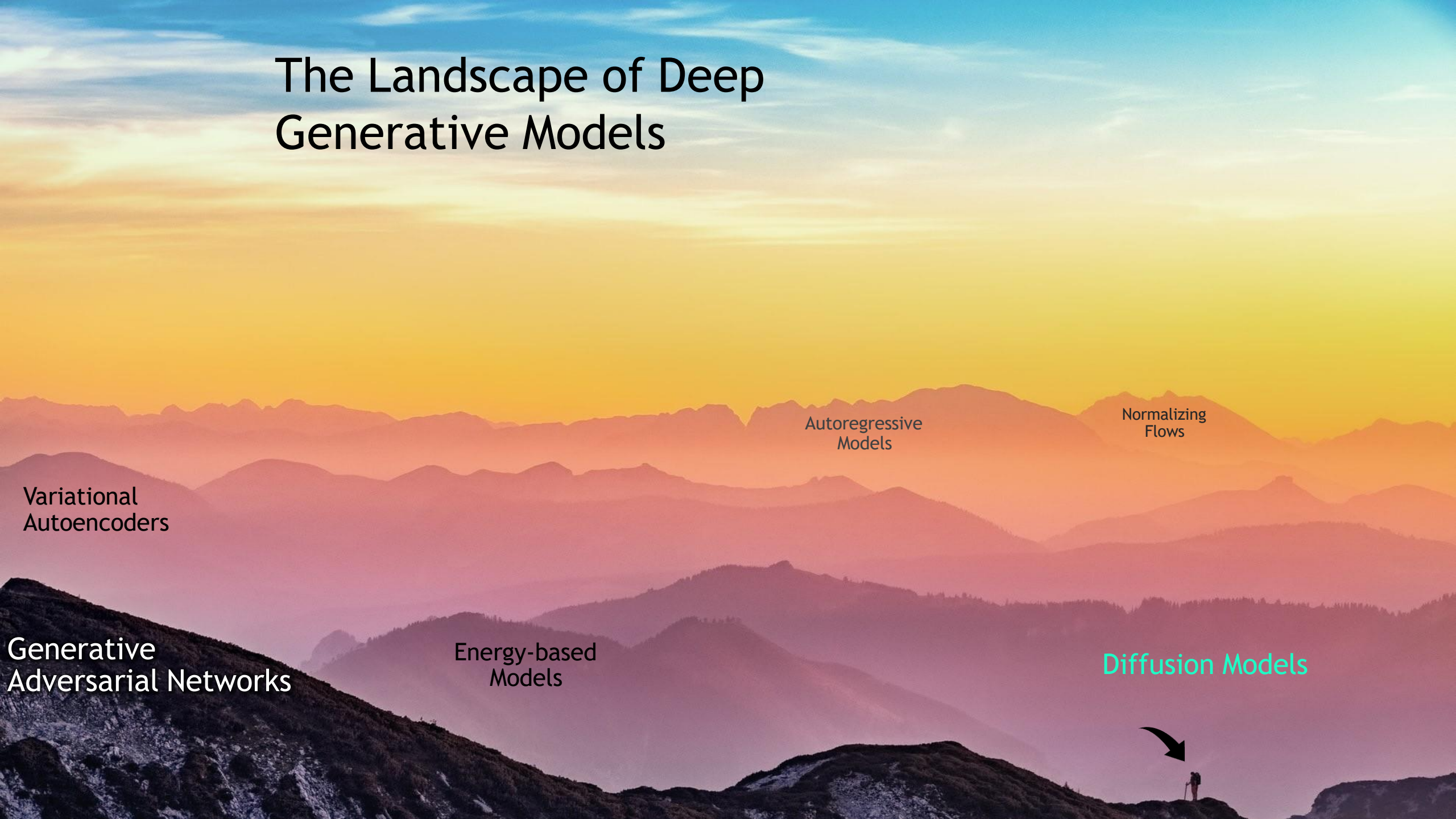


BigGAN, 2019

Imagen, 2022


The Landscape of Deep Generative Models
Autoregressive Models
Normalizing Flows
Variational Autoencoders
Generative Adversarial Networks
Energy-based Models

The Landscape of Deep Generative Models



Diffusion Model

2D Domain

2020.06	OpenAI 2022.04	Google 2022.05	2022.06
DDPM	DALLE2	Imagen	Stable Diffusion

Diffusion Model for Conditional Generation

2020.06 DDPM

2022.04 DALLE2

2022.05 Imagen

2022.06 Stable Diffusion

and more...

2D Domain

- Conditional Generation
  - **Inpainting**
  - Outpainting
  - Image to Image Generation
  - Text to Image Generation

Diffusion Model for Conditional Generation

2D Domain

2020.06	2022.04	2022.05	2022.06	
DDPM	DALLE2	Imagen	Stable Diffusion	and more...

- Conditional Generation
  - Inpainting
  - **Outpainting**
  - Image to Image Generation
  - Text to Image Generation

Diffusion Model for Conditional Generation

2020.06 DDPM

2022.04 DALLE2

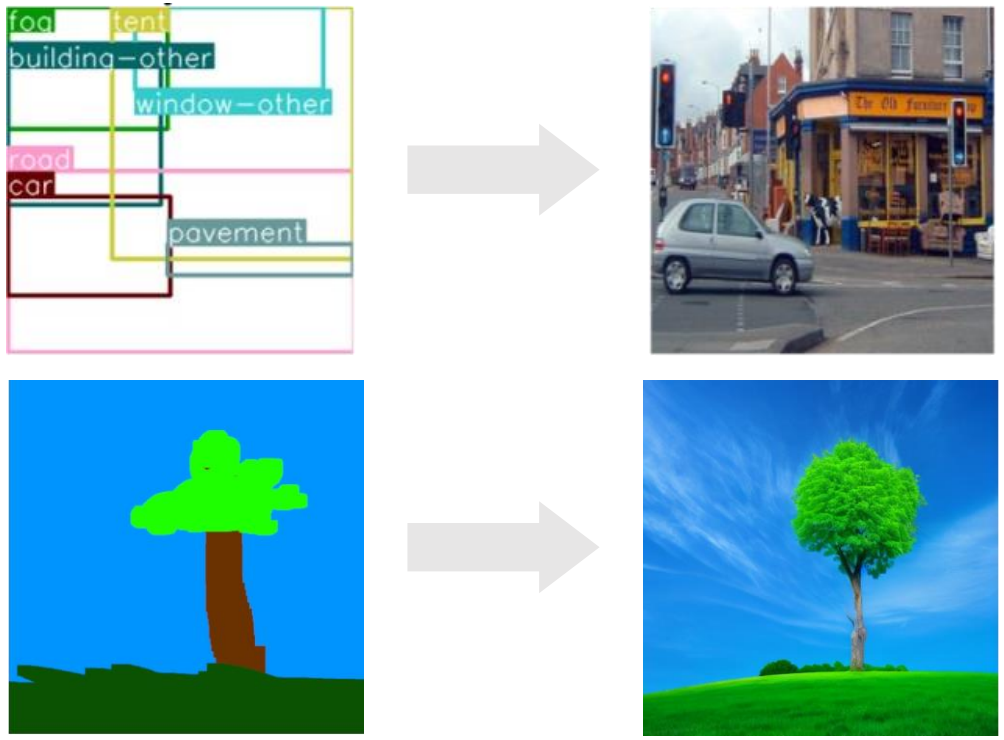
2022.05 Imagen

2022.06 Stable Diffusion

and more...

2D Domain

- Conditional Generation
  - Inpainting
  - Outpainting
  - **Image to Image Generation**
  - Text to Image Generation



Diffusion Model for Conditional Generation

2D Domain

2020.06 DDPM

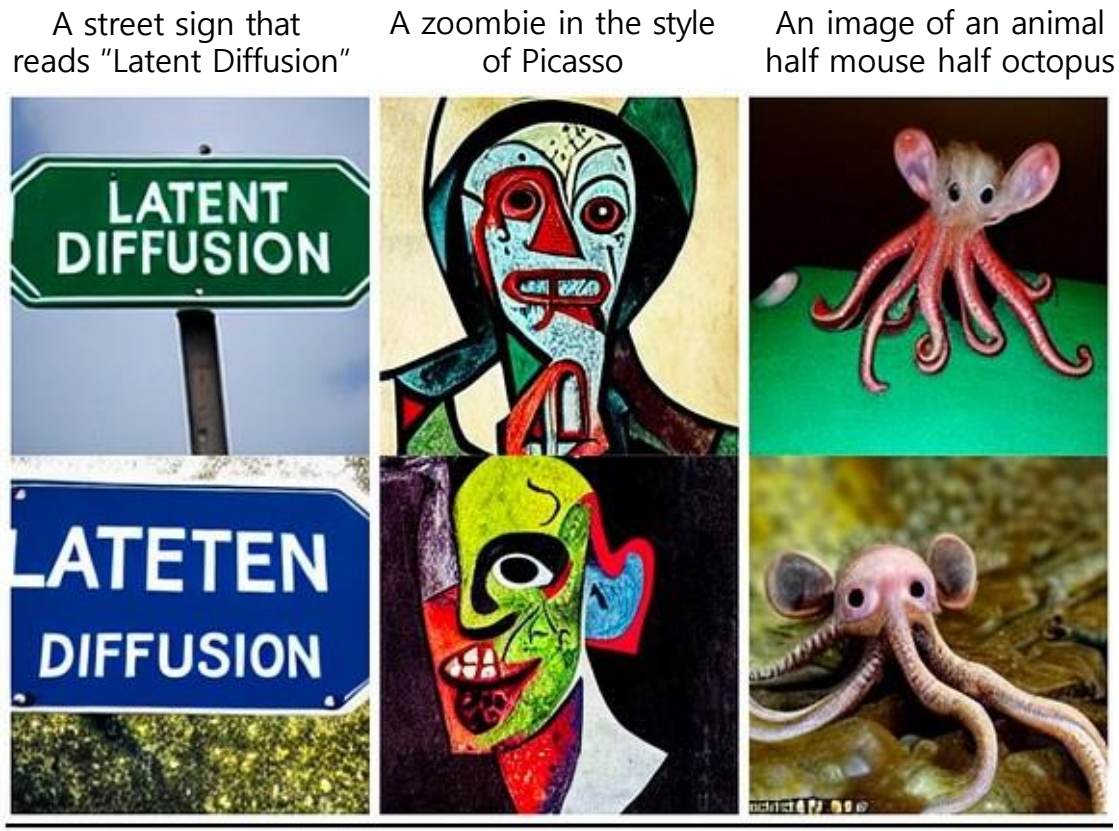
2022.04 DALLE2

2022.05 Imagen

2022.06 Stable Diffusion

and more...

- Conditional Generation
  - Inpainting
  - Outpainting
  - Image to Image Generation
  - **Text to Image Generation**



Diffusion Model

2021~
3D Diffusion

- A 3D diffusion process can be used to generate an object from point clouds, meshes, or latent spaces.

2021 PVD

2021
Text2Mesh

2023
Dreamfusion

2023
Magic3D

2023
ProlificDreamer

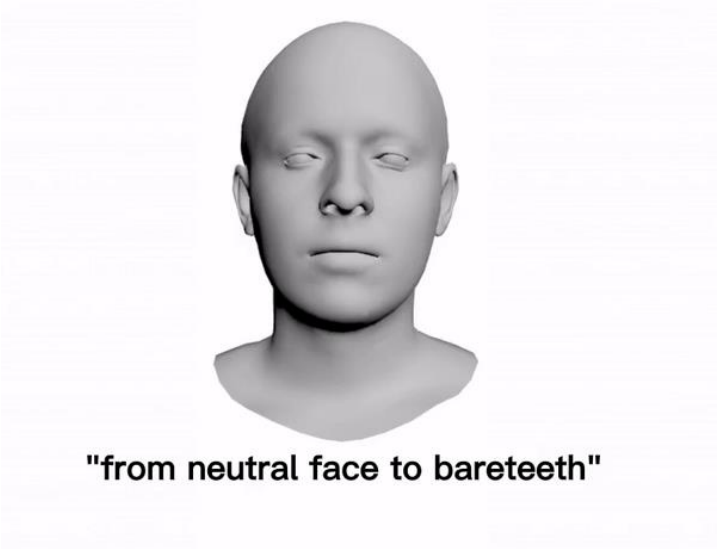
2023
MVdream

Diffusion Model

2021~
3D Diffusion

2023~
4D Diffusion

- Extend the diffusion process domain to 4D, including space and time.



2023
4D Facial Expression

2023
Align Your Gaussian

Diffusion Model

2021~
3D Diffusion

2023~
4D Diffusion

- Extend the diffusion process domain to 4D, including space and time.

2023
4DGen

stable-diffusion Public

Watch 572 | Fork 10.4k | Star 70.2k

main | 1 Branch | 0 Tags | Go to file t | Add file | Code

rromb Update sampler.py 21f890f · 3 years ago 33 Commits

assets	Release under CreativeML Open RAIL M License	3 years ago
configs	stable diffusion	3 years ago
data	stable diffusion	3 years ago
ldm	Update sampler.py	3 years ago
models	add configs for training unconditional/class-conditional ldms	4 years ago
scripts	add dpm-solver support (much faster than plms)	3 years ago
LICENSE	Release under CreativeML Open RAIL M License	3 years ago
README.md	Release under CreativeML Open RAIL M License	3 years ago
Stable_Diffusion_v1_Model_Card.md	Release under CreativeML Open RAIL M License	3 years ago
environment.yaml	Release under CreativeML Open RAIL M License	3 years ago
main.py	add configs for training unconditional/class-conditional ldms	4 years ago
notebook_helpers.py	add code	4 years ago
setup.py	add code	4 years ago

README License

Stable Diffusion

Stable Diffusion was made possible thanks to a collaboration with Stability AI and Runway and builds upon our previous work:

High-Resolution Image Synthesis with Latent Diffusion Models
Robin Rombach*, Andreas Blattmann*, Dominik Lorenz, Patrick Esser, Björn Ommer
CVPR '22 Oral | *GitHub* | *arXiv* | *Project page*

About

A latent text-to-image diffusion model

ommer-lab.com/research/latent-diffusio...

Readme
View license
Activity
Custom properties
70.2k stars
572 watching
10.4k forks
Report repository

Releases

No releases published

Packages

No packages published

Contributors 8

Languages

Jupyter Notebook 88.8% Python 11.1%
Shell 0.1%

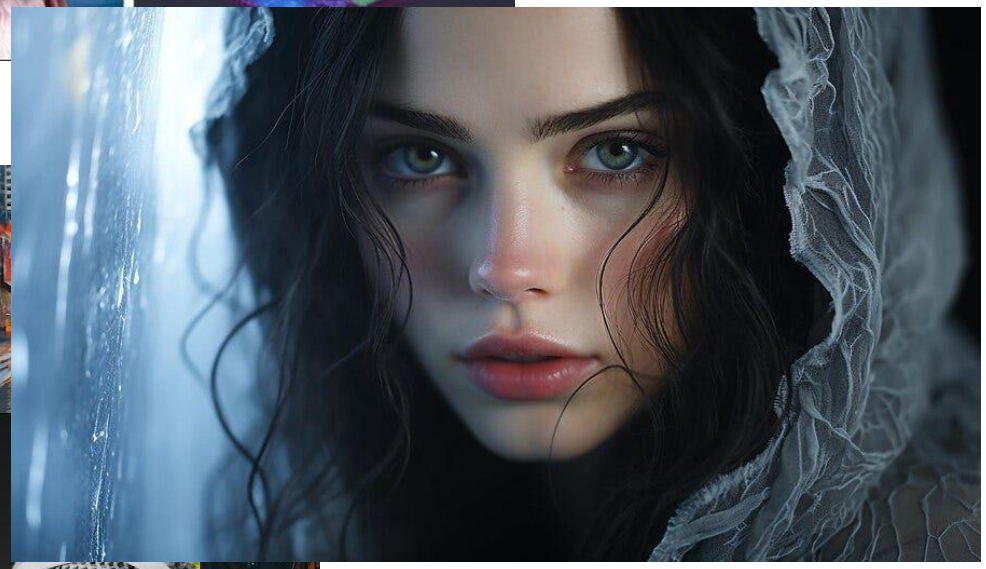
https://github.com/CompVis/stable-diffusion

Commercial software

SDXL
Stability AI 2023.7

DALLE 3
OpenAI 2023.10

And more!!!



Midjourney v6
2023.12

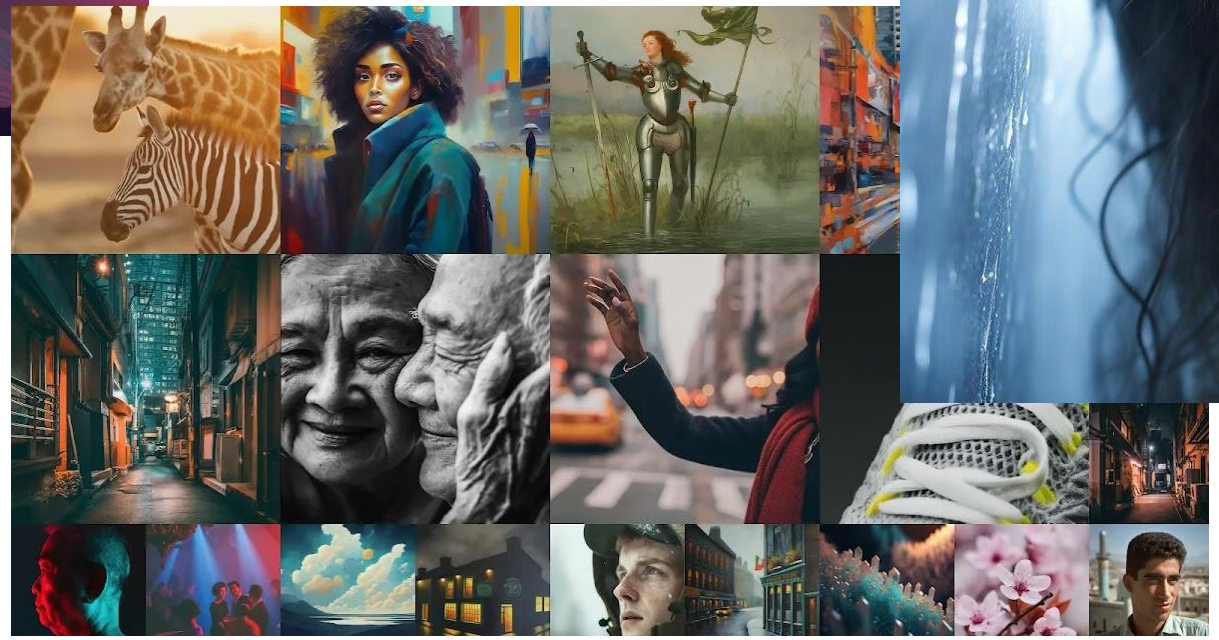


Imagen 2
Google 2023.11

Landscape Highlights of Diffusion Models

basic principles

- Diffusion probabilistic models (Sohl-Dickstein et al., 2015)
- Noise-conditioned score network (**NCSN**; Yang & Ermon, 2019)
- Denoising diffusion probabilistic models (**DDPM**; Ho et al. 2020)

conditional & high-res image generation

- Classifier-guided conditional generation (Dhariwal and Nichole, 2021)
- Classifier-free Diffusion Guidance (Ho and Salimans, 2022)
- Latent-space Diffusion (**StableDiffusion**; Rombach and Blattmann et al., 2022)
- Edify Image from Nvidia 2024 (https://arxiv.org/abs/2411.07126?utm_source=chatgpt.com)

new applications

- Planning with Diffusion for Flexible Behavior Synthesis (**Diffuser**; Janner et al., 2022)
- DreamFusion: Text-to-3D using 2D Diffusion (Poole and Jain et al., 2022)
- Make-A-Video: Text-to-Video Generation without Text-Video Data (Singer et al., 2022)
- PhysDiff: Physics-Guided Human Motion Diffusion Model (Yuan et al. 2023)
- Magicanimate: Temporally Consistent Human Image Animation using Diffusion Model (Xu et al., 2024)

Content

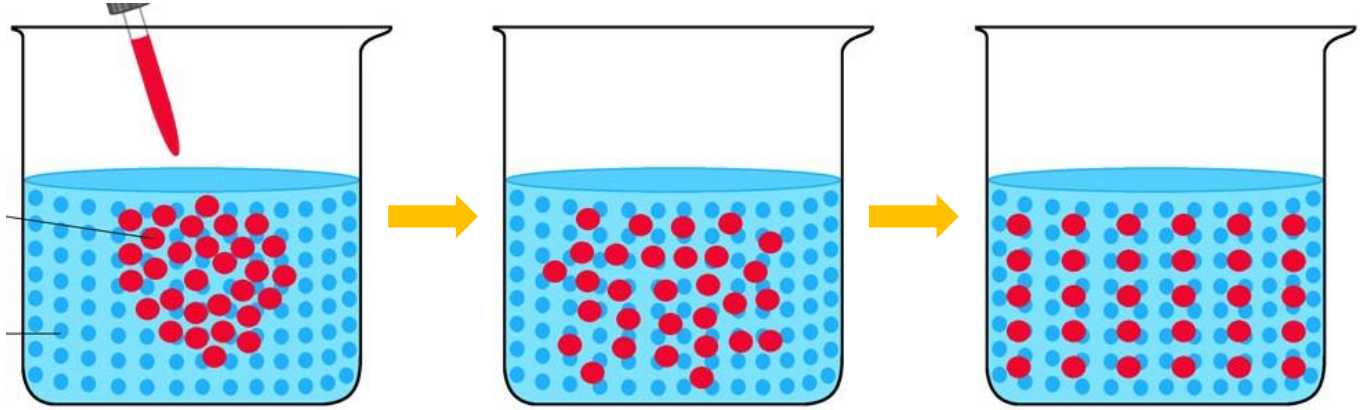
- Diffusion Model Basics
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Variants of Diffusion Models
  - Denoising Diffusion Implicit Model (DDIM)
  - Latent Diffusion Models
  - Conditional Diffusion Models
- Applications of Diffusion Models

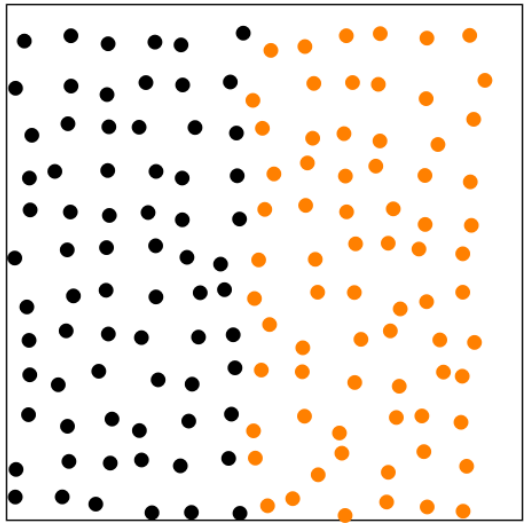
Content

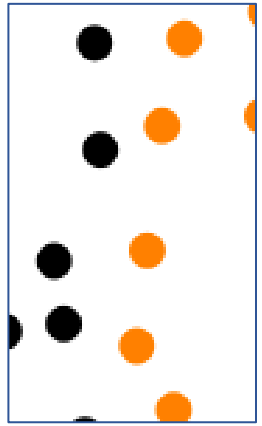
- Diffusion Model Basics
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Variants of Diffusion Models
  - Denoising Diffusion Implicit Model (DDIM)
  - Latent Diffusion Models
  - Conditional Diffusion Models
- Applications of Diffusion Models

Diffusion Process

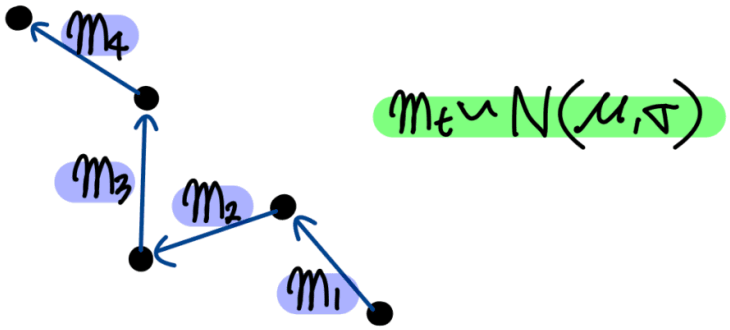
- Diffusion models are inspired by non-equilibrium thermodynamics.
- For a small fraction of the time, it is difficult to determine whether particles are moving in the direction of mixing or in the opposite direction.



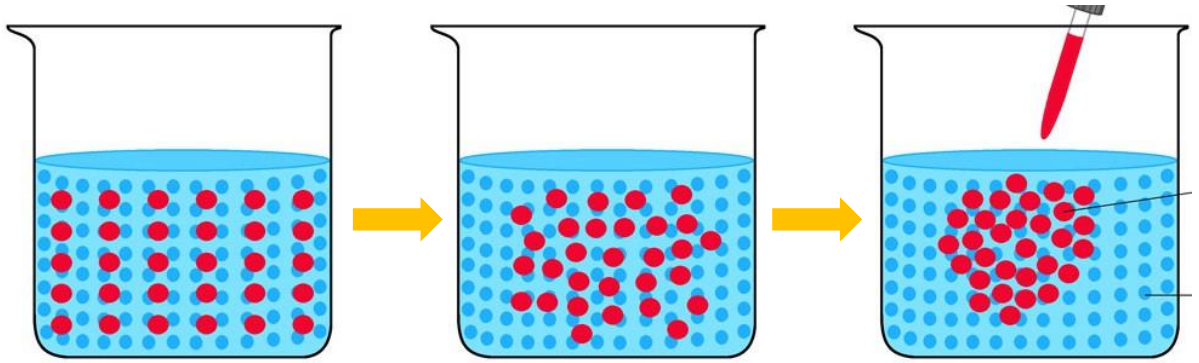




Diffusion Process

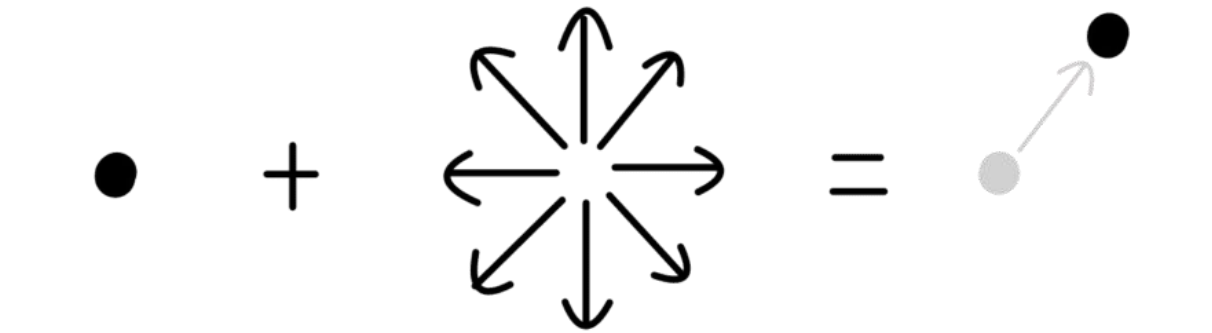


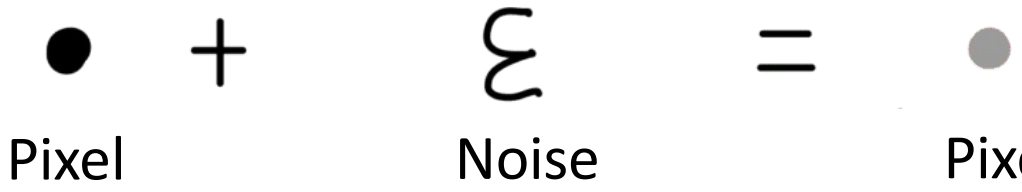
- If we look at the movement of a single molecule on a very short time scale, it follows a Gaussian distribution.
- Since the direction of mixing and the opposite direction are the same in a very short time, the opposite direction also follows a Gaussian distribution.



Diffusion Process

- Just as we viewed the molecule's motion as a Gaussian-distributed noise, we add a Gaussian-distributed noise to the pixel.

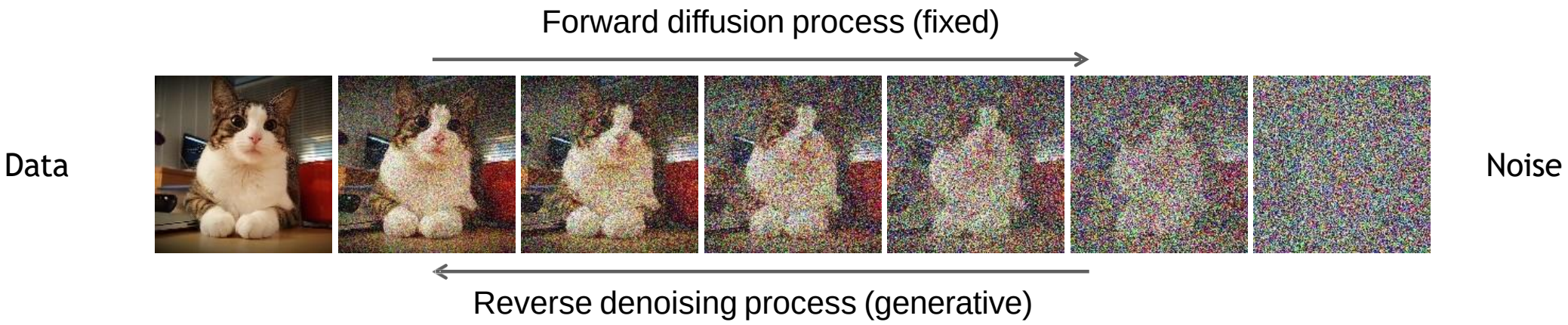




Denoising Diffusion Models

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



The Denoising Diffusion Process

image from dataset

The “forward diffusion” process: add Gaussian noise each step

x_0 → x_1 → …

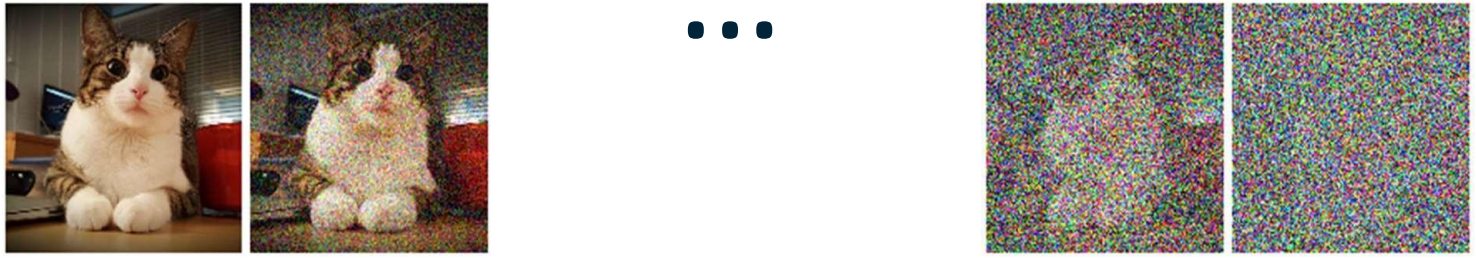
The Denoising Diffusion Process

image from dataset

The “forward diffusion” process: add Gaussian noise each step

noise $\mathcal{N}(0, I)$

x_0 → x_1 → ... → x_{T-1} → x_T

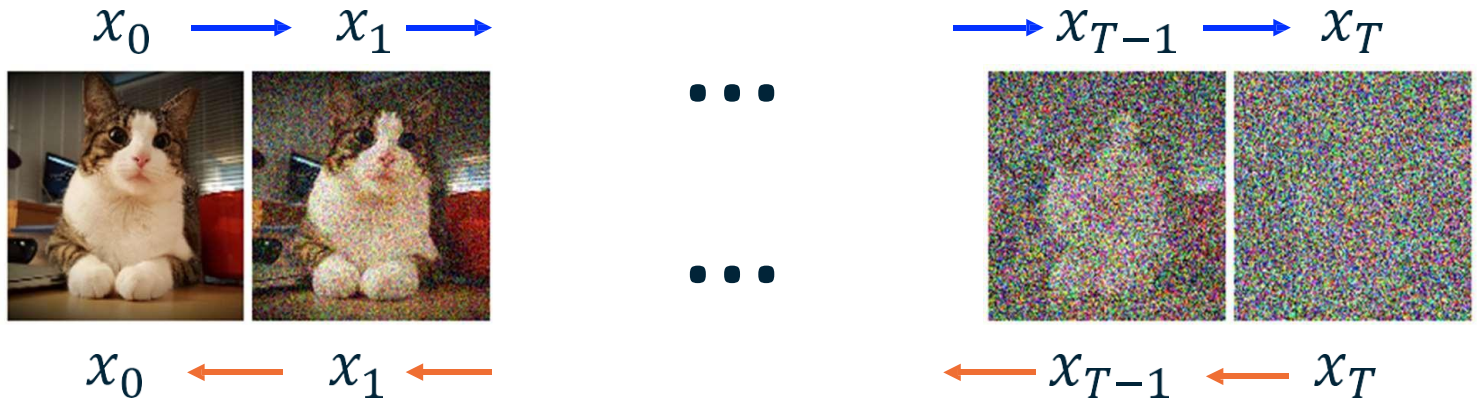


The Denoising Diffusion Process

image from dataset

The "forward diffusion" process: add Gaussian noise each step

noise $\mathcal{N}(0, I)$



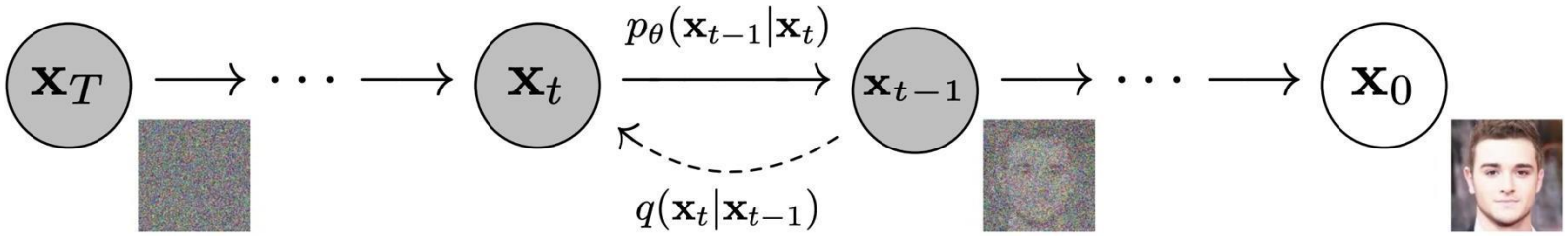
The "denoising diffusion" process: generate an image from noise by *denoising* the gaussian noises

Learning to generate by denoising

Ties/inspiration form Annealed Imporantce Sampling in physics

Denoising Diffusion Models

- what we often see about diffusion models



$$q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)=\mathcal{N}\left(\mathbf{x}_t ; \sqrt{\bar{\alpha}_t} \mathbf{x}_0,\left(1-\bar{\alpha}_t\right) \mathbf{I}\right)\Big)$$

$$\mathbf{x}_t=\sqrt{\bar{\alpha}_t} \mathbf{x}_0+\sqrt{\left(1-\bar{\alpha}_t\right)} \epsilon$$

Forward diffusion process

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

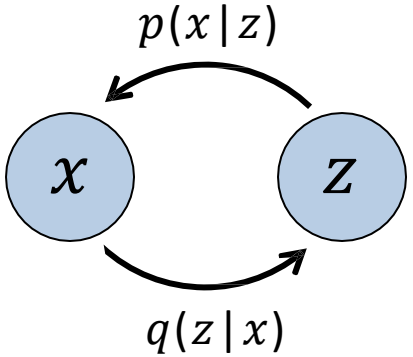
$$p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)=\mathcal{N}\left(\mathbf{x}_{t-1} ; \mu_\theta\left(\mathbf{x}_t, t\right), \sigma_t^2 \mathbf{I}\right)$$

Reverse denoising process

- Firstly, we will see that denoising diffusion is a stack of VAEs

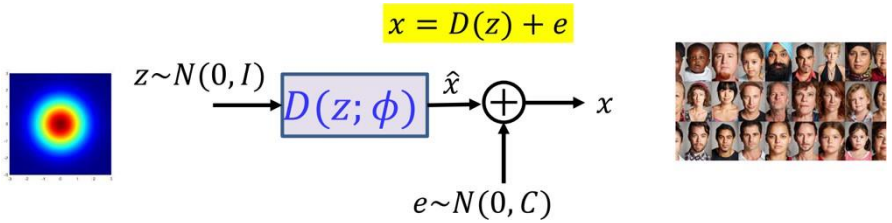
Recap: Variational Autoencoders

- **Encoder**: an inference model that approximates the posterior $q(z \mid x)$
- **Decoder**: a generative model that transforms a Gaussian variable z to real data
- **Training**: maximize the ELBO



Recap: Variational Autoencoders

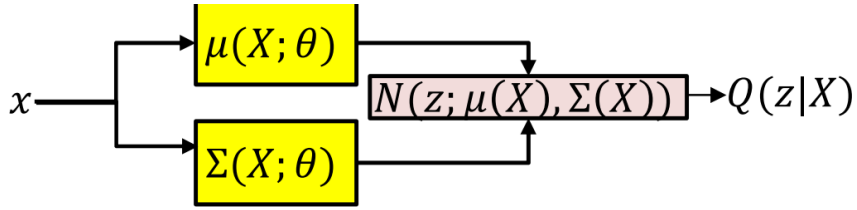
Decoder: transforms a Gaussian variable to real data



$$p(x|z) = \mathcal{N}(\mu(z), \sigma^2 I)$$

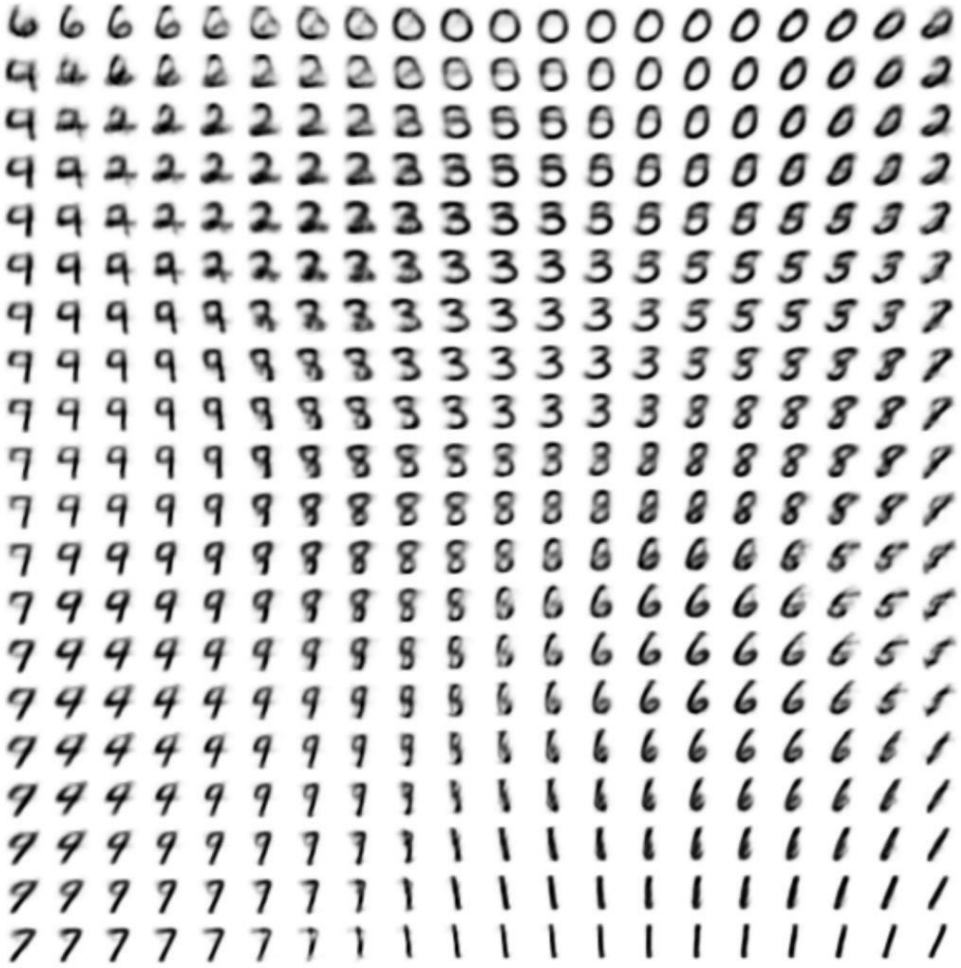
where $\mu(z) = D(z; \phi)$ is the decoder output (mean), and σ^2 is a learned or fixed variance.

Encoder: an inference model approximates the posterior, i.e. Gaussian



VAEs are good, but…

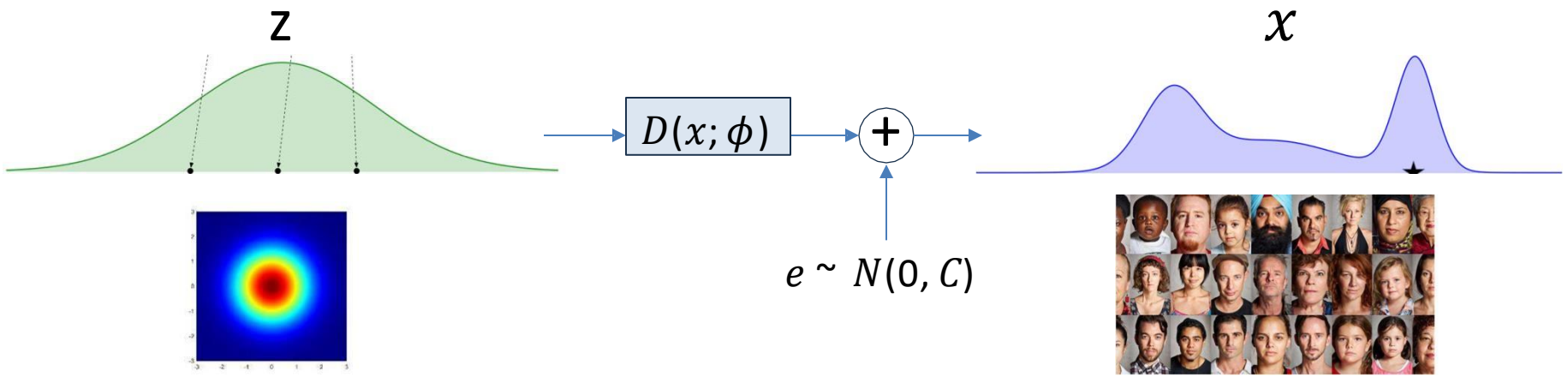
- Blurry results



Kingma et al. Auto-Encoding Variational Bayes

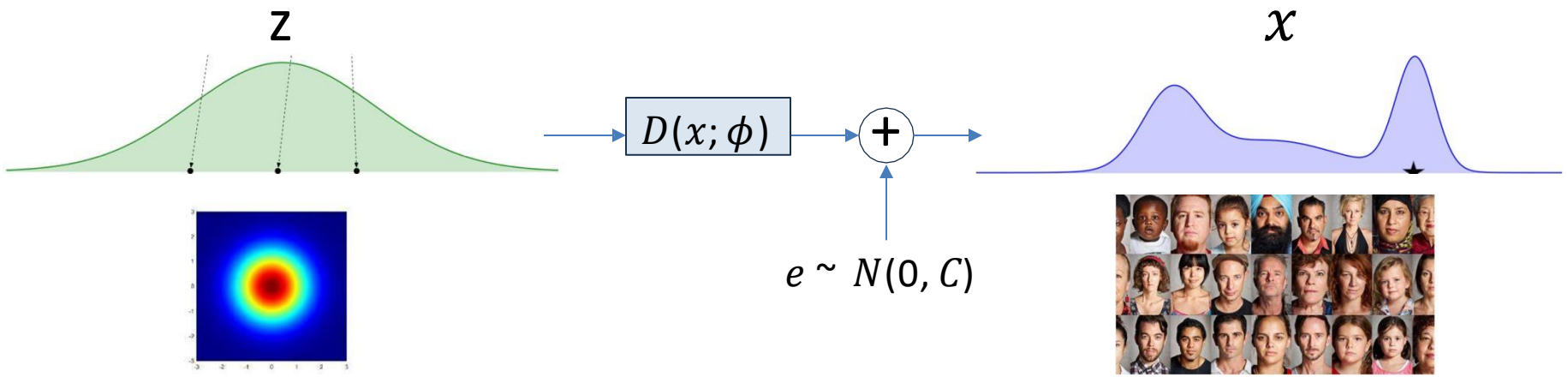
Limitations of VAEs

- Decoder must transform a standard Gaussian all the way to the target distribution in **one-step**
  - Often too large a gap
  - Blurry results are generated



Limitations of VAEs

- Decoder must transform a standard Gaussian all the way to the target distribution in **one-step**
  - Often too large a gap
  - Blurry results are generated

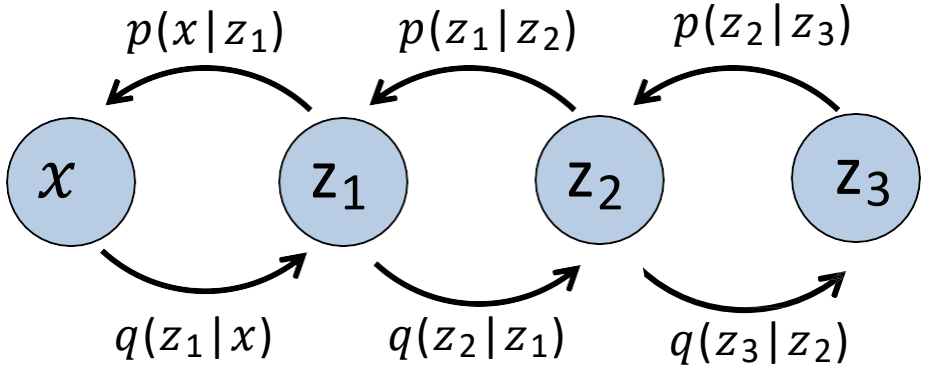


- Solution: have some intermediate latent variables to reduce the gap of each step

Hierarchical VAEs

- Hierarchical VAEs - Stacking VAEs on top of each other
  - Multiple (T) intermediate latent
  - Joint distribution $$p(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}_1) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1} \mid \boldsymbol{z}_t)$$
  - Posterior $$q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T} \mid \boldsymbol{x}) = q_{\phi}(\boldsymbol{z}_1 \mid \boldsymbol{x}) \prod_{t=2}^{T} q_{\phi}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1})$$

- Better likelihood achieved!



Sønderby et al. Ladder Variational Networks. 2016

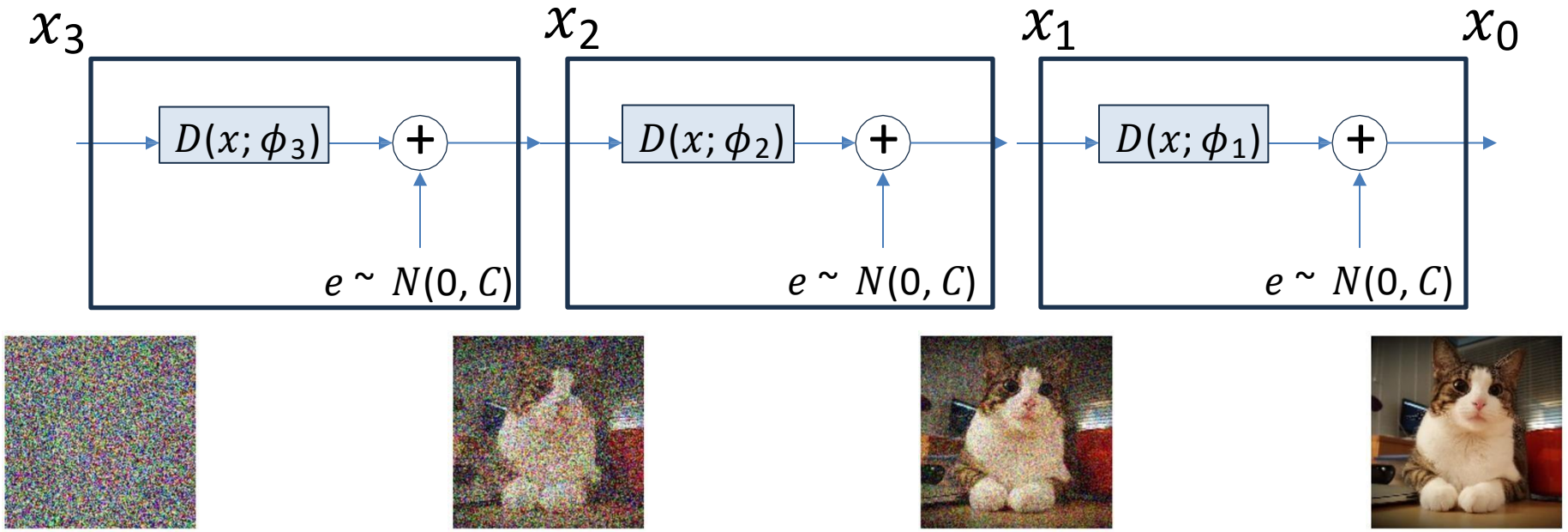
Stacking VAEs

- Each step, the decoder removes part of the noise

$$x_i = D(x_{i+1}; \phi_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, C)$$

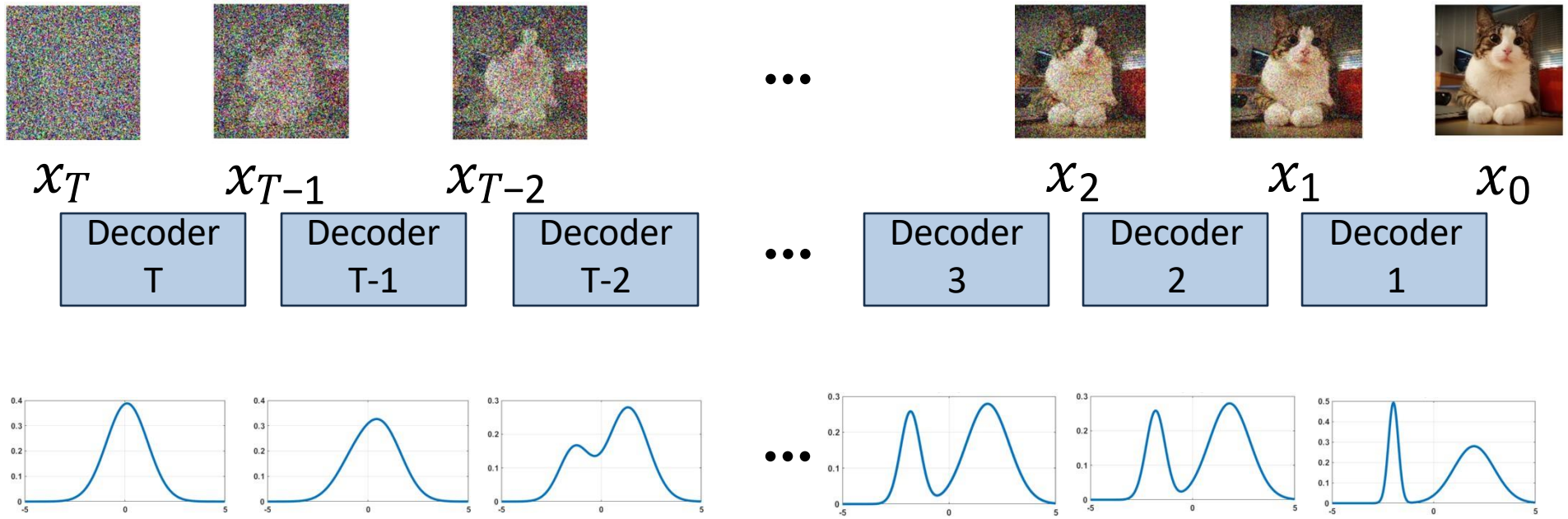
- Provides a seed model closer to final distribution

$$p(x_i|x_{i+1}) = \mathcal{N}(D(x_{i+1}), C)$$



Stacking VAEs

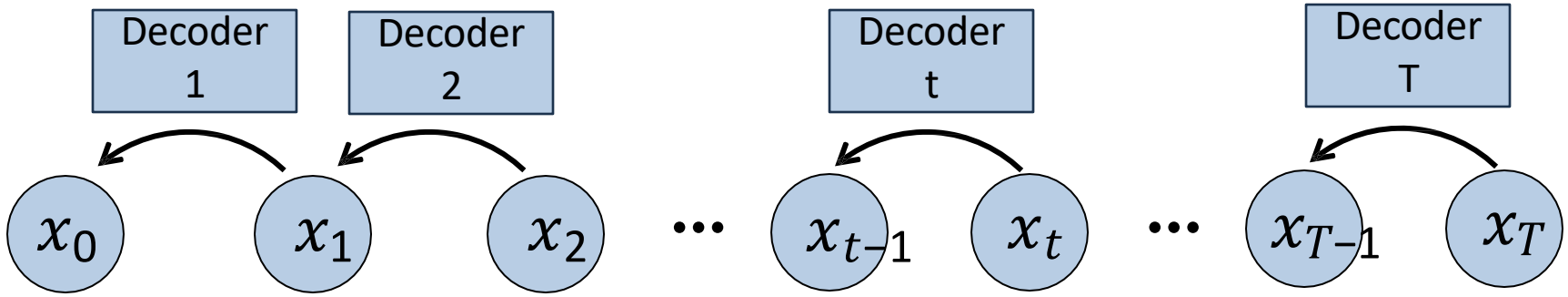
- We can have many many steps (in total T)…
- Each step incrementally recovers the final distribution



- Looks familiar?

Diffusion Models are Stacking VAEs

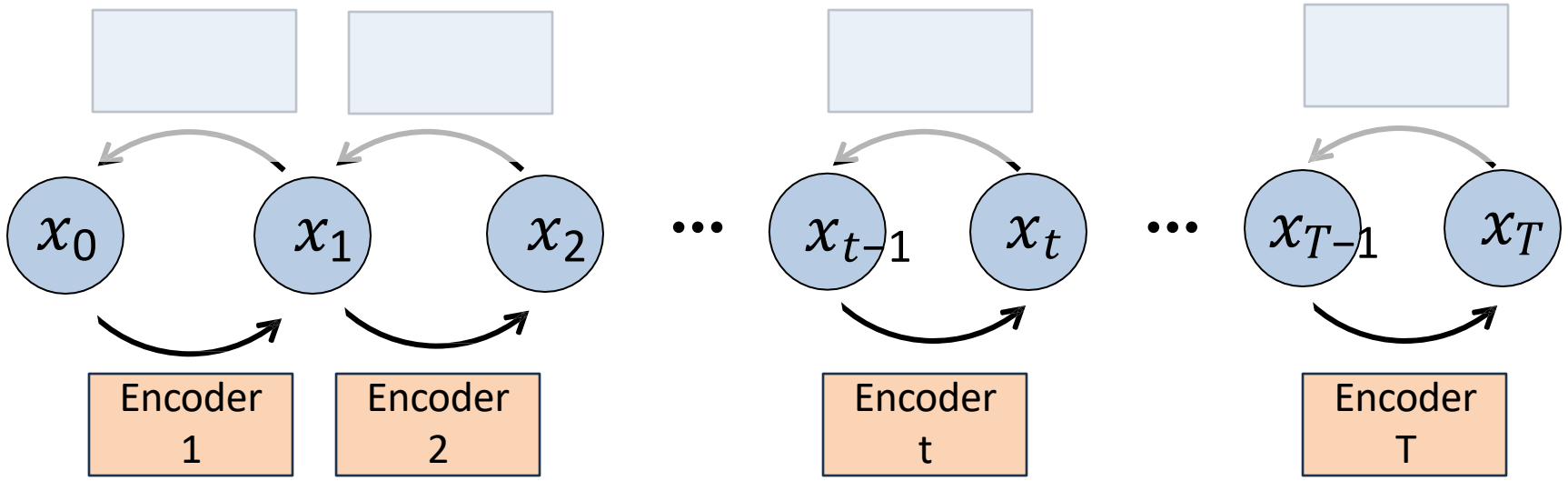
- Diffusion models are special cases of Stacking VAEs



- The reverse denoising process is the stack of decoders
- What about encoders?

Diffusion Models are Stacking VAEs

- Diffusion models are special case of Stacking VAEs

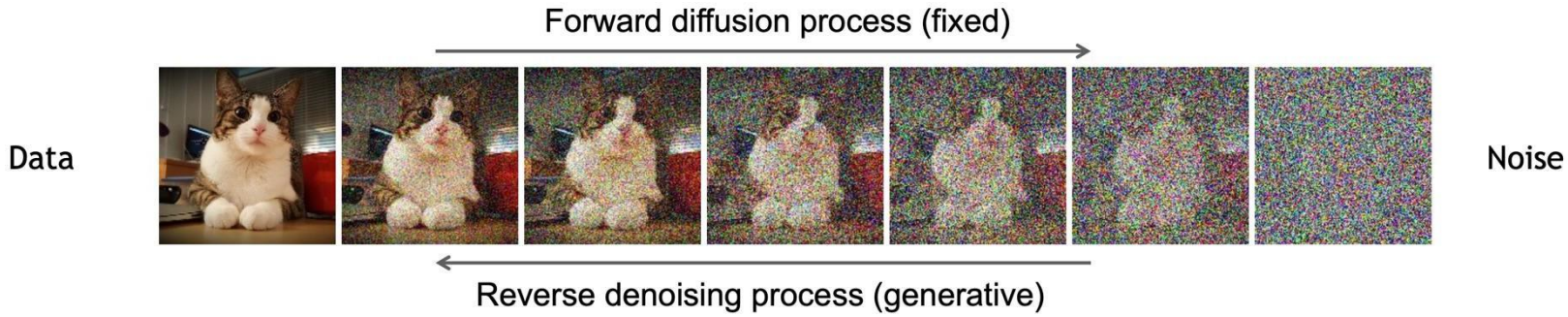


- In VAEs, encoders are learned with KL-divergence between the posterior and the prior
- Suffers from the 'posterior-collapse' issue
- Diffusion models use **fixed inference encoders**

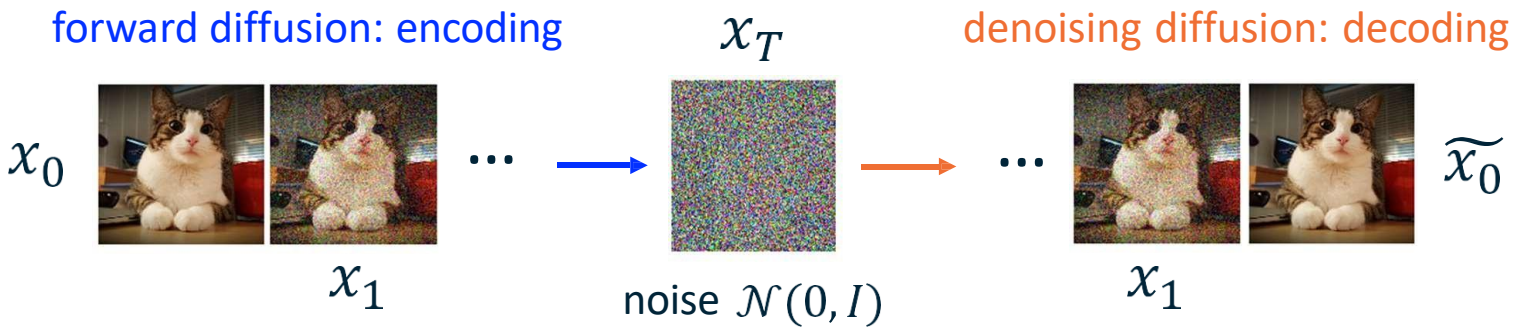
Chen et al. Variational Lossy Autoencoder. 2016

Denoising Diffusion Models

- Diffusion models have two processes
- **Forward diffusion process** gradually adds noise to input
- **Reverse denoising process** learns to generate data by denoising



Connection to VAEs



Known / predefined:

$q(x_{1:T}|x_0)$

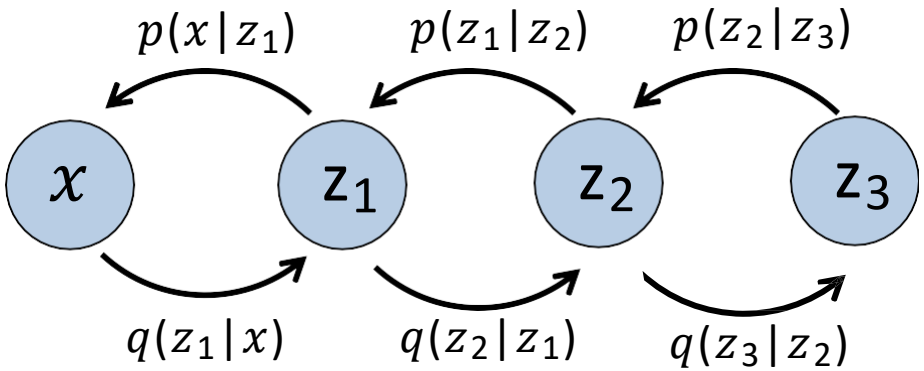
Unknown / learned:

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

Similar to VAEs, use the denoising decoding process to generate new images.

Connection with Hierarchical VAEs

- Diffusion models are special case of Hierarchical VAEs
  - Fixed inference models in forward process
  - Latent variables have same dimension as data
  - ELBO is decomposed to each timestep: faster to train
  - Model is trained with some weighting of ELBO



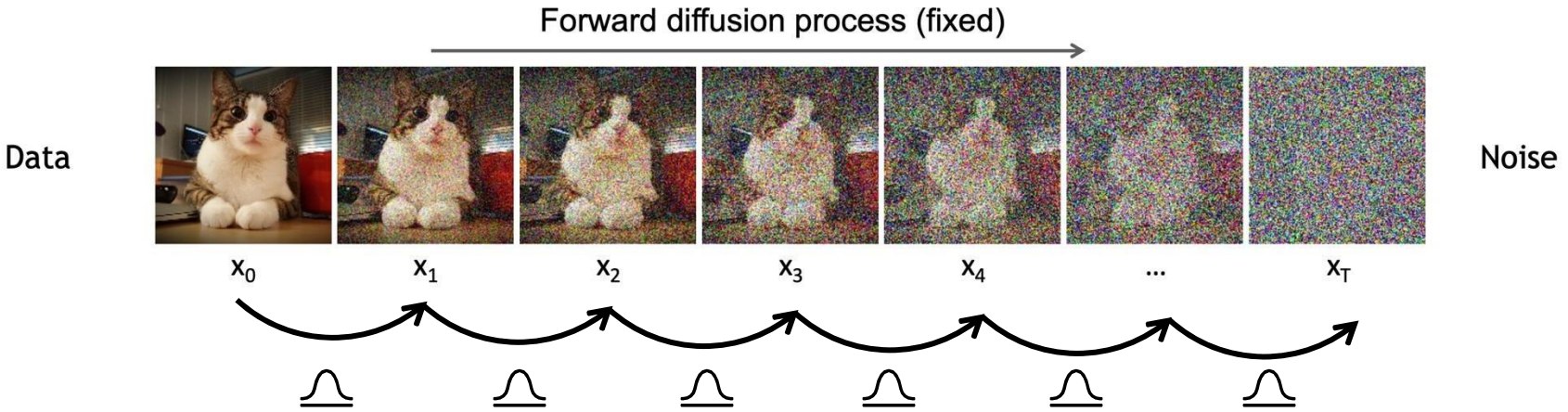
Ho et al. Denoising Diffusion Probabilistic Models. 2020.
Slide credit to: Ruiqi Gao CS231N

Forward Diffusion Process

- Forward diffusion process is stacking **fixed** VAE encoders
  - gradually adding Gaussian noise according to schedule β_t

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

$$q\left(\mathbf{x}_{1: T} \mid \mathbf{x}_0\right)=\prod_{t=1}^T q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)$$



The Diffusion (Encoding) Process

The **known** forward process $x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_T$

$$q(x_{1:\mathrm{T}}|x_0) = \prod_{\mathrm{t}=1}^{\mathrm{T}} q(x_{\mathrm{t}}|x_{\mathrm{t}-1})$$

Probability Chain Rule (Markov Chain)

The Diffusion (Encoding) Process

The **known** forward process $x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_T$

$$q(x_{1:\mathrm{T}}|x_0) = \prod_{\mathrm{t}=1}^{\mathrm{T}} q(x_\mathrm{t}|x_{\mathrm{t}-1})$$ Probability Chain Rule (Markov Chain)

$$q(x_\mathrm{t}|x_{\mathrm{t}-1}) = \mathcal{N}(x_\mathrm{t}; (\sqrt{1-\beta_\mathrm{t}}\ )x_{\mathrm{t}-1}, \beta_\mathrm{t} I)$$ Conditional Gaussian

The Diffusion (Encoding) Process

The **known** forward process $x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_T$

$$q(x_{1:\mathrm{T}}|x_0) = \prod_{\mathrm{t}=1}^{\mathrm{T}} q(x_\mathrm{t}|x_{\mathrm{t}-1})$$ Probability Chain Rule (Markov Chain)

$$q(x_\mathrm{t}|x_{\mathrm{t}-1}) = \mathcal{N}(x_\mathrm{t}; (\sqrt{1-\beta_\mathrm{t}}\,)x_{\mathrm{t}-1}, \beta_\mathrm{t} I)$$ Conditional Gaussian

Notation: A Gaussian distribution "for" x_t

The Diffusion (Encoding) Process

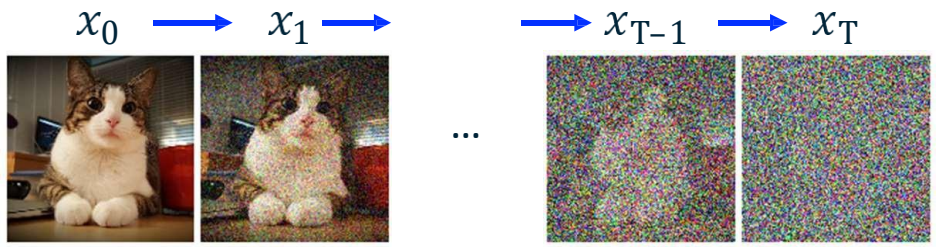
The **known** forward process $x_0 \longrightarrow x_1 \longrightarrow \cdots \longrightarrow x_T$

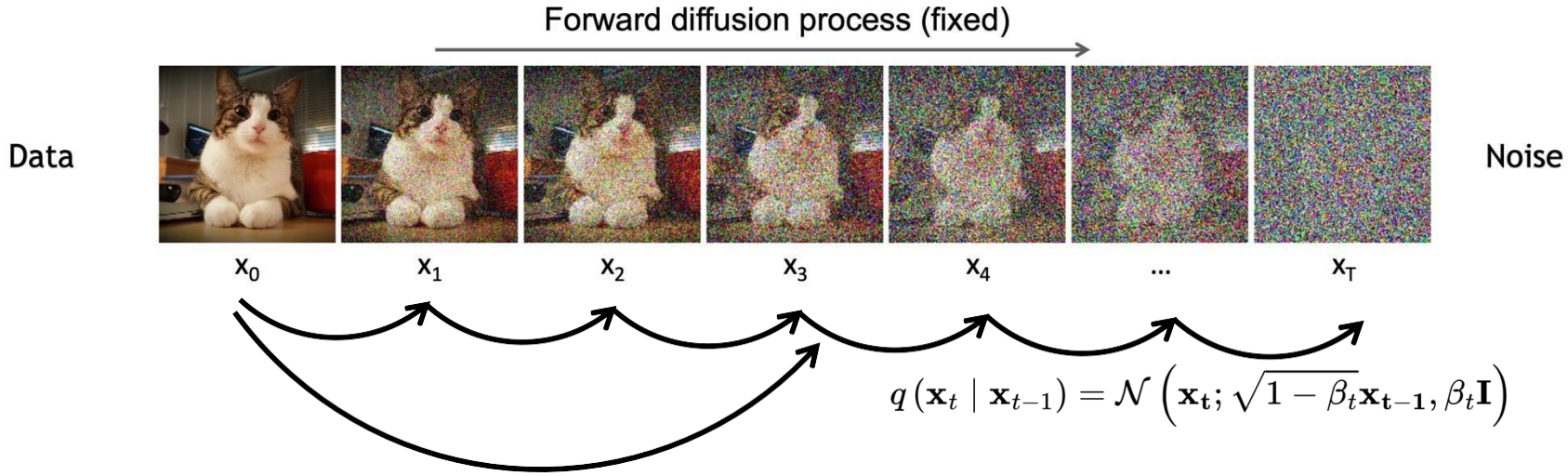
$$q(x_{1:\mathrm{T}}|x_0) = \prod_{\mathrm{t}=1}^{\mathrm{T}} q(x_\mathrm{t}|x_{\mathrm{t}-1})$$ Probability Chain Rule (Markov Chain)

$$q(x_\mathrm{t}|x_{\mathrm{t}-1}) = \mathcal{N}(x_\mathrm{t}; (\sqrt{1-\beta_\mathrm{t}}\ )x_{\mathrm{t}-1}, \beta_\mathrm{t} I)$$ Conditional Gaussian

β_t is the *variance schedule* at the diffusion step t

$0 < \beta_1 < \beta_2 < \cdots < \beta_\mathrm{T} < 1$, typical value range $[0.0001, 0.02]$, with $T = 1000$





- **Nice Property:** The forward process allows sampling of x_t at arbitrary timestep t in closed form:

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

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

- The noise schedule (β_t values) is designed such that

$$q(\mathbf{x}_T \mid \mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})) \quad \text{as} \quad \bar{\alpha}_T \to 0$$

DDPM: Noise Schedule

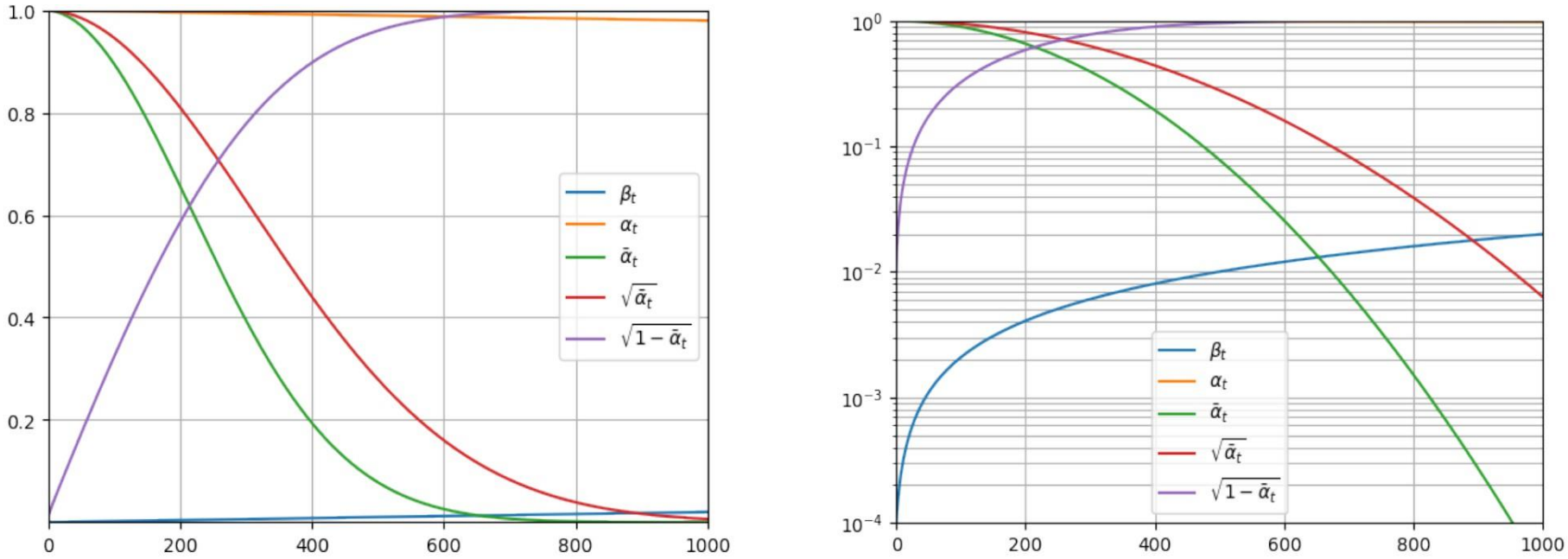


Figure 2: Parameter values for $\beta = [10^{-4}, 0.02]$ over 1000 time steps t using a linear schedule. The information in the two figures are the same, but the right-hand side uses log-scale on the y-axis to show the speed of which $\bar{\alpha}_t$ goes towards zero.

Str¨umke et al. Lecture Notes in Probabilistic Diffusion Models. 2020.

Generative Learning by Denoising

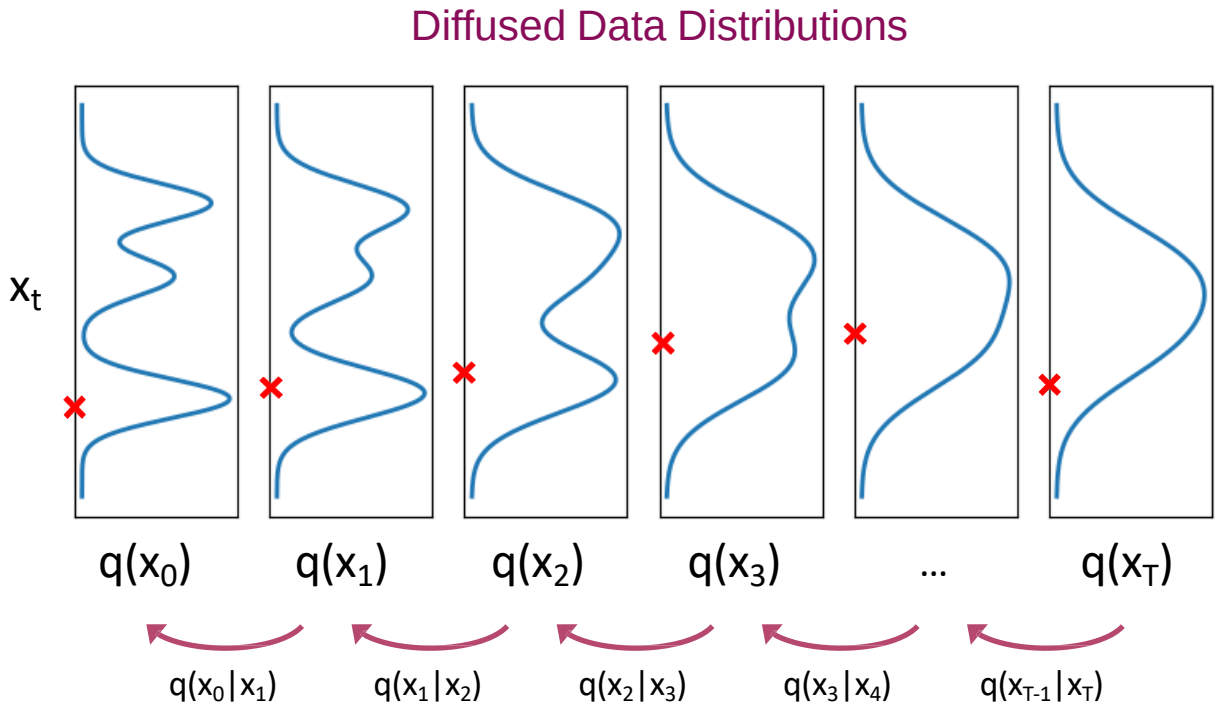
Diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Generation:

Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}$

True Denoising Dist.



In general, $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)$ is intractable.

Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$?

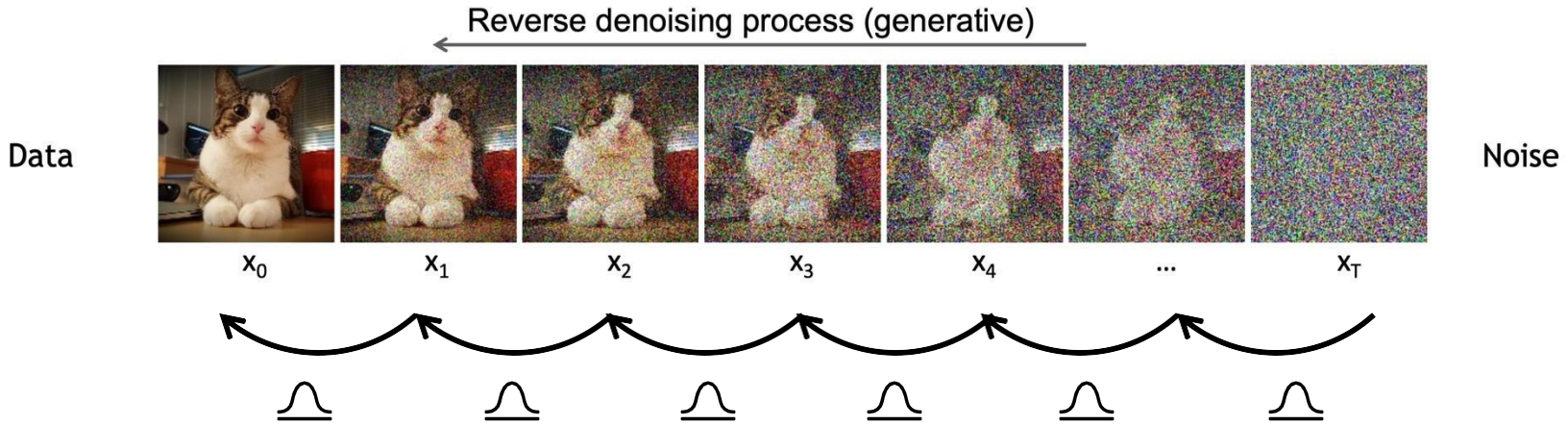
Yes, we can use a Gaussian distribution **if** β_t is small in each forward diffusion step.

– The purpose of our stack of VAE decoders!

slide from https://cvpr2022-tutorial-diffusion-models.github.io/

Reverse Denoising Process

- Reverse diffusion process is stacking **learnable** VAE decoders
  - Predicting the mean and std of *added* Gaussian Noise



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

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

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

Similar to the generative model in hierarchical VAEs.

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:\mathrm{T}}) = p(x_\mathrm{T}) \prod_{\mathrm{t}=1}^{\mathrm{T}} p_\theta(x_{\mathrm{t}-1} | x_\mathrm{t})$$

Probability Chain Rule (Markov Chain)

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:\mathrm{T}}) = p(x_\mathrm{T}) \prod_{\mathrm{t}=1}^{\mathrm{T}} p_\theta(x_{\mathrm{t}-1}|x_\mathrm{t})$$ Probability Chain Rule (Markov Chain)

$$p_\theta(x_{\mathrm{t}-1}|x_\mathrm{t}) = \mathcal{N}(x_{\mathrm{t}-1}; \mu_\theta(x_\mathrm{t}, t), \Sigma_\theta(x_\mathrm{t}, t))$$ Conditional Gaussian

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$ Probability Chain Rule (Markov Chain)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$ Conditional Gaussian

Trainable network
(U-net, Denoising Autoencoder)

Shared Across All Timesteps

Assume fixed / known variance
(simplification)

Want to learn time- dependent mean

How do we form a learning objective?

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

High-level intuition: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

High-level intuition: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

The learning objective: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

High-level intuition: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

The learning objective: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

What does it look like? $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right)$

$$\mu_q(t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right), \qquad \epsilon \sim \mathcal{N}(0, I)$$

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

High-level intuition: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

The learning objective: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

What does it look like? $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right)$

$$\mu_q(t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right), \qquad \epsilon \sim \mathcal{N}(0, I)$$

Recall: Gaussian reparameterization trick

The "ground truth" noise that brought x_{t-1} to x_t

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

High-level intuition: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

The learning objective: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

What does it look like? $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right)$

$$\mu_q(t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right), \qquad \epsilon \sim \mathcal{N}(0, I)$$

Assuming identical variance $\Sigma_q(t)$, we have:

$$\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) = \text{argmin}_\theta w||\mu_q(t) - \mu_\theta(x_t, t)||^2$$

Should be variance-dependent, but constant works better in practice

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$

High-level intuition: derive a *ground truth denoising distribution* $q(x_{t-1}|x_t, x_0)$ and train a neural net $p_\theta(x_{t-1}|x_t)$ to match the distribution.

The learning objective: $\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$

What does it look like? $q(x_{t-1}|x_t, x_0) = \mathcal{N}\left(x_{t-1}; \mu_q(t), \Sigma_q(t)\right)$

$$\mu_q(t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon\right), \qquad \epsilon \sim \mathcal{N}(0, I)$$

Assuming identical variance $\Sigma_q(t)$, we have:

$$\text{argmin}_\theta D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) = \text{argmin}_\theta w||\mu_q(t) - \mu_\theta(x_t, t)||^2$$

Simplified learning objective: $\text{argmin}_\theta||\epsilon - \epsilon_\theta(x_t, t)||^2$

Predict the one-step noise that was added (and remove it)!

The Denoising (Decoding) Process

The **learned** denoising process $x_0 \leftarrow x_1 \leftarrow \cdots \leftarrow x_T$

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$ Probability Chain Rule (Markov Chain)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_q(t))$$ Conditional Gaussian

Assume fixed / known variance

How did we arrive at the learning objective?
Let's go back to the basics of variational models …

Learning the Denoising Model

- Denoising models are trained with variational upper bound (negative ELBO), as VAEs

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta\left(\mathbf{x}_0\right)\right] \leq \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right] =: L$$

- which derives to (Sohl-Dickstein et al. ICML 2015 and Ho et al. NeurIPS 2020):

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

- where tractable posterior distribution (closed-form)

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

Condition on x_0 makes it tractable!
Can only use during training!

$$\text{where } \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1-\bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{1-\beta_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t} \mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t$$

Ho et al. Denoising Diffusion Probabilistic Models. 2020.

Learning the Denoising Model

- Denoising models are trained with variational upper bound (negative ELBO), as VAEs

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta\left(\mathbf{x}_0\right)\right] \leq \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right] =: L$$

- which derives to :

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

constant

Scaling

Ho et al. actually drop this term entirely during training

- Where tractable posterior distribution (closed-form)

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

$$\text{where } \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}\mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

Ho et al. Denoising Diffusion Probabilistic Models. 2020.

Learning the Denoising Model

- Denoising models are trained with variational upper bound (negative ELBO), as VAEs

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta\left(\mathbf{x}_0\right)\right] \leq \mathbb{E}_{q(\mathbf{x}_0) q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\left[-\log \frac{p_\theta\left(\mathbf{x}_{0:T}\right)}{q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right)}\right] =: L$$

- which derives to :

$$L = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\theta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{-\log p_\theta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}]$$

constant

Scaling

- Where tractable posterior distribution (closed-form)

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

$$\text{where } \tilde{\mu}_t\left(\mathbf{x}_t, \mathbf{x}_0\right) := \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1-\bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{1-\beta_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t} \mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \beta_t$$

Ho et al. Denoising Diffusion Probabilistic Models. 2020.

Parameterizing the Denoising Model

Since both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Gaussian distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2\right] + C$$

Given $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$. DDPM Ho et al. NeurIPS 2020 observe that (equivalent to the equation in previous slide):

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boxed{\epsilon_\theta(\mathbf{x}_t, t)}\right)$$

Trainable network predicts the noise mean

With this parameterization

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

slide from https://cvpr2022-tutorial-diffusion-models.github.io/

Training Objective Weighting

Trading likelihood for perceptual quality

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \Big[\underbrace{\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon, t)||^2 \Big]$$

The time dependent λ_t ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small t's.

DDPM Ho et al. NeurIPS 2020 observe that simply setting $\lambda_t = 1$ improves sample quality. So, they propose to use:

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

slide from https://cvpr2022-tutorial-diffusion-models.github.io/

Summary

Training and Sample Generation

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
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

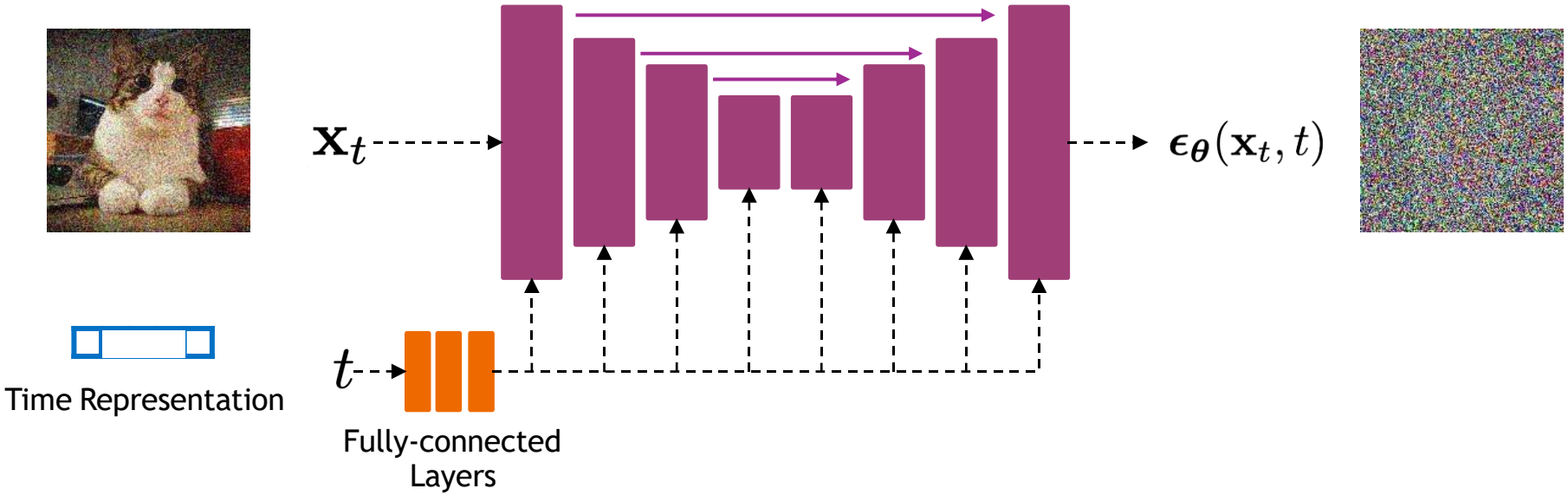
Algorithm 2 Sampling

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

Implementation Considerations

Network Architectures

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.

Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see Dhariwal and Nichol NeurIPS 2021)

slide from https://cvpr2022-tutorial-diffusion-models.github.io/