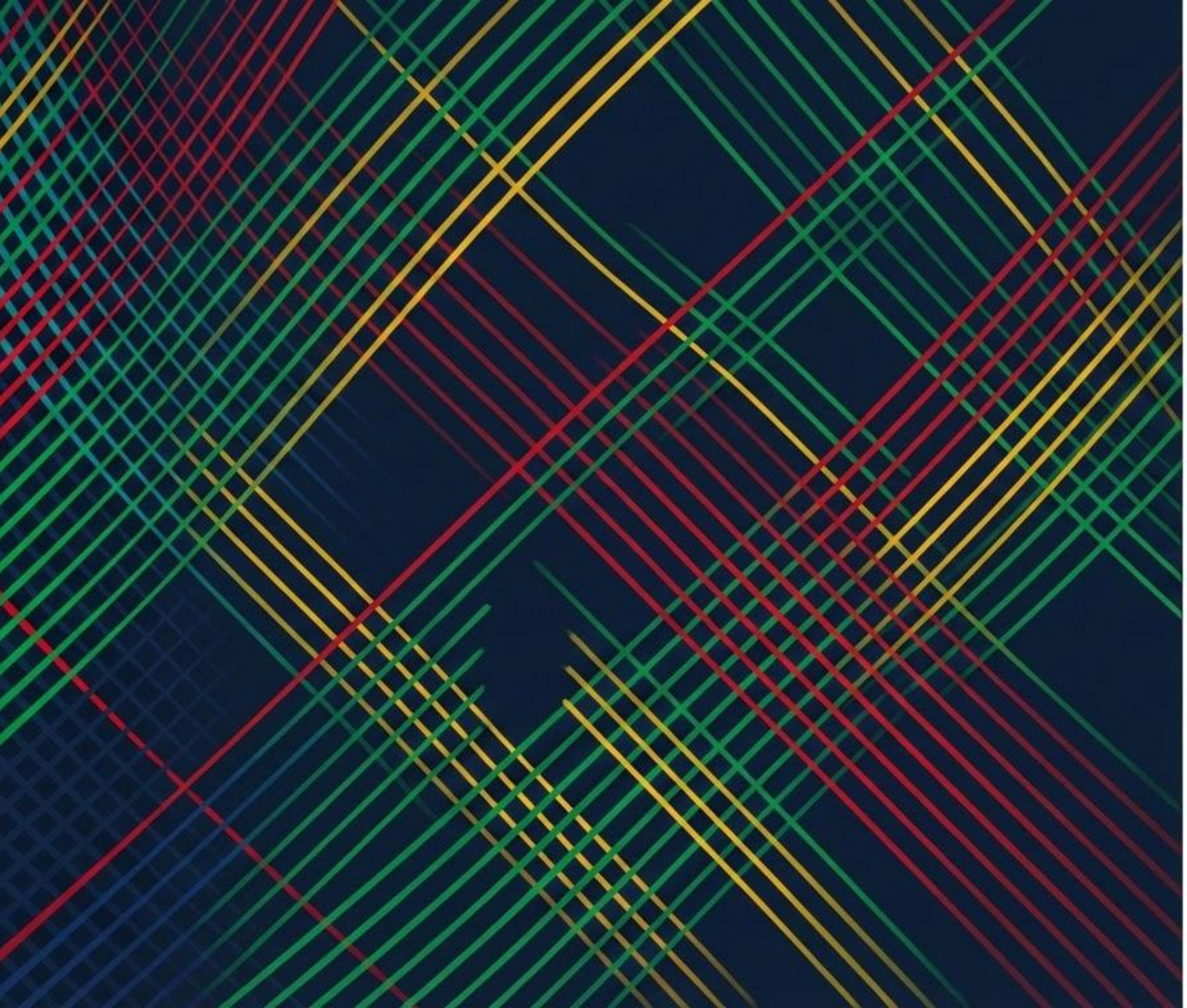


Diffusion

Presented By: Massa Baali

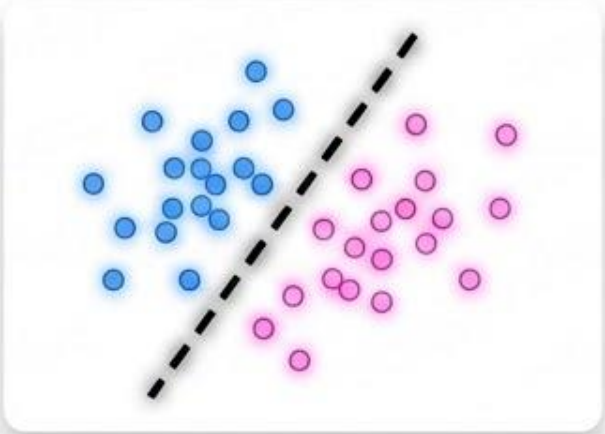
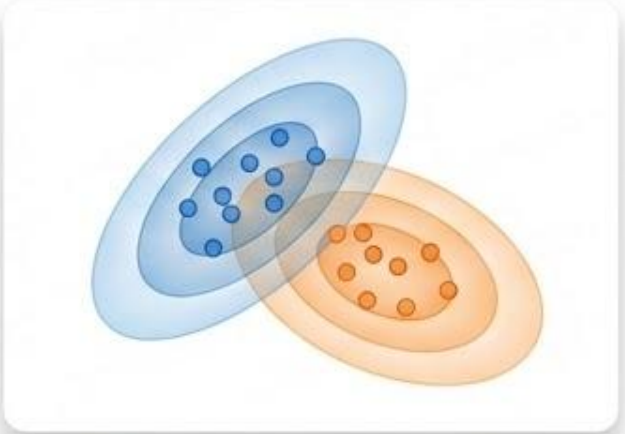
Carnegie Mellon University

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Discriminative vs. Generative Models

https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning/

Discriminative model		Generative model	
Goal	Directly estimate $P(y\|\mathbf{x})$	**Goal**	Estimate $P(\mathbf{x}\|y)$ to then deduce $P(y\|\mathbf{x})$
What's learned	Decision boundary	**What's learned**	Probability distributions of the data
Illustration		**Illustration**	
Examples	Regressions, SVMs	**Examples**	GDA, Naive Bayes

Deep Generative learning

Learning to generate data

Samples from a Data Distribution

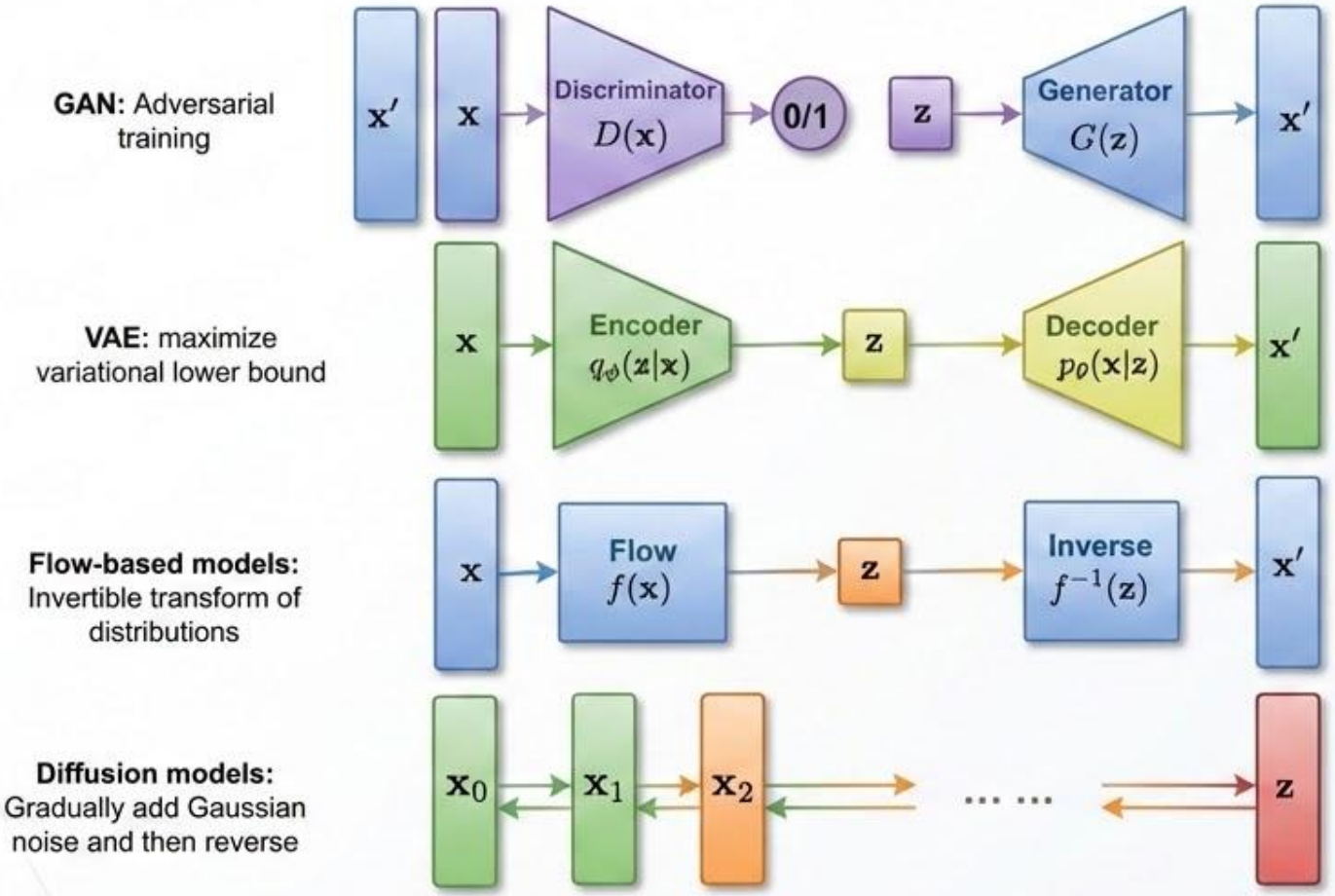
Train

Neural Network

Sample

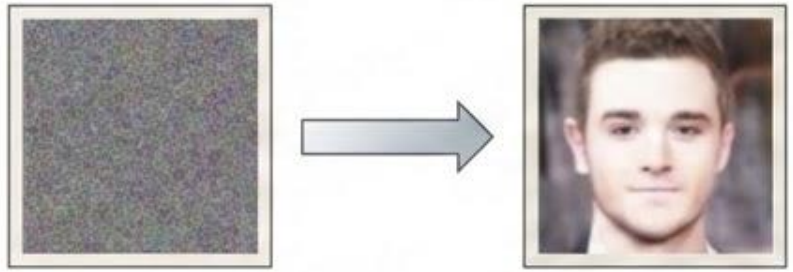
Slide credit to: https://cvpr2022-tutorial-diffusion-models.github.io/

Generative Models



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

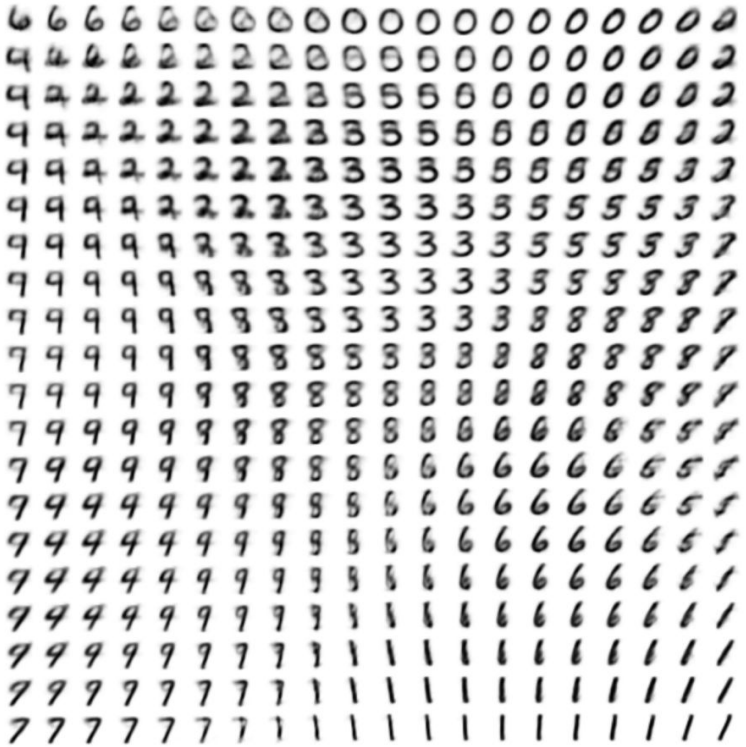
Limitations of VAEs



The decoder must transform a standard Gaussian *all the way* to the sample from the target distribution in **one step**.

- Too large a gap to bridge in one step
- The decoder has to do ALL the work at once

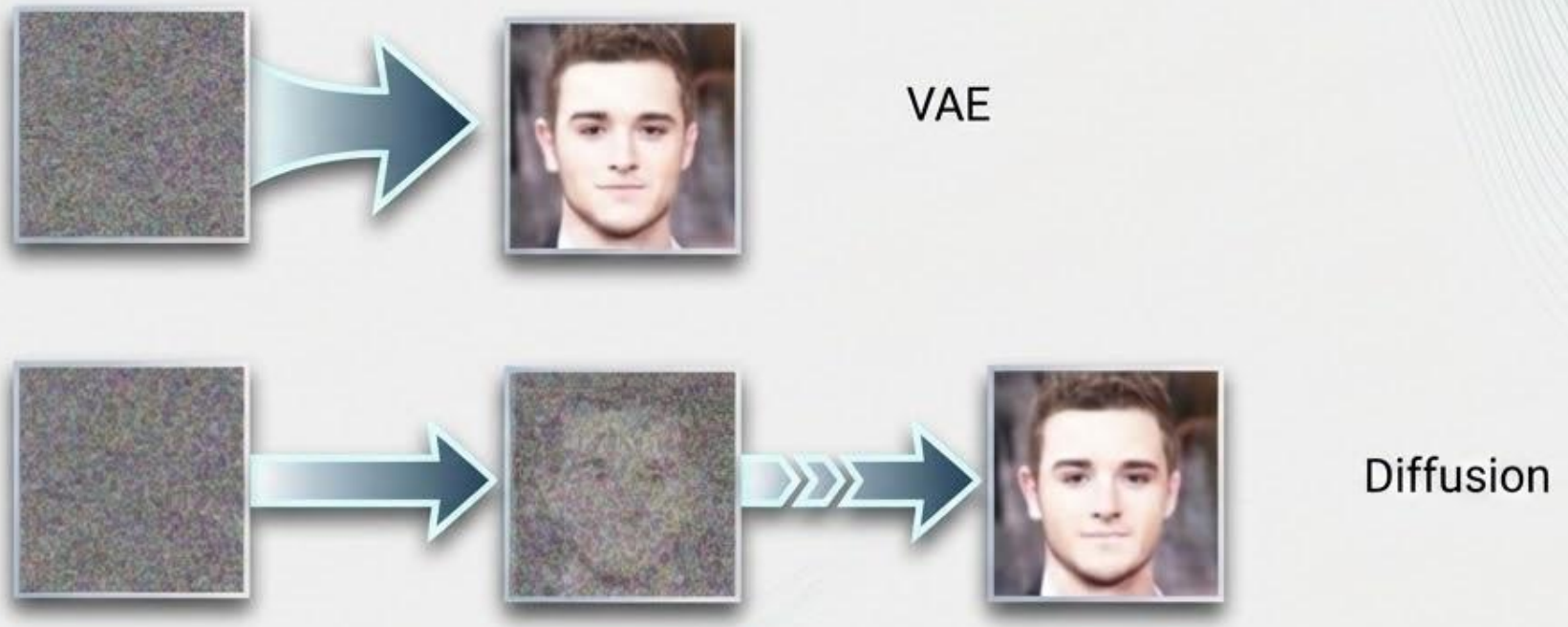
Limitations of VAEs



Result: Blurry, Low-Quality Images

Kingma et al. Auto-Encoding Variational Bayes

From Single-Step to Multi-Step Generation



Hierarchical VAEs

- Stack multiple VAEs with intermediate latent variables
- Each level: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_t$
- Each step is a small, learnable transformation

Diffusion Models: A Special Case

- Can be viewed as a hierarchical VAE with specific choices:
  - All latent variables have the same dimension as the original image
  - Forward process (encoder) is fixed (just adds Gaussian noise)
  - Only learn the reverse process (decoder/denoising)
  - Many steps (e.g., T = 1000)

Poll 1

Why do VAEs produce blurry images?

A) Not enough training data 

B) Bad optimizer choice 

C) One-step jump from noise to image is too hard 

D) Network architecture is too small 

Poll 1

Why do VAEs produce blurry images?

A) Not enough training data

B) Bad optimizer choice

C) One-step jump from noise to image is too hard ✓

D) Network architecture is too small

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

What are diffusion models?

Inspired by diffusion processes

(Brownian motion)

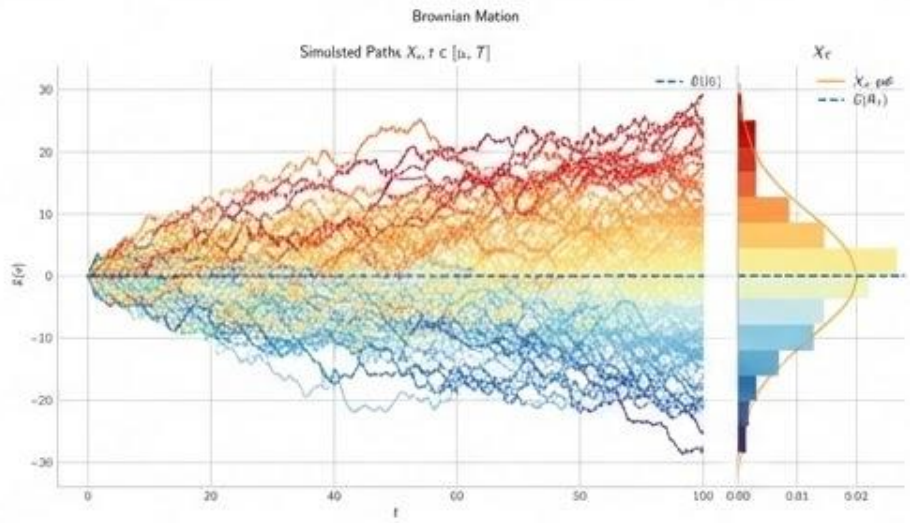
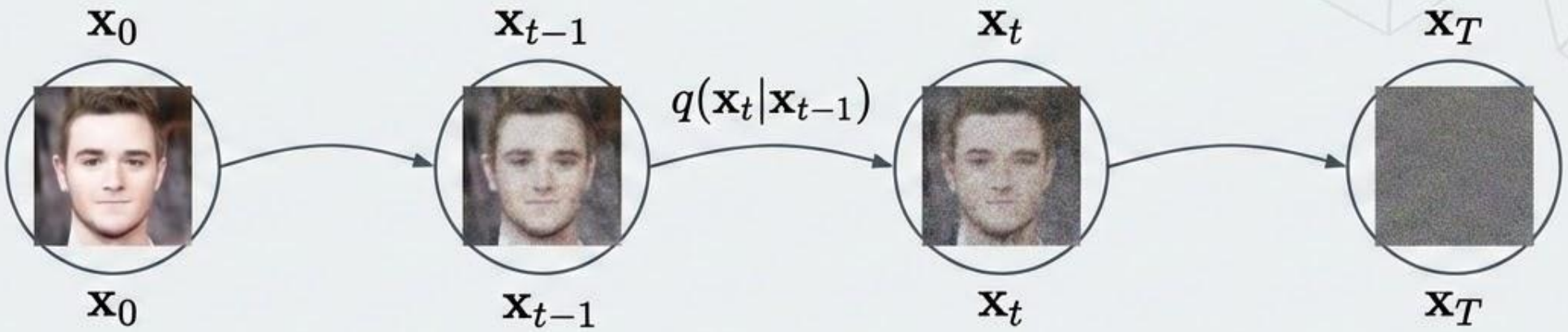


Image credit:
https://quantgirluk.github.io/Understanding-Quantitative-Finance/brownian_motion.html

t=0 t=T/2 t=T

Adapted from: Ho, J., Jain, A. and Abbeel, P., 2020. Denoising diffusion probabilistic models.

Forward Process



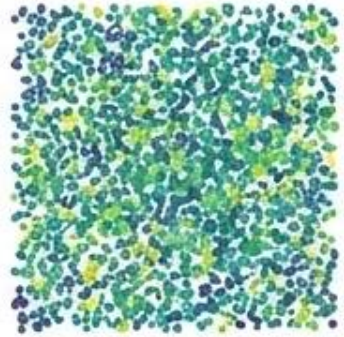
Gradual Noise Addition: Over t time steps, noise is progressively added to an image.

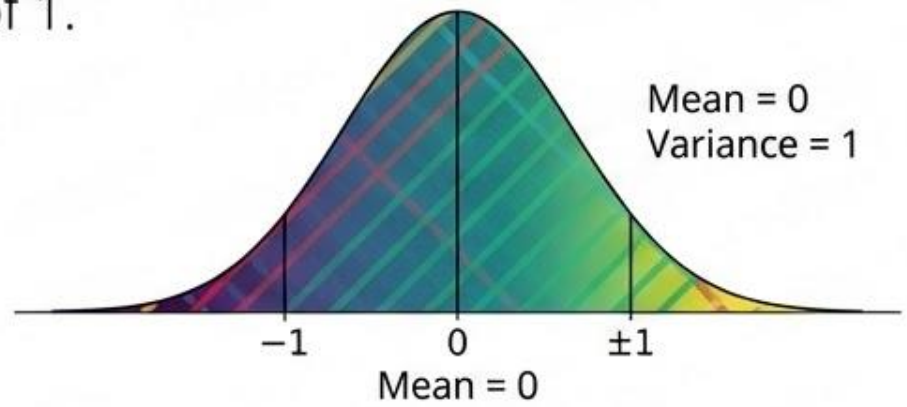
Markov Chain Model: The process, denoted by q, takes the form of a Markov Chain where the distribution at a particular time step only depends on the sample from the immediate previous step.

Adapted from: https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

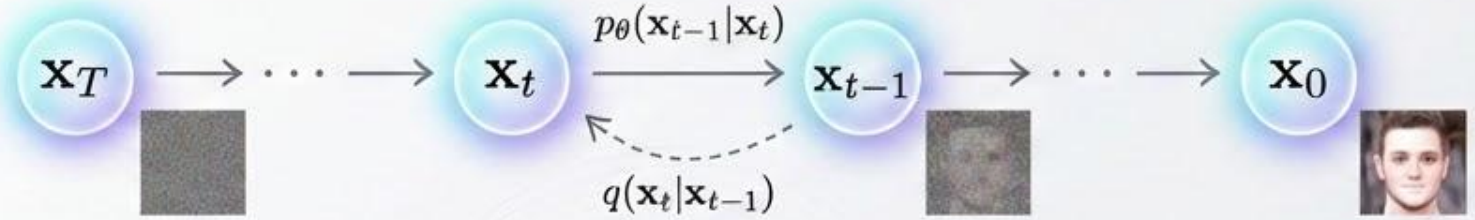
How do we add noise?

- **Gaussian Noise:** The noise added in diffusion models follows a Gaussian (normal) distribution. This means at each diffusion step, we inject some random noise that has the familiar "bell curve" distribution.
- **Standard Normal Sampling:** We sample the noise from a standard normal distribution $\mathcal{N}(0, \boldsymbol{I})$, which has a mean of 0 and a variance of 1.





Reverse Process



≋ **Denoise:** The model learns to gradually remove noise from the data at each step.

🖧 **Reconstruction:** Aims to reconstruct the original image or data from its noisy version.

🎯 The main goal of training a diffusion model is learning the reverse process specifically training pθ(xt−1 | xt).

Adapted from:
https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

Training diffusion models

Denoising diffusion models estimate a noise vector $\epsilon \in \mathbb{R}^n$ from a given noise level $\sigma > 0$ and noisy input $x_\sigma \in \mathbb{R}^n$ such that for some x_0 in the data manifold $\mathcal{K}$,

$$x_\sigma \approx x_0 + \sigma\epsilon$$

A *denoiser* $\epsilon_\theta : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^n$ is learned by minimizing

$$\mathcal{L}(\theta) := \mathbf{E}_{x_0,\sigma,\epsilon} \left\| \epsilon_\theta(x_0 + \sigma\epsilon, \sigma) - \epsilon \right\|^2$$

- x_0 is sampled from training data
- σ is sampled from *training noise schedule* (more on this later)
- ϵ is sampled from $\mathcal{N}(0, I_n)$ (i.i.d. Gaussian)

Slide credit to: https://www.practical-diffusion.org/

Training noise schedules

What levels of noise to use?

In practice, noise levels range from $\sigma_{min} = 0.01$ to $\sigma_{max}=100$

Sampled uniformly from a *noise schedule* (typically 100-1000 discrete noise levels)

$$\sigma \sim \{\sigma_1, \ldots, \sigma_{1000}\}$$

Typically follow a log-scale $\lambda = \log(\sigma)$

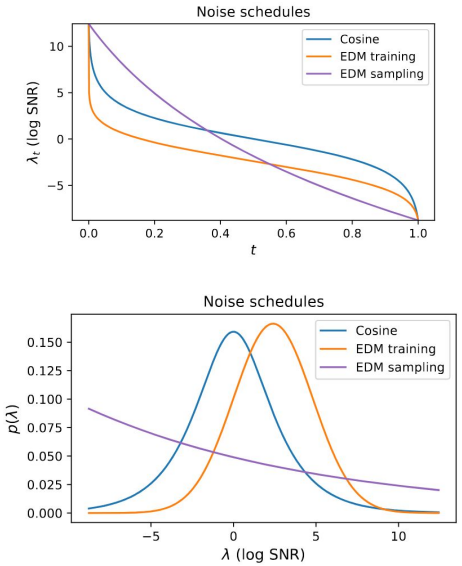


Image credit: Understanding Diffusion Objectives as the ELBO with Simple Data Augmentation https://arxiv.org/pdf/2303.00848

Slide credit to: https://www.practical-diffusion.org/

Training Pseudocode

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

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

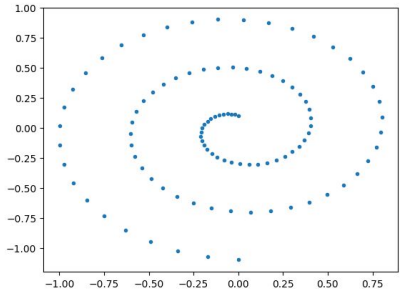
6: **until** converged

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

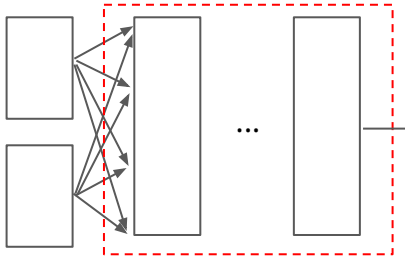
Toy Model

2D spiral dataset



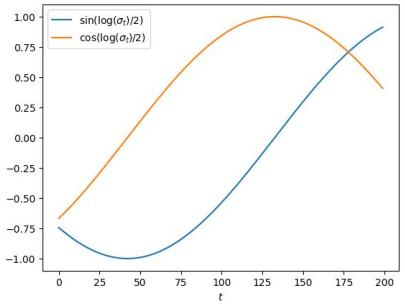
Add noise

Fully Connected MLP



Predict Noise

2D noise-level embedding



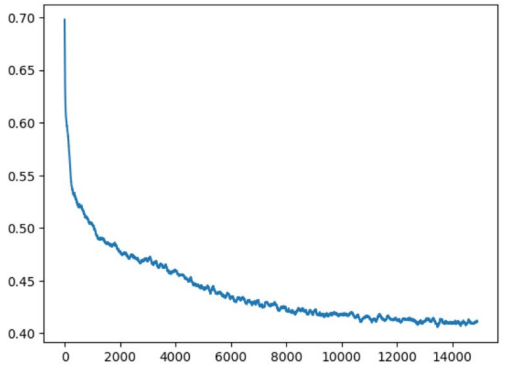
Noise-level embedding

```python
def get_sigma_embeds(sigma):
    sigma = sigma.unsqueeze(1)
    return torch.cat([torch.sin(torch.log(sigma)/2),
                      torch.cos(torch.log(sigma)/2)], dim=1)

class TimeInputMLP(nn.Module):
    def __init__(self, dim, hidden_dims):
        super().__init__()
        layers = []
        for in_dim, out_dim in pairwise((dim + 2,) + hidden_dims):
            layers.extend([nn.Linear(in_dim, out_dim), nn.GELU()])
        layers.append(nn.Linear(hidden_dims[-1], dim))
        self.net = nn.Sequential(*layers)
        self.input_dims = (dim,)

    def rand_input(self, batchsize):
        return torch.randn((batchsize,) + self.input_dims)

    def forward(self, x, sigma):
        sigma_embeds = get_sigma_embeds(sigma)         # shape: b x 2
        nn_input = torch.cat([x, sigma_embeds], dim=1) # shape: b x (dim + 2)
        return self.net(nn_input)

model = TimeInputMLP(dim=2, hidden_dims=(16,128,128,128,128,16))
```


Slide credit to: https://www.practical-diffusion.org/

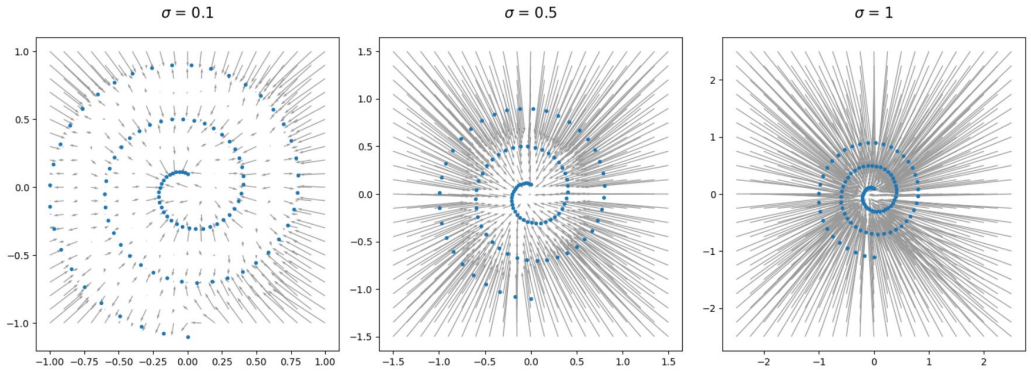
Training the Toy Model

```python
def training_loop(loader  : DataLoader,
                  model   : nn.Module,
                  schedule: Schedule,
                  epochs  : int = 10000):
    optimizer = torch.optim.Adam(model.parameters())
    for _ in range(epochs):
        for x0 in loader:
            optimizer.zero_grad()
            sigma, eps = generate_train_sample(x0, schedule)
            eps_hat = model(x0 + sigma * eps, sigma)
            loss = nn.MSELoss()(eps_hat, eps)
            optimizer.backward(loss)
            optimizer.step()
```



Training loss over 15000 epochs, smoothed with moving average

The learned denoiser $\epsilon_\theta(x, \sigma)$ can be visualized as a vector field parameterized by the noise level σ, by plotting $x - \sigma\epsilon_\theta(x, \sigma)$ for different x and levels of σ.



Plot of predicted $\hat{x}_0 = x - \sigma\epsilon_\theta(x, \sigma)$ for different x and σ

The model predicts expected x0 given x and σ

Slide credit to: https://www.practical-diffusion.org/

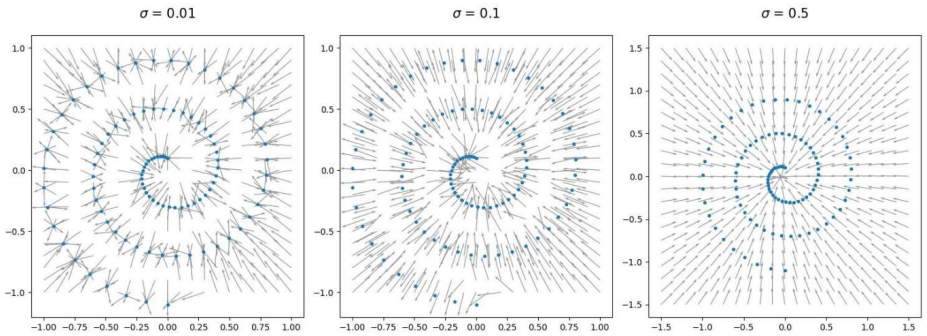
What did the model learn?

The *ideal denoiser* is the minimizer of training loss, a function of data distribution $\mathcal{K}$ and noise level σ

$$\epsilon^* := \arg\min_{\epsilon_\theta} \mathbf{E}_{x_0,\sigma,\epsilon} \left\| \epsilon_\theta(x_0 + \sigma\epsilon, \sigma) - \epsilon \right\|^2$$

For finite $\mathcal{K}$, there is a closed-form solution:

$$\epsilon^*(x_\sigma, \sigma) = \frac{\sum_{x_0 \in \mathcal{K}} (x_\sigma - x_0) \exp(-\left\| x_\sigma - x_0 \right\|^2 / 2\sigma^2)}{\sigma \sum_{x_0 \in \mathcal{K}} \exp(-\left\| x_\sigma - x_0 \right\|^2 / 2\sigma^2)}$$

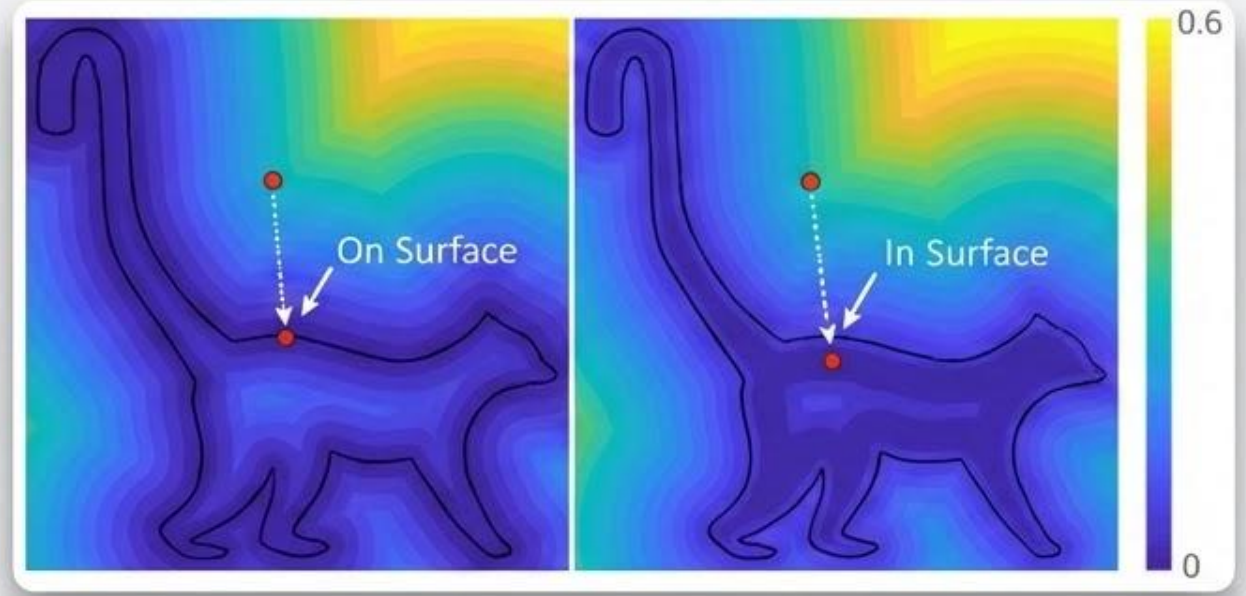


Plot of direction of $\epsilon^*(x, \sigma)$ for different x and σ

Slide credit to: https://www.practical-diffusion.org/

What did the model learn?

Denoiser model learns the gradient of a σ-smoothed distance function



$$\mathrm{dist}_{\mathcal{K}}(x,\sigma) := \operatorname*{softmin}_{x_0 \in \mathcal{K}^\epsilon}{}^{2}_{\sigma} \|x_0 - x\|^2 = -\sigma^2 \log\left(\sum_{x_0 \in \mathcal{K}} \exp\left(-\frac{\|x_0 - x\|^2}{2\sigma^2}\right)\right)$$

Theorem

For all $\sigma > 0$ and $x \in \mathbb{R}^n$, we have

$$\frac{1}{2}\nabla_x \mathrm{dist}_x^2(x,\sigma) = \sigma\epsilon^*(x,\sigma).$$

Image credit: Fast Evaluation of Smooth Distance Constraints on Co-Dimensional Geometry https://arxiv.org/pdf/2108.10480
Slide credit to: https://www.practical-diffusion.org/

Poll 2

What does the neural network actually predict during training?

- A) The clean image x_0
- B) The noisy image x_t
- C) The noise ε that was added
- D) The next timestep t+1

Poll 2

What does the neural network actually predict during training?

- A) The clean image x_0
- B) The noisy image x_t
- C) The noise ε that was added
- D) The next timestep t+1

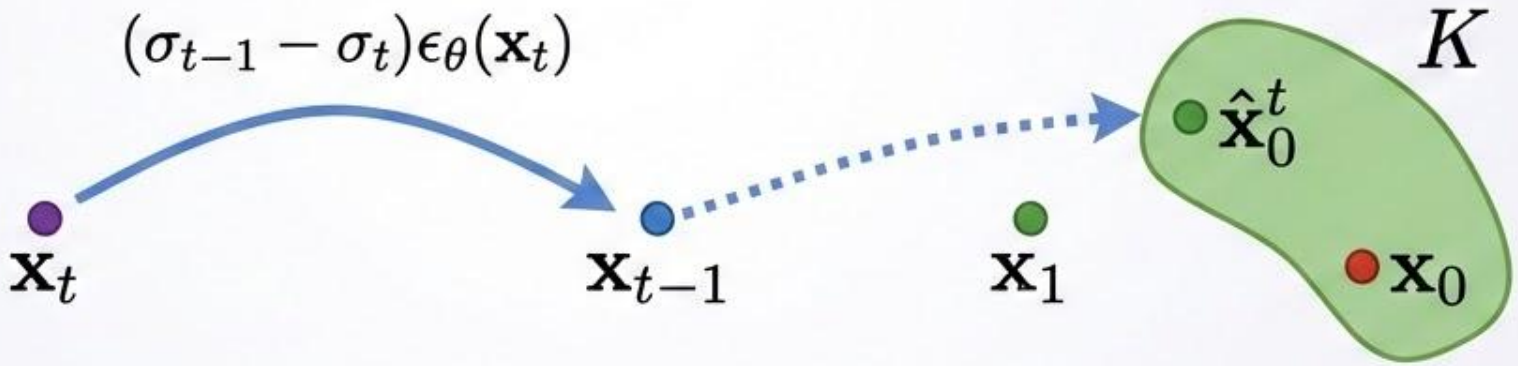
Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Sampling from diffusion models

Given noisy $\mathbf{x}_\sigma$ and noise level σ, the learned denoiser $\epsilon_\theta(\mathbf{x}_\sigma, \sigma)$ estimates

$$\mathbf{x}_0 \approx \hat{\mathbf{x}}_0(\mathbf{x}_\sigma, \sigma) := \mathbf{x}_\sigma - \sigma\epsilon_\theta(\mathbf{x}_\sigma, \sigma)$$



Slide credit to: https://www.practical-diffusion.org/

Sampling

Algorithm 2 Sampling

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

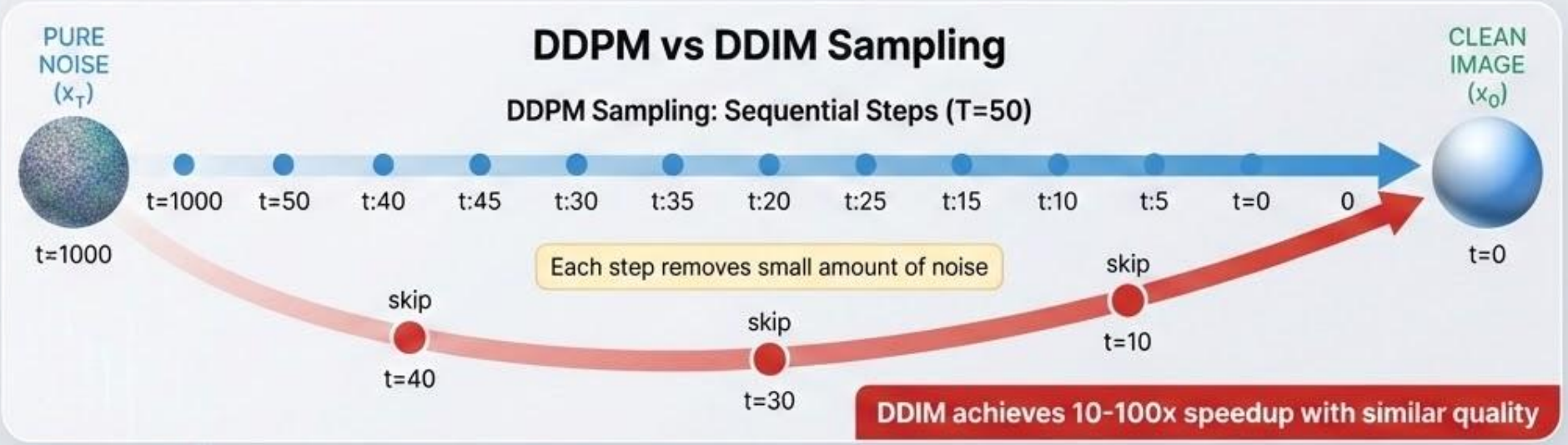
The Speed Problem

DDPM Challenge:

- Must run model T=1000 times sequentially
- Each step removes tiny amount of noise
- Generation takes minutes per image

DDIM Solution:

- Skip timesteps intelligently
- Use 10-50 steps instead of 1000
- 10-100× faster with similar quality



Denoising Diffusion Implicit Models (DDIM)

Limitations of DDPM Inference

🔗 **Sequential Denoising:** Must process each timestep in reverse to remove Gaussian noise. This happens due to the markov chain structure of the reverse process.

Algorithm 2 Sampling

1: $x_T \sim \mathcal{N}(0, I)$ ▷ *Initial isotropic gaussian noise sampling*
2: **for** t = T, ..., 1 **do**
3: $z \sim \mathcal{N}(0, I)$ if $t > 1$ else $z = 0$ ▷ *Sample random noise (if not last step)*
4: $\tilde{\epsilon} = \epsilon_\theta(x_t, t)$ ▷ *Estimated noise in current noisy data*
5: $\tilde{x_0} = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t}\tilde{\epsilon})$ ▷ *Estimated x_0 from estimated noise*
6: $\tilde{\mu} = \mu_t(x_t, \tilde{x_0}) \left(= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)\right)$ ▷ *Mean for previous step sampling*
7: $x_{t-1} = \tilde{\mu} + \sigma_t z$ ▷ *Previous step sampling*
8: **end for**
9: **return** x_0

DDIM Overview

Deterministic Sampling: DDIM uses a non-Markovian process allowing for fewer timesteps. It modifies the inference step by modifying the reverse process, making the process deterministic.

$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{“ predicted } \boldsymbol{x}_0\text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\boldsymbol{x}_t)}_{\text{“direction pointing to } \boldsymbol{x}_t\text{”}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}}$$

Implementing the sampler

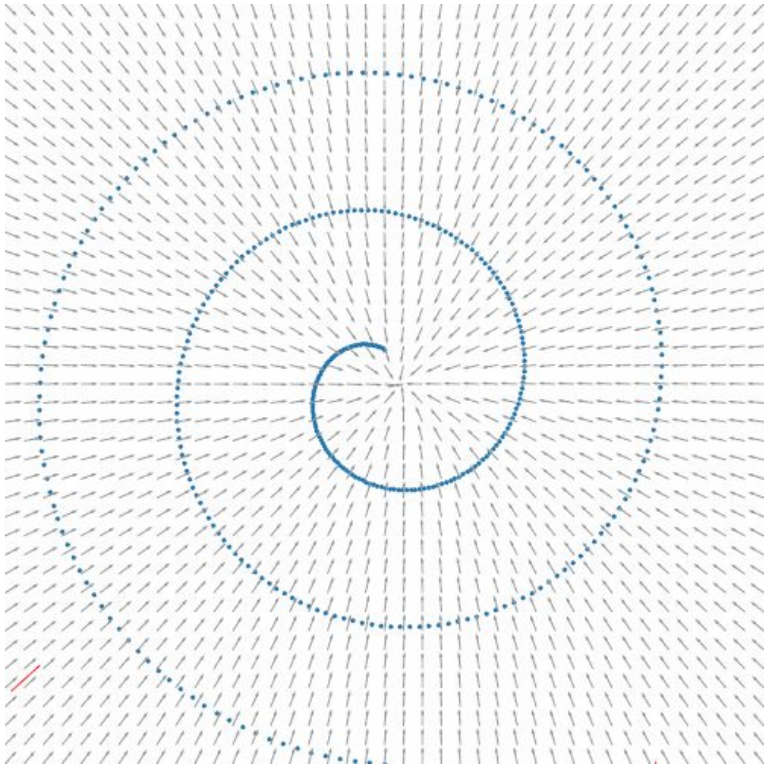
Algorithm 1 DDIM sampler

for $t = N, \ldots, 1$ **do**

$$x_{t-1} \leftarrow x_t + (\sigma_{t-1} - \sigma_t)\epsilon_\theta(x_t, \sigma_t)$$

return x_0

```
@torch.no_grad()
def samples_ddim(model      : torch.nn.Module,
                 sigmas     : torch.FloatTensor,
                 xt         : Optional[torch.FloatTensor] = None,
                 batchsize  : int = 1):
    model.eval()
    accelerator = Accelerator()
    xt = model.rand_input(batchsize).to(accelerator.device) * sigmas[0]
    for i, (sig, sig_prev) in enumerate(pairwise(sigmas)):
        xt = xt - (sig - sig_prev) * model.predict_eps(xt, sig.to(xt))
    return xt
```



Slide credit to: https://www.practical-diffusion.org/

Poll 3

DDIM is 10-50× faster than DDPM because:

- A) It uses a smaller neural network
- B) It skips timesteps intelligently
- C) It trains faster
- D) It uses a better GPU

Poll 3

DDIM is 10-50× faster than DDPM because:

- A) It uses a smaller neural network
- B) It skips timesteps intelligently
- C) It trains faster
- D) It uses a better GPU

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Continuous vs Discrete Diffusion Processes

	Discrete Processes	Continuous Processes
Timesteps	$t \in \{1, \ldots, T\}$	$t \in [0, 1]$
Noise levels	$\sigma_t \in \{\sigma_1, \ldots, \sigma_T\}$	$\sigma : [0, 1] \to \mathbb{R}_{\geq 0}$
Diffused samples	$x_t \in \{x_1, \ldots, x_T\}$	$\boldsymbol{x} : [0, 1] \to \mathbb{R}^d$
	$\boldsymbol{x}_t = \boldsymbol{x}_0 + \sigma_t \boldsymbol{\epsilon},$ $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$	$\boldsymbol{x}(t) = \boldsymbol{x}(0) + \sigma(t)\boldsymbol{\epsilon},$ $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),\ \sigma(0) = 0$

 Slide credit to: https://www.practical-diffusion.org/

Continuous Diffusion Processes

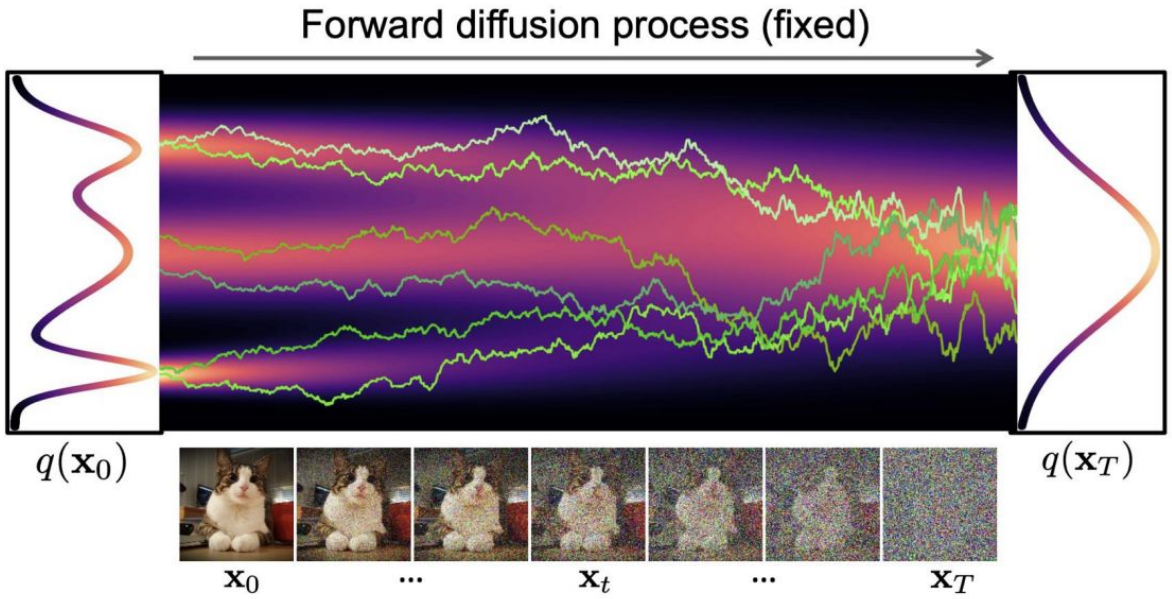
To simplify notation, from here on we will use **continuous processes** everywhere but may use **subscripts** for the "time" of the diffusion process when convenient, i.e.

$$\sigma_t := \sigma(t) \in \mathbb{R}_{\geq 0}$$
$$x_t := x(t) \in \mathbb{R}^d$$

$$x_t \overset{\text{dist}}{=} x_0 + \sigma_t \epsilon,$$
$$\epsilon \sim \mathcal{N}(0, I),$$
$$\sigma_0 = 0$$

Slide credit to: https://www.practical-diffusion.org/

Generative Reverse SDEs



drift term

diffusion term

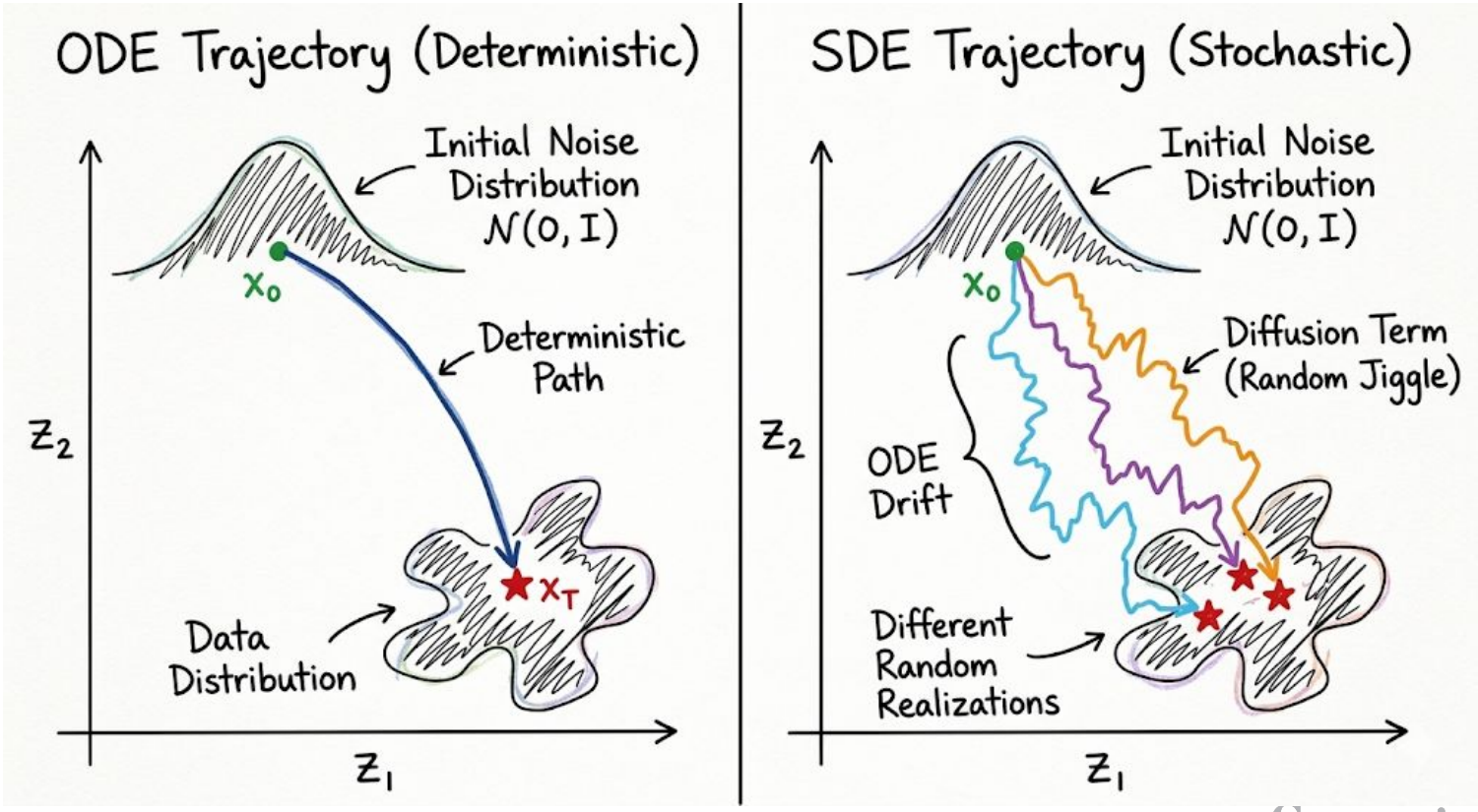
$$\mathrm{d}\mathbf{x}_t = \left[-\frac{1}{2}\beta(t)\mathbf{x}_t - \beta(t)\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)\right]\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\bar{\boldsymbol{\omega}}_t$$

"Score Function"

Slide credit to: https://cvpr2022-tutorial-diffusion-models.github.io/

ODE vs. SDE

Transformations in two ways.



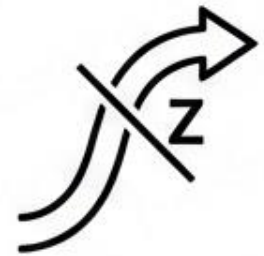
The Score Function: Why It Matters

Traditional Approach:

Learn $p(x)$ directly → requires normalizing constant $\mathbf{Z}$ → Z is intractable for complex data

Score-Based Approach:

Learn $\nabla_x \log p(x)$ instead → no Z needed! → Z cancels in gradient



The Score Function: Why It Matters

Connection to Diffusion:

Predicting noise $\varepsilon \iff$ Estimating score $\nabla_x \log p(x)$

Same objective, different interpretation

Score tells us direction toward high probability

Denoising Score Matching

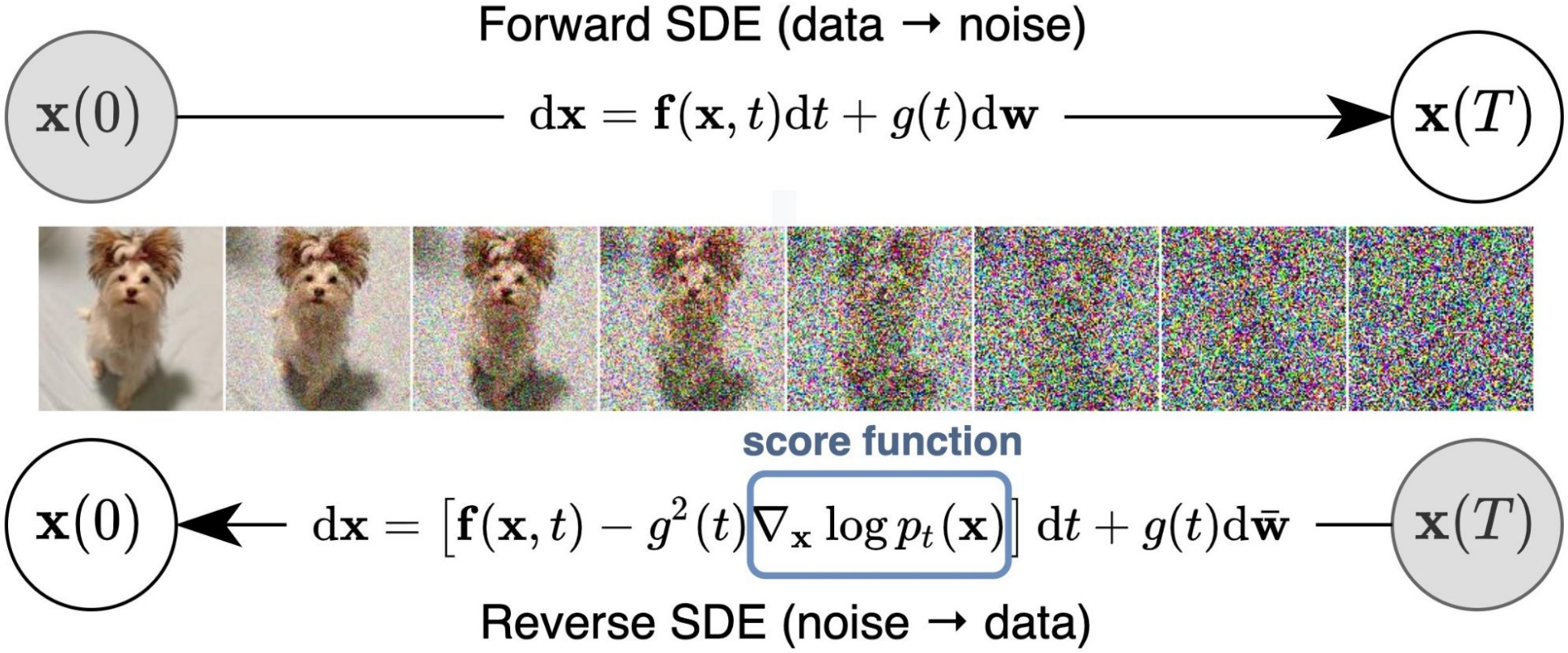
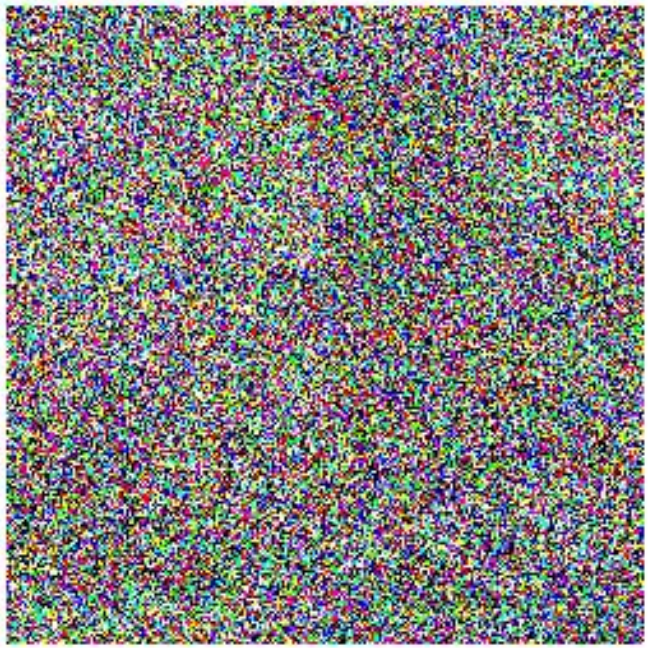


Figure credit to: https://yang-song.net/blog/2021/score/

Generative Reverse SDEs



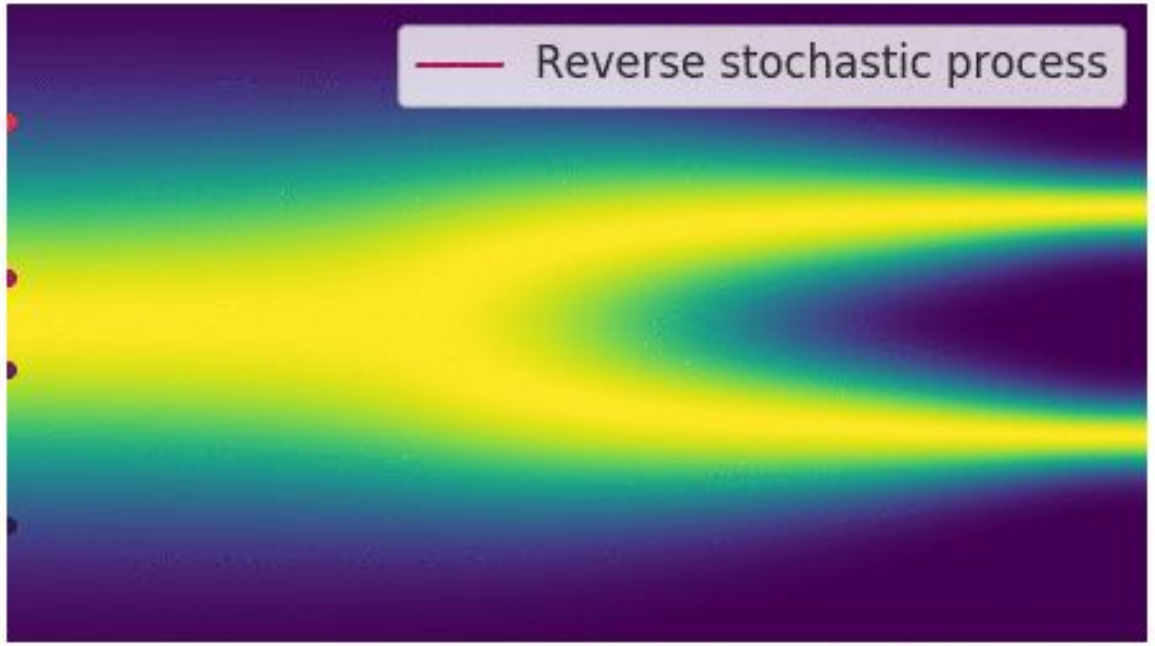


Figure credit to: https://yang -song.net/blog/2021/score/

Poll 4

The score function $\nabla \log p(x)$ points toward:

- A) Random directions
- B) Low probability regions
- C) High probability regions (data)
- D) The origin

Poll 4

The score function $\nabla \log p(x)$ points toward:

- A) Random directions
- B) Low probability regions
- C) High probability regions (data)
- D) The origin

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Conditional Generation: The Control Problem

Unconditional Diffusion:

- Sample: Random noise → Random image from training distribution
- No control over output class/content

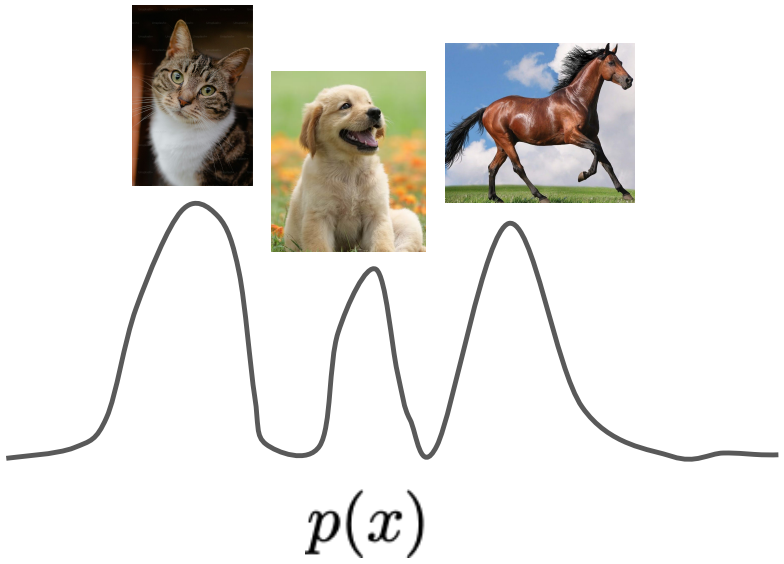
The Need for Control:

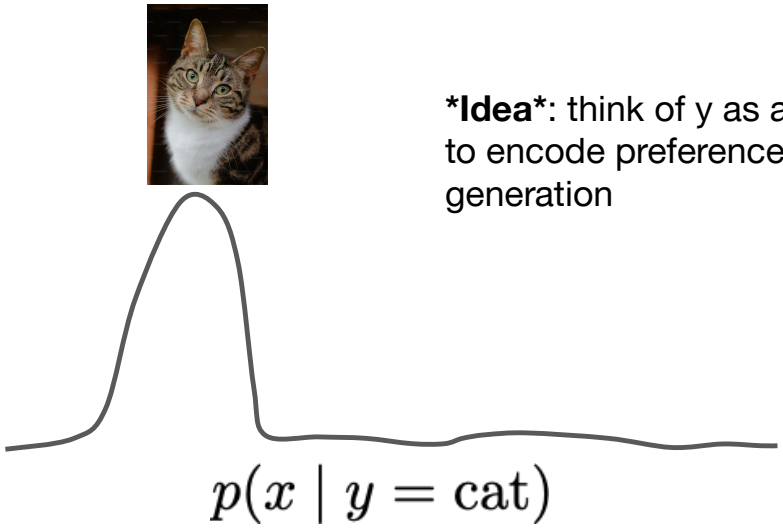
- "Draw a sunset over mountains"
- "Create a portrait in Van Gogh style"

Conditional Diffusion (overview)

Previously: sample from $p(x)$

Today: Sample from $p(x \mid y)$ for some condition y





Idea: think of y as a way to encode preferences in generation

Slide credit to: https://www.practical-diffusion.org/

Conditional Generation: The Control Problem

1. **Classifier Guidance:** Train separate classifier, use gradients
   - ✗ Requires training extra model
   - ✗ Classifier can be noisy at high t
2. **Classifier-Free Guidance (CFG):** Train one model for both
   - ✓ Single model
   - ✓ Better quality

Classifier Guidance (Dhariwal Nicol 2021)

Learn an unconditional score function and a classifier

$$p_t(x_t \mid c) = \frac{p_t(x_t)p_t(c \mid x_t)}{p(c)}$$

$$\implies \nabla_x \log p_t(x_t \mid c) = \underbrace{\nabla_x \log p_t(x_t)}_{\text{score function}} + \nabla_x \log \underbrace{p_t(c \mid x_t)}_{\text{classifier}}$$

Slide credit to: https://www.practical-diffusion.org/

Classifier-Free Guidance (Ho Salimans, 2022)

Same idea, just train one conditional score function instead with an extra class c = ∅

$$\nabla_x \log \tilde{p}_t(x_t \mid c) := \underbrace{\nabla_x \log p_t(x_t)}_{\text{score function}} + \gamma \nabla_x \log \underbrace{p_t(c \mid x_t)}_{\text{classifier}}, \gamma > 1$$

$$= \nabla_x \log p_t(x_t) + \gamma(\nabla_x \log p_t(x_t \mid c) - \nabla_x \log p_t(x_t))$$

$$= (1-\gamma)\nabla_x \log p_t(x_t) + \gamma \nabla \log p_t(x \mid c)$$

$$= (1-\gamma)\nabla_x \log p_t(x_t \mid c = \emptyset) + \gamma \nabla \log p_t(x \mid c)$$

Takeaway: training is identical to training conditional diffusion model, but with a dummy class c = ∅ used during training as well.

Slide credit to: https://www.practical-diffusion.org/

Classifier-free Guidance → sample quality higher

OpenAI's GLIDE model, without guidance (left) and with guidance (right)

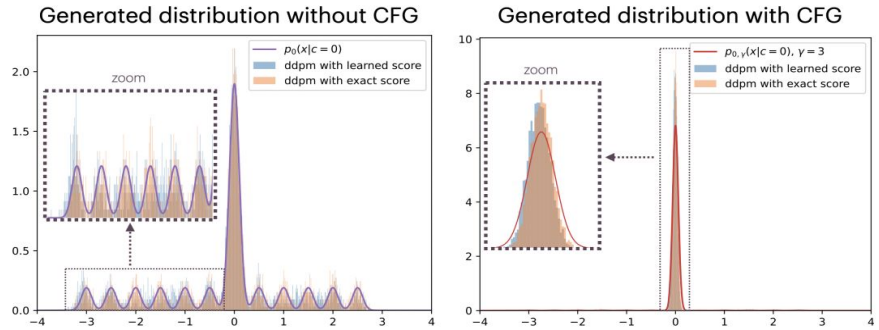
Slide credit to: https://www.practical-diffusion.org/

What is Classifier-free Guidance “doing”?

Theory is still developing but empirically:

1. Increasing guidance γ leads to sharper “low-temperature” sampling (i.e. higher density regions of distribution contain higher perceived quality samples)

2. Learned score functions tend to have more significant error i low density regions, and so this error is less apparent with guidance



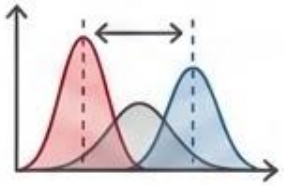
CFG is a predictor corrector, Bradley and Nakkiran, 2024, and Guiding a Diffusion Model with a Bad Version of Itself, Karras et al. 2024

Slide credit to: https://www.practical-diffusion.org/

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Performance Metrics



Fréchet Inception Distance (FID)

Measures the distance between feature vectors of real and generated images; lower scores indicate better image quality and similarity to real images.

Inception Score (IS)

Assesses the diversity and clarity of generated images using a pre-trained model; higher scores denote better image clarity and variety.

Precision and Recall

Evaluates the quality and diversity of generated images; high precision indicates realistic images, while high recall shows variety close to the actual dataset.

Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Latent DDPM

Why Latent Space?

Efficiency: Running diffusion models directly in pixel space is computationally expensive. Latent Diffusion Models (LDMs) operate in a compressed latent space, drastically reducing computational cost and complexity.

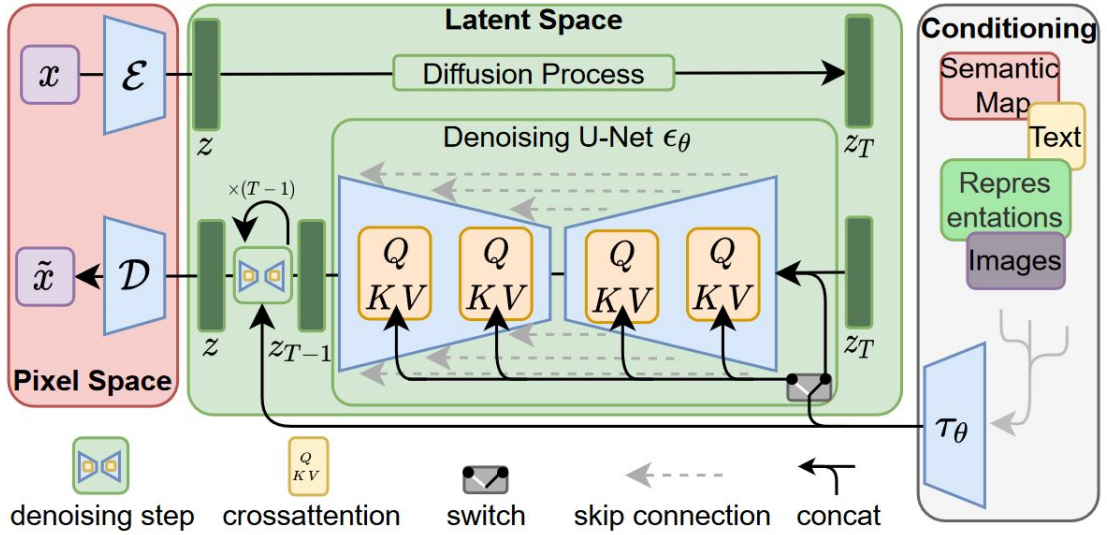
Data Compression: LDMs capture the essential ***features*** of high-dimensional data like images in a more compact and structured form, enhancing processing efficiency.

Improved Performance: Working in latent space focuses the model on **relevant aspects** of the *data*, improving both the efficiency and effectiveness of the generation process.

Latent DDPM

- Map data into latent space using VAE or any other similar approach

1. **Encoder:** Compresses input data into a latent representation.
2. **Decoder:** Reconstructs the original data from the latent representation.



Outline

- Why diffusion? VAEs and the one-step problem
- The diffusion process: forward, reverse, and training
- Toy model: seeing diffusion learn on a 2D spiral
- Sampling: DDPM and the speed problem
- Diffusion Models from Stochastic Differential Equations and Score Matching Perspective
- Conditional generation and classifier-free guidance
- Performance Metrics: FID, IS, Precision and Recall
- Latent diffusion
- Applications of Diffusion Models

Fine-grained control of diffusion models

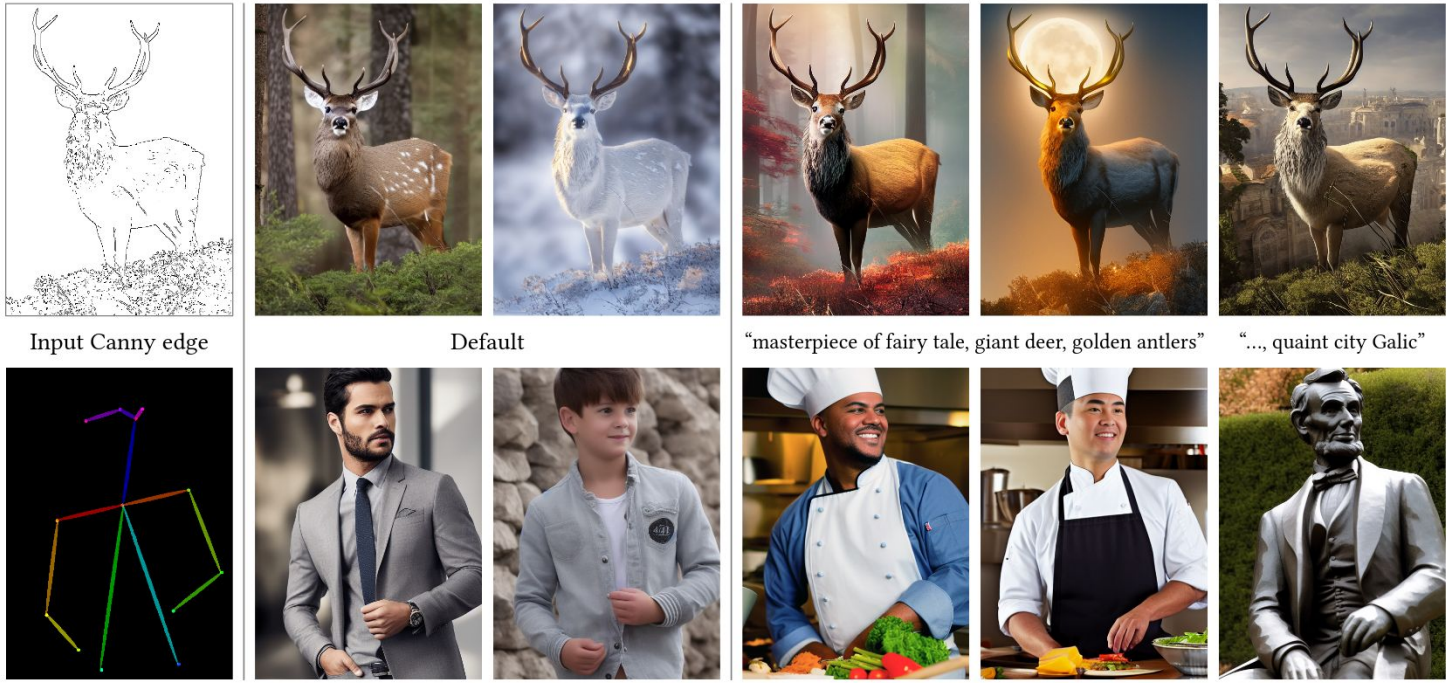


Figure 1: Controlling Stable Diffusion with learned conditions. ControlNet allows users to add conditions like Canny edges (top), human pose (bottom), *etc*., to control the image generation of large pretrained diffusion models. The default results use the prompt "a high-quality, detailed, and professional image". Users can optionally give prompts like the "chef in kitchen".

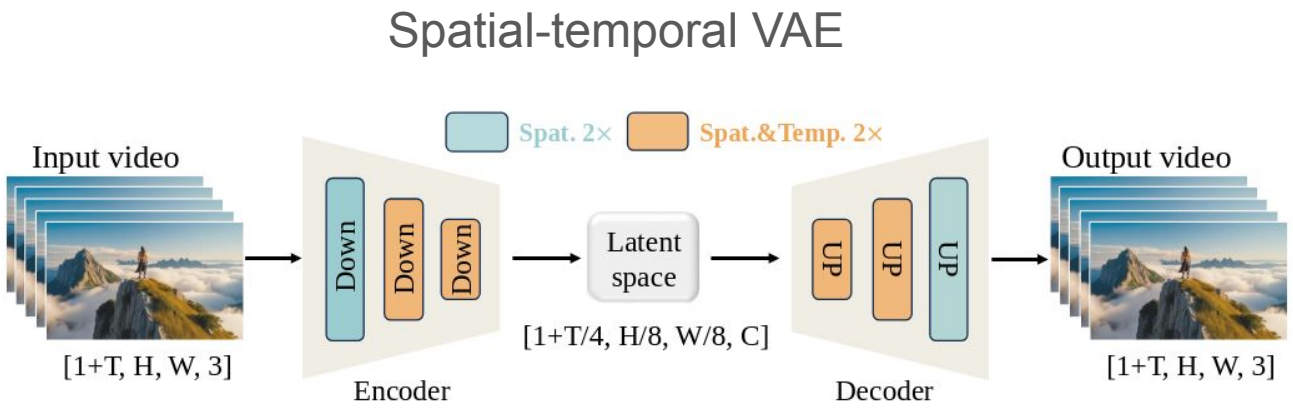
Image credit: Adding Conditional Control to Text-to-Image Diffusion Models https://arxiv.org/pdf/2302.05543

Image editing and composition

Figure 1. With just a few images (typically 3-5) of a subject (left), *DreamBooth*—our AI-powered photo booth—can generate a myriad of images of the subject in different contexts (right), using the guidance of a text prompt. The results exhibit natural interactions with the environment, as well as novel articulations and variation in lighting conditions, all while maintaining high fidelity to the key visual features of the subject.

Image credit: DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation https://arxiv.org/pdf/2208.12242

How does video generation work?



Challenges:

1. Temporal consistency and how to best perceptually compress motion
2. Autoregressive generation of arbitrary-length videos

Image credit: WAN 2.1 https://arxiv.org/pdf/2503.20314

Slide credit to: https://www.practical-diffusion.org/

Generating visual illusions

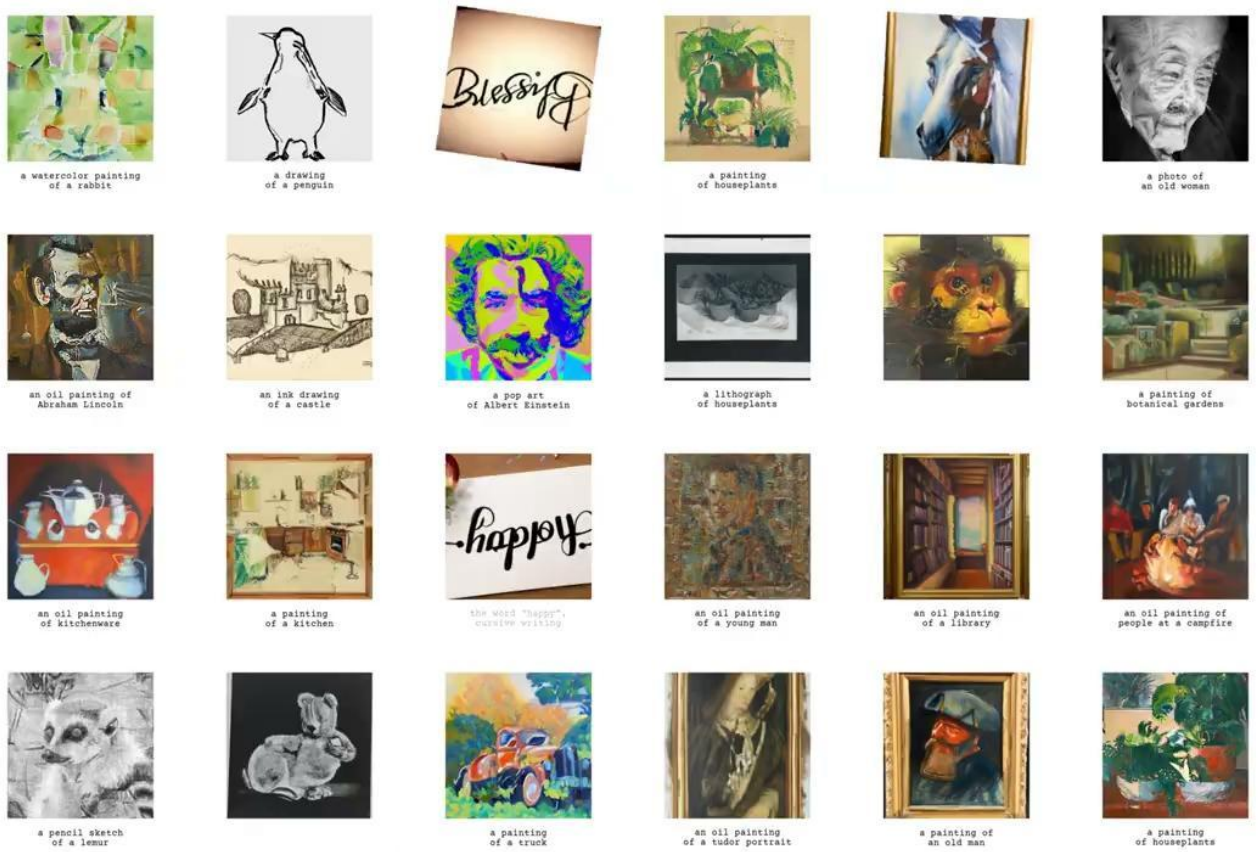


Image credit: Visual anagrams https://dangeng.github.io/visual_anagrams/

Key Takeaways

1. Many Small Steps Beat One Big Jump

VAEs try to generate in one step and fail (blurry images). Diffusion uses 1000 tiny steps instead. Each small change is easy to learn. Result: sharp, high-quality outputs.

2. Forward is Fixed, Reverse is Learning

Forward process: add noise using a fixed equation (no training needed). Reverse process: train neural network to predict the noise. Training objective: MSE loss.

3. Three Views, Same Model

Denoising, score-based, and SDEs are equivalent perspectives. Understanding one helps understand all.

Thank you :-)