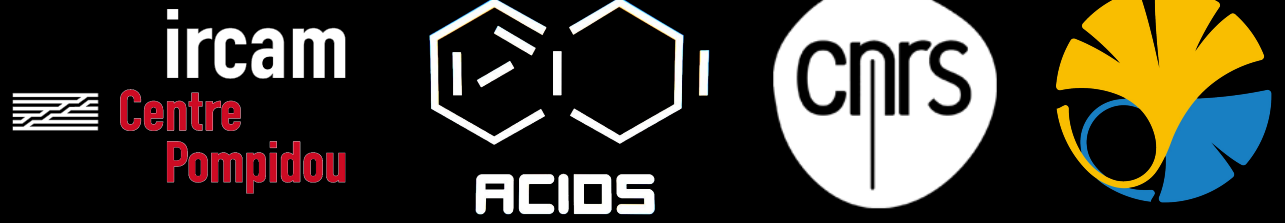


Diffusion models

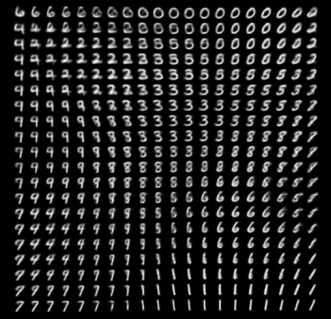
Creative Machine Learning - Course 10

Pr. Philippe Esling
esling@ircam.fr

Brief history of AI

Families of generative models that we will learn

1

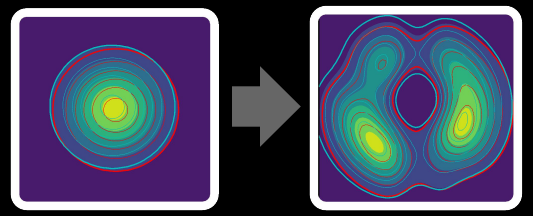


Variational Auto-Encoder (VAE)

Random variables, distributions, independence

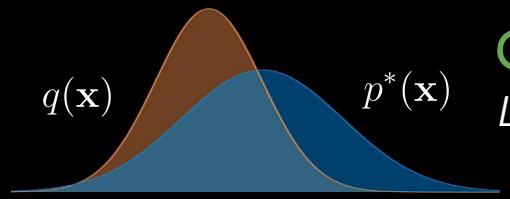
Normalizing flows

Bayes' theorem, likelihood, conjugate priors



2

3



Generative Adversarial Network (GAN)

Latent variables, probability graphs

Diffusion models

Latent variables, probability graphs

4

2015 - *Generative model*

First wave of interest in generating data
Led to current model craze (VAEs, GANs, Diffusion)

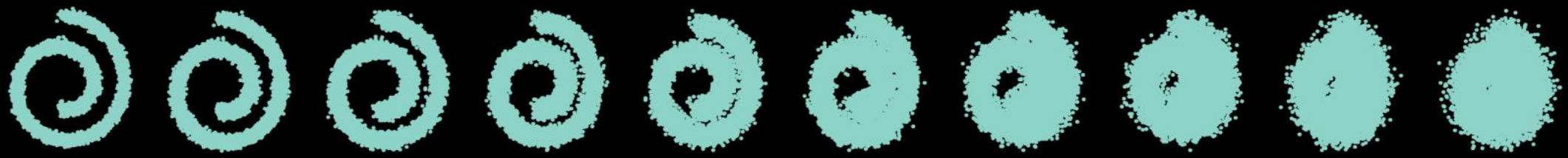
2012 onwards Deep learning era

Introduction

Diffusion models

- Stochastic models inspired by the physical process of diffusion.
- Iteratively add noise to the data and learn to reverse this process.

$\mathbf{x}_0 + \mathcal{N}(0, \sigma)$ ---- Diffusion process ----------> $\mathbf{x}_T$



Data ← Learning ? → Prior

Goal | Learn the conditional (denoising) distribution at each step

- Distribution explains what is the data at the next step given current data
- Implicitly generative learning (generate probable data from Gaussian noise)

Diffusion agenda

Concepts that we will see today

Theory

1. Score matching
2. Langevin dynamics
3. Denoising score matching

Diffusion

1. Formalization of diffusion models
2. Forward process properties
3. Implementing the reverse process
4. Model probability and training loss

Applications

1. Denoising Diffusion Probabilistic Models (DDPM)
2. Applications of diffusion

Score matching

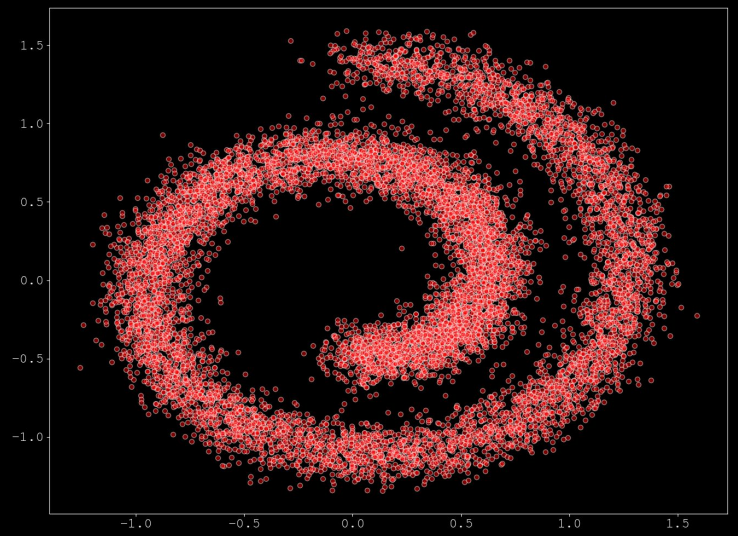
Introducing the **score matching** idea

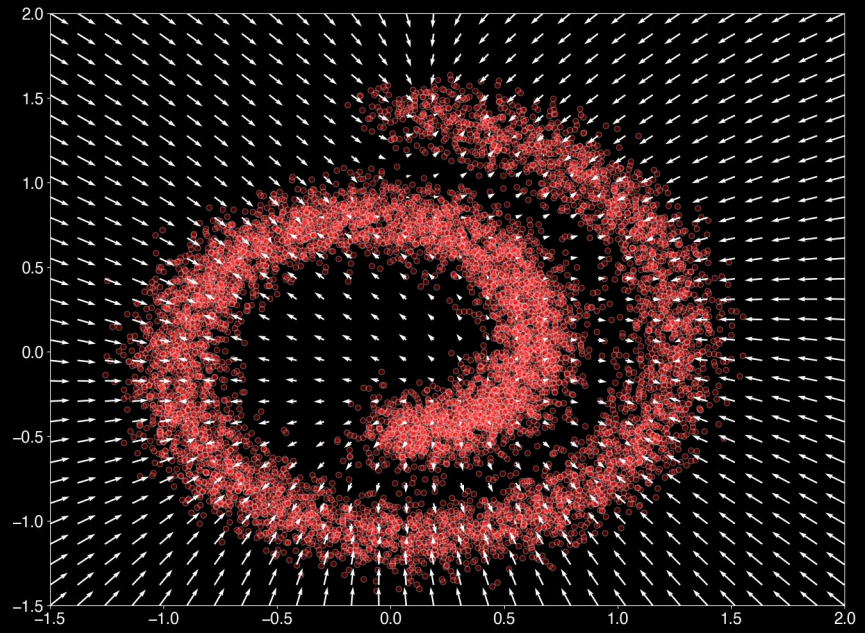
We usually try to learn the distribution $\log p(\mathbf{x})$ ---------- $\mathcal{F}_\theta(\mathbf{x}) \approx \log p(\mathbf{x})$

How about trying to model the **gradients** $\nabla_\mathbf{x} \log p(\mathbf{x})$

Still aim to learn a model $\mathcal{F}_\theta(\mathbf{x})$ but trying to obtain $\mathcal{F}_\theta(\mathbf{x}) \approx \nabla_x \log p(x)$

$\log p(\mathbf{x})$





$\nabla_\mathbf{x} \log p(\mathbf{x})$

Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research, 695-709.

Score matching

Shifting our training objectives

Instead of trying to learn the distribution $\log p(\mathbf{x})$

Try to approximate the **gradients** $\nabla_{\mathbf{x}} \log p(\mathbf{x})$

Still aim to learn a model $\mathcal{F}_\theta(\mathbf{x})$ but trying to obtain $\mathcal{F}_\theta(\mathbf{x}) \approx \nabla_x \log p(x)$

Objective (naive)

$$\mathcal{L}_{mse} = E_{\mathbf{x}\sim p(\mathbf{x})}\left[\left\|\mathcal{F}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\right\|_2^2\right]$$

Can be shown that it is equivalent to optimize

$$\mathcal{L}_{matching} = E_{\mathbf{x}\sim p(\mathbf{x})}\left[\operatorname{tr}\left(\nabla_{\mathbf{x}}\mathcal{F}_\theta(\mathbf{x})\right) + \frac{1}{2}\left\|\mathcal{F}_\theta(\mathbf{x})\right\|_2^2\right]$$

$\operatorname{tr}\left(\nabla_{\mathbf{x}}\mathcal{F}_\theta(\mathbf{x})\right)$ is the *trace of the Jacobian of* $\mathcal{F}_\theta(\mathbf{x})$

$\left\|\mathcal{F}_\theta(\mathbf{x})\right\|_2^2$ is the *norm of the model output*

Hyvärinen, A. (2005). Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research, 695-709.

Langevin dynamics

Issue: how to sample (generate new points)

How can we use $\mathcal{F}_\theta(\mathbf{x}) \approx \nabla_x \log p(x)$ in order to obtain samples $\mathbf{x} \sim p(\mathbf{x})$

Idea #1: Use gradient descent to find maximum probability

$$\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \dashrightarrow \mathbf{x}_{t+1} = \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) \text{ ------ } \mathcal{F}_\theta(\mathbf{x})$$

This just gives us **maximum probability** but **not true samples**

Special case of Langevin dynamics

It is shown that we can rely on Langevin dynamics for sampling

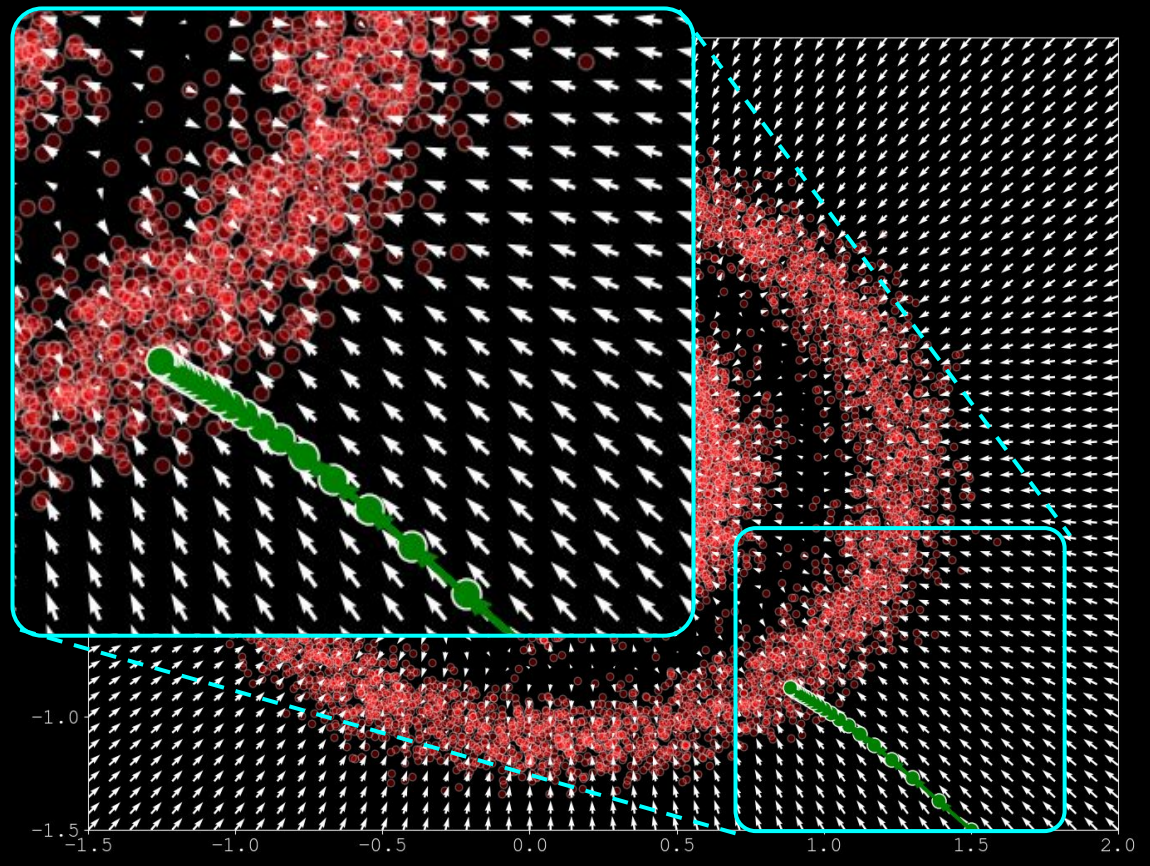
$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\epsilon}{2} \nabla_{\mathbf{x}_t} log p(\mathbf{x}_t) + \sqrt{\epsilon} \mathbf{z}_t$$

Under the conditions $\epsilon \to 0, t \to \inf$ it was shown that $\mathbf{x} \sim p(\mathbf{x})$

Welling, M., & Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of ICML* (pp. 681-688).

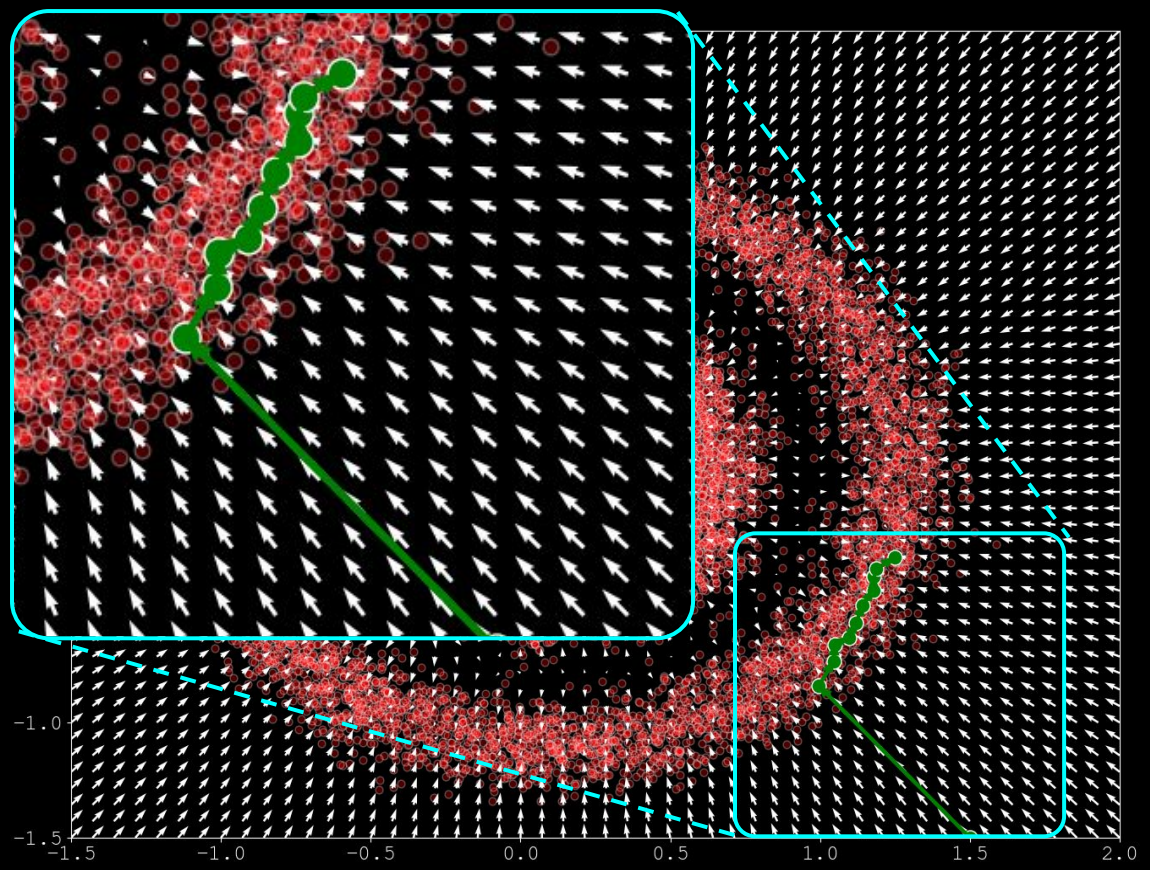
Langevin dynamics

Gradient descent



$$\mathbf{x}_{t+1} = \mathbf{x}_t + \epsilon \nabla_{\mathbf{x}_t} \log\, p(\mathbf{x}_t)$$

Langevin dynamics



$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\epsilon}{2} \nabla_{\mathbf{x}_t} logp(\mathbf{x}_t) + \sqrt{\epsilon}\mathbf{z}_t$$

Sliced score matching

Issue: computing efficiency of score matching

Inside the score matching objective $\mathcal{L}_{matching} = E_{\mathbf{x}\sim p(\mathbf{x})}\left[\operatorname{tr}\left(\nabla_{\mathbf{x}}\mathcal{F}_\theta(\mathbf{x})\right) + \frac{1}{2}\left\|\mathcal{F}_\theta(\mathbf{x})\right\|_2^2\right]$

> The computation of $\operatorname{tr}\left(\nabla_{\mathbf{x}}\mathcal{F}_\theta(\mathbf{x})\right)$ is extremely costly
>
> Notably, computing the Jacobian of $\mathcal{F}_\theta(\mathbf{x})$ is O(N2+N)

Improved loss with **sliced** score matching

Using *random projections* to approximate $\operatorname{tr}\left(\nabla_{\mathbf{x}}\mathcal{F}_\theta(\mathbf{x})\right)$

$$E_{\mathbf{v}\sim\mathcal{N}(0,1)}E_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbf{v}^T\nabla_{\mathbf{x}}\mathcal{F}_\theta(\mathbf{x})\mathbf{v} + \frac{1}{2}\left\|\mathbf{v}^T\mathcal{F}_\theta(\mathbf{x})\right\|_2^2\right]$$

where $\mathbf{v} \sim \mathcal{N}(0,1)$ is a set of random vectors

Largely reduces the computational cost of score matching

Song, Y., Garg, S., Shi, J., & Ermon, S. (2020, August). Sliced score matching: A scalable approach to density and score estimation. In Uncertainty in AI, PMLR (pp. 574-584)..

Denoising score matching

A completely different view on score matching

Goal | First discussed in the context of *denoising auto-encoders*

Allows to completely remove the use of $\nabla_{\mathbf{x}} \mathcal{F}_\theta(\mathbf{x})$

General idea

Corrupt the input $\mathbf{x}$ with noise $\tilde{\mathbf{x}} = \mathbf{x} + \mathcal{N}(0, \sigma)$

Defines a new distribution $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$

It was shown that obtaining $\mathcal{F}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

Can be found by minimizing the following objective

$$E_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} E_{\mathbf{x} \sim p(\mathbf{x})} \left[\| \mathcal{F}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \|_2^2 \right]$$

Vincent, P. (2011). A connection between score matching and denoising autoencoders. Neural computation, 23(7), 1661-1674.

Denoising score matching

Important recent discovery

Note that in order to have $\mathcal{F}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log q_\sigma(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

We have to consider that the **noise is small enough** so $q_\sigma(\mathbf{x}) \approx p(\mathbf{x})$

If we choose the noise distribution to be $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}} \mid \mathbf{x}, \sigma^2 \mathbf{I})$

Resulting gradient

$$\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = -\frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2}$$

Final simplified training objective

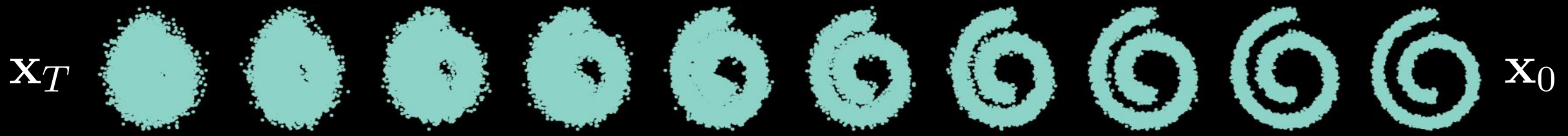
$$\mathcal{L}(\theta; \sigma) = E_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} E_{\mathbf{x} \sim p(\mathbf{x})} \left[\left\| \mathcal{F}_\theta(\tilde{\mathbf{x}}) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \right\|_2^2 \right]$$

Song, Y., & Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. In NeurIPS 2019 (pp. 11918-11930).

Diffusion models

Model structure

Forward (diffusion) process $q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)$



Reverse (learned) process $p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)$

Overall idea

Based on our input data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ define series of variables $\mathbf{x}_1, \cdots, \mathbf{x}_T$

Variables are obtained by gradually adding small amounts of noise

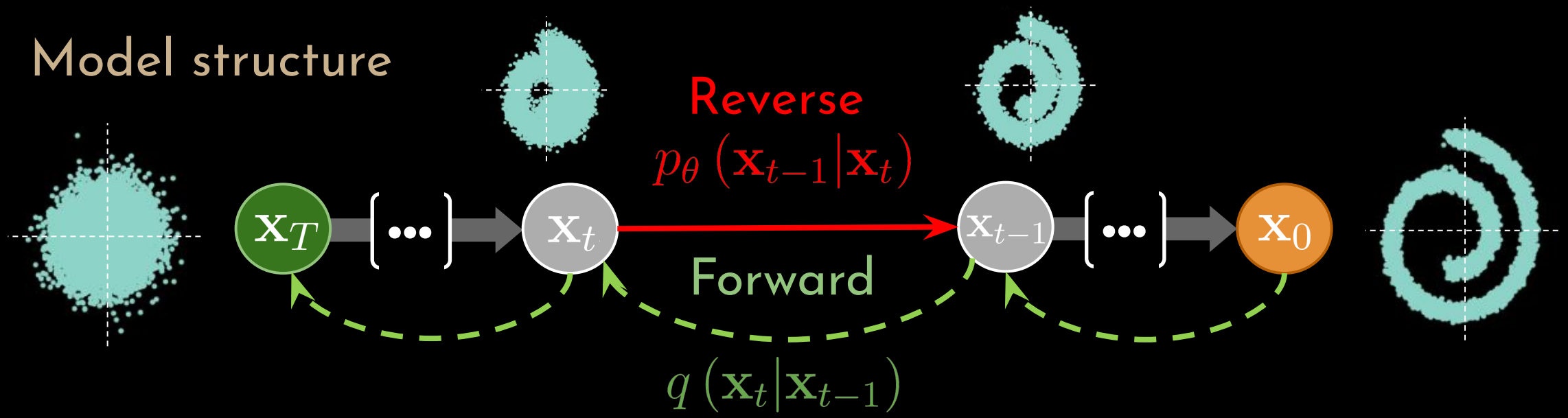
This defines a (known) **forward diffusion process** $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$

Goal is to **learn the reverse (denoising) process** $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

Sohl-Dickstein, J., et al. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. arXiv preprint arXiv:1503.03585.

Diffusion models

Model structure

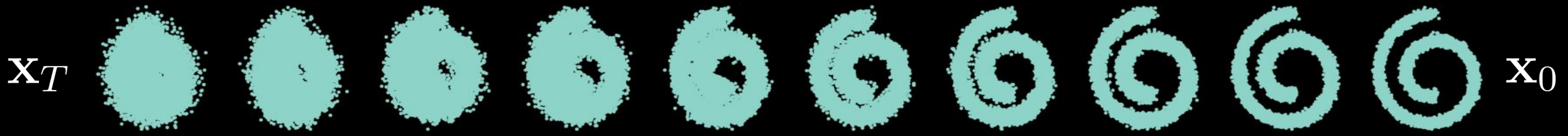


Decomposing the model

1. **Forward diffusion** process $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$
2. **Reverse (learned)** process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$
3. Model probability and sampling
4. Training losses and theory

Forward process

Forward (diffusion) process $q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)$



Goal of the forward process

Transform data distribution $q(\mathbf{x}_0)$ into tractable (noise) distribution $\pi(\mathbf{y})$

Perform by *repeated application* of **Markov diffusion kernel** $T_\pi(\mathbf{y} \mid \mathbf{y}'; \beta)$

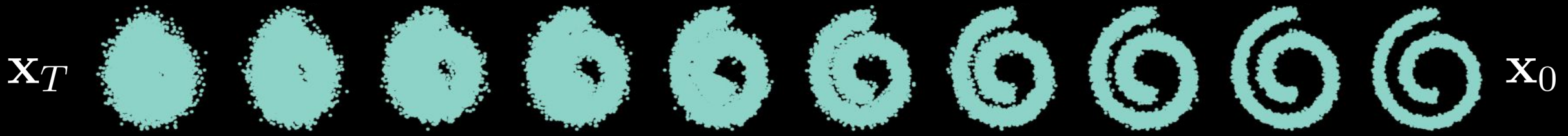
Transition probability $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = T_\pi(\mathbf{x}_t \mid \mathbf{x}_{t-1}; \beta_t)$

Use a **Gaussian kernel**, with given **diffusion rates (schedule)** $\beta_1, \cdots, \beta_T$

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

Forward process

Forward (diffusion) process $q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)$



Based on our Gaussian diffusion kernel $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

Diffusion process

$$q(\mathbf{x}_{0:T}) = q(\mathbf{x}_0)\prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

Arbitrary sampling of forward process

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

using notations :

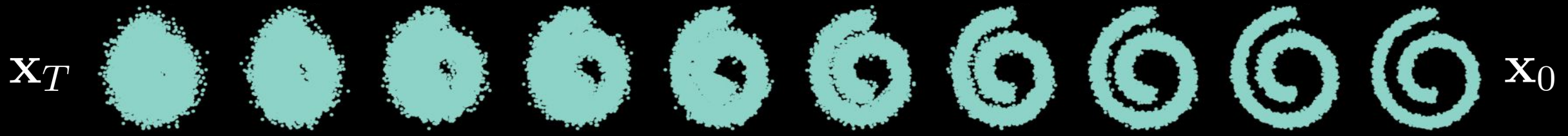
$$\epsilon_i = \mathcal{N}_t(\mathbf{0};\mathbf{I})$$

$$\bar{\alpha}_t = \prod_{s=1} \alpha_s$$

$$\alpha_t = 1 - \beta_t$$

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

Reverse process



Reverse (learned) process $p_\theta\left(\mathbf{x}_{t-1}|\mathbf{x}_t\right)$ ⟶

Starting from our tractable distribution $p(\mathbf{x}_T) = \pi(\mathbf{x}_T)$

Reverse process $p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

We can break down each transition as conditional Gaussians

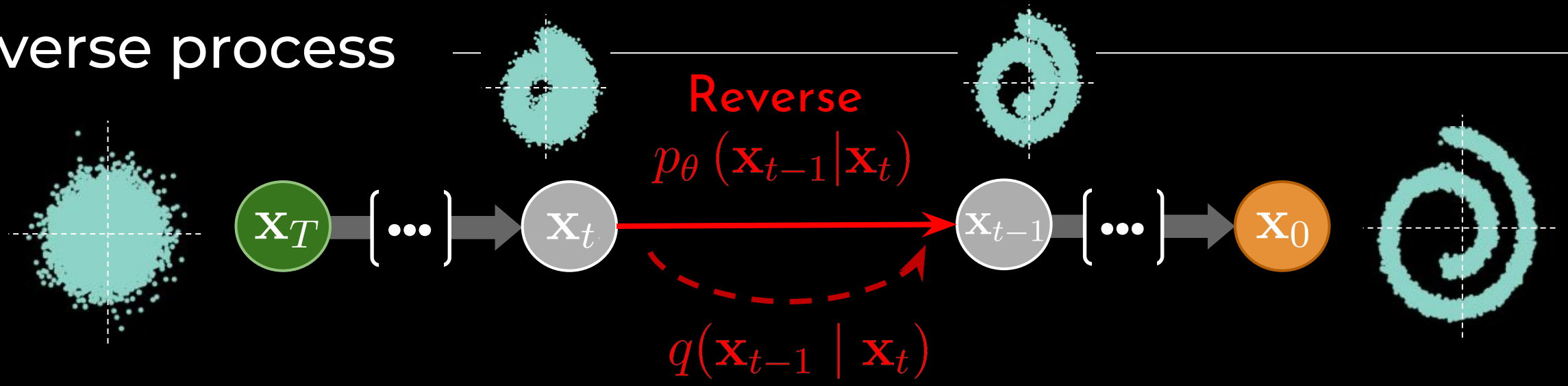
$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$$

Therefore, training only needs to learn two functions

Mean $\mu_\theta(\mathbf{x}_t, t)$ **Covariance** $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t)$

These functions will be **parametrized by deep networks**

Reverse process



Again, we can't easily estimate the true reverse process $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$

But it is tractable when conditioned on $\mathbf{x}_0$:

Using Bayes : $q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right)=q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0\right) \frac{q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_0\right)}{q\left(\mathbf{x}_t \mid \mathbf{x}_0\right)}$

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

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

Training losses

Model probability

The complete (reverse) model probability is $p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$

As usual, this integral is **intractable**

Meeting back the variational trick

We can use the same trick as the development of variational inference

$$p_\theta(\mathbf{x}_0) = \int q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} d\mathbf{x}_{1:T} \longrightarrow \mathcal{L} = \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right]$$

Using the same array of tools (Jensen inequality), we can obtain

$$K = -\mathbb{E}_q [D_{KL}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)) + H_q(\mathbf{X}_T | \mathbf{X}_0) - H_q(\mathbf{X}_1 | \mathbf{X}_0) - H_p(\mathbf{X}_T)]$$

Sohl-Dickstein, J., et al. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. arXiv preprint arXiv:1503.03585.

Modern evolution

Properties of the original formulation

- Very interesting (and different) take on generative modeling
- Versatile *density estimation* (no form) limitations

Issues

- Slow inference (Markov chain denoising)
- Low quality of results on complex data
- Rather complex model and loss formulation

Deep Denoising Probabilistic Models (DDPM)

1. Proposing more efficient *forward process* parametrization
2. Further improvements through *variance reduction* in loss
3. Simplifying the loss to *denoising score matching*

Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. arXiv preprint arXiv:2006.11239.

DDPM - Parametrization

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

Or we have $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t$

Hence we can rewrite $\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right)$

Novel **parametrization** for the *mean*

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

Now the model is trained at outputting directly a *noise function* $\epsilon_\theta(\mathbf{x}_t, t)$ *which approximates the noise* ϵ_t

Introduce **fixed variance function**

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$$

DDPM - Variance reduction

Rewriting loss as sum of KL

$$\mathcal{L} = \mathbb{E}_q \left[\mathcal{L}_T + \sum_{t>1} \mathcal{L}_{t-1} + \mathcal{L}_0 \right]$$

$$\mathcal{L}_0 = -\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)$$ Data

$$\mathcal{L}_T = D_{KL}(q(\mathbf{x}_T \mid \mathbf{x}_0) \| p(\mathbf{x}_T))$$ Prior

$$\mathcal{L}_{t-1} = D_{KL}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t))$$

Reverse process steps

Further simplifying to denoising score matching

Based on the proposed parametrization, this further simplifies into

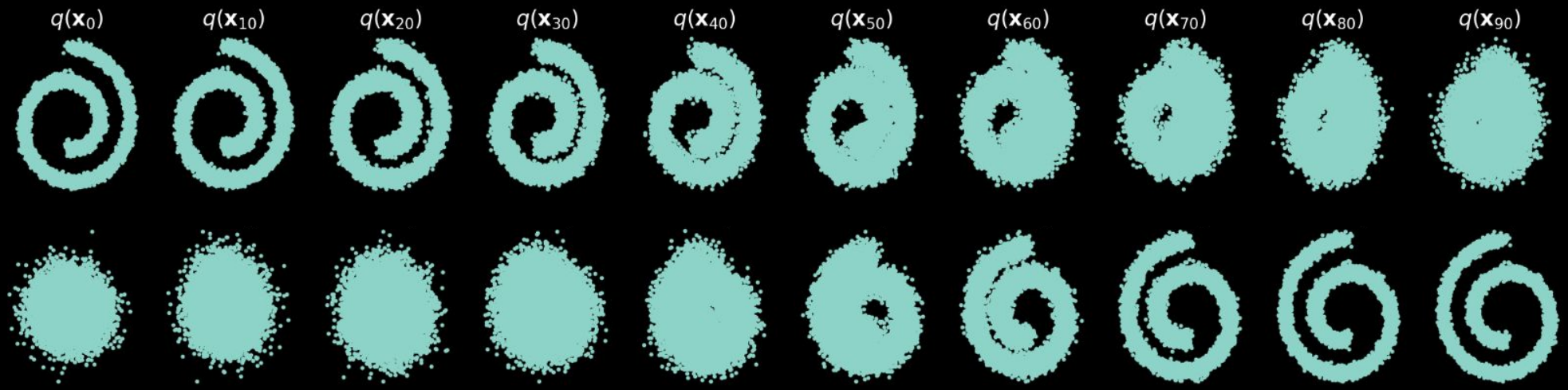
$$\mathcal{L}_{t-1} - C = \mathbb{E}_{\mathbf{x}_0,\epsilon} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} \| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) \|^2 \right]$$

Leads to an extremely simple training objective (*denoising score matching*)

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

Diffusion process

Results of trained models



Results for image generation

Flashback on denoising score matching

Denoising score matching

Approximate the score by predicting the noise of corrupted inputs

$$\mathcal{L}(\theta;\sigma) = E_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})} E_{\mathbf{x}\sim p(\mathbf{x})} \left[\left\| \mathcal{F}_\theta(\tilde{\mathbf{x}}) + \frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2} \right\|_2^2 \right]$$

Link with diffusion

For a gaussian $\mathbf{x} \sim \mathcal{N}(\mu, \sigma^2 \mathbf{I})$, we have $\nabla_{\mathbf{x}} \log p(\mathbf{x}_t) = -\frac{\mathbf{x} - \mu}{\sigma}$

Remembering that $q(\mathbf{x}_t \mid \mathbf{x}_0) \sim \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}\right)$

$$\mathbf{s}_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t) = \mathbb{E}_{q(\mathbf{x}_0)} \left[\nabla_{\mathbf{x}_t} q(\mathbf{x}_t \mid \mathbf{x}_0) \right] = \mathbb{E}_{q(\mathbf{x}_0)} \left[-\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1 - \bar{\alpha}_t}} \right] = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sqrt{1 - \bar{\alpha}_t}}$$

Song, Y., & Ermon, S. (2019). Generative modeling by estimating gradients of the data distribution. In NeurIPS 2019 (pp. 11918-11930).

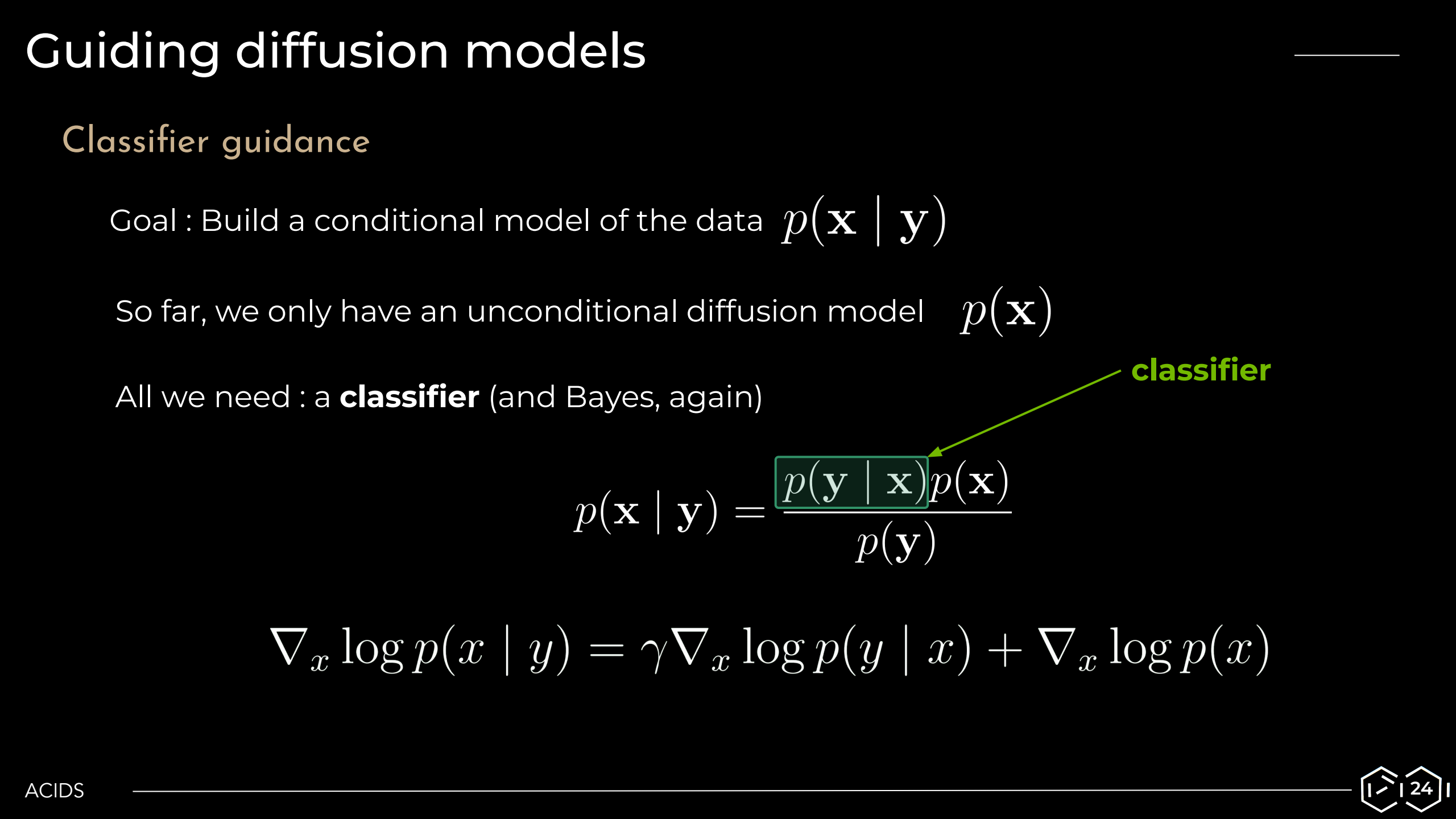
Guiding diffusion models

Classifier guidance

Goal : Build a conditional model of the data $p(\mathbf{x} \mid \mathbf{y})$

So far, we only have an unconditional diffusion model $p(\mathbf{x})$

All we need : a **classifier** (and Bayes, again)



$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}$$

$$\nabla_x \log p(x \mid y) = \gamma \nabla_x \log p(y \mid x) + \nabla_x \log p(x)$$

Guiding diffusion models

Classifier-free guidance

Issues

- Requires an external classifier
- The classifier must be noise-robust because it is applied to noisy versions of the the input

Solution

Train the diffusion model as both a conditional and unconditional generative model by applying drop out to the conditioning

$$p(\mathbf{x})$$
$$p(\mathbf{x} \mid \mathbf{y})$$

Using Bayes :

$$\nabla_x \log p(y \mid x) = \nabla_x \log p(x \mid y) - \nabla_x \log p(x)$$

Guiding diffusion models

Classifier-free guidance

$$\begin{aligned}\nabla_x \log p(x \mid y) &= \gamma \nabla_x \log p(y \mid x) + \nabla_x \log p(x) \\ &= (1-\gamma)\nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y)\end{aligned}$$

$\gamma = 0$ Unconditional

$\gamma = 1$ Conditional

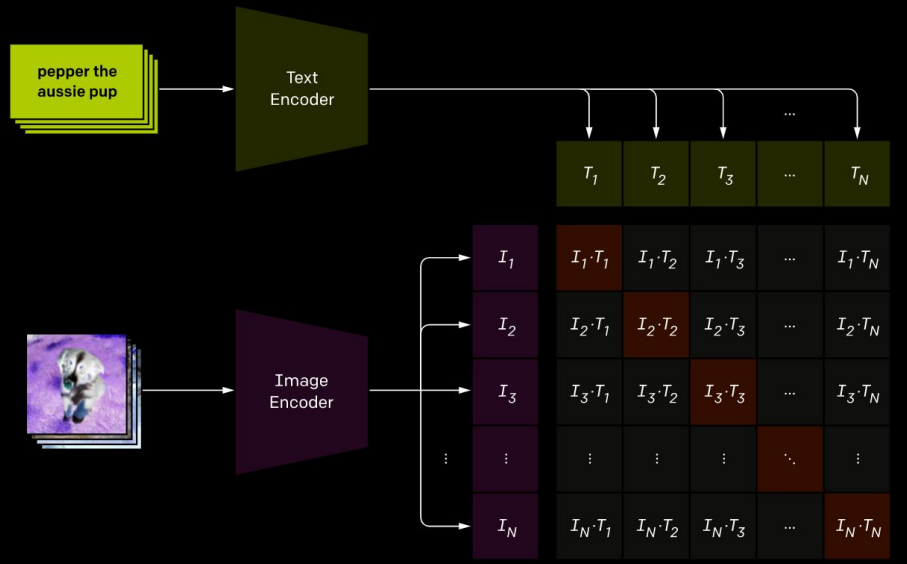
$\gamma > 1$ Magic happens

- Performs better than classifier guidance
- Specifically with strong factor
- No post-hoc conditioning

Extended applications

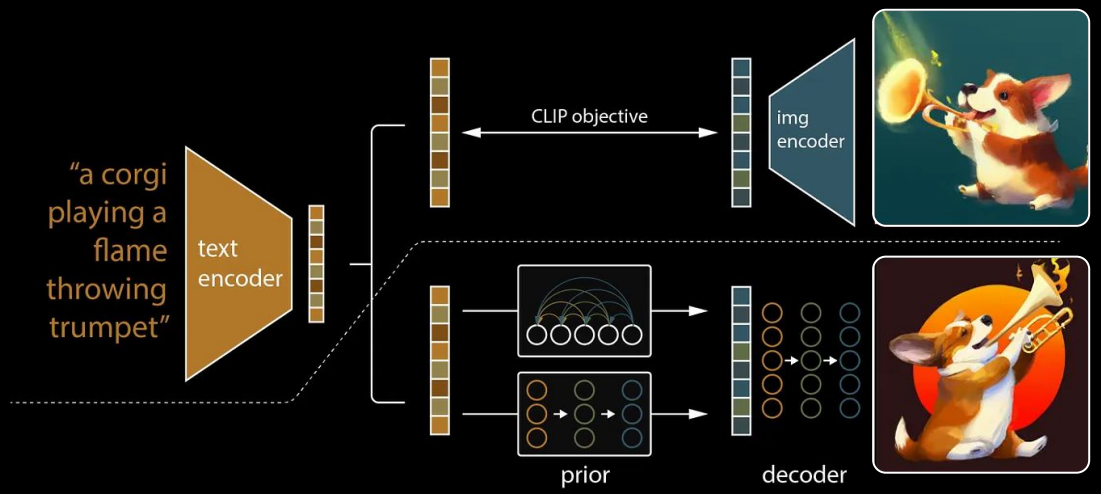
CLIP Multimodality

Learns text-to-image association



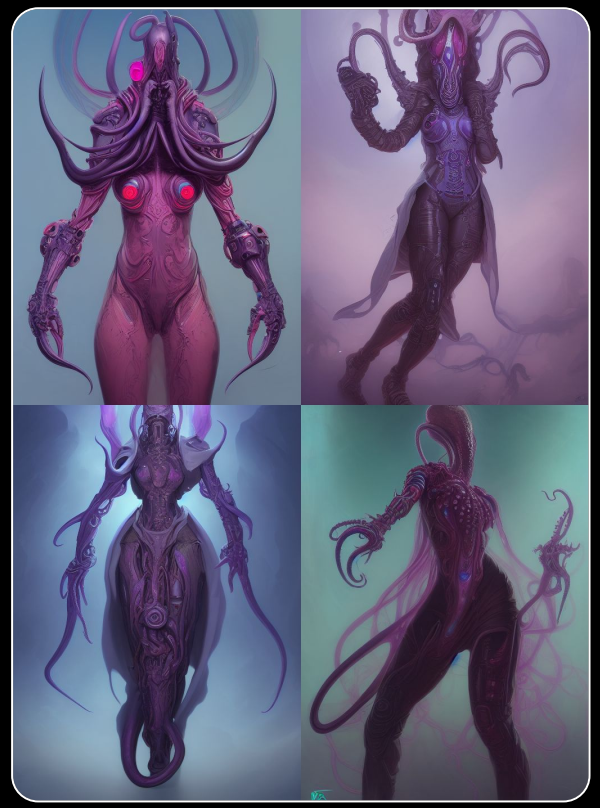
Generation

CLIP Conditionning

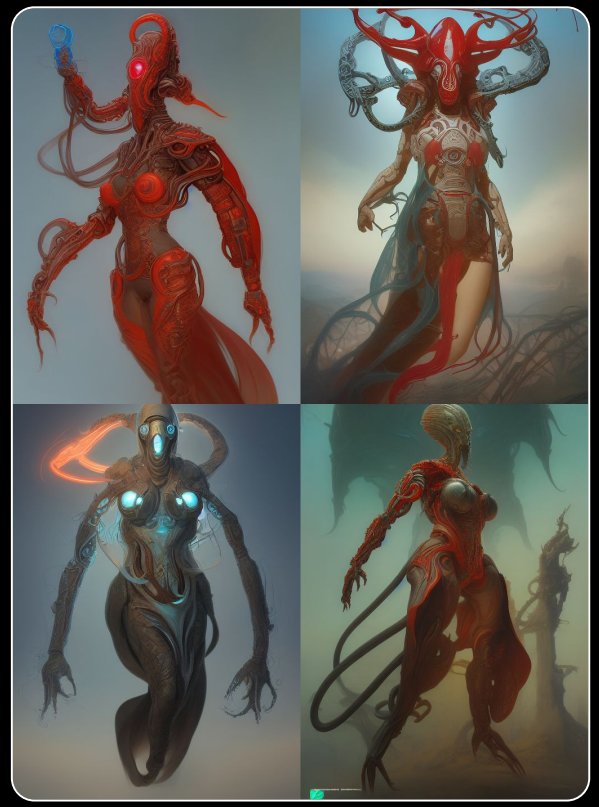


Extended applications

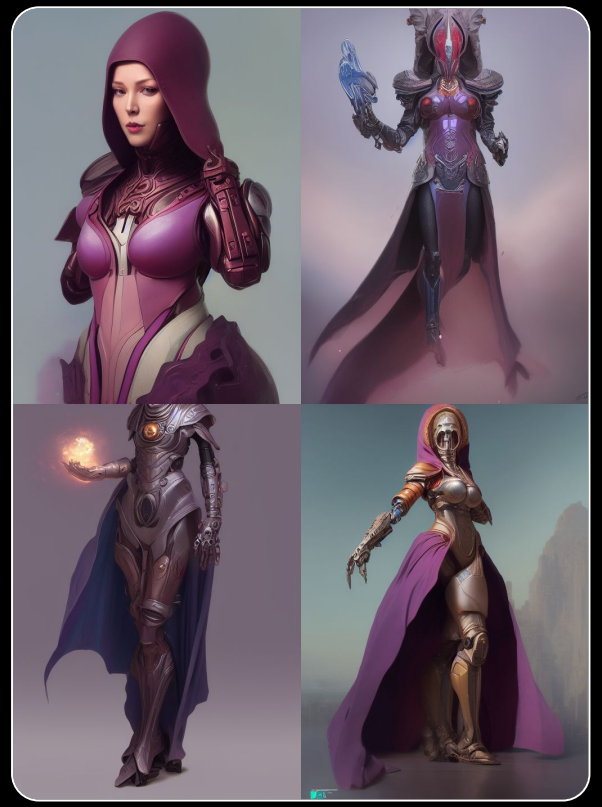
Negative prompts addition to image generation



Original



– Purple



– Tentacles

https://github.com/AUTOMATIC1111/stable-diffusion-webui-feature-showcase

Extended applications

Outpainting

Inpainting

https://github.com/AUTOMATIC1111/stable-diffusion-webui-feature-showcase

Extended applications

Sketching and styling

Rant and shout-out to being open-source

- All of these fantastic extensions come from the community
- Unique advantage of being open-source
  - Bug finding, cross-platform testing, extensions coding
  - Idea sharing, free pedagogy
- Keep all of these in mind for your next project

https://github.com/AUTOMATIC1111/stable-diffusion-webui-feature-showcase

Research directions

Improving theory

Theoretical understanding is still not complete.
Provide a more thorough theoretical analysis of these models, s
Understanding conditions under which we capture the distribution
Convergence properties of the learning algorithm.

Model efficiency

Main challenges is the computational cost of sampling
Reduce this cost with more efficient sampling or reducing number of steps.
Also target reducing the training computational cost

Complex datatypes

Extends to other types of data, such as audio, video, and 3D data.
Analyze the interactions of cross-modalities data

Model quality

Improve the quality of samples through better architectures
Improve the training methods and convergence

Beyond generation

Tasks as denoising, inpainting, super-resolution, and anomaly detection.
Explore these and other potential applications of diffusion models.