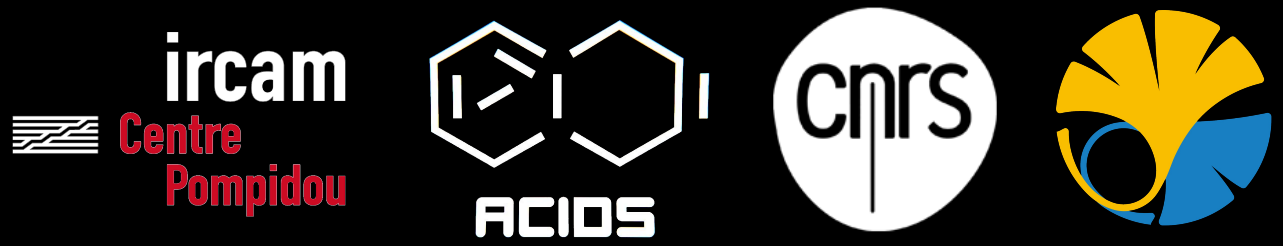


Diffusion models

Creative Machine Learning - Course 10

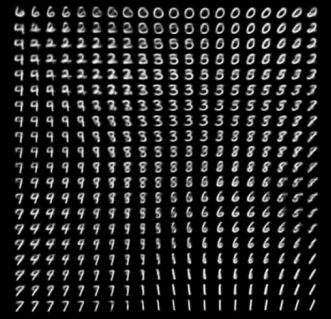
Pr. Philippe Esling esling@ircam.fr

Nils Demerlé demerle@ircam.fr

Brief history of AI

Families of generative models that we will learn

1

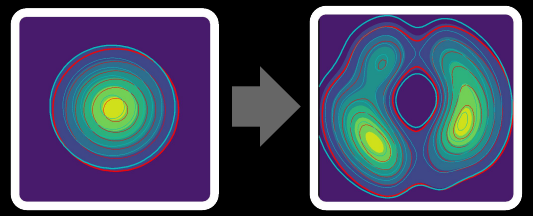


Variational Auto-Encoder (VAE)

Random variables, distributions, independence

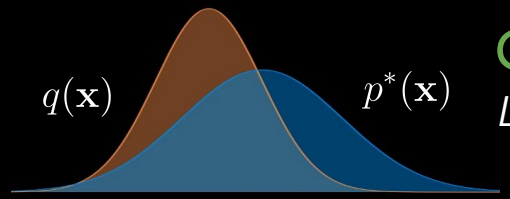
Normalizing flows

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)

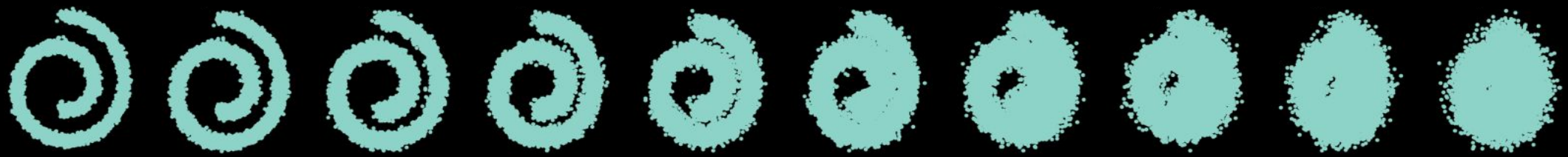
2012 onwards Deep learning era

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. Continuous time diffusion models with SDE
3. The simpler EDM formulation
4. Training

Applications

1. Classifier free guidance for conditioning
2. Applications of diffusion

Score matching

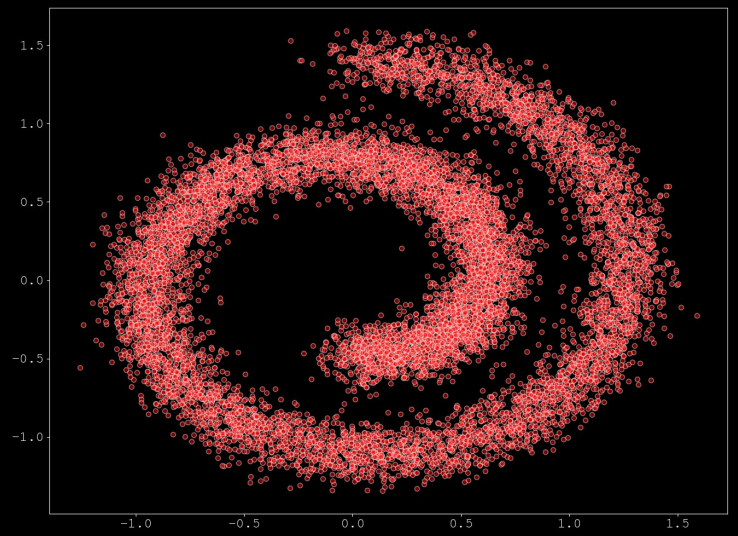
Introducing the **score matching** idea

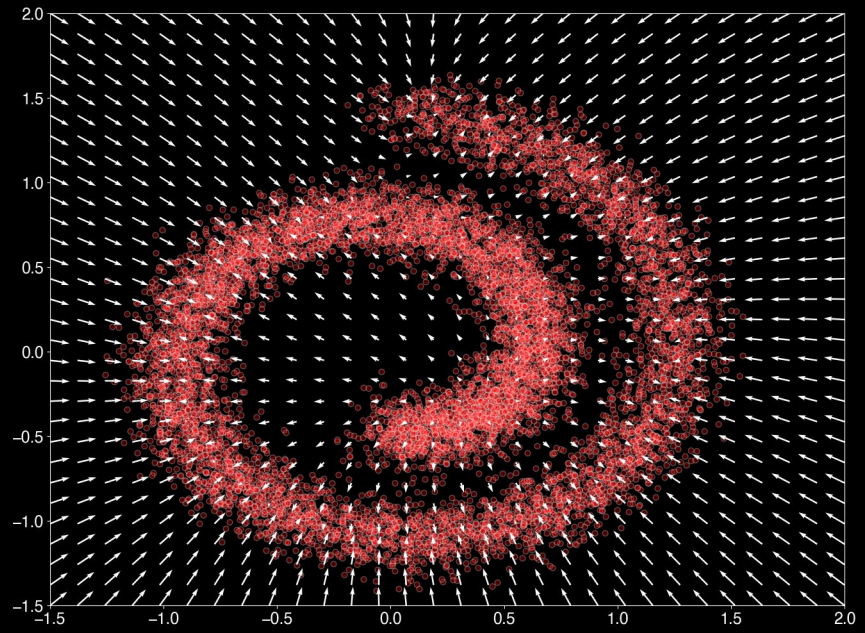
We usually try to learn the distribution $\log p(\mathbf{x})$ ---------- $\mathcal{F}_\theta(\mathbf{x}) \approx \log p(\mathbf{x})$

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.

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).

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

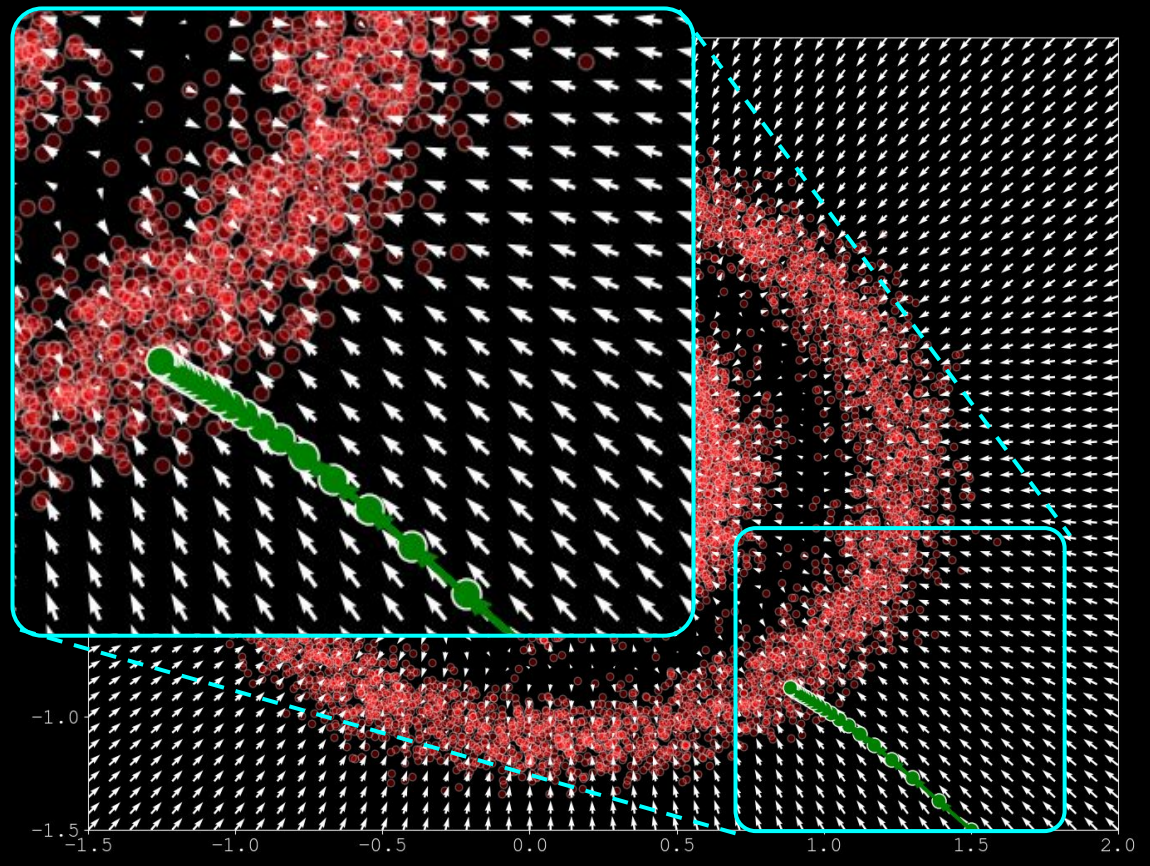
$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\epsilon}{2} \nabla_{\mathbf{x}_t} log p(\mathbf{x}_t) + \sqrt{\epsilon} \mathbf{z}_t$$

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).

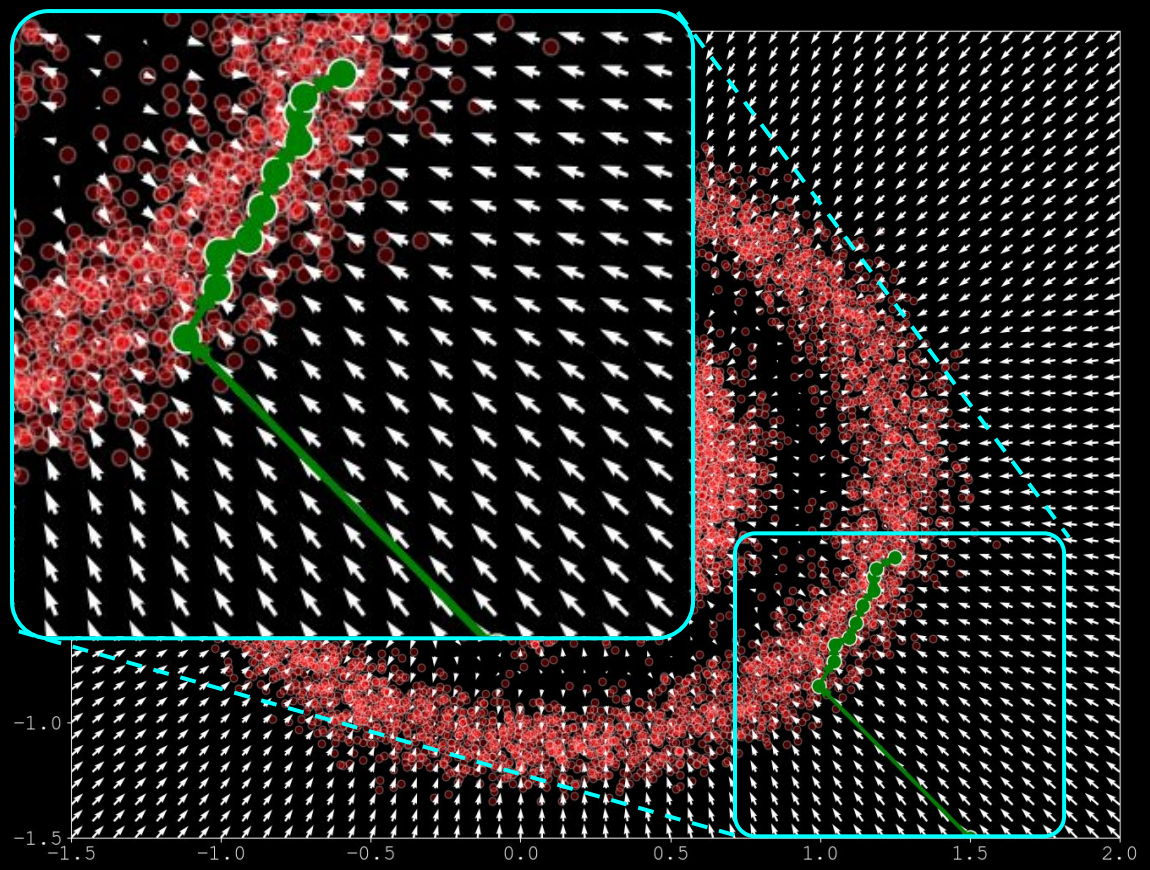
Langevin dynamics

Gradient descent



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

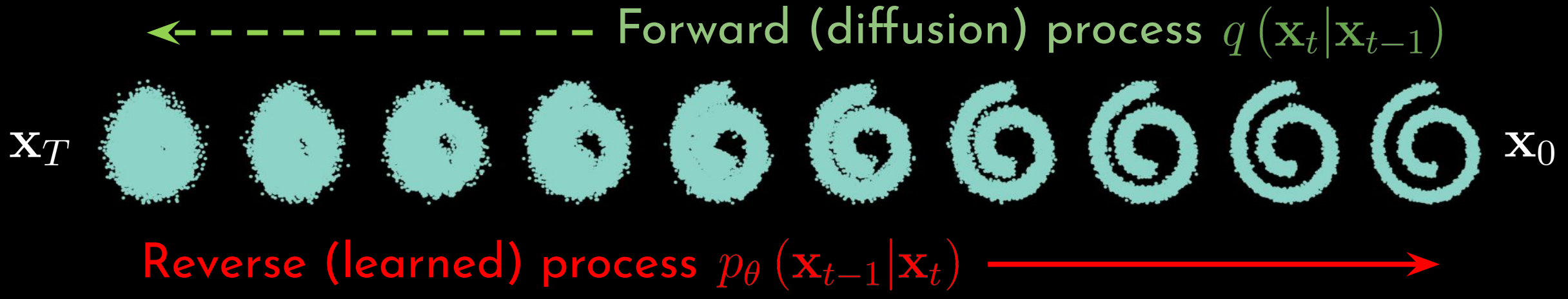
Langevin dynamics



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

Diffusion models

Model structure



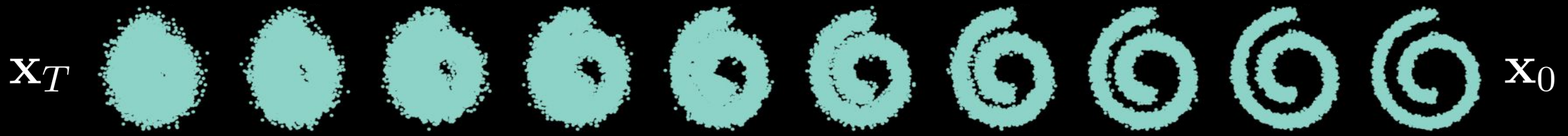
1. **Discrete time diffusion models**
2. **Continuous time diffusion with SDEs and density score**
3. **Simplified formulation : EDM**
4. **Training losses and implementation**

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

Discrete time 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.

Discrete time diffusion models

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

$\mathbf{x}_T$  $\mathbf{x}_0$

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)** $\beta_1, \cdots, \beta_T$

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

in practice $\mathbf{x_t} = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon$ with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Discrete time diffusion models

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

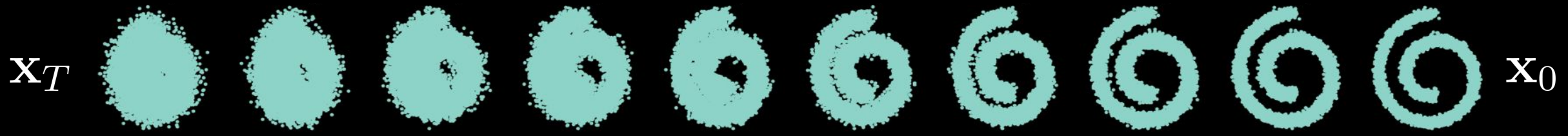
$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$$

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**

Discrete time diffusion models



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

Mean $\mu_\theta(\mathbf{x}_t, t)$ and Covariance $\Sigma_\theta(\mathbf{x}_t, t)$ **are trained variational inference tricks**

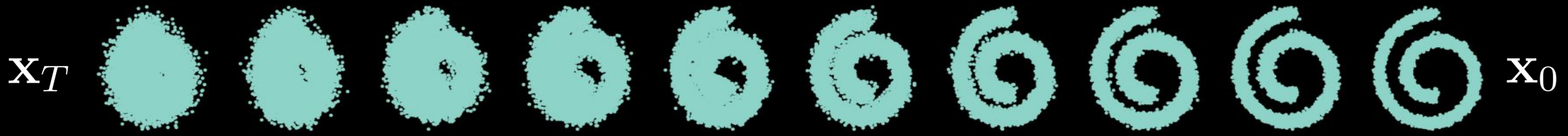
Issues

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

Multiple approaches have been proposed to simplify the objective and improve the training dynamics.

Here, we will focus on continuous time diffusion formulation that explicitly rely on the score of the distribution, estimated through denoising score matching.

Continuous time diffusion



Diffusion process $x(t), t \in [0, T]$ where t is a continuous time variable

Built such that $x(0) \sim p_{data}(\mathbf{x})$ and $x(T) \sim \pi(\mathbf{x})$

General model

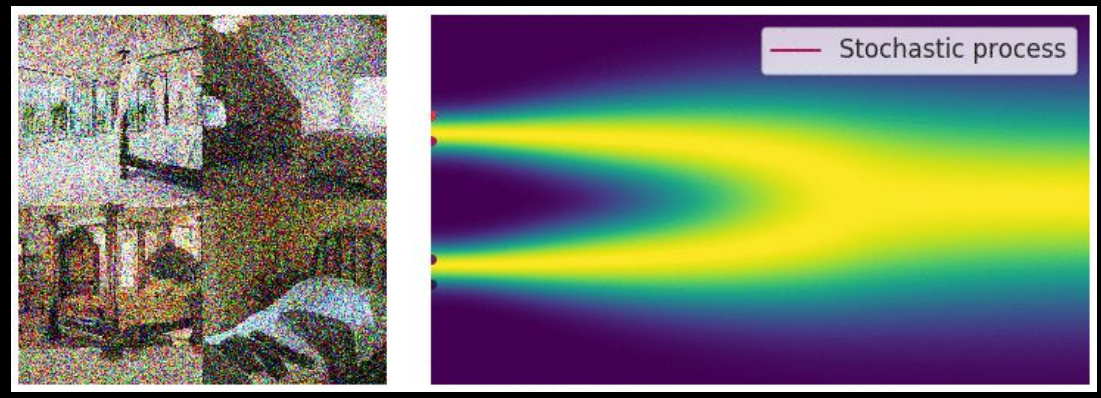
$$d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)d\mathbf{w}$$

drift coefficient

diffusion coefficient

Solution : collection of random variables $\{\mathbf{x}(t)\}_{t\in[0,T]}$

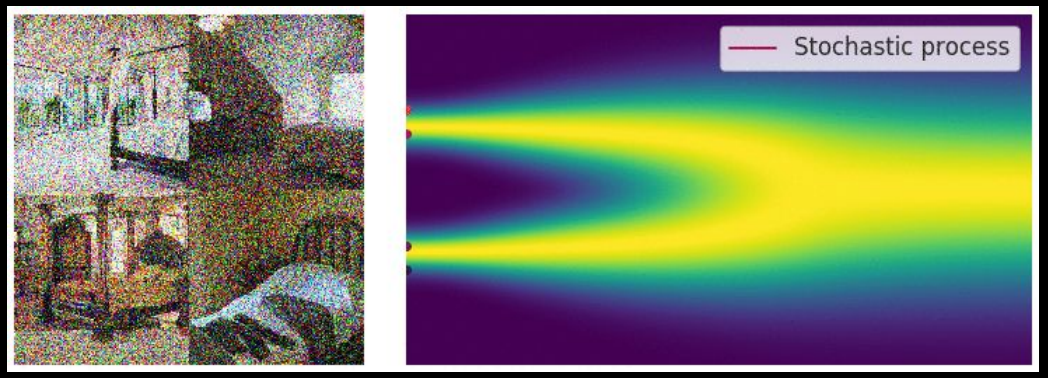
The series of random variables traces trajectory that transforms samples $x(0) \sim p_{data}(\mathbf{x})$ by adding noise until $x(T) \sim \pi(\mathbf{x})$



Continuous time diffusion

"Forward" SDE

$$d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)d\mathbf{w}$$



How to chose f and g ?

The drift and diffusion coefficients are hand-designed to match a specific noising scheme. For instance, we can retrieve our initial discrete diffusion :

discrete forward

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

→

Making $N \to +\infty$ this correspond to :

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\mathbf{w}$$

Continuous time diffusion

"Forward" SDE

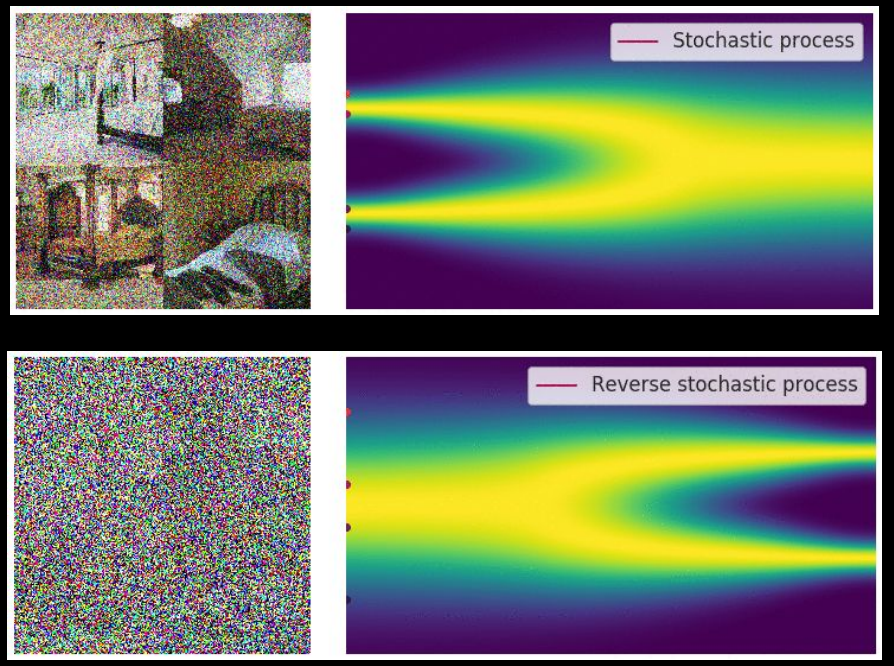
$$d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)d\mathbf{w}$$

How to model the reverse process?

Corresponding reverse SDE

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

we retrieve the score



The (noise-dependent) score can be estimated with denoising score matching

Given an approximated score function $\mathbf{s}_\theta(\mathbf{x}, t)$ we can solve with simple Euler :

Sampling

Sample $x(T) \sim \pi(\mathbf{x})$ then successively apply :

$$\Delta\mathbf{x} \leftarrow \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\mathbf{s}_\theta(\mathbf{x}, t)\right]\Delta t + g(t)\sqrt{|\,\Delta t\,|}\epsilon$$

update $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$ and $t \leftarrow t + \Delta t$

EDM Formulation

Let's simplify all this

$\mathbf{x}_T$  $\mathbf{x}_0$

- Noise schedule $\sigma(t)$ such that $\sigma(0) = 0$ and $\sigma(N) = \sigma_{max}$
- Intermediate distributions $\mathbf{x}_t \sim p(\mathbf{x_t}, \sigma(t))$ obtained by adding i.i.d gaussian noise to the original data : $\mathbf{x}_t = \mathbf{x}_0 + \sigma(t)\epsilon$

ODE Formulation $\mathrm{d}\boldsymbol{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}; \sigma(t))\mathrm{d}t$

- The ODE is the deterministic counterpart of the SDE. The only stochasticity during sampling comes from the prior sampling
- For a positive step Δ_t , the ODE nudges the sample away from the data
- For negative step $-\Delta_t$, the ODE pushes the sample towards $p_{data}(\mathbf{x})$

EDM Formulation

Let's simplify all this

$\mathbf{x}_T$  $\mathbf{x}_0$

ODE Formulation $$\mathrm{d}\boldsymbol{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}; \sigma(t))\mathrm{d}t$$ **(1)**

Estimating the score

If a denoiser minimize for all σ : $$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\, \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\sigma^2\mathbf{I})} \left\| D(\boldsymbol{x}+\boldsymbol{\epsilon};\sigma) - \boldsymbol{x} \right\|_2^2$$ **(2)**

then $$\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x};\sigma) = (D(\boldsymbol{x};\sigma) - \boldsymbol{x})/\sigma^2$$ **(3)**

To summarize

- Train a denoiser $D(\boldsymbol{x};\sigma)$ to minimize **(2)** for different noise levels
- Iterative sampling using the update obtained by combining **(1)** and **(3)**

$$\frac{d\mathbf{x}}{dt} = \dot{\sigma(t)} \frac{\mathbf{x} - D(\mathbf{x}, \sigma(t))}{\sigma(t)}$$

EDM Formulation

Training

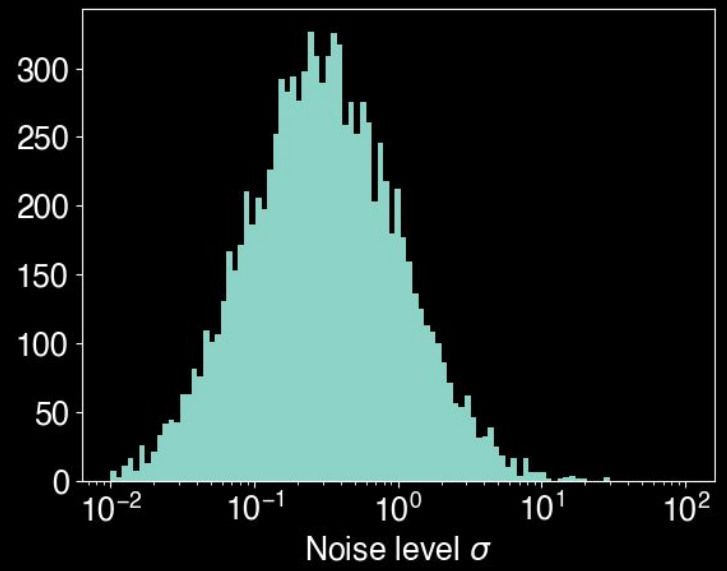
Training noise levels

High noise : training target is impossible to recover

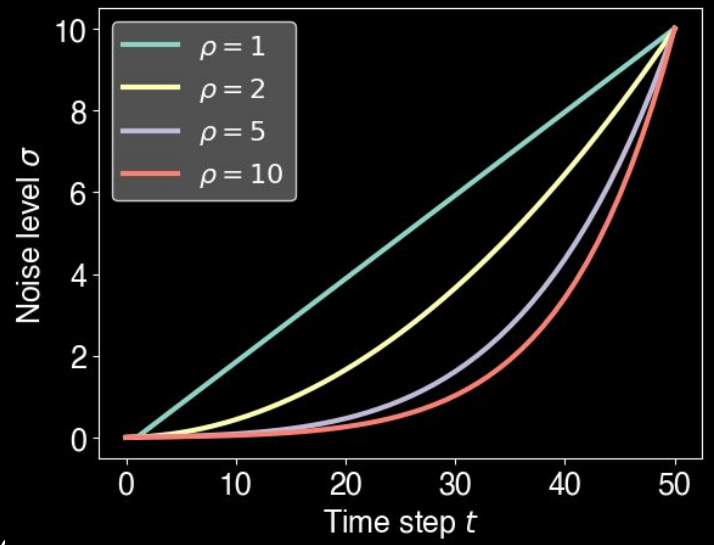
Low noise : irrelevant to predict vanishing noise

➔ focus on intermediate levels with lognormal

$$ln(\sigma) \sim \mathcal{N}(P_{mean}, P_{std}^2)$$



$P_{mean} = -1.2, P_{std} = 1.2$



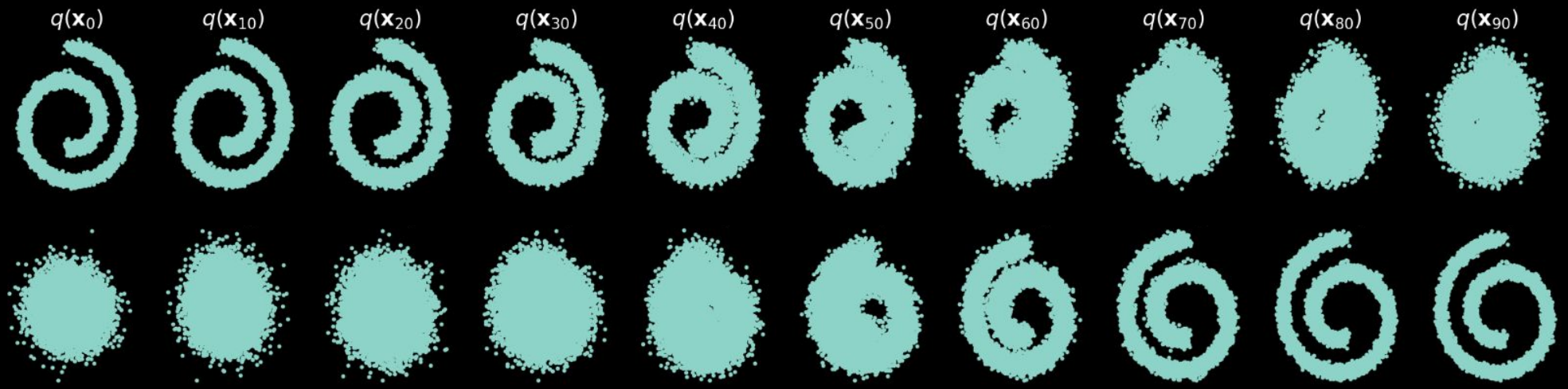
Sampling scheme

Take smaller steps at lower noise levels to reduce the discretization error

$$\sigma_{0<i<=N} = (\sigma_{min}^{1/\rho} + \frac{i}{N-1}(\sigma_{max}^{1/\rho} - \sigma_{min}^{1/\rho}))^{\rho}$$

Diffusion process

Results of trained models



Results for image generation

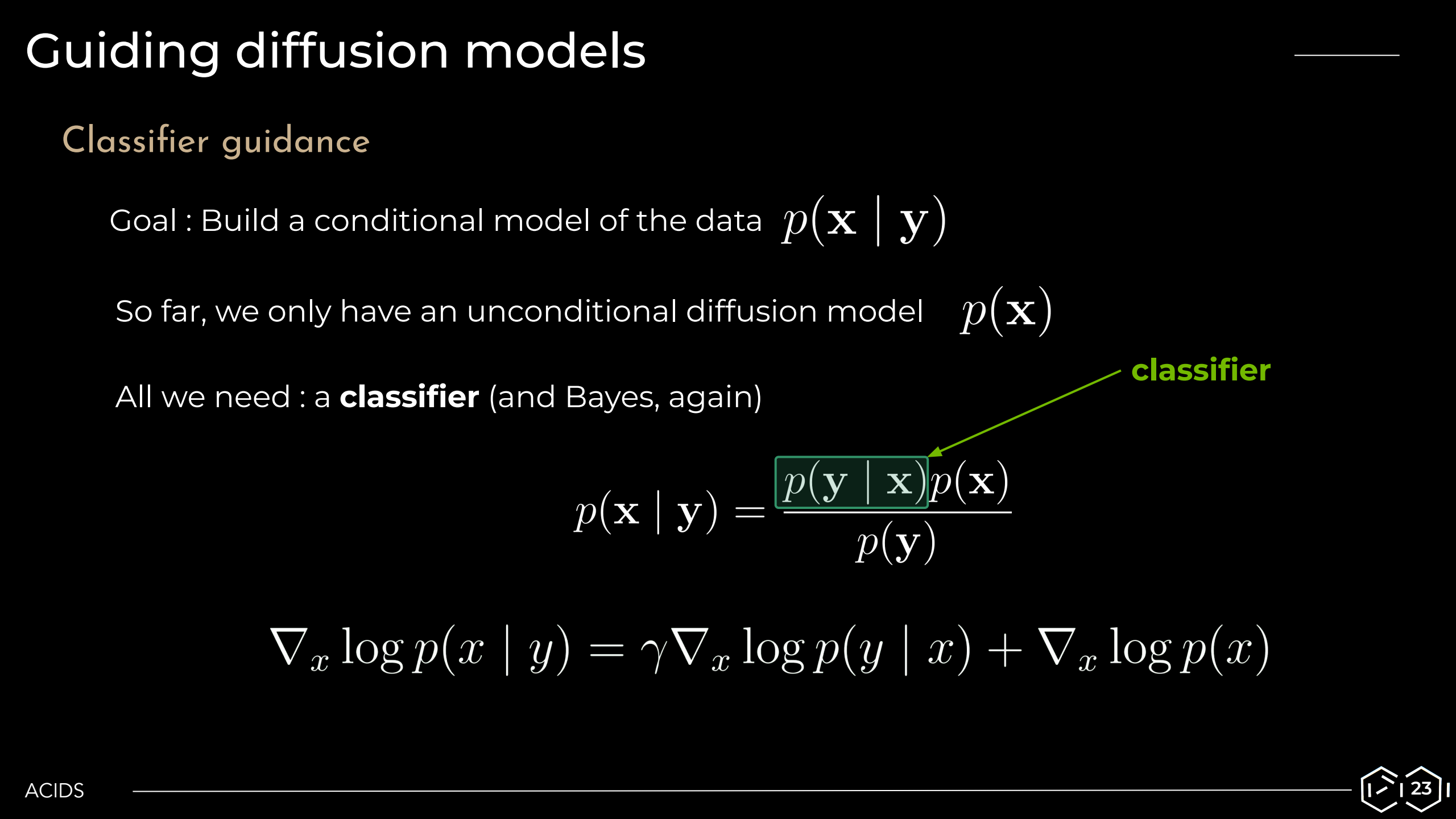
Guiding diffusion models

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)

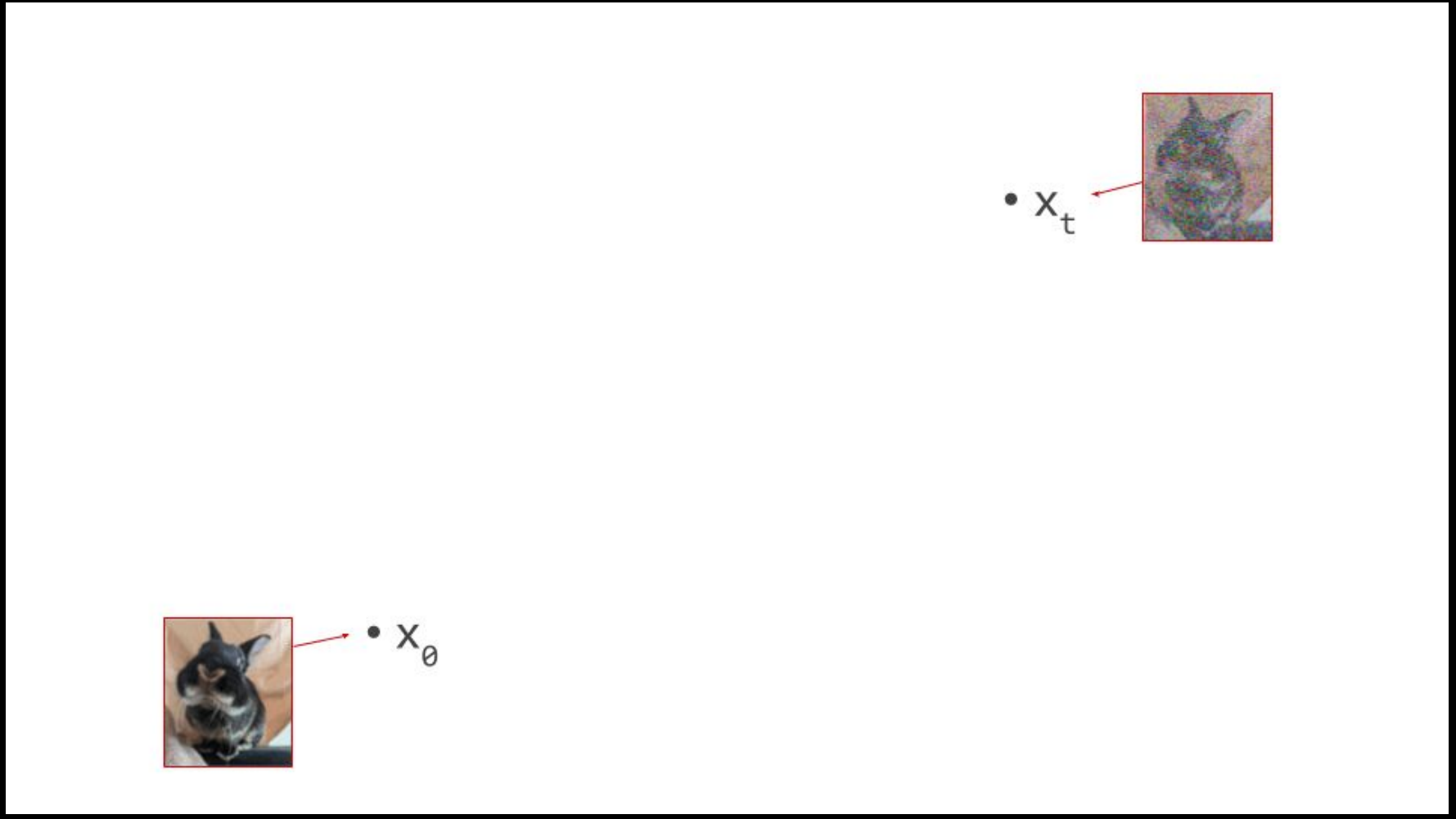


$$p(\mathbf{x} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}$$

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

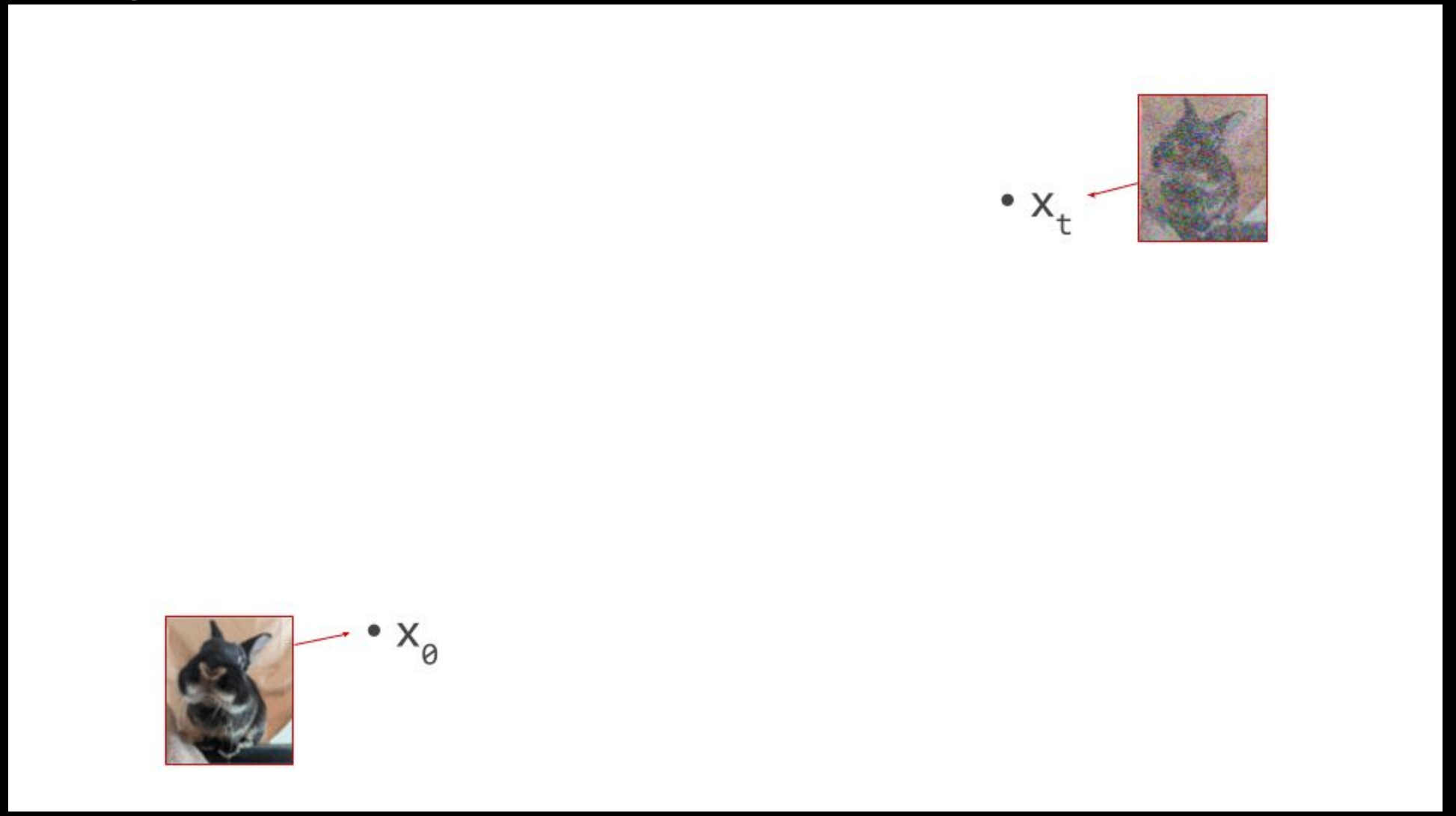
Guiding diffusion models

Without classifier guidance



Guiding diffusion models

With classifier guidance



Guiding diffusion models

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

Issues

- Requires an external classifier $\log p(y \mid x)$
- The classifier must be noise-robust because it is applied to noisy versions of the the input

Classifier-free guidance

Train the diffusion model as both a conditional model $p(\mathbf{x} \mid \mathbf{y})$ and unconditional model $p(\mathbf{x})$ by applying drop out to the conditioning

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

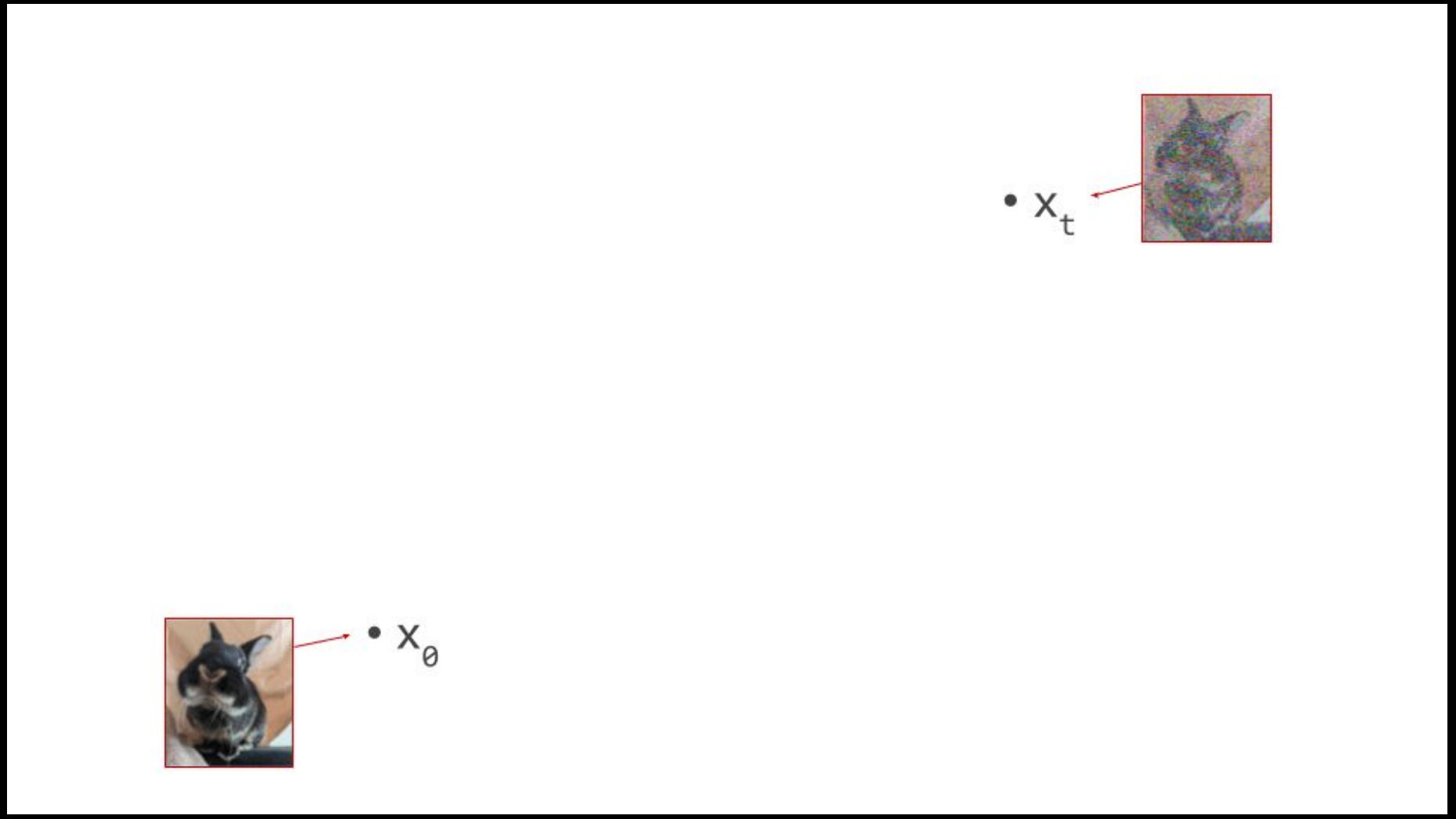
$\gamma = 1$ Conditional

$\gamma > 1$ Magic happens

- Performs better than classifier guidance
- Specifically with strong factor
- No additional training

Guiding diffusion models

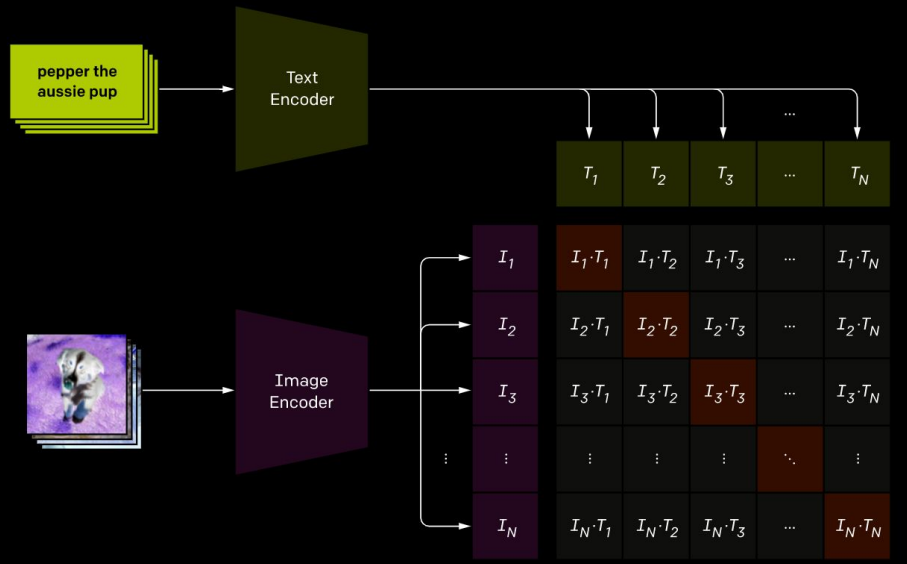
Classifier-free guidance



Extended applications

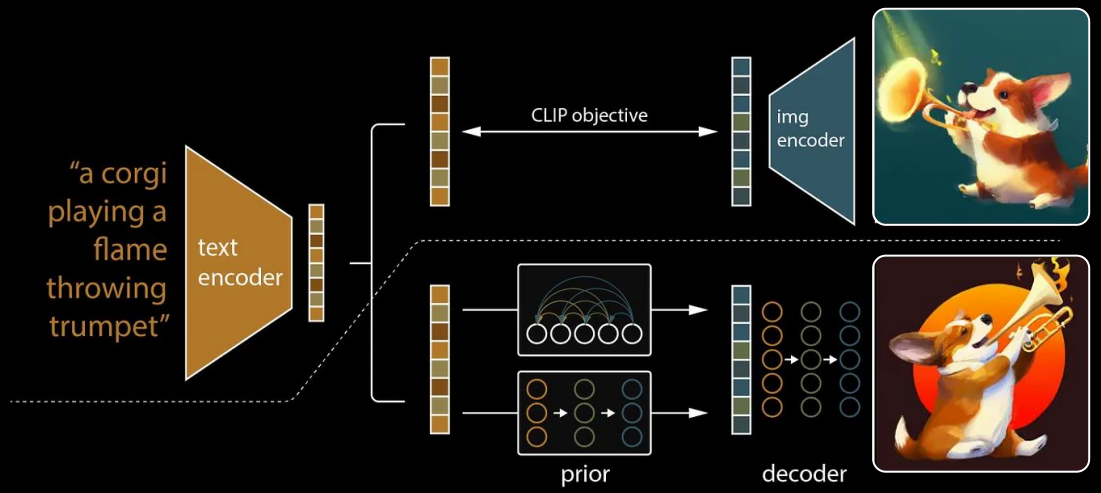
CLIP Multimodality

Learns text-to-image association



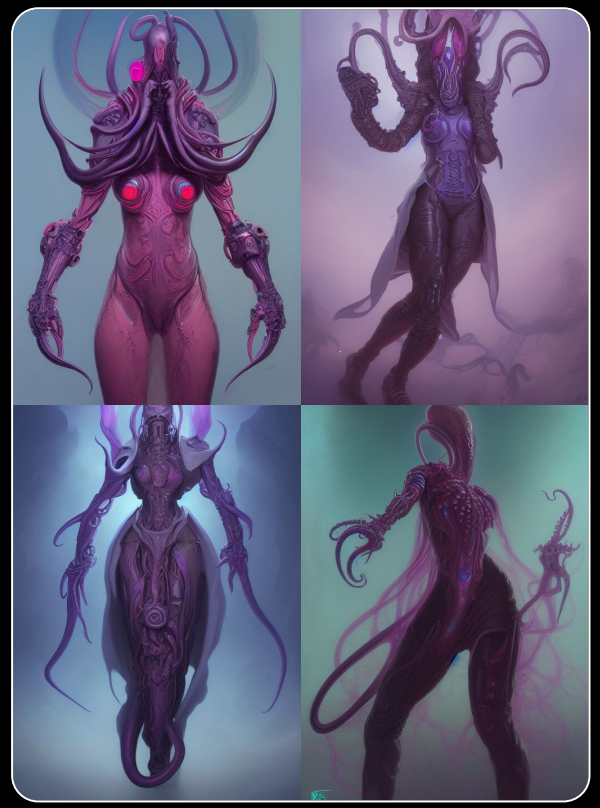
Generation

CLIP Conditionning

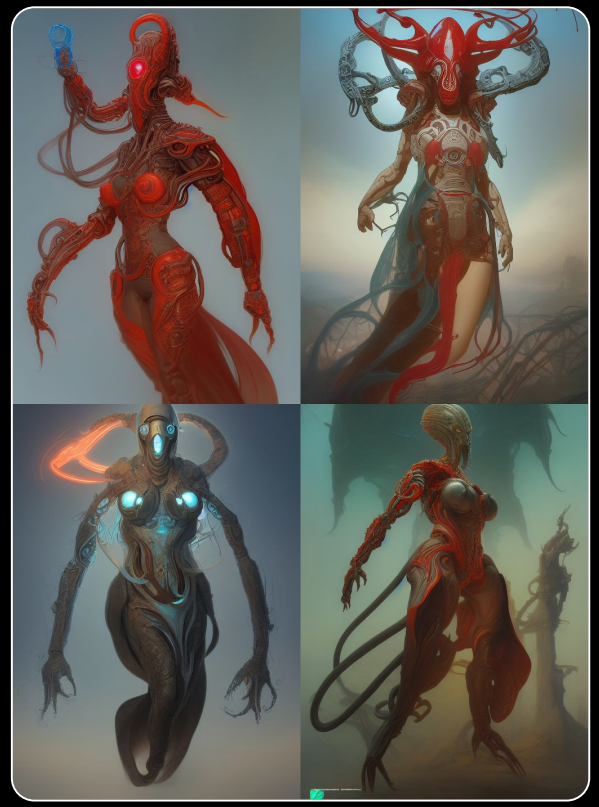


Extended applications

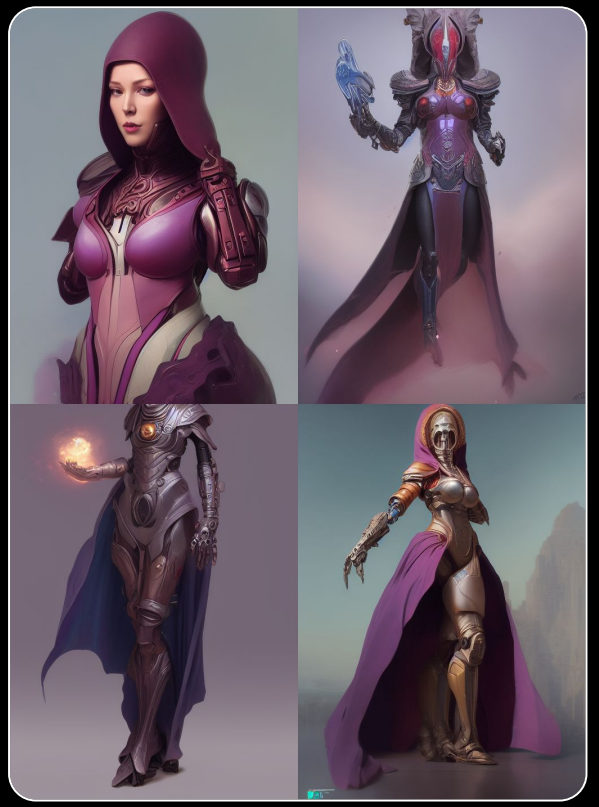
Negative prompts addition to image generation



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.