Diffusion Models

M. Soleymani
Sharif University of Technology
Spring 2024

Most slides have been adapted from:

Vahdat et al., Denoising Diffusion-based Generative Modeling Tutorial, CVPR 2022

Calvin Luo, Deep Learning Course, Brown University

Outline

- Denoising Diffusion Models
  - Forward Diffusion Process
  - Diffusion Kernel
  - Reverse Denoising Process
- Score Matching

Denoising Diffusion Models

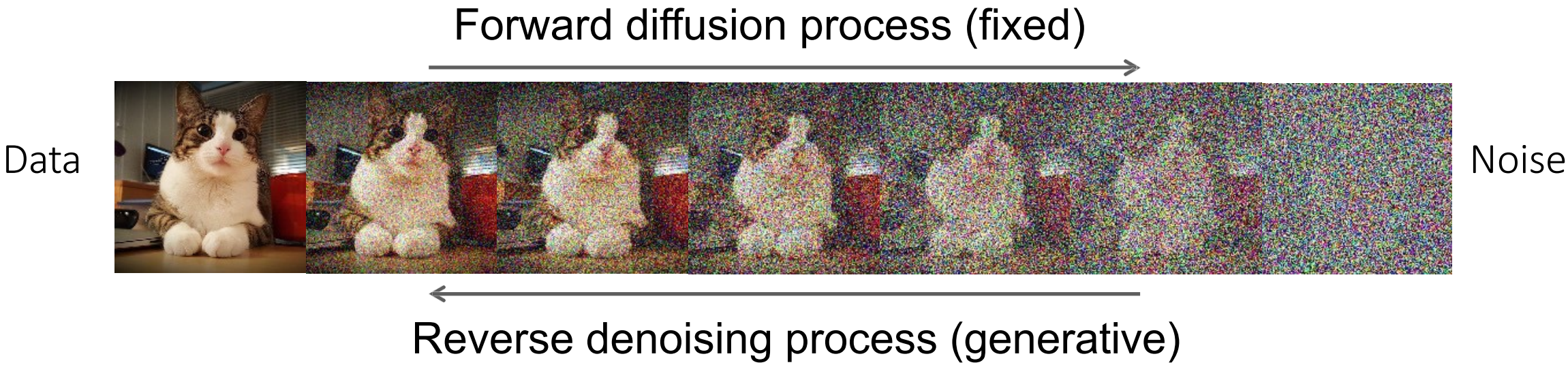
Emerging as powerful generative models, outperforming GANs

"Diffusion Models Beat GANs on Image Synthesis"
Dhariwal & Nichol, OpenAI, 2021

"Cascaded Diffusion Models for High Fidelity Image Generation"
Ho et al., Google, 2021

Denoising Diffusion Models

- Learning to generate by denoising
- Denoising diffusion models consist of two processes:
  - Forward diffusion process that gradually adds noise to input
  - Reverse denoising process that learns to generate data by denoising



Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

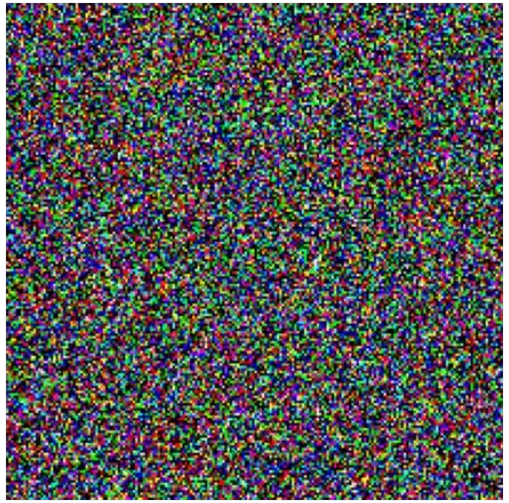
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

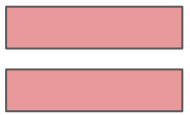
Diffusion Models: An Observation

adding steady amounts of Gaussian noise eventually corrupts an image into something indistinguishable from a random Gaussian sample.

Diffusion Models: An Observation

adding steady amounts of Gaussian noise eventually corrupts an image into something indistinguishable from a random Gaussian sample.

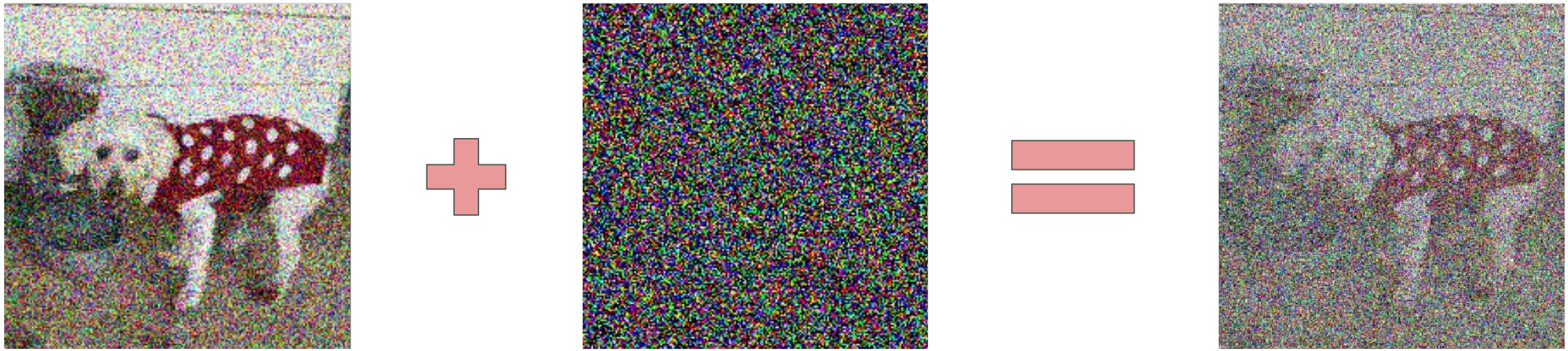




One Step

Diffusion Models: An Observation

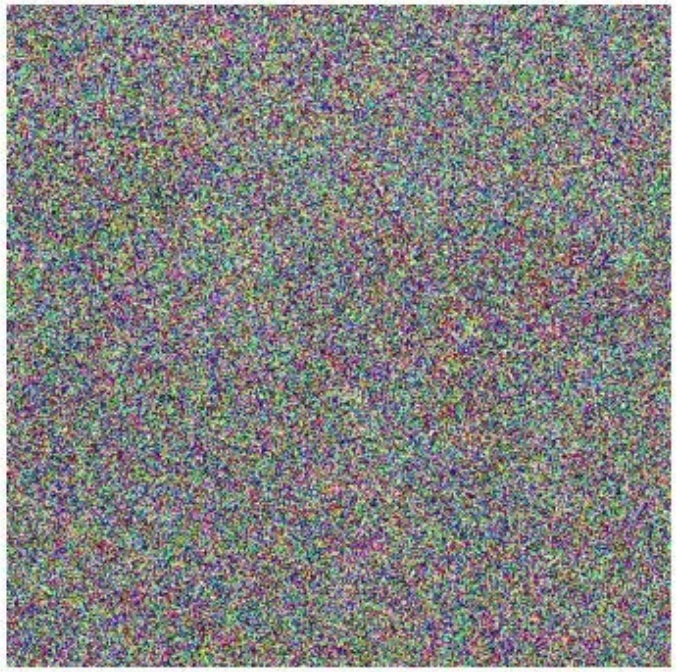
adding steady amounts of Gaussian noise eventually corrupts an image into something indistinguishable from a random Gaussian sample.



Another Step

Diffusion Models: An Observation

adding steady amounts of Gaussian noise eventually corrupts an image into something indistinguishable from a random Gaussian sample.

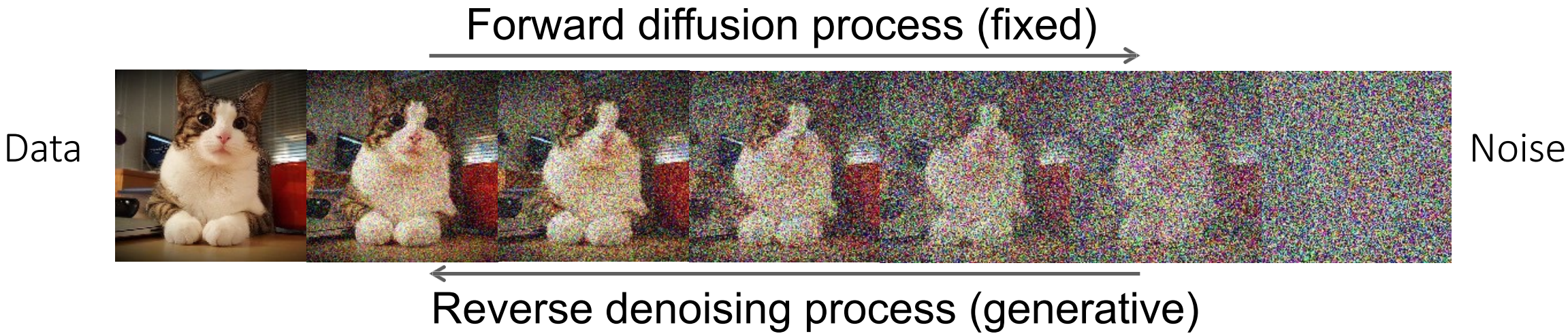


Many Steps

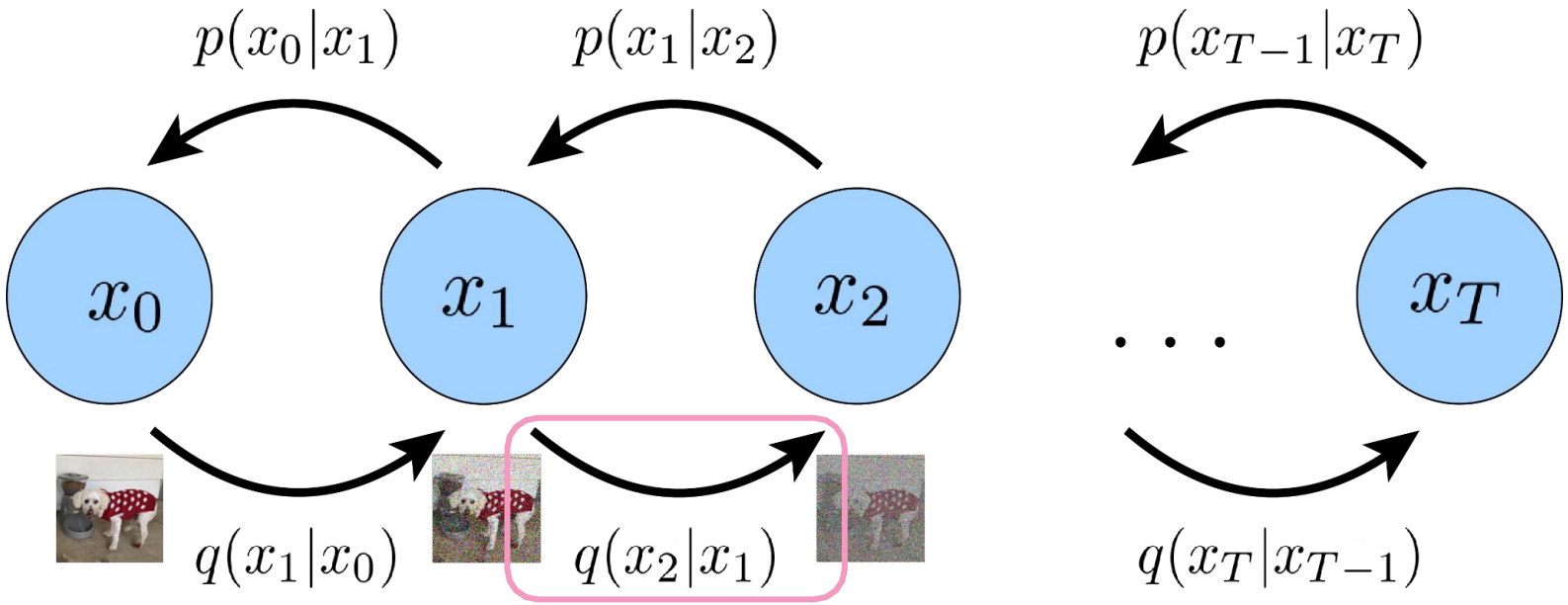
Diffusion Models: An Generative Process

adding steady amounts of Gaussian noise eventually corrupts an image into something indistinguishable from a random Gaussian sample.

- Diffusion models simply learn to **reverse** this procedure over many timesteps



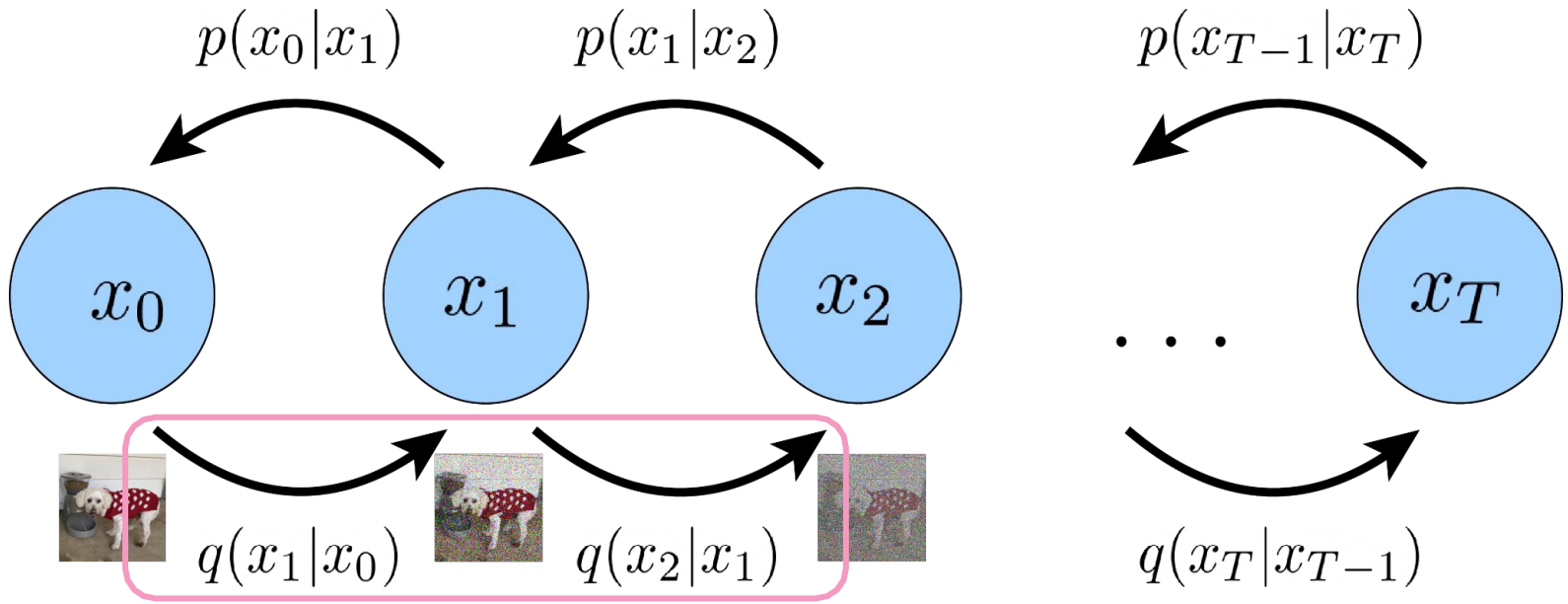
Let's take a look at one encoding

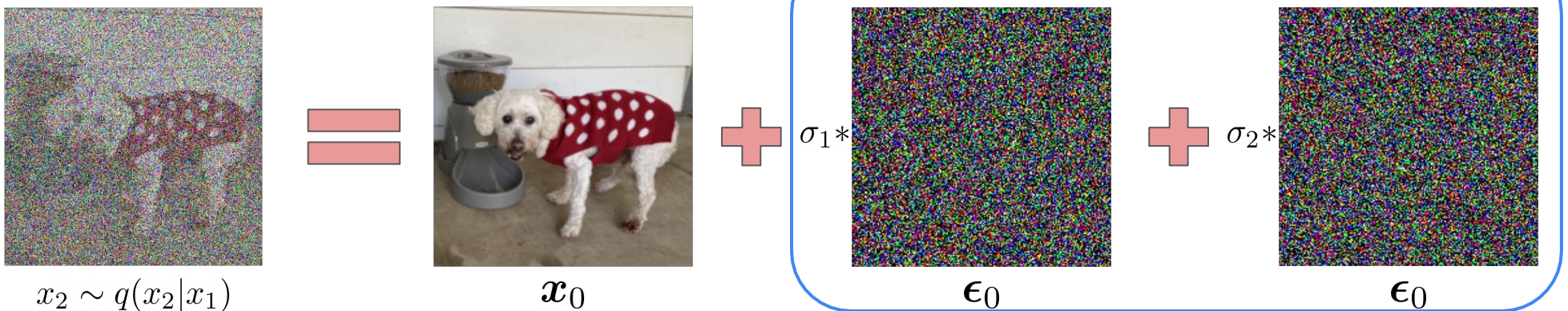


$x_2 \sim q(x_2|x_1)$ $\quad$ x_1 $\quad$ ϵ_0

reparam. trick!

Let's take a look at one encoding



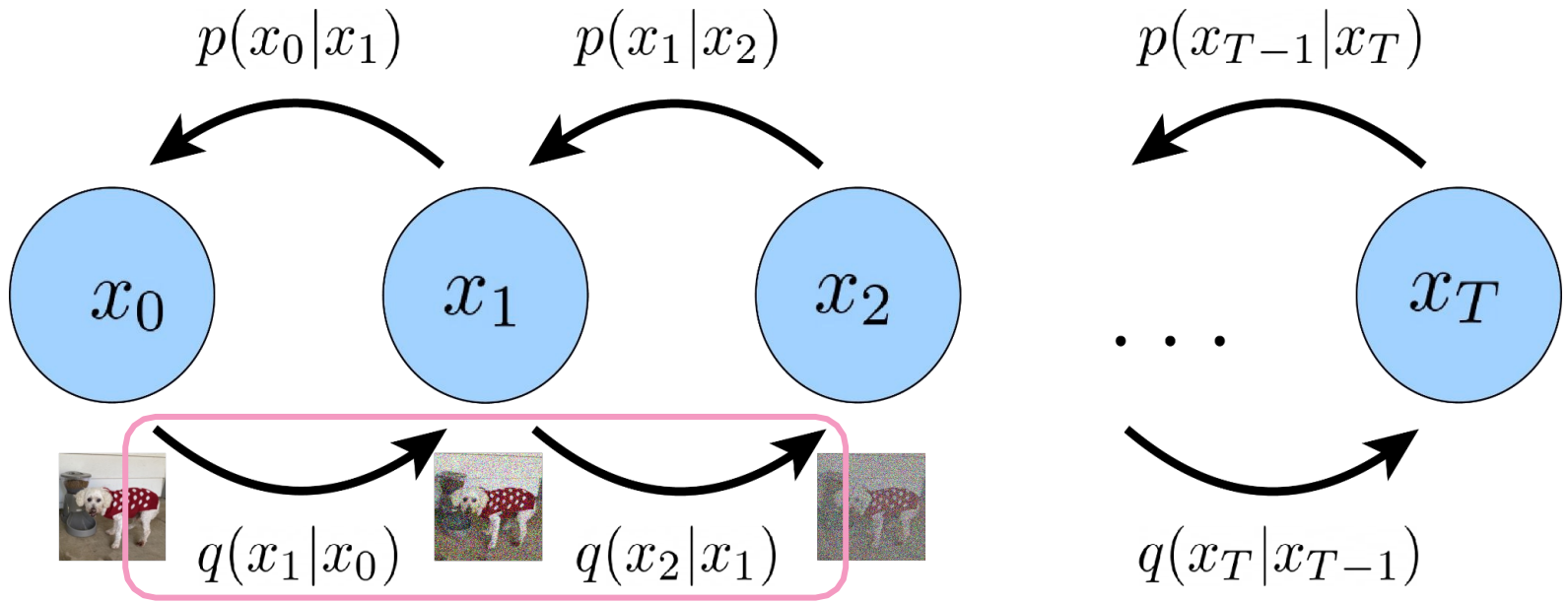


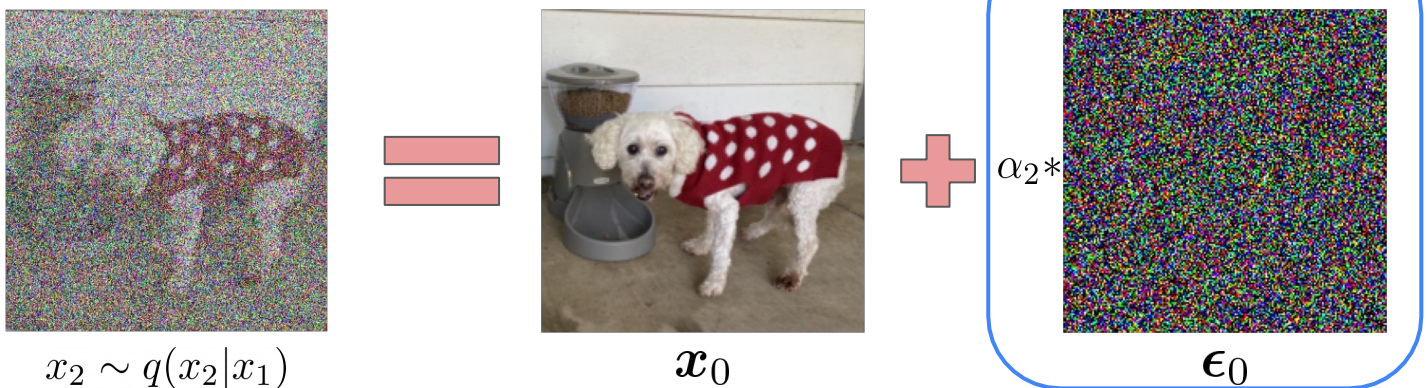
$x_2 \sim q(x_2|x_1)$ $\quad$ $\boldsymbol{x}_0$ $\quad$ $\boldsymbol{\epsilon}_0$ $\quad$ $\boldsymbol{\epsilon}_0$

Aggregate into 1 sample!

reparam trick!

Let's take a look at one encoding





$x_2 \sim q(x_2|x_1)$ $\quad$ $\boldsymbol{x}_0$ $\quad$ $\boldsymbol{\epsilon}_0$

where,

$$\alpha_2 = \sqrt{\sigma_1^2 + \sigma_2^2}$$

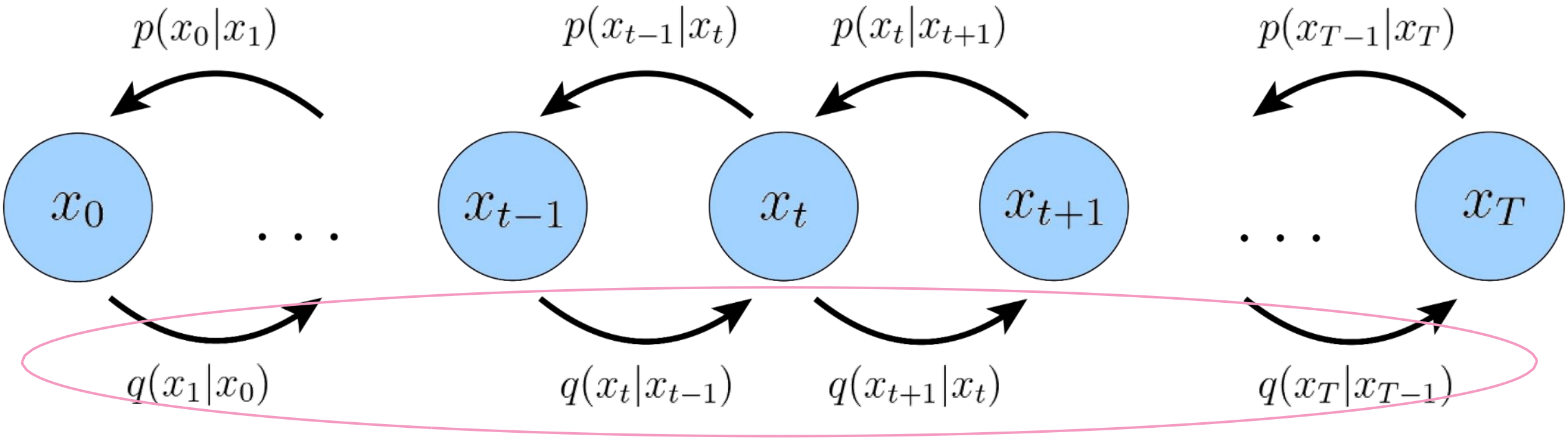
and,

$$q(x_2|x_0) = \mathcal{N}(x_2|x_0, \alpha_2^2)$$

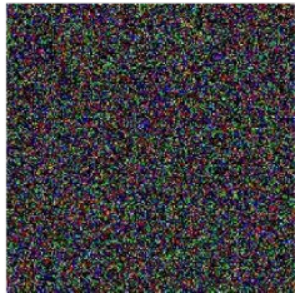
Aggregate into 1 sample!

reparam. trick!

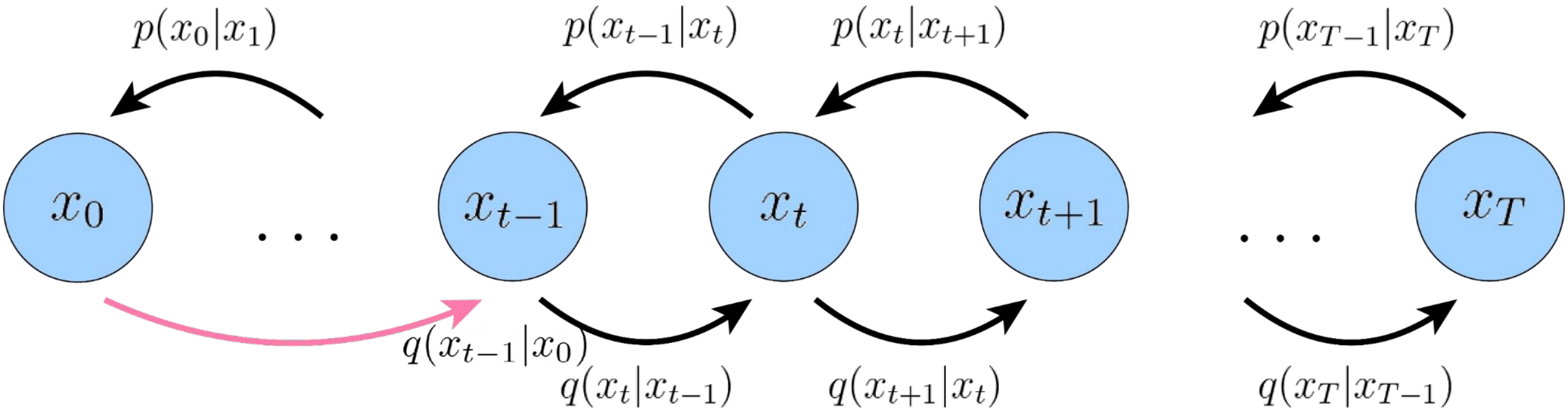
Let’s take a look at one encoding

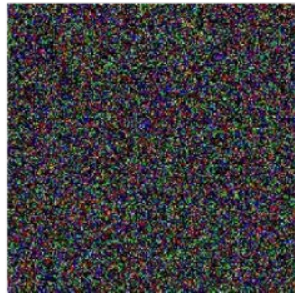


Individual (known) Gaussians!

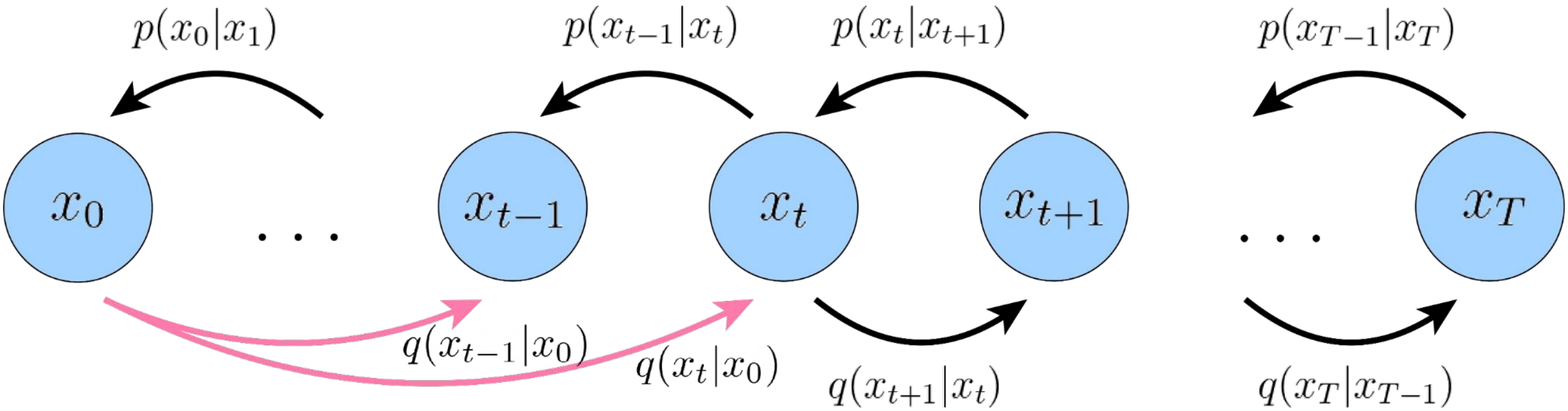


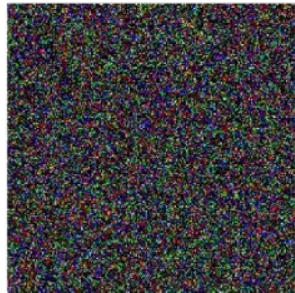
Let’s take a look at one encoding



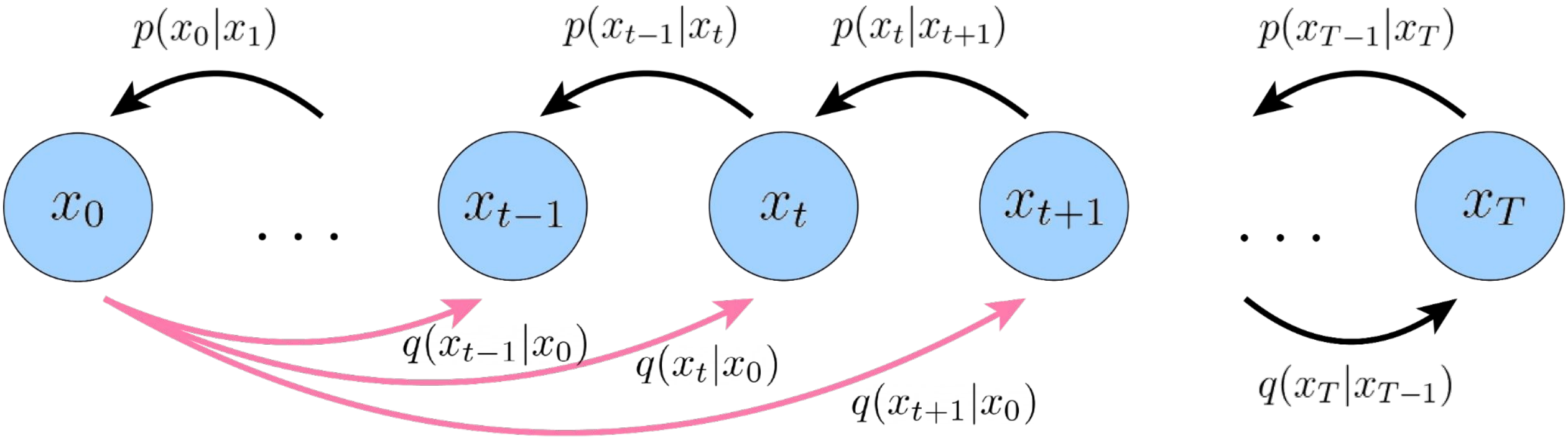


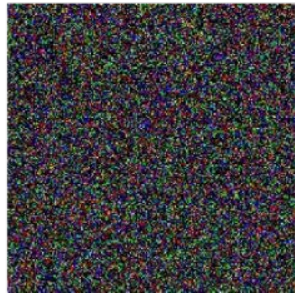
Let’s take a look at one encoding





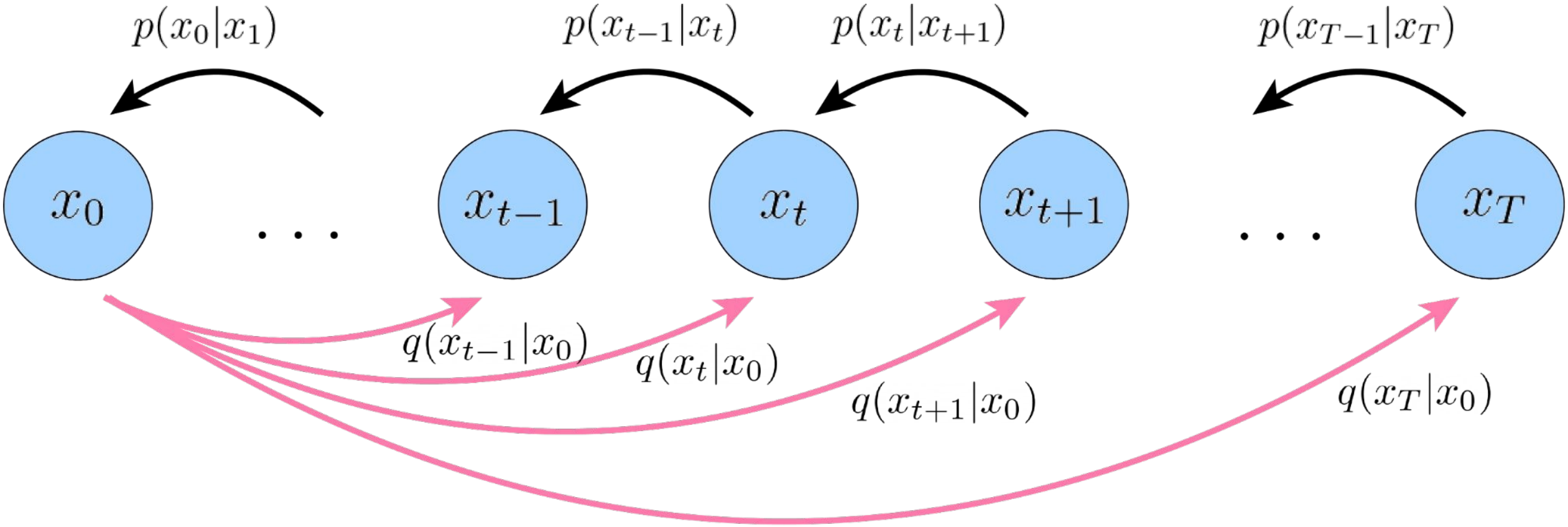
Let’s take a look at one encoding

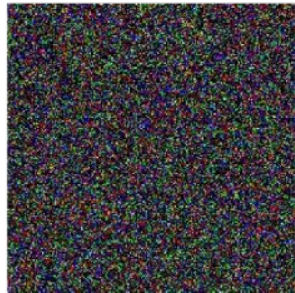




$q(x_t|x_0)$ is a Gaussian, for arbitrary t !

$q(x_t|x_0) = \mathcal{N}(x_t|x_0, \alpha_t^2 \mathrm{I})$, where $\alpha_0, \alpha_1, \ldots \alpha_T$ are all known/fixed.



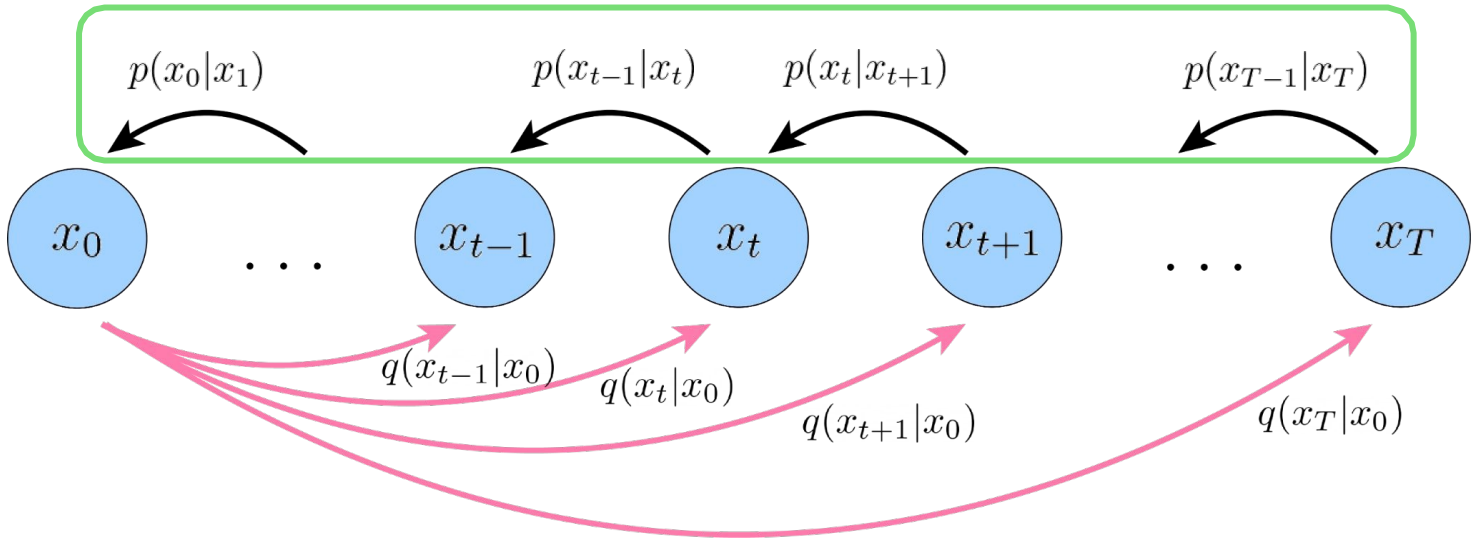


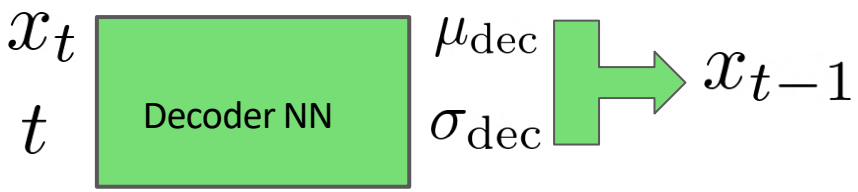
Diffusion Models

A diffusion model is implemented as a single neural network (the decoder)

Quick Quiz: The dataset we are given is only clean images x_0 - but the decoder takes in x_t as input. How do we get our x_t's to train on?

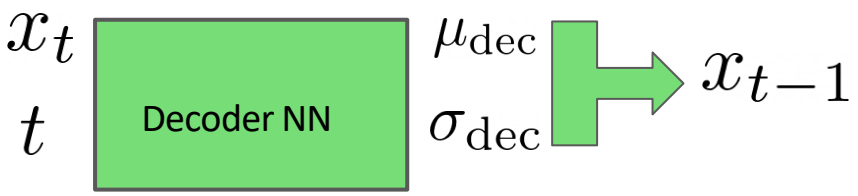
How do we optimize our decoder? How do we know we are outputting good x_{t-1}?





Optimization?

We want to learn a denoising decoder:

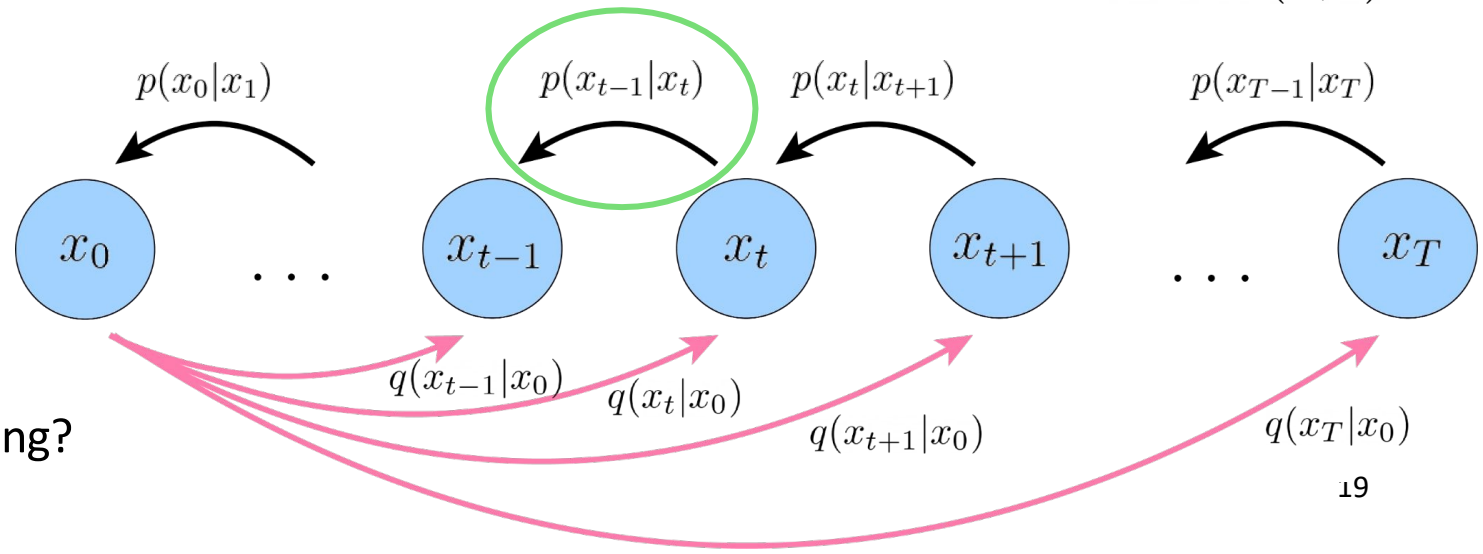


$$\hat{x}_{t-1} = \mu_{\text{dec}} + \sigma_{\text{dec}} * \epsilon$$

reparam. trick!

$$\epsilon \sim \mathcal{N}(0, \mathrm{I})$$

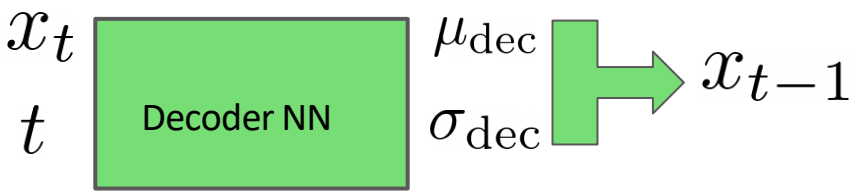
But what is the form of x_{t-1}?



…can we formulate this as supervised learning?

Optimization?

We want to learn a denoising decoder:



$$\hat{x}_{t-1} = \mu_{\text{dec}} + \sigma_{\text{dec}} * \epsilon$$

reparam. trick!

$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

But what is the form of x_{t-1}?

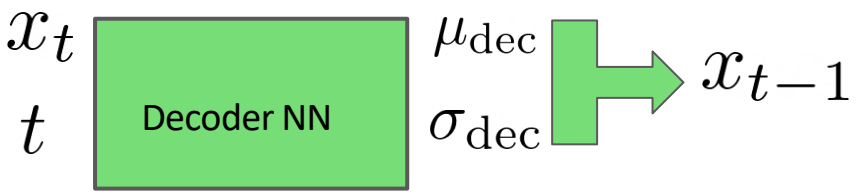
Recall that:

$q(x_t|x_0)$ is a Gaussian, for arbitrary t!

$q(x_t|x_0) = \mathcal{N}(x_t|x_0, \alpha_t^2 \mathrm{I})$, where $\alpha_0, \alpha_1, \ldots \alpha_T$ are all known/fixed.

Optimization?

We want to learn a denoising decoder:



$$\hat{x}_{t-1} = \mu_{\text{dec}} + \sigma_{\text{dec}} * \epsilon$$

reparam. trick!

$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

But what is the form of x_{t-1}?

Recall that:

$$q(x_{t-1}|x_0) = \mathcal{N}(x_{t-1}|x_0, \alpha_{t-1}^2 \mathrm{I})$$

$$\therefore x_{t-1} = x_0 + \alpha_{t-1} * \epsilon$$

reparam. trick!

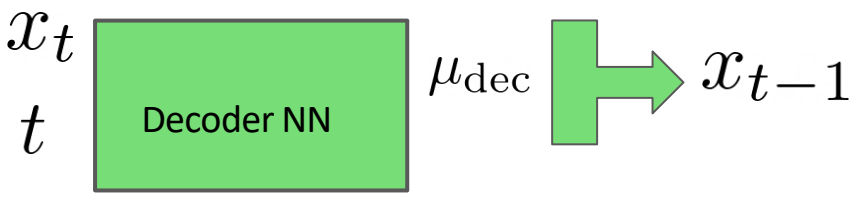
$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

Do we really need to predict σ_{dec}?

What is the ground truth signal for μ_{dec}

Optimization?

We want to learn a denoising decoder:



$$\hat{x}_{t-1} = \hat{x}_0 + \alpha_{t-1} * \epsilon$$

reparam. trick!

$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

But what is the form of x_{t-1}?

Recall that:

$$q(x_{t-1}|x_0) = \mathcal{N}(x_{t-1}|x_0, \alpha_{t-1}^2 \mathrm{I})$$

$$\therefore x_{t-1} = x_0 + \alpha_{t-1} * \epsilon$$

reparam. trick!

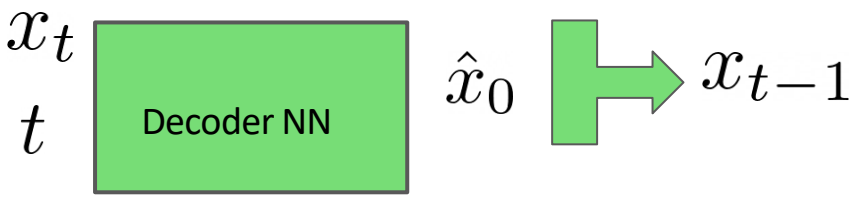
$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

Do we really need to predict σ_{dec}?

What is the ground truth signal for μ_{dec}

Optimization?

We want to learn a denoising decoder:



$$\hat{x}_{t-1} = \hat{x}_0 + \alpha_{t-1} * \epsilon$$

reparam. trick!

$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

But what is the form of x_{t-1}?

Recall that:

$$q(x_{t-1}|x_0) = \mathcal{N}(x_{t-1}|x_0, \alpha_{t-1}^2 \mathrm{I})$$

$$\therefore x_{t-1} = x_0 + \alpha_{t-1} * \epsilon$$

reparam. trick!

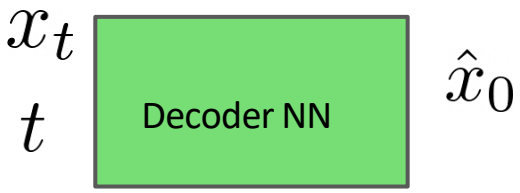
$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

Do we really need to predict σ_{dec}?

What is the ground truth signal for μ_{dec}

Optimization?

We want to learn a denoising decoder:



So in the end, a diffusion model is simply *one* Neural Network that predicts a clean image x_0 from arbitrary noisified image x_t.

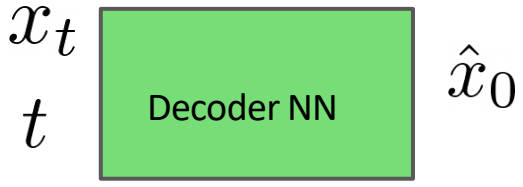
Loss Objective:

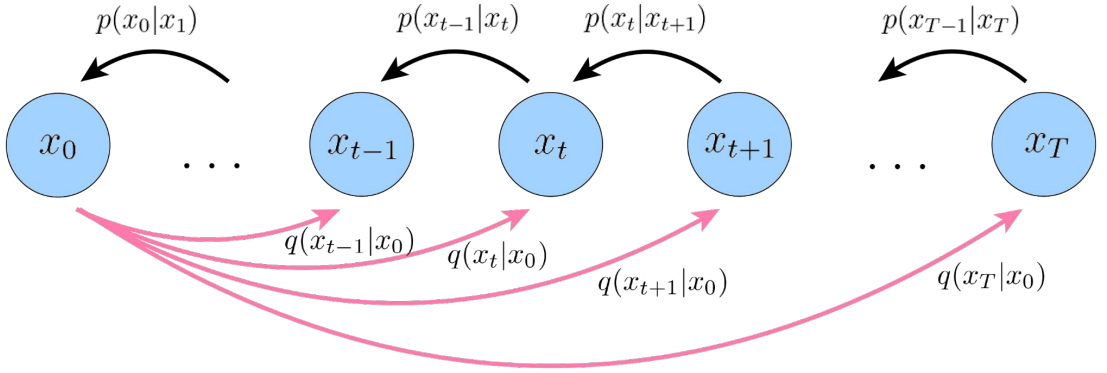
$$\arg\min_{\boldsymbol{\theta}} \; ||x_0 - \hat{x}_\theta(x_t, t)||^2$$

Diffusion Models: A Summary

A Diffusion Model is:

- One NN that predicts a clean image from a noisy version of the image

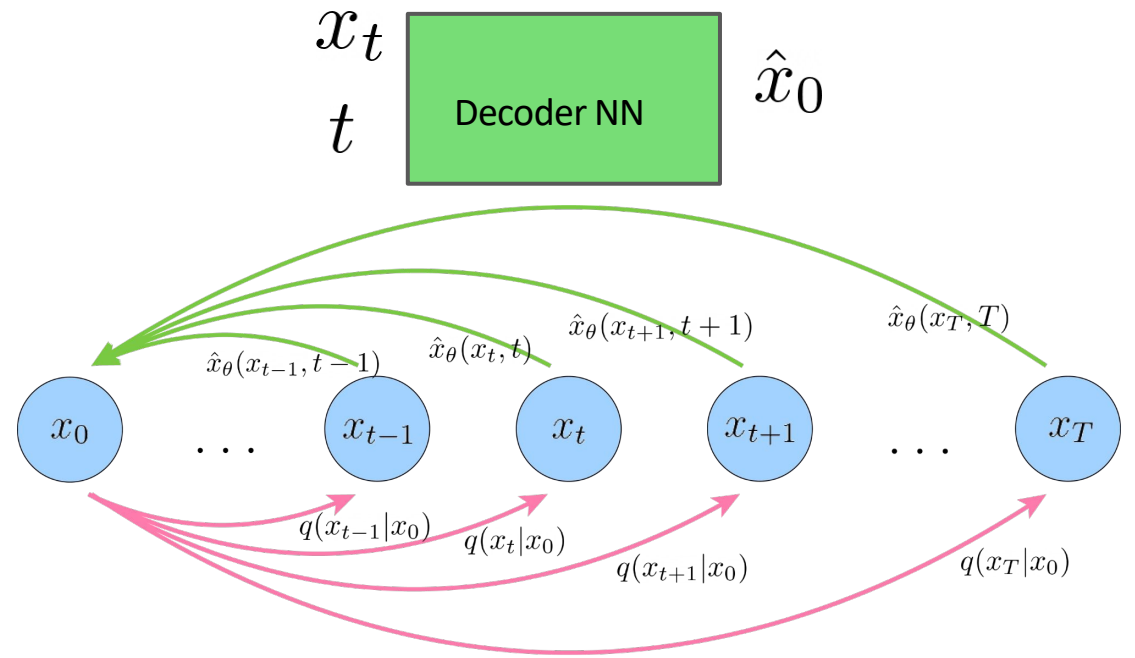




Diffusion Models: A Summary

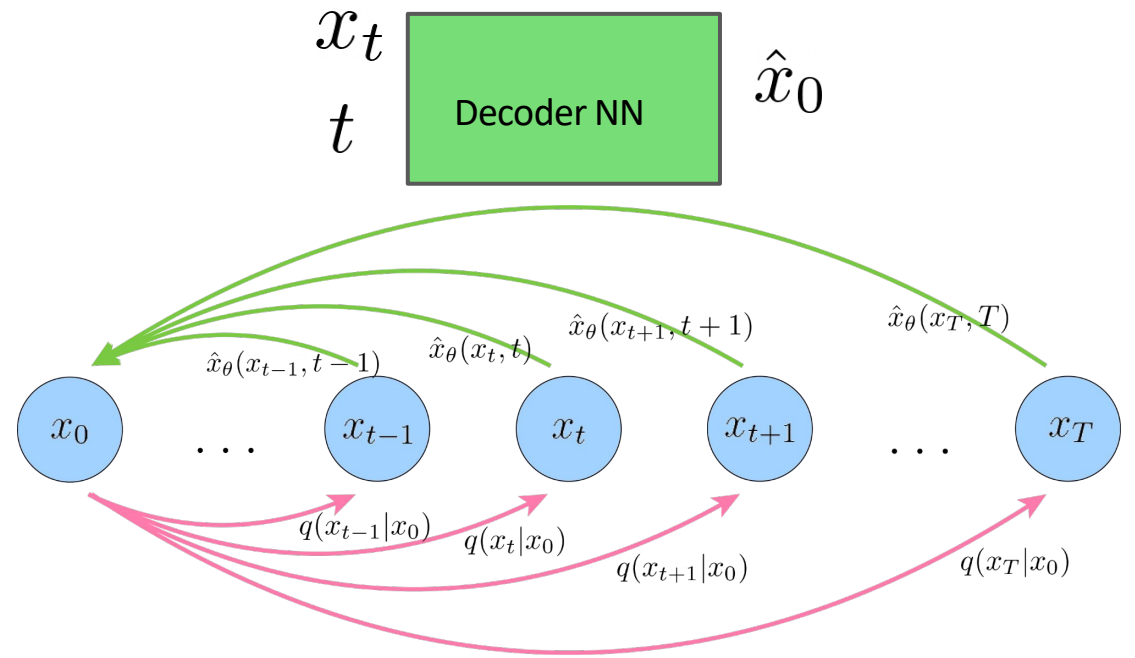
A Diffusion Model is:

- One NN that predicts a clean image from a noisy version of the image

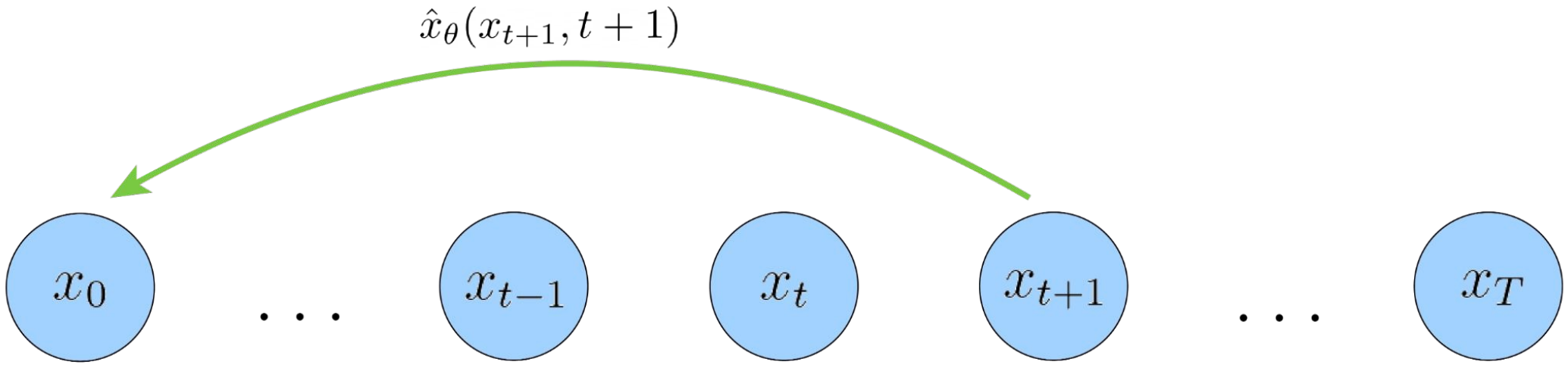


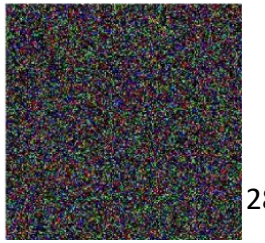
Diffusion Models: A Summary

How do we perform sampling?

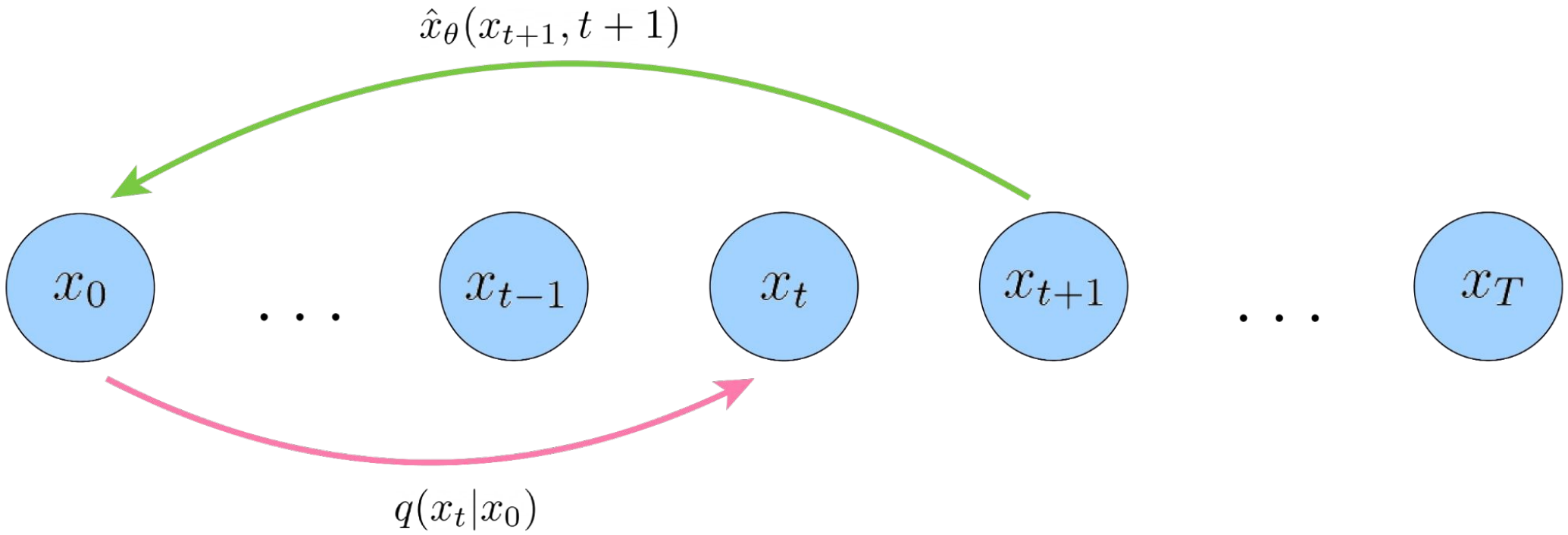


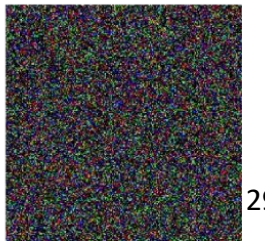
Sampling



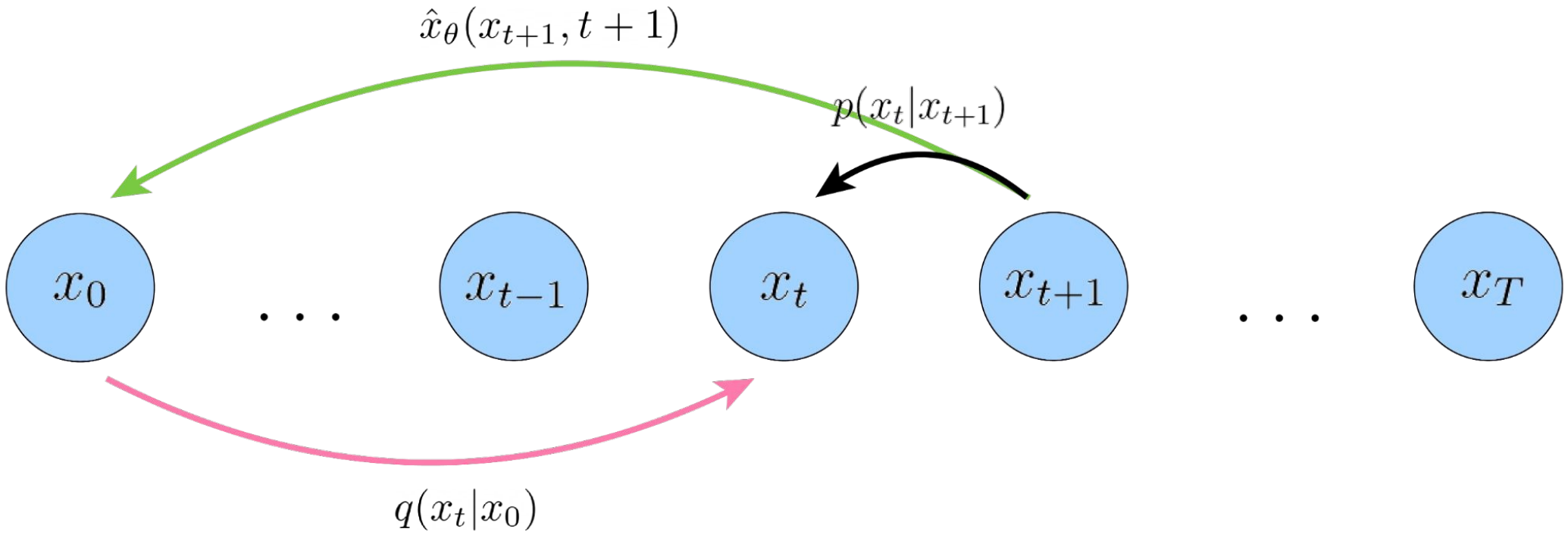


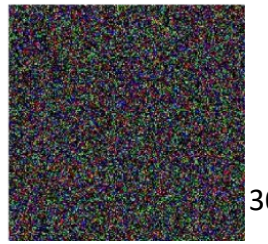
Sampling



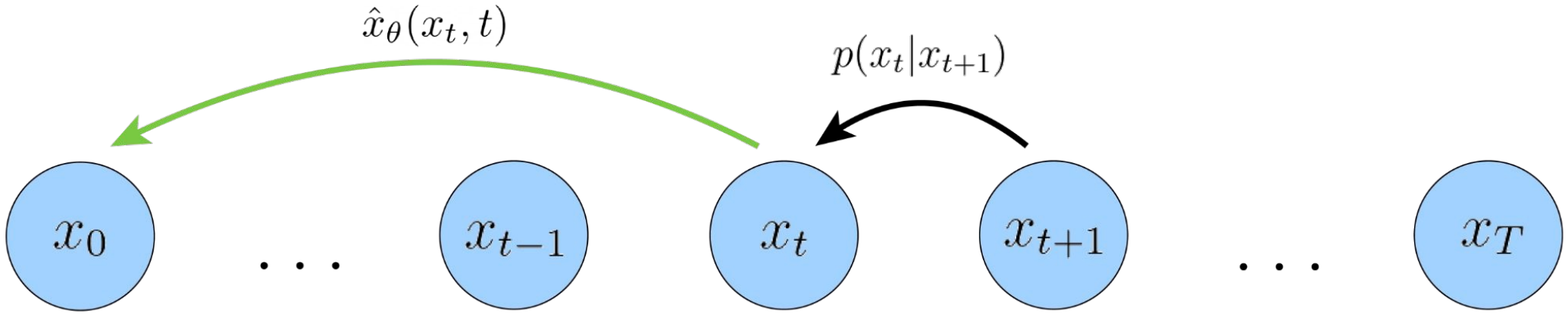


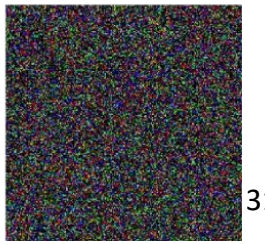
Sampling



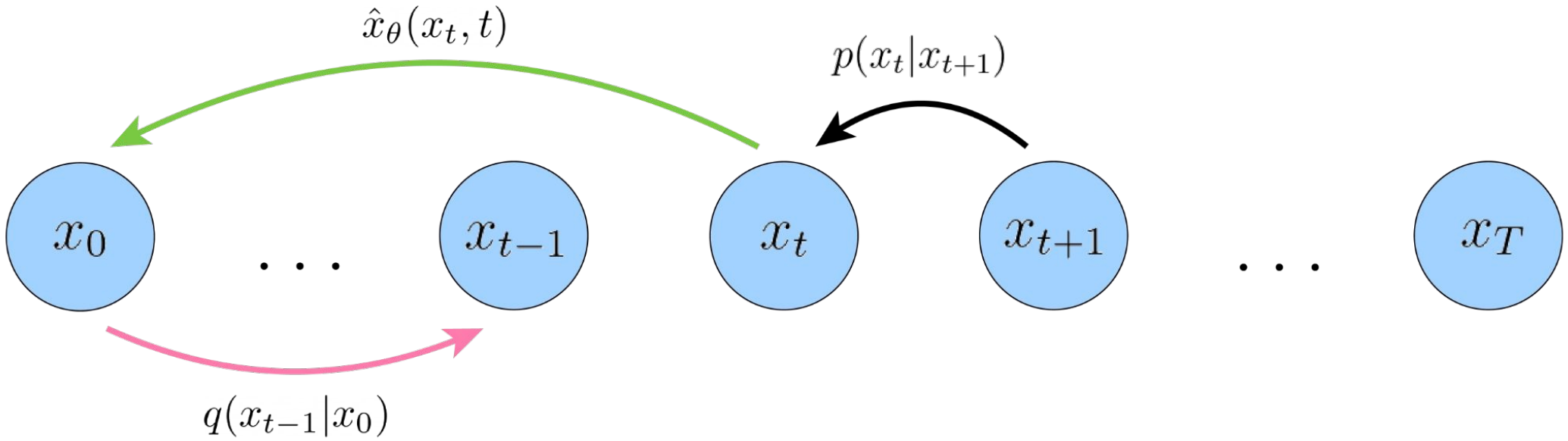


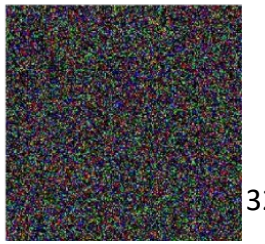
Sampling



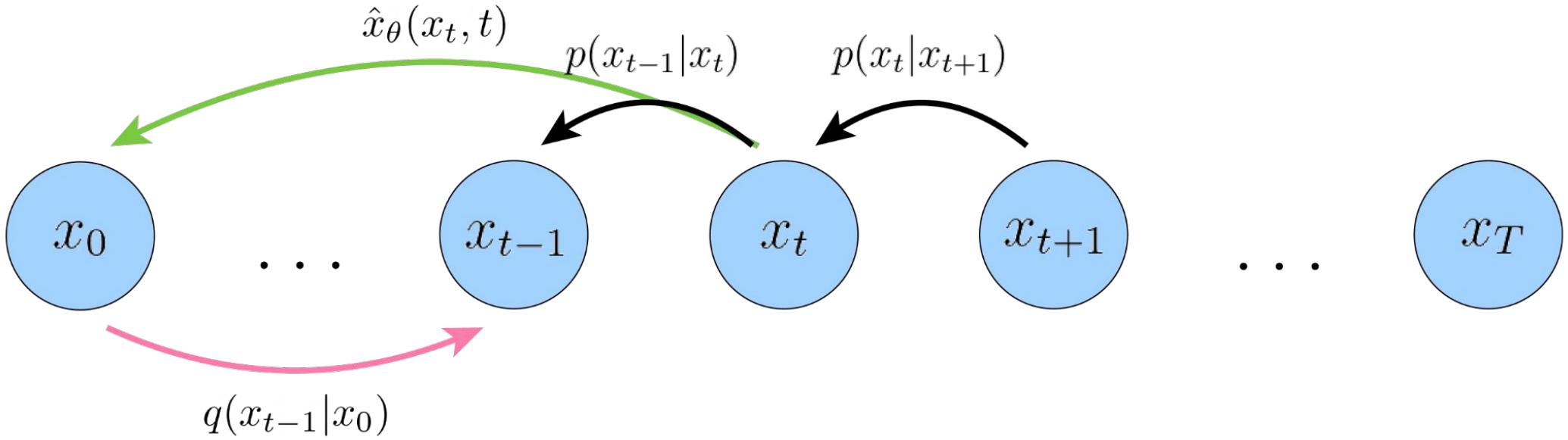


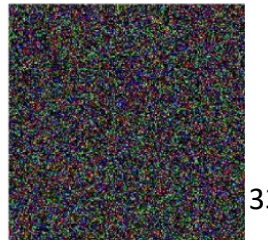
Sampling



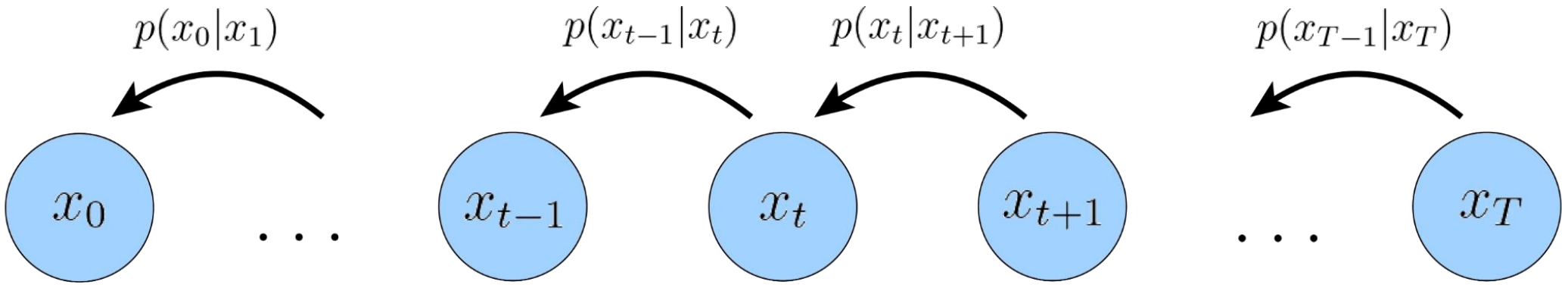


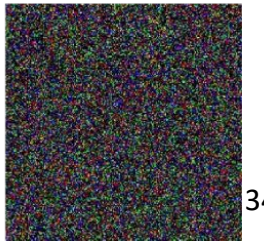
Sampling


$\hat{x}_\theta(x_t, t)$
$p(x_{t-1}|x_t)$
$p(x_t|x_{t+1})$
x_0
$\cdots$
x_{t-1}
x_t
x_{t+1}
$\cdots$
x_T
$q(x_{t-1}|x_0)$



Sampling





Pseudocode

Algorithm 1 Training

repeat
  $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$
  $t \sim \texttt{Uniform}(1, ..., T)$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
    $\nabla_\theta ||\boldsymbol{x}_0 - \hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_0 + \alpha_t \boldsymbol{\epsilon}, t)||^2$
until converged

Algorithm 2 Sampling

$\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, ..., 1$:
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\boldsymbol{\epsilon} = 0$
  $\boldsymbol{x}_{t-1} = \hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t, t) + \alpha_{t-1}\epsilon$
end for
return $\boldsymbol{x}_0$

Another Viewpoint: Predicting Noise

It turns out, training a diffusion model can be done using a different interpretation:

- Predicting noise (instead of predicting image)

Noise Predictor

Recall that our objective is to predict $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0$

Further recall that $\boldsymbol{x}_t = \boldsymbol{x}_0 + \alpha_t \boldsymbol{\epsilon}_0$

reparam. trick!

$\epsilon \sim \mathcal{N}(0, \mathrm{I})$

Therefore $\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_t - \alpha_t \boldsymbol{\epsilon}_0$

- But since the model already knows x_t and α_t, we can just predict unknown ϵ_0

We can have a Neural Network $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{\epsilon}_0$

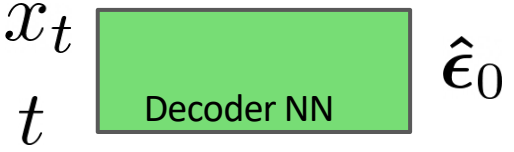


Image and Noise?

They are the same!

What does it mean intuitively?

For arbitrary $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$, we can rewrite it as $\boldsymbol{x}_t = \boldsymbol{x}_0 + \alpha_t \boldsymbol{\epsilon}_0$

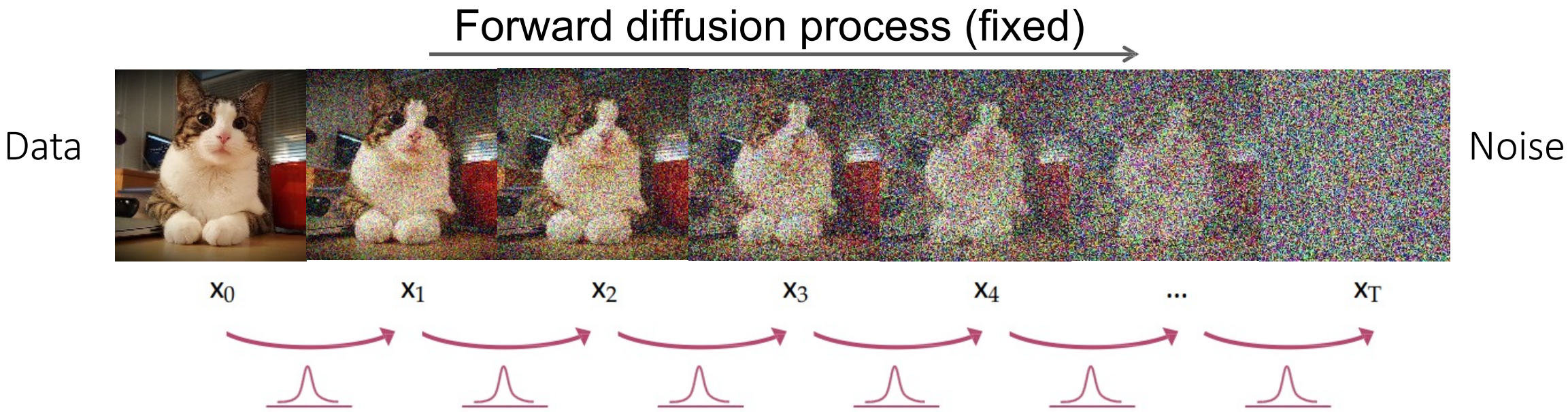
Predicting $\boldsymbol{x}_0$ *determines* $\boldsymbol{\epsilon}_0$ and vice-versa, since they sum to the same thing!

$\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$ $\boldsymbol{x}_0$ $\boldsymbol{\epsilon}_0$

More Exact and Theoretical Perspective

Forward Diffusion Process

- The formal definition of the forward process in T steps:



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

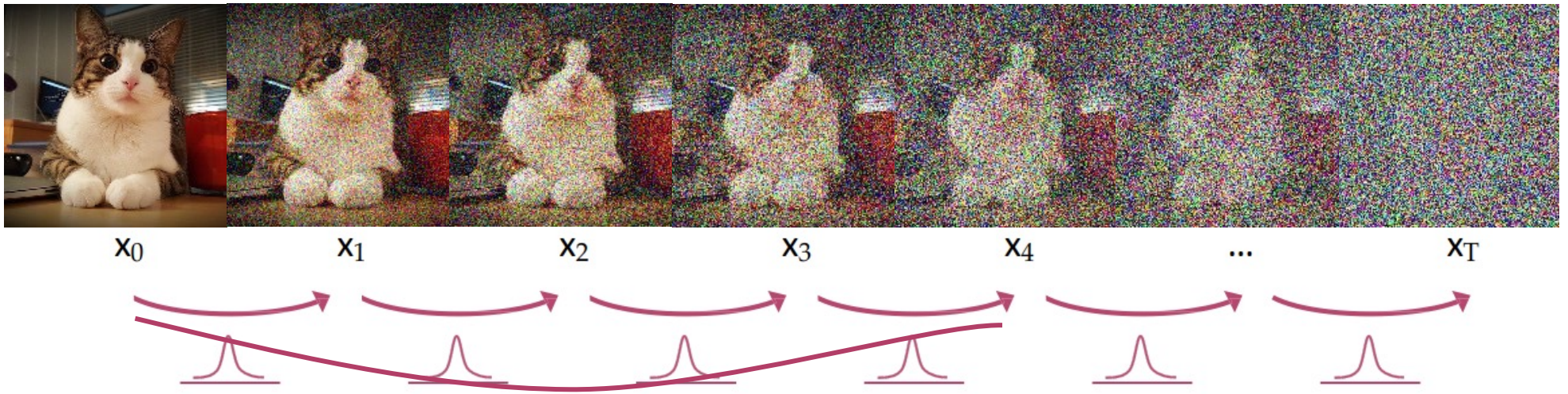
mean variance

➡ Sample: $x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$

where, $\epsilon_{t-1} \sim \mathcal{N}(0, \mathbf{I})$

Diffusion Kernel

Data



Noise

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

mean variance

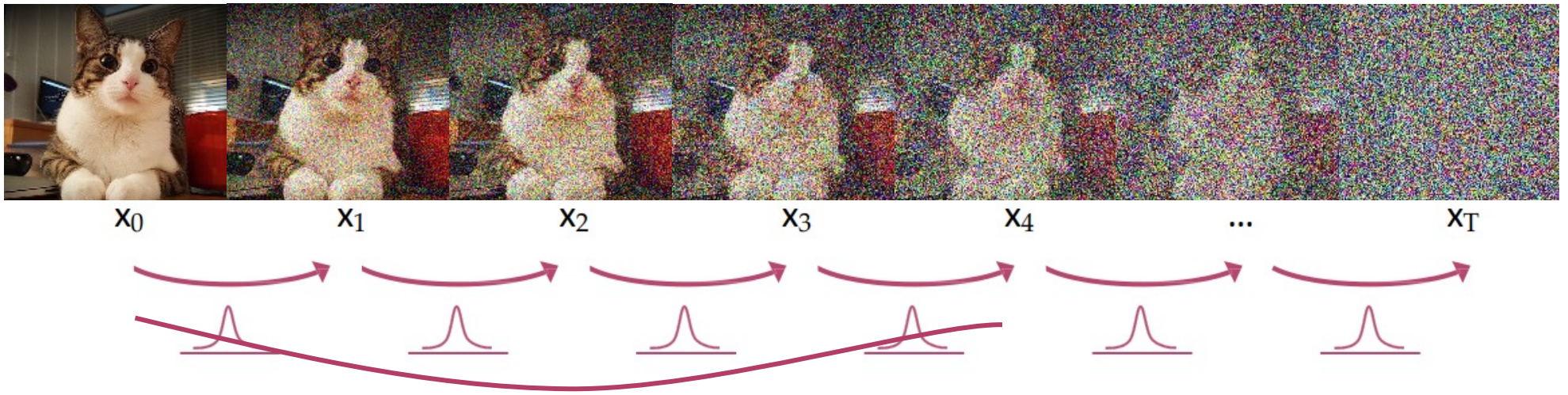
➡ Sample: $x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$

where, $\epsilon_{t-1} \sim \mathcal{N}(0, \mathbf{I})$

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

Diffusion Kernel

Data



Noise

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad \text{Sample: } x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1}$$

mean variance

where, $\epsilon_{t-1} \sim \mathcal{N}(0, \mathbf{I})$

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

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

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

Diffusion Kernel

$$
\begin{aligned}
\boldsymbol{x}_t &= \sqrt{\alpha_t}\boldsymbol{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}^*_{t-1} \\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}^*_{t-2}\right) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}^*_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1}}\boldsymbol{\epsilon}^*_{t-2} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}^*_{t-1} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{\sqrt{\alpha_t - \alpha_t\alpha_{t-1}}^2 + \sqrt{1-\alpha_t}^2}\boldsymbol{\epsilon}_{t-2} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1} + 1 - \alpha_t}\boldsymbol{\epsilon}_{t-2} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\boldsymbol{\epsilon}_{t-2} \\
&= \dots \\
&= \sqrt{\prod_{i=1}^{t}\alpha_i}\boldsymbol{x}_0 + \sqrt{1-\prod_{i=1}^{t}\alpha_i}\boldsymbol{\epsilon}_0 \\
&= \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_0 \\
&\sim \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\,\mathbf{I})
\end{aligned}
$$

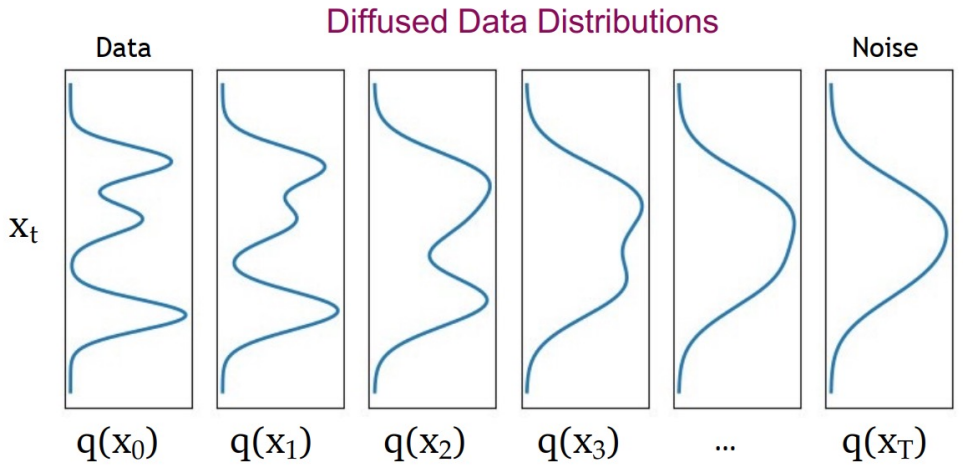
Calvin Luo, Understanding Diffusion Models: A Unified Perspective, 2022

What happens to a distribution in the forward diffusion?

- So far, we discussed the diffusion kernel $q(x_t|x_0)$ but what about $q(x_t)$?

$$\underbrace{q(x_t)}_{\text{Diffused data dist.}} = \int \underbrace{q(x_0, x_t)}_{\text{Joint dist.}} dx_0 = \int \underbrace{q(x_0)}_{\text{Input data dist.}} \underbrace{q(x_t|x_0)}_{\text{Diffusion kernel}} dx_0$$

The diffusion kernel is Gaussian convolution.



Generative Learning by Denoising

- Recall, that the diffusion parameters are designed such that $q(x_T) \approx \mathcal{N}(x_T; 0, I)$
- Generation:

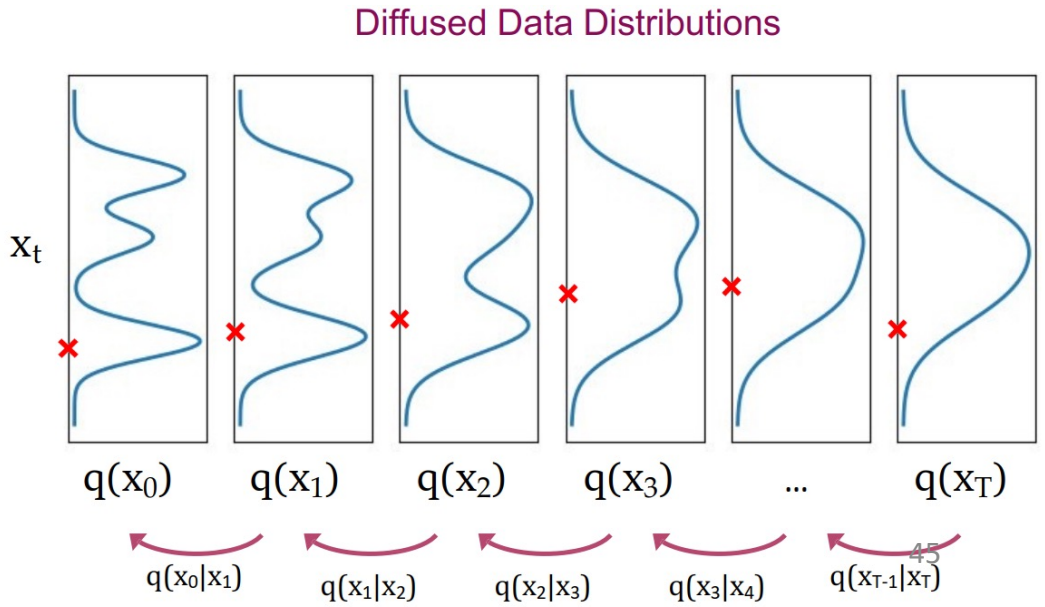
  Sample $x_T \sim \mathcal{N}(x_T; 0, I)$

  Iteratively sample $x_{t-1} \sim q(x_{t-1}|x_t)$ (True denoising dist.)

- In general, $q(x_{t-1}|x_t) \propto q(x_{t-1})q(x_t|x_{t-1})$ is intractable

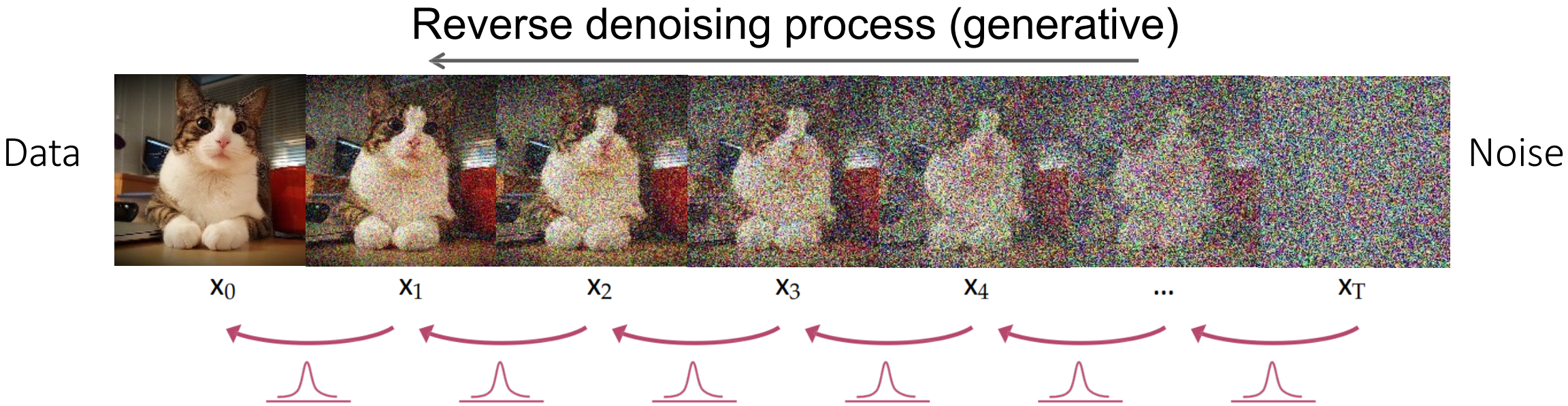
- Can we approximate $q(x_{t-1}|x_t)$?
  - Yes, we can use a Normal dist.

    if β_t is small in each forward diffusion step.



Reverse Denoising Process

- Formal definition of forward and reverse processes in T steps:



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

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

Trainable network

(U-net, Denoising Autoencoder)

Autoencoder predicts the mean of the denoised image x(t-1) given x(t).

Denoising Autoencoders

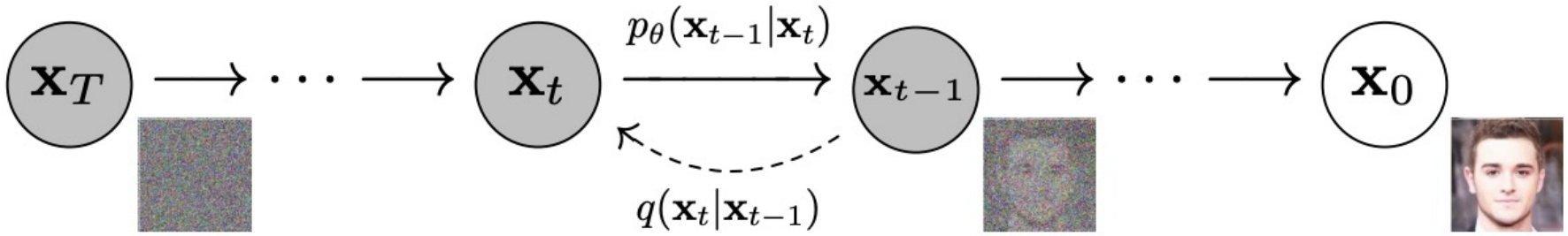
- Learn to denoise or recover x from noise-corrupted data $\hat{x} = x + \epsilon$.
- Learn to project noise-corrupted data back onto the data manifold
- Optimize a reconstruction objective:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathop{\mathbb{E}}_{(x,\tilde{x})\sim p} \|x - g_\theta(f_\phi(\tilde{x}))\|^2$$

Vincent et. al., JMLR 2010

Generative denoising model

- Can we turn denoising autoencoders or U-Nets into a generative model?
- Idea: construct a Markov chain of progressively less noisy samples:



Ho, Jain, and Abbeel, Neurips 2020

- What if each transition $p_\theta(x_{t-1}|x_t)$ were given by a denoising autoencoder?

Learning generative model

- Want to model the distribution of $q(x_0)$, where $x_0 \in \mathbb{R}^d$.
- Construct a Markov chain $x_0, \dots, x_T \in \mathbb{R}^d$ (with base distribution $p(x_T) = \mathcal{N}(0, I)$)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}|\mu_\theta(x_t, t), \sigma_t^2 I)$$

- Marginal distribution over x_0:

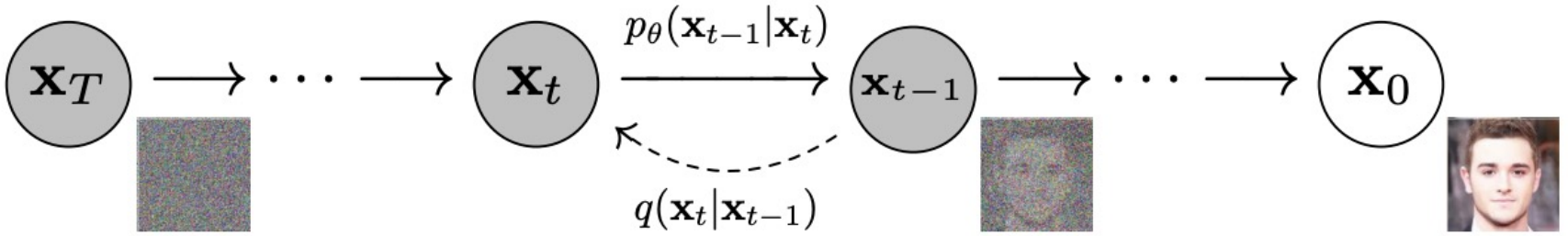
$$p_\theta(x_0) = \int p_\theta(x_0, \dots, x_T) dx_1, \dots, x_T$$

- Learn the parameters so that $q(x_0) \approx p_\theta(x_0)$ by maximum likelihood:

$$\underset{\theta}{\operatorname{argmax}} \underset{x_0 \sim q}{\mathbb{E}} [\log p_\theta(x_0)] = \underset{x_0 \sim q}{\mathbb{E}} \left[\log \int p_\theta(x_0, \dots, x_T) dx_1, \dots, x_T \right]$$

- Likelihood is intractable and thus use the variational approximation!

Posterior Approximation

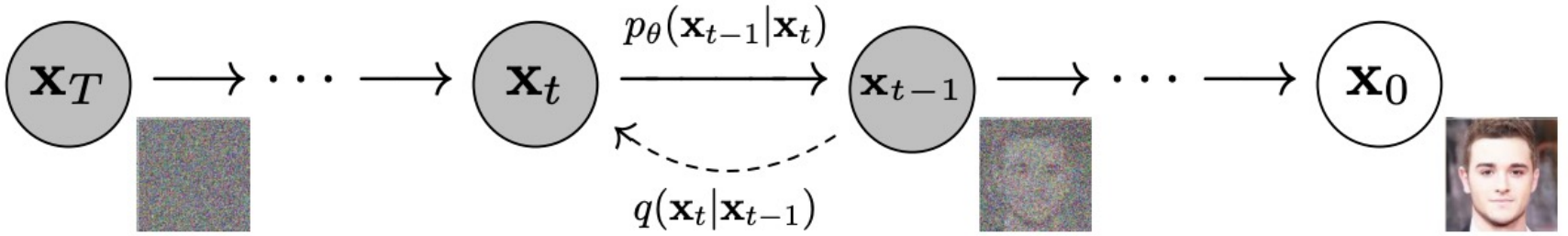


- We want to model the distribution of $q(x_0)$, where $x \in \mathbb{R}^d$.
  - Learn the parameters so that $q(x_0) \approx p_\theta(x_0)$.

- Marginal distribution over x_0 is $p_\theta(x_0) = \int p_\theta(x_0, \dots, x_T) dx_1, \dots, x_T$ that is intractable

- Utilize $q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$ in ELBO (Jensen inequality)

ELBO



$q(x_0) = p_{data}(x_0)$

$$\underset{\theta}{\operatorname{argmax}} \underset{x_0 \sim q}{\mathbb{E}} [\log p_\theta(x_0)]$$

$$= \underset{x_0 \sim q(x_0)}{\mathbb{E}} \left[\log \int q(x_{1:T}|x_0) \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} dx_{1:T} \right]$$

$$= \underset{x_0 \sim q(x_0)}{\mathbb{E}} \left[\log \underset{q(x_{1:T}|x_0)}{\mathbb{E}} \left[\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right] \right]$$

$$\geq \underset{\substack{x_0 \sim q(x_0) \\ x_{1:T} \sim q(x_{1:T}|x_0)}}{\mathbb{E}} \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right]$$

Unpacking the ELBO

$$\mathop{\mathbb{E}}_{\substack{x_0 \sim q(x_0) \\ x_{1:T} \sim q(x_{1:T}|x_0)}} \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right]$$

$$= \mathop{\mathbb{E}}_{\substack{x_0 \sim q(x_0) \\ x_{1:T} \sim q(x_{1:T}|x_0)}} \left[\log \frac{p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)}{\prod_{t=1}^{T} q(x_t|x_{t-1}, x_0)} \right]$$

$$= \mathop{\mathbb{E}}_{x_{0:T} \sim q} \left[\log p(x_T) + \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1}, x_0)} \right]$$

$$= \mathop{\mathbb{E}}_{x_{0:T} \sim q} \left[\log p(x_T) + \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} \frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} \right]$$

$$= \mathop{\mathbb{E}}_{x_{0:T} \sim q} \left[\log p(x_T) + \sum_{t=2}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} + \sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_0)}{q(x_t|x_0)} + \log p_\theta(x_0|x_1) - \log q(x_1|x_0) \right]$$

$$= \mathop{\mathbb{E}}_{x_{0:T} \sim q} \left[\log \frac{p(x_T)}{q(x_T|x_0)} + \sum_{t>1} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t, x_0)} + \log p_\theta(x_0|x_1) \right].$$

Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Closed Form Conditionals

- The forward process: $q(x_t|x_{t-1}) = \mathcal{N}\left(x_t;\ \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$.
- The reverse process: $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}|\mu_\theta(x_t,t), \sigma_t^2 I)$
- The ELBO: $\mathop{\mathbb{E}}_{x_{0:T}\sim q}\left[\log\frac{p(x_T)}{q(x_T|x_0)} + \sum_{t>1}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)} + \log p_\theta(x_0|x_1)\right]$.
- Conditionals have closed form: $q(x_{t-1}|x_t,x_0) = \frac{q(x_t|x_{t-1},x_0)q(x_{t-1}|\,x_0)}{q(x_t|\,x_0)} = \mathcal{N}\left(x_{t-1};\tilde{\mu}(x_t,x_0),\tilde{\beta}_t I\right)$.

  Where: $\tilde{\mu}(x_t,x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$, $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$.
  - $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

A Reconstruction Objective

- The ELBO: $\mathop{\mathbb{E}}_{x_{0:T}\sim q}\left[\log\frac{p(x_T)}{q(x_T|x_0)}+\sum_{t>1}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}+\log p_\theta(x_0|x_1)\right]$

- Optimization of the second term:

$$\mathop{\mathbb{E}}_{x_{0:T}\sim q}\left[\sum_{t>1}\log\frac{p_\theta(x_{t-1}|x_t)}{q(x_{t-1}|x_t,x_0)}\right]$$

$$=-\sum_{t>1}\mathop{\mathbb{E}}_{x_0,\epsilon}\left[D_{KL}\left(q(x_{t-1}|x_t,x_0)\;||\;p_\theta(x_{t-1}|x_t)\right)\right]$$

$$=-\sum_{t>1}\frac{1}{2\sigma_t^2}\underbrace{\mathop{\mathbb{E}}_{x_0,\epsilon}\left[\|\tilde{\mu}(x_t,x_0)-\mu_\theta(x_t,t)\|^2\right]}_{L_t}+C$$

$$D_{KL}(p||q)=\frac{1}{2}\left[\log\frac{|\Sigma_q|}{|\Sigma_p|}-k+(\boldsymbol{\mu_p}-\boldsymbol{\mu_q})^T\Sigma_q^{-1}(\boldsymbol{\mu_p}-\boldsymbol{\mu_q})+tr\left\{\Sigma_q^{-1}\Sigma_p\right\}\right]$$

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Reparameterization

- We can directly compute $q_t(x_t|x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I\right)$.
- $\tilde{\mu}(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$ $\xrightarrow{x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{(1-\bar{\alpha}_t)}\epsilon}$ $\tilde{\mu}(x_t, \epsilon) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$
- Re-parameterize $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$.

$$L_{t-1} = \mathop{\mathbb{E}}_{\substack{x_0 \sim q \\ \epsilon \sim \mathcal{N}(0,I)}}\left[\frac{1}{2\sigma_t^2}\|\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)\|^2\right]$$

$$= \mathop{\mathbb{E}}_{\substack{x_0 \sim q \\ \epsilon \sim \mathcal{N}(0,I)}}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Training objective weighting

$$\mathop{\mathbb{E}}_{\substack{x_0 \sim q(x_0) \\ \epsilon \sim \mathcal{N}(0,I) \\ t \sim U(1,T)}} \left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)}}_{\lambda_t} \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{(1-\bar{\alpha}_t)} \epsilon, t \right) \right\|^2 \right]$$

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

However, this weight is often very large for small ts

Ho et al. NeurIPS 2020 observe that $\lambda_t = 1$ improves sample quality

More advanced weighting: Choi et al., Perception prioritized Training of Diffusion Models, CVPR 2022

How do we train? (summary version)

- What is the loss function? (Ho et al. NeurIPS 2020)

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

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

U-Net autoencoder takes x(t) as input and simply predict a noise. The goal of the training is to generate a noise pattern that is unit normal. Very similar to VAE, right?

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

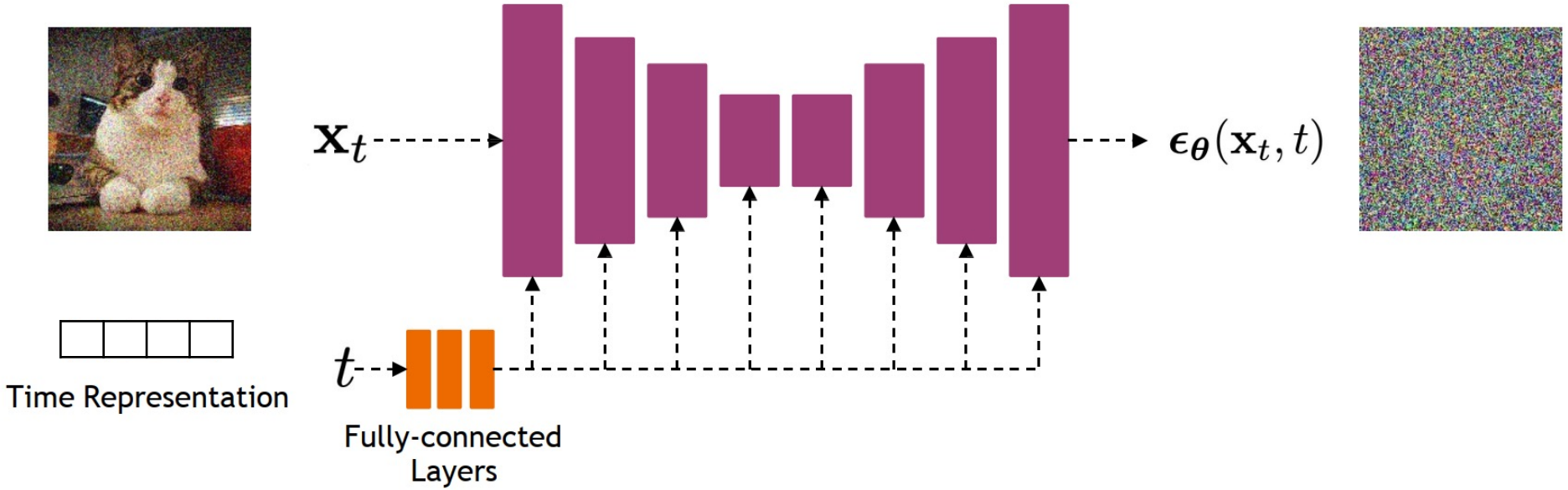
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t,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

- Intuitively: During forward process we add noise to image. During reverse process we try to predict that noise with a U-Net and then subtract it from the image to denoise it.

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Implementation Considerations

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



- 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 Dharivwal and Nichol NeurIPS 2021)

Diffusion Parameters

- Nosie schedule

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

Data  Noise

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

- Above, β_t and σ_t^2 control the variance of the forward diffusion and reverse denoising processes respectively.
- Often a linear schedule is used for β_t and σ_t^2 is set equal to β_t. Slowly increase the amount of added noise.
- Kingma et al. NeurIPS 2022 introduce a new parameterization of diffusion models using signal-to-noise ratio (SNR), and show how to learn the noise schedule by minimizing the variance of the training objective.
- We can also train σ_t^2 while training the diffusion model by minimizing the variational bound (Improved DPM by Nichol and Dhariwal ICML 2021) or after training the diffusion model (Analytic-DPM by Bao et al. ICLR 2022).

Summary – Denoising Diffusion Probabilistic Models

- Diffusion process can be reversed if the variance of the gaussian noise added at each step of the diffusion is **small enough**.
- To **reverse** the process we train a U-Net that takes input: current noisy image and timestamp, and **predicts the noise** map.
- Training goal is to make sure that the predicted noise map at each step is unit gaussian (Note that in VAE we also required the latent space to be unit gaussian).
- During sampling/generation, subtract the predicted noise from the noisy image at time t to generate the image at time $t-1$ (with some weighting).

The devil is in the details:

- Network architectures
- Objective weighting
- Diffusion parameters (i.e., noise schedule)

See "Elucidating the Design Space of Diffusion-Based Generative Models" by Karras et al." for important design decisions.

Analogies

- Training is like Denoising Score Matching.
- Sampling is like Annealed Langevin Dynamics.
- Can we think of a denoising diffusion model as a model trained to optimally step through the annealing levels of the Langevin sampling procedure?

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

(Like Denoising Score Matching)

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
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** $\mathbf{x}_0$

(Like Annealed Langevin Dynamics)

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T$.
1: Initialize $\tilde{\mathbf{x}}_0$
2: **for** $i \leftarrow 1$ to L **do**
3: $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
4: **for** $t \leftarrow 1$ to T **do**
5: Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
6: $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\, \mathbf{z}_t$
7: **end for**
8: $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
9: **end for**
return $\tilde{\mathbf{x}}_T$

Score-based generative modeling

- Stochastic Gradient Langevin Dynamics (SGLD): sample from $p(x)$ using only $\nabla_x \log p(x)$ in a Markov chain of updates:

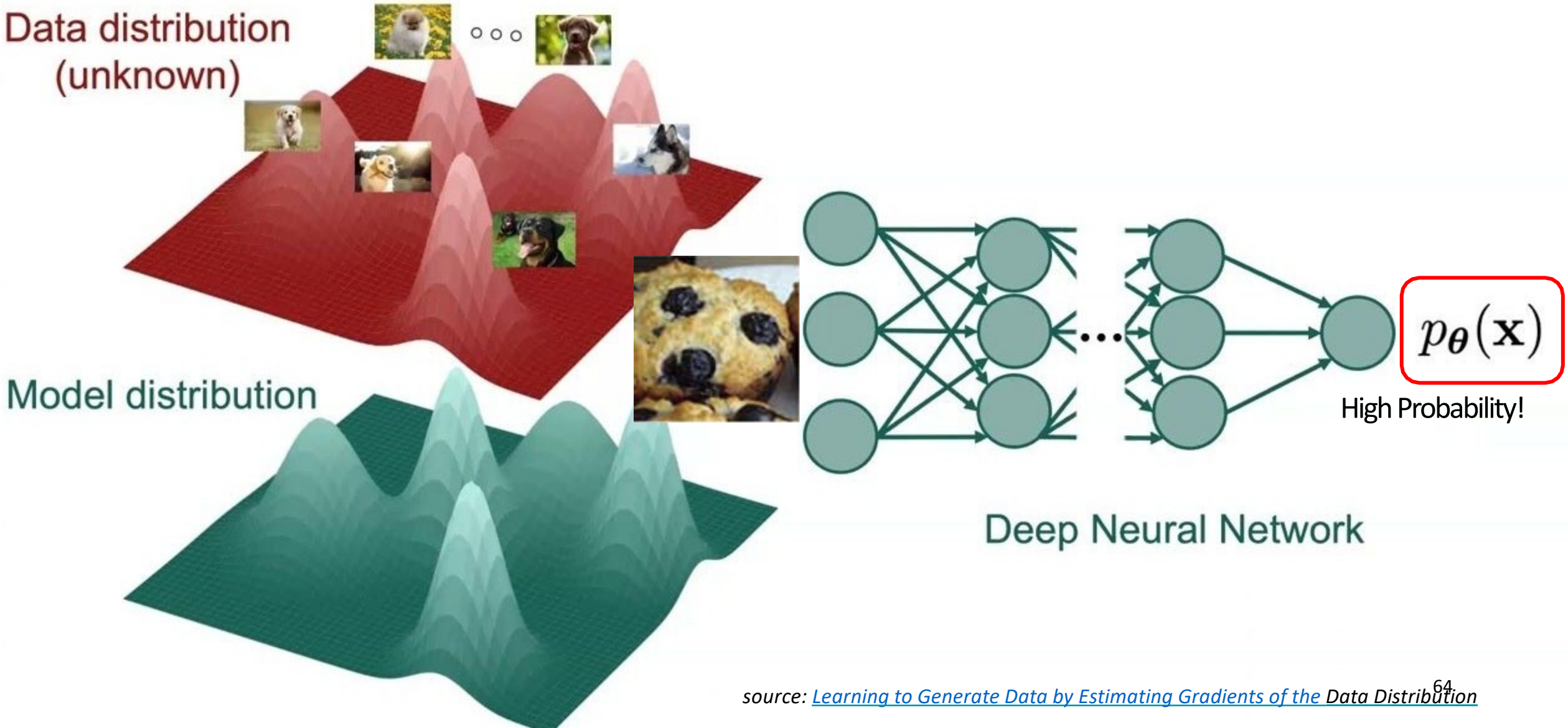
$$x_t = x_{t-1} + \frac{\delta}{2}\nabla_x \log p(x) + \sqrt{\delta}\epsilon_t \qquad \text{where } \epsilon_t \sim N(0, I)$$

When $T \to \infty$ and $\delta \to 0$ it converges to true probability distribution $p(x)$

Score-based generative modeling: use the score network $s_\theta(x)$ to approximate $\nabla_x \log p_{data}(x)$ in the above updates

Song and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", *NeurIPS*, 2019

Recap: Generative Modeling



source: Learning to Generate Data by Estimating Gradients of the Data Distribution

Probability-based Generative Modeling

Why is modeling the **probability** hard in GenMo?

Probabilities, and therefore model outputs, have to be:

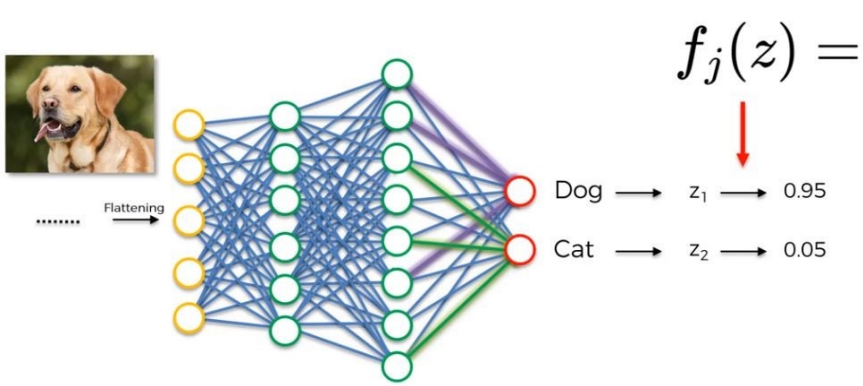
- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$

Probability-based **Discriminative** Modeling

What about modeling the **probability** for classifiers?

Probabilities, and therefore model outputs, have to be:

- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$



$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

How did we do this for discriminative modeling?

- We have a **pre-defined** list of **all** possible outputs (labels), just softmax over it!

Probability-based **Generative** Modeling

Why is modeling the **probability** hard in Generative Modeling?

Probabilities, and therefore model outputs, have to be:

- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$

In Generative Modeling we do not model the probability over all labels for an image…

We model the **probability of all possible images** - there's no way we can pass everything through our model and softmax over the result!

Probability-based **Generative** Modeling

Why is modeling the **probability** hard in GenMo?

Probabilities, and therefore model outputs, have to be:

- Non-negative: $0 \leq p_\theta(\boldsymbol{x})$
- Less than or equal to 1: $p_\theta(\boldsymbol{x}) \leq 1$
- Sum to 1 for the **entire** space: $\int_{\boldsymbol{x}} p_\theta(\boldsymbol{x})\, d\boldsymbol{x} = 1$

This puts a lot of architectural burden on our network, to output ***valid*** probabilities!

…but neural nets are really good at producing *flexible*, *unconstrained* values

A simple observation…

All probability distributions can be written as:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})}$$

This is a concept from thermodynamics, where the $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is a flexible, unconstrained value called the ***energy***.

Z_θ is a normalization constant, computed as: $Z_{\boldsymbol{\theta}} = \int_{\boldsymbol{x}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})} d\boldsymbol{x}$

We have divorced the probability rules from what we need to model!

Energy-based Generative Modeling

Idea: Let's just model the energy function $f_\theta(x)$ using a flexible neural network!

$f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called an *energy-based model (EBM)* or unnormalized probabilistic model.

How do we train our energy function?

- We can try interpreting it as a probability: $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})}$

- Then we can maximize log likelihood as before: $\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i)$

- What is the problem with this? $Z_{\boldsymbol{\theta}} = \int_{\boldsymbol{x}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})} d\boldsymbol{x}$

Intractable for complex parameterizations!

Another simple observation…

How do we avoid calculating the normalization constant?

Remember that Z_θ is a *constant* that only depends on parameters θ

Then, if we take the input gradient of the log of the probability:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \qquad \left(\frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})}\right)$$

Another simple observation…

How do we avoid calculating the normalization constant?

Remember that Z_θ is a *constant* that only depends on parameters θ

Then, if we take the input gradient of the log of the probability:

$$\begin{aligned}
\nabla_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \nabla_{\boldsymbol{x}} \log\left(\frac{1}{Z_{\boldsymbol{\theta}}} e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})}\right) \\
&= \nabla_{\boldsymbol{x}} \log \frac{1}{Z_{\boldsymbol{\theta}}} + \nabla_{\boldsymbol{x}} \log e^{-f_{\boldsymbol{\theta}}(\boldsymbol{x})} \\
&= -\nabla_{\boldsymbol{x}} f_{\boldsymbol{\theta}}(\boldsymbol{x}) \\
&\approx \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})
\end{aligned}$$

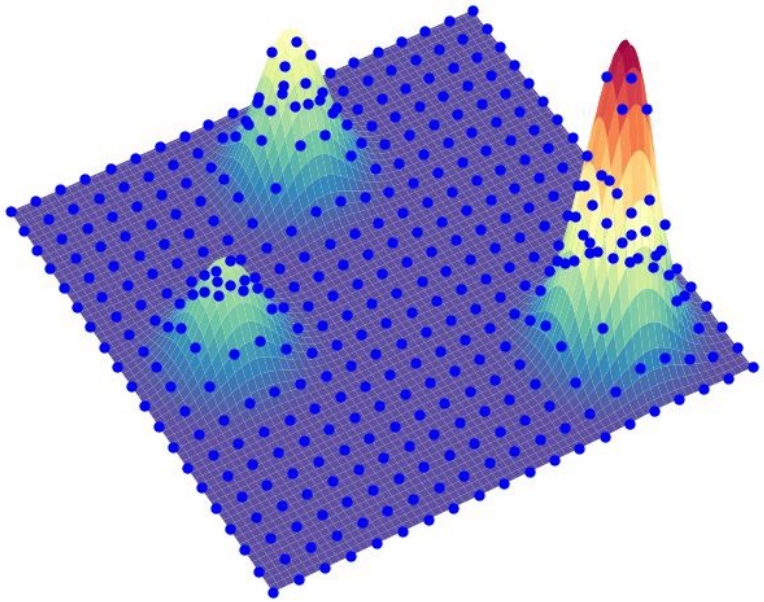
The $\nabla_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called the ***score function***.

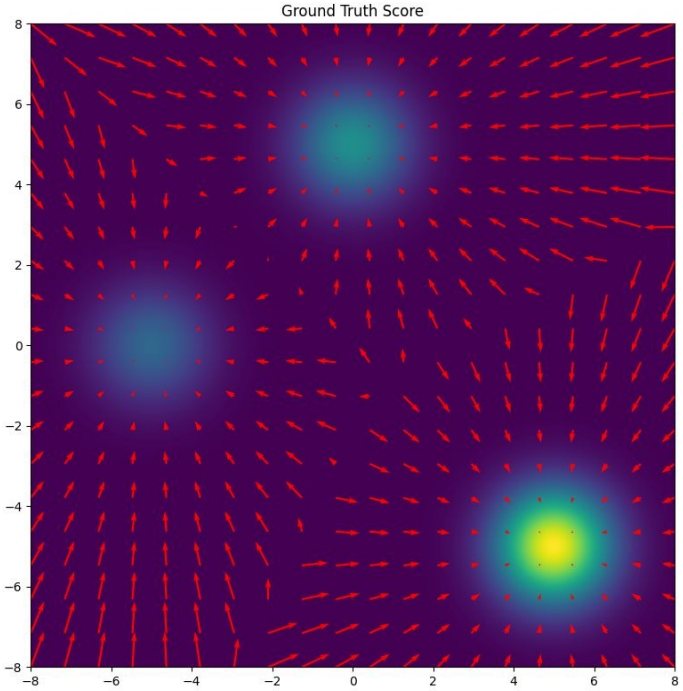
Score Functions

What are score functions?

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

Intuitively, it describes how to move in **data space** to improve the (log) likelihood.





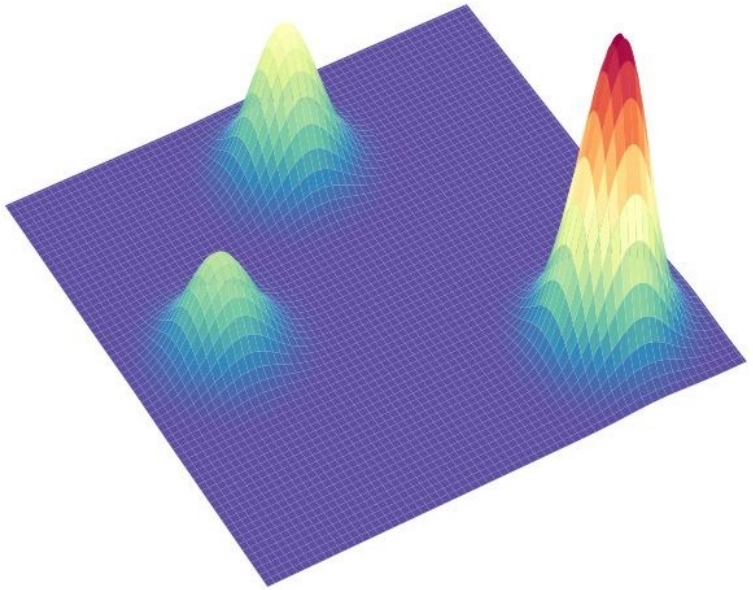
Score Functions

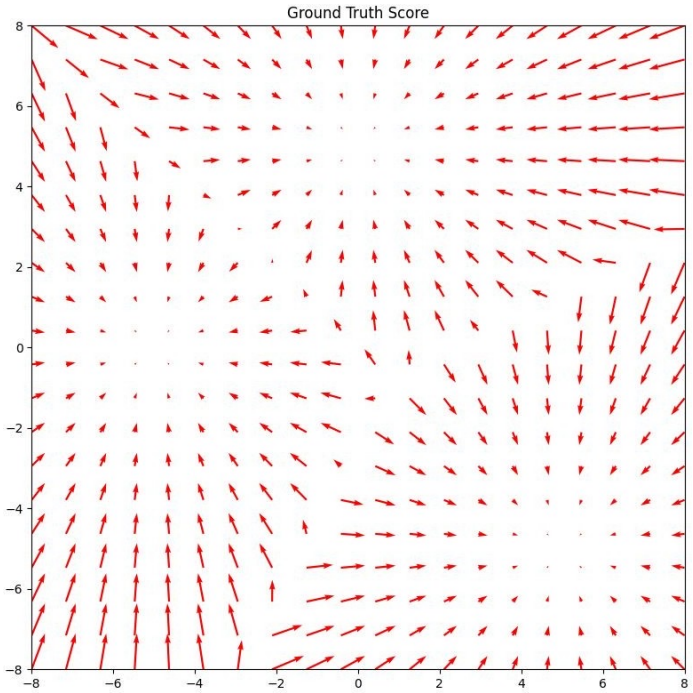
What are score functions?

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

$p_{data}(\boldsymbol{x})$

Intuitively, it describes how to move in **data space** to improve the (log) likelihood.





Score-based Generative Modeling

Idea: Let's just model the score function $s_\theta(\boldsymbol{x})$ using a flexible neural network!

The score is still an unconstrained value, which is attractive to model directly.

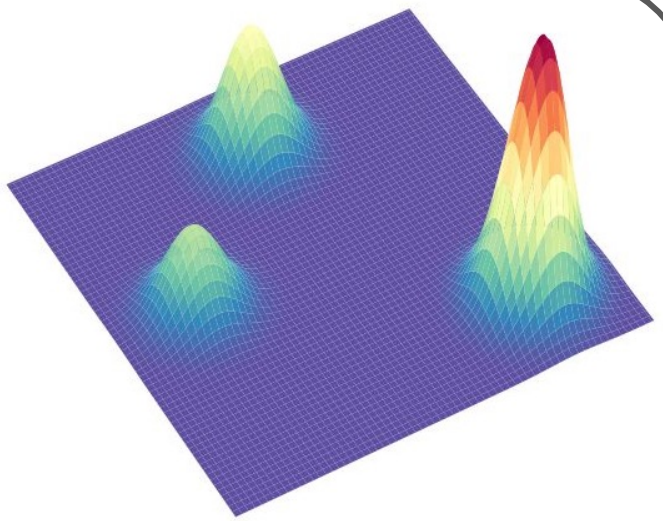
How do we train our score function?

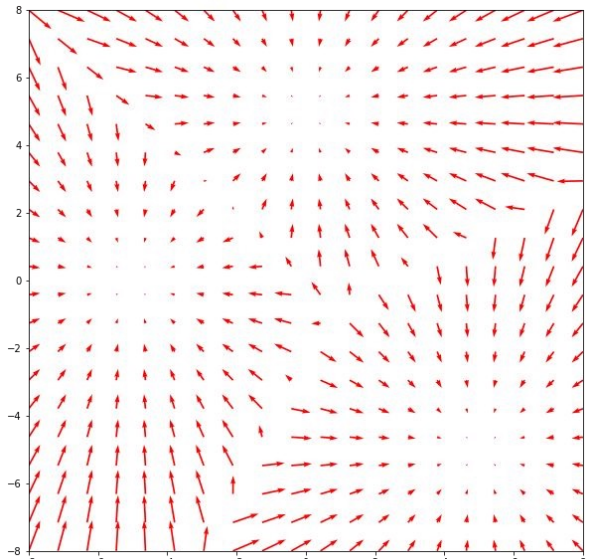
- Minimize the Fisher Divergence between the ground truth and predicted score

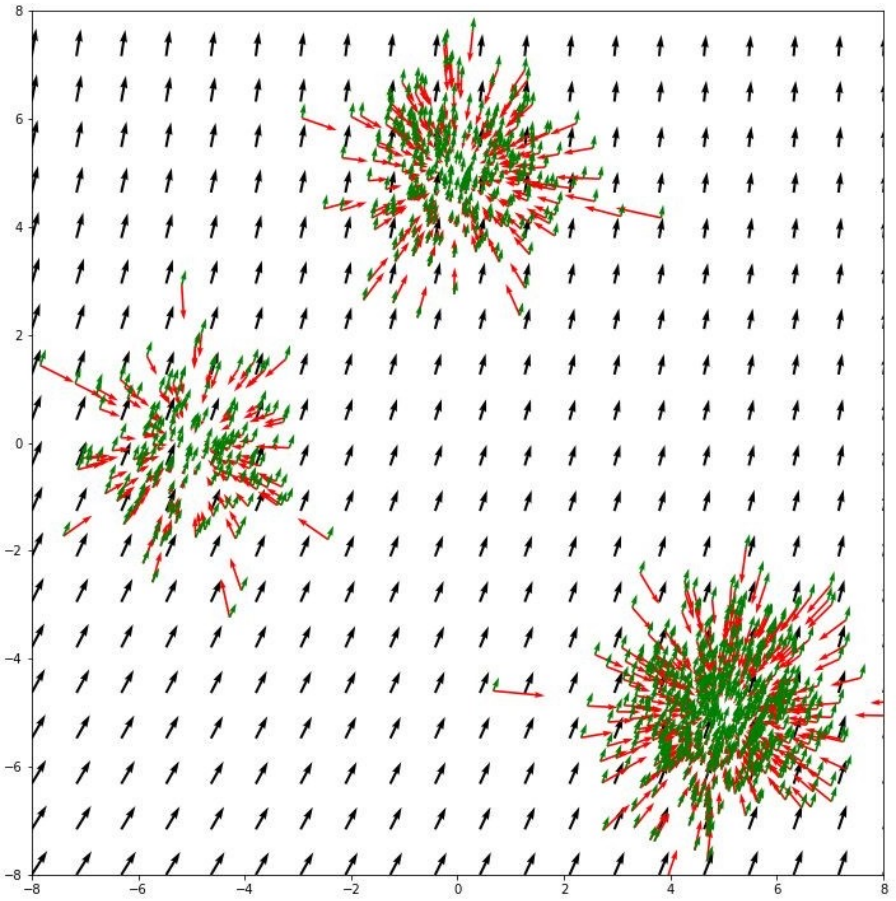
$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s_\theta}(\boldsymbol{x})\|_2^2\right]$$

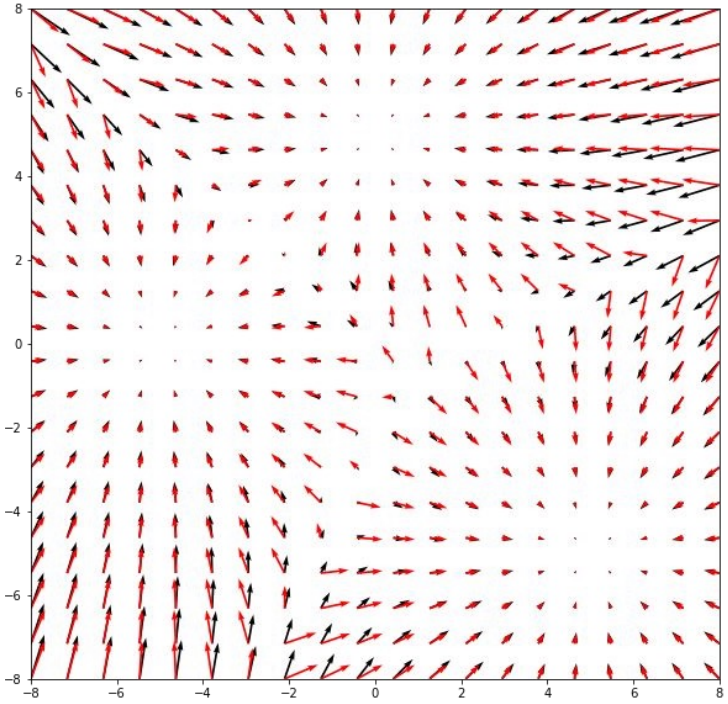
- Intuitively this is simply minimizing the L2-distance between our score model and the ground truth score

Score-based Generative Modeling





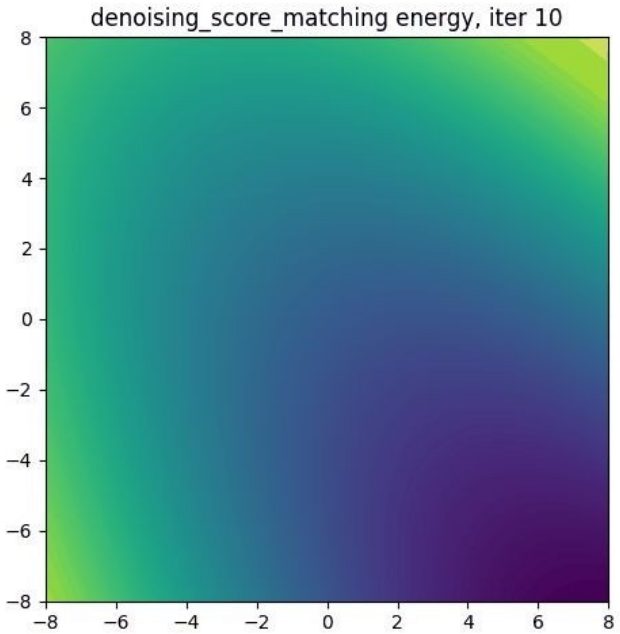


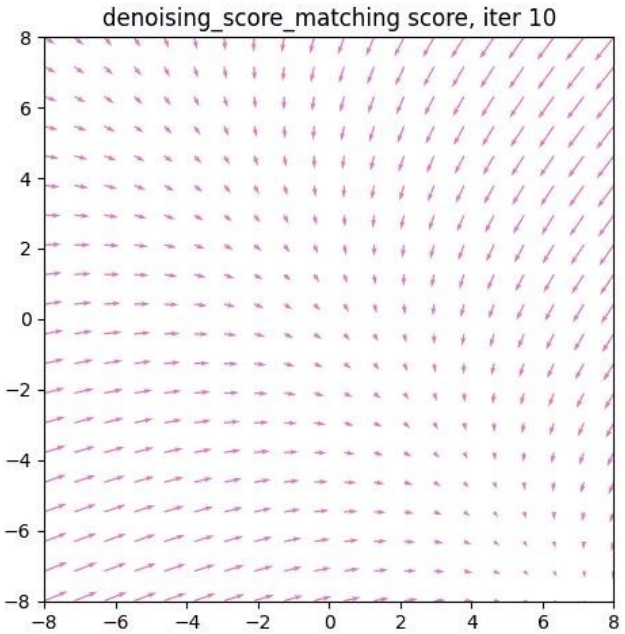


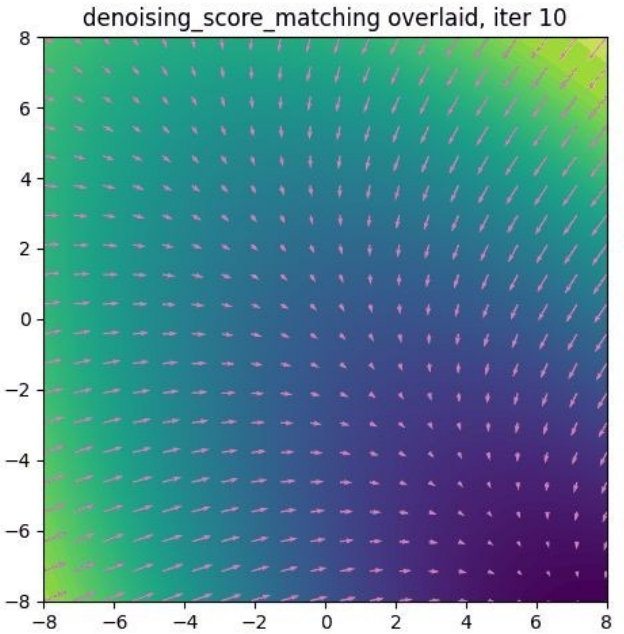
Score-based Generative Modeling

To drive the point home, score-based generative modeling is a way to implicitly model the energy function $f_\theta(\boldsymbol{x})$

We can







Score-based generative modeling

- Stochastic Gradient Langevin Dynamics (SGLD): sample from $p(x)$ using only $\nabla_x \log p(x)$ in a Markov chain of updates:

$$x_t = x_{t-1} + \frac{\delta}{2}\nabla_x \log p(x) + \sqrt{\delta}\epsilon_t \qquad \text{where } \epsilon_t \sim N(0, I)$$

When $T \to \infty$ and $\delta \to 0$ it converges to true probability distribution $p(x)$

Score-based generative modeling: use the score network $s_\theta(x)$ to approximate $\nabla_x \log p(x)$ in the above updates

Song and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", *NeurIPS*, 2019

Score function sampling

- Learn a score function s_θ: $\mathbb{R}^d \to \mathbb{R}^d$ to approximate $s_\theta(x) = \nabla_x \log p(x)$.
- Sample via Langevin dynamics:
  - $x_{t+1} = x_t + \frac{\delta}{2} \nabla_x \log p(x_t) + \sqrt{\delta}\epsilon_t$
    $= x_t + \frac{\delta}{2} s_\theta(x_t) + \sqrt{\delta}\epsilon_t$
- In continuous limit as $\delta \to 0$, $D(x_t || p) \to 0$ as $t \to \infty$.
- How long will I need to run this Markov chain? A long long time.

Score-based Generative Modeling

What is the problem with this optimization objective?

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

How do we get the ground truth score value for complex distributions?

- This is not generally assumed to be known

- The score tells how to change each element of the input to increase its likelihood
  - For images, this is like saying how we can change each pixel to make each image more image-like. We don't really have access to this function!

Score Matching

Fortunately, there are a class of methods called score matching that minimize the Fisher Divergence without needing to know the ground-truth score!

Ground Truth Score Matching: $$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

Hyvarinen Score Matching: $$\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

Sliced Score Matching: $$\mathbb{E}_{p_{\mathrm{v}}}\mathbb{E}_{p(\boldsymbol{x})}\left[\mathbf{v}^{\intercal}\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\mathbf{v} + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

Denoising Score Matching: $$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})}[\|\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - \mathbf{s}_\theta(\tilde{\mathbf{x}})\|_2^2]$$

Hyvarinen Score Matching

Hyvarinen (2005) utilized integration by parts to remove the unknown $\nabla \log p(\boldsymbol{x})$

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|\nabla \log p(\boldsymbol{x}) - \boldsymbol{s_\theta}(\boldsymbol{x})\|_2^2\right]$$

$$L(\theta) = \frac{1}{2}\int p(\boldsymbol{x})\left[s_\theta(\boldsymbol{x})^\top s_\theta(\boldsymbol{x}) - 2s_\theta(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x}) + \nabla \log p(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x})\right] d\boldsymbol{x}$$

$$= \frac{1}{2}\int p(\boldsymbol{x})\left[s_\theta(\boldsymbol{x})^\top s_\theta(\boldsymbol{x}) - 2s_\theta(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x})\right] d\boldsymbol{x}$$

$$\int p(\boldsymbol{x}) s_\theta(\boldsymbol{x})^\top \nabla \log p(\boldsymbol{x}) d\boldsymbol{x} = \int s_\theta(\boldsymbol{x})^\top \nabla p(\boldsymbol{x}) d\boldsymbol{x}$$

log-deriv trick:

$$p(\boldsymbol{x})\nabla \log p(\boldsymbol{x}) = \nabla p(\boldsymbol{x})$$

$$= \left[p(\boldsymbol{x}) s_\theta(\boldsymbol{x})\right]_{-\infty}^{\infty} - \int p(\boldsymbol{x}) \nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}) d\boldsymbol{x}$$

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\mathrm{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

Hyvarinen Score Matching

$$\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

We have gotten rid of unknown $\nabla \log p(\boldsymbol{x})$ 🎉

But now we have to compute $\operatorname{tr}(\nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}))$

When our data has high dimensionality, this is **not cheap** - many nested backprops!

$$\nabla_{\mathbf{x}} s_\theta(\mathbf{x}) = \begin{pmatrix} \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,1}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,2}(\mathbf{x})}{\partial x_3} \\ \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_1} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_2} & \frac{\partial s_{\theta,3}(\mathbf{x})}{\partial x_3} \end{pmatrix}$$

figure link

Sliced Score Matching

Song (2019) utilized random projections to estimate the expensive $\operatorname{tr}(\nabla_{\boldsymbol{x}} s_\theta(\boldsymbol{x}))$

Hyvarinen: $\mathbb{E}_{p(\boldsymbol{x})}\left[\operatorname{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

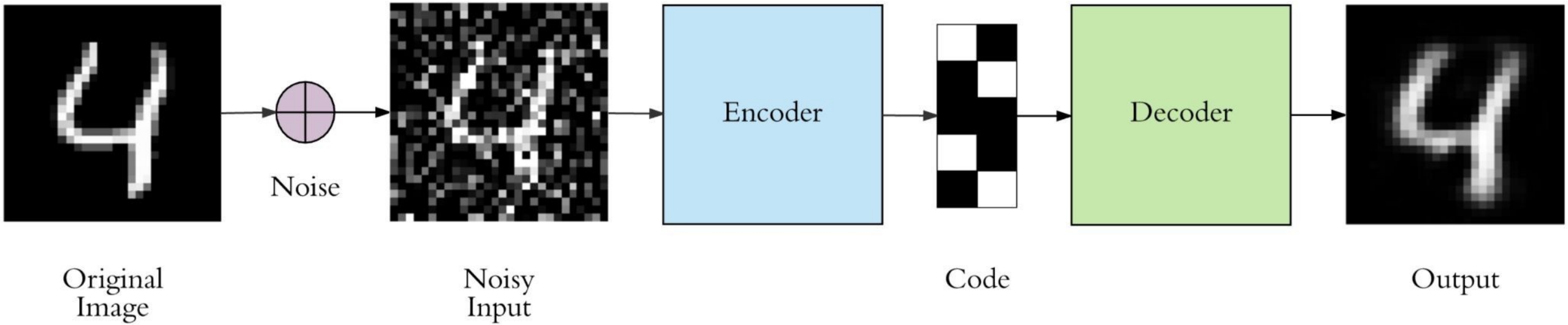
Sliced Score Matching: $\mathbb{E}_{p_{\mathbf{v}}}\mathbb{E}_{p(\boldsymbol{x})}\left[\mathbf{v}^\intercal \nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\mathbf{v} + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$

Intuition - when we project to a lower dimension, the problem becomes tractable.

- $p_{\mathbf{v}}$ is a simple distribution of random vectors, e.g. the multivariate std. normal.

Denoising Autoencoders

Denoising Autoencoders work as follows:



The rationale is that this minimizes “memorization” - the input is corrupted from the start!

Denoising Score Matching

Vincent (2010) proved that matching the score for a noisy perturbation of the input can also minimize the score for the ground truth estimator.

The intuition is that following the gradient of some simple Gaussian perturbation of an input should move us towards the original clean input.

$$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})}[\|\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - \mathbf{s}_\theta(\tilde{\mathbf{x}})\|_2^2]$$

With a simple gaussian for $q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\tilde{\boldsymbol{x}}; \boldsymbol{x}, \sigma^2)$

We know that indeed: $\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = -\frac{1}{\sigma^2}(\tilde{x} - x)$

Denoising Score Matching

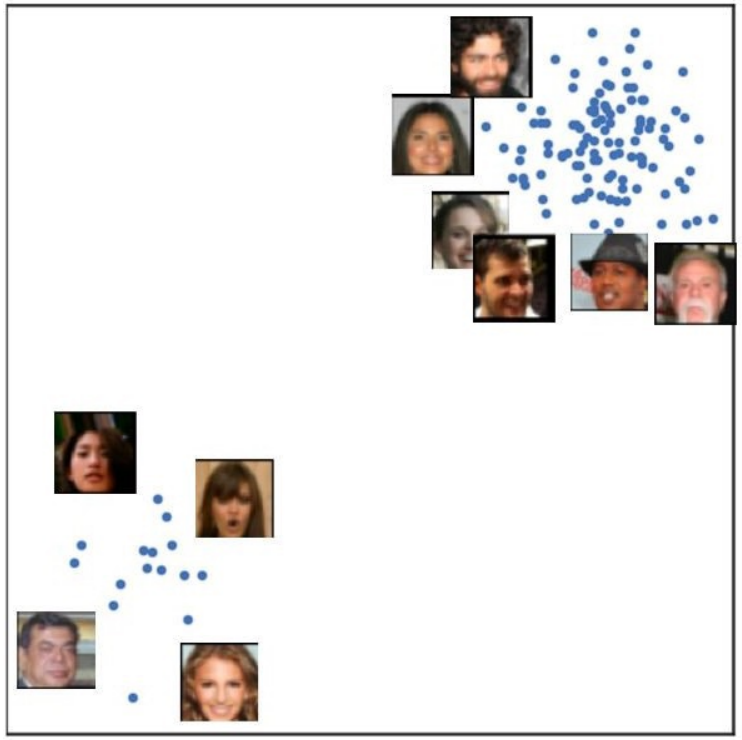
$$\mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})}[\|\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) - \mathbf{s}_\theta(\tilde{\mathbf{x}})\|_2^2]$$

A simple algorithm:

- Take in your input sample
- Perturb it with some Gaussian noise
- Compute the score estimate for the noisy sample
- Compare it with the ground truth score computed by the noising Gaussian

$$\nabla_{\tilde{x}} \log q_\sigma(\tilde{x}|x) = -\frac{1}{\sigma^2}(\tilde{x} - x)$$

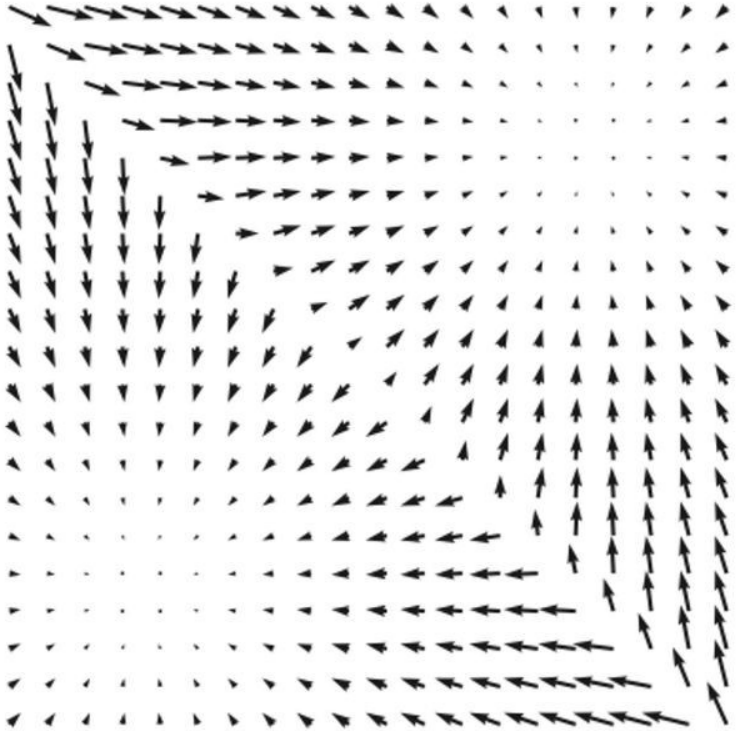
Score-based Generative Modeling: Summary



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

score matching



Scores

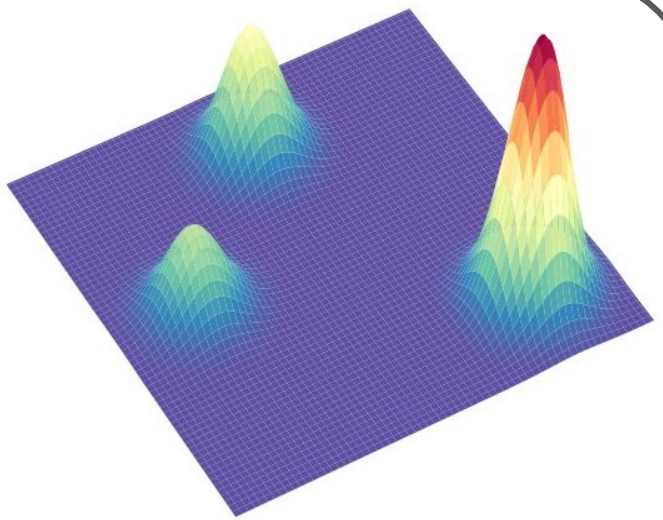
$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

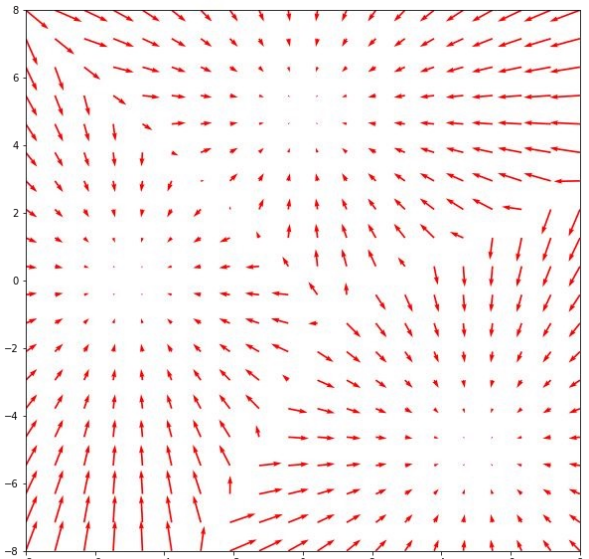
Langevin dynamics

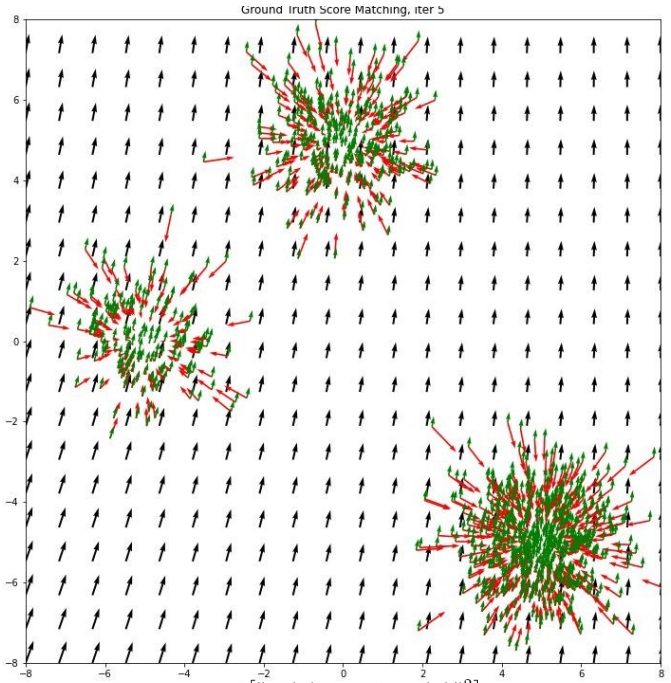
New samples

source: Generative Modeling by Estimating Gradients of the Data Distribution

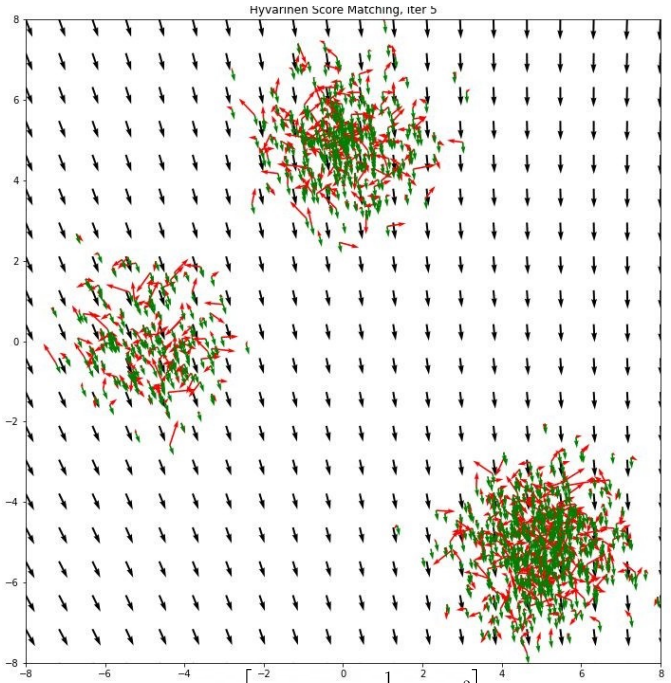
Score Matching



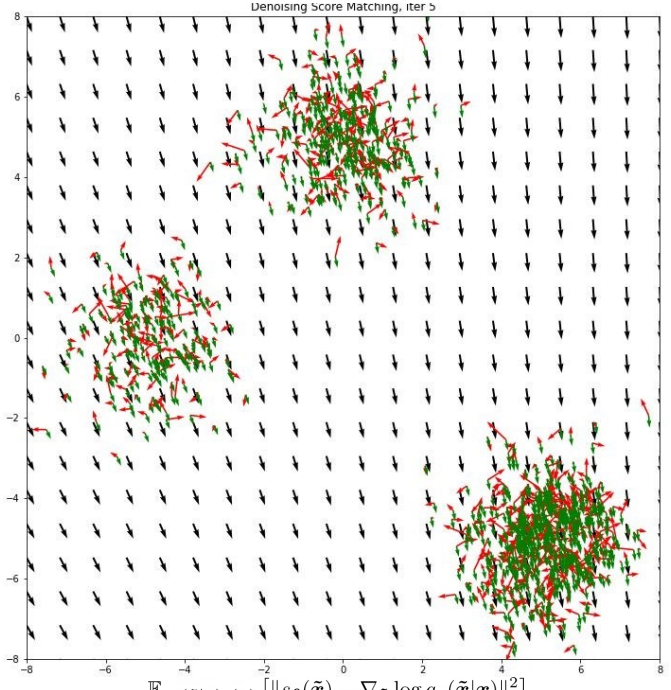




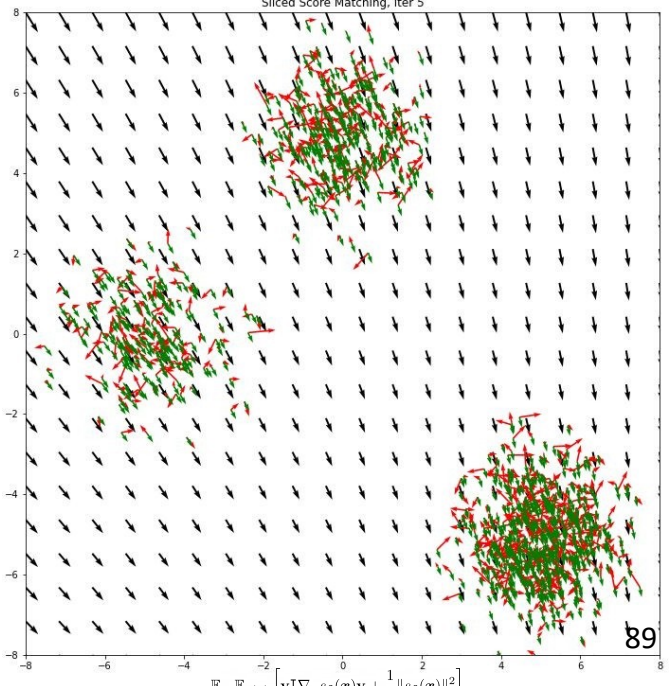
$$\mathbb{E}_{p(\boldsymbol{x})}\left[\|s_{\boldsymbol{\theta}}(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})\|_2^2\right]$$



$$\mathbb{E}_{p(\boldsymbol{x})}\left[\mathrm{tr}(\nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})) + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$



$$\mathbb{E}_{q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})p(\boldsymbol{x})}\left[\|s_{\boldsymbol{\theta}}(\tilde{\boldsymbol{x}}) - \nabla_{\tilde{\boldsymbol{x}}} \log q_\sigma(\tilde{\boldsymbol{x}}|\boldsymbol{x})\|_2^2\right]$$



$$\mathbb{E}_{p_v}\mathbb{E}_{p(\boldsymbol{x})}\left[\mathbf{v}^\intercal \nabla_{\boldsymbol{x}} s_{\boldsymbol{\theta}}(\boldsymbol{x})\mathbf{v} + \frac{1}{2}\|s_{\boldsymbol{\theta}}(\boldsymbol{x})\|_2^2\right]$$

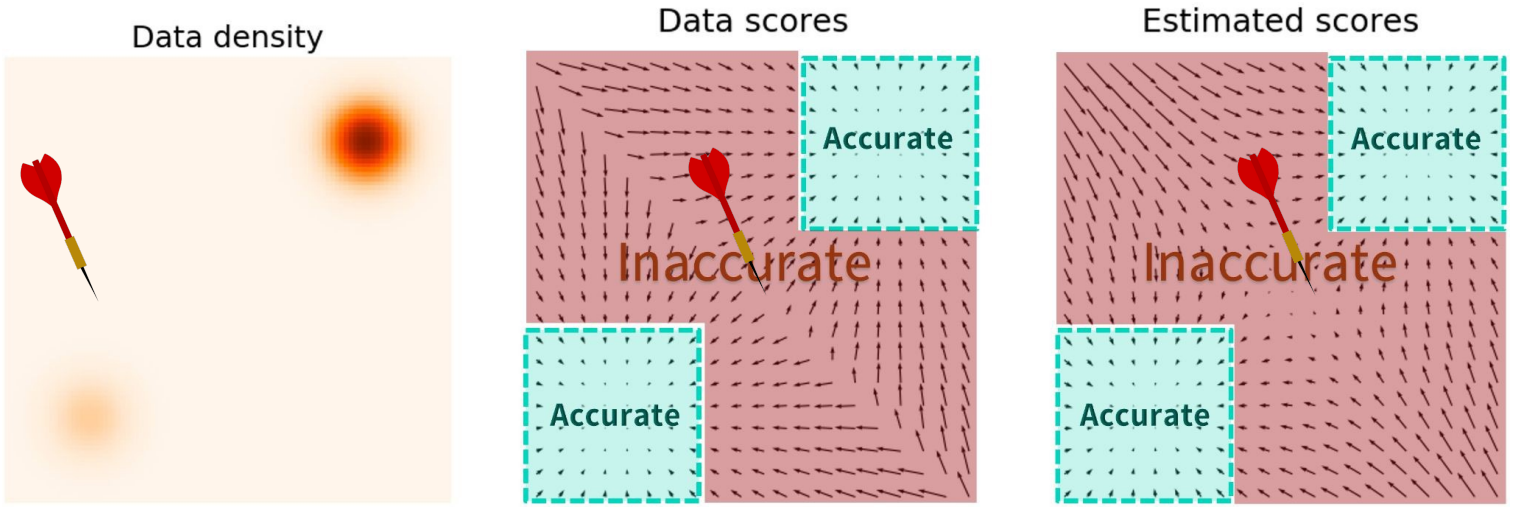
Score-based Generative Modeling

What is the problem with vanilla score-matching?

$$\mathbb{E}_{p(\boldsymbol{x})}[\|\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) - \boldsymbol{s}_\theta(\boldsymbol{x})\|_2^2] = \int p(\boldsymbol{x})\|\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) - \boldsymbol{s}_\theta(\boldsymbol{x})\|_2^2 \mathrm{d}\boldsymbol{x}$$

Our model of the score will not learn the low-density regions well

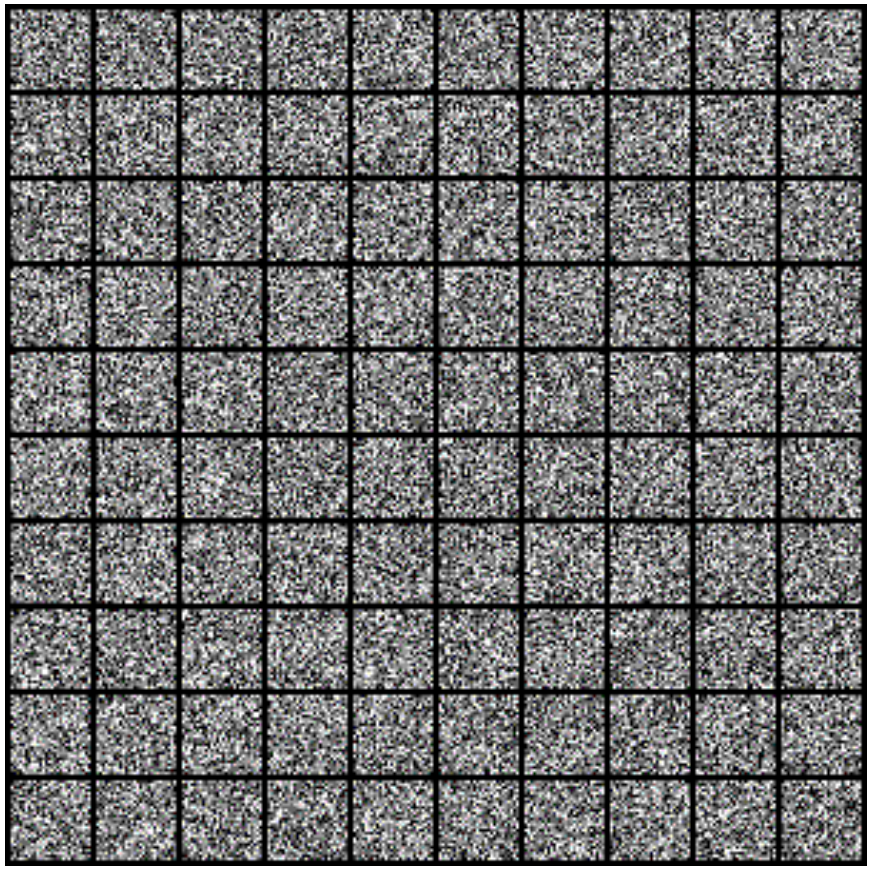
perform sampling with Langevin Dynamics we most likely will initialize as noise!

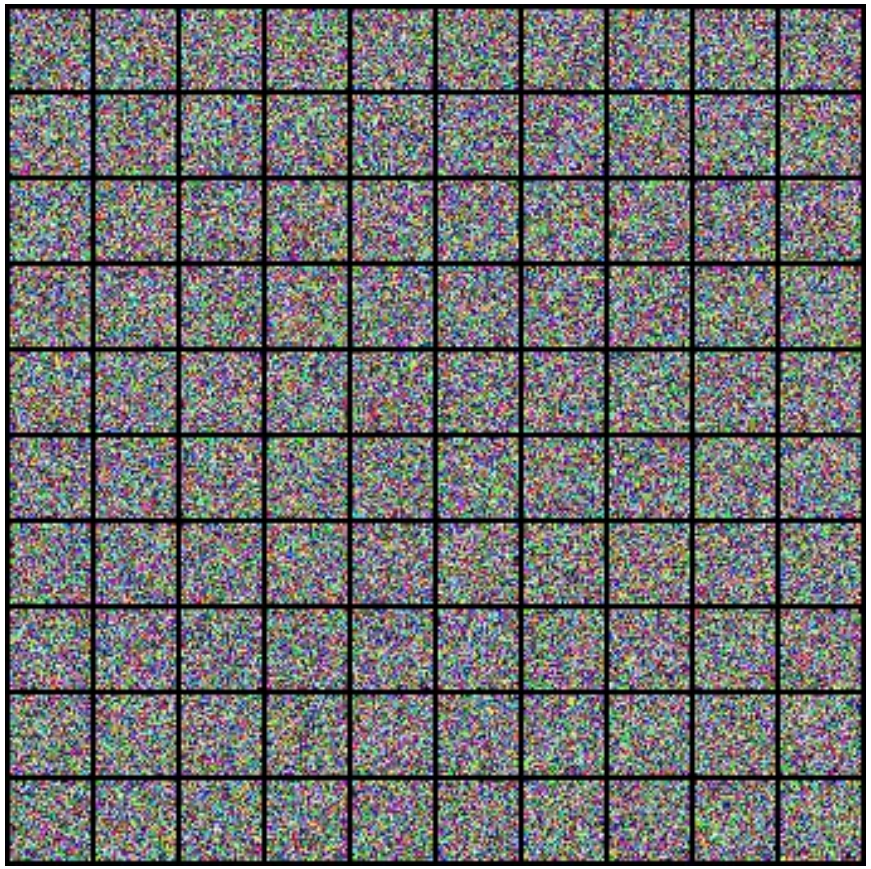


source: Generative Modeling by Estimating Gradients of the Data Distribution

Langevin Dynamics in Practice

- Sampling from MNIST (left) and CIFAR-10 (right) Using Langevin Dynamics



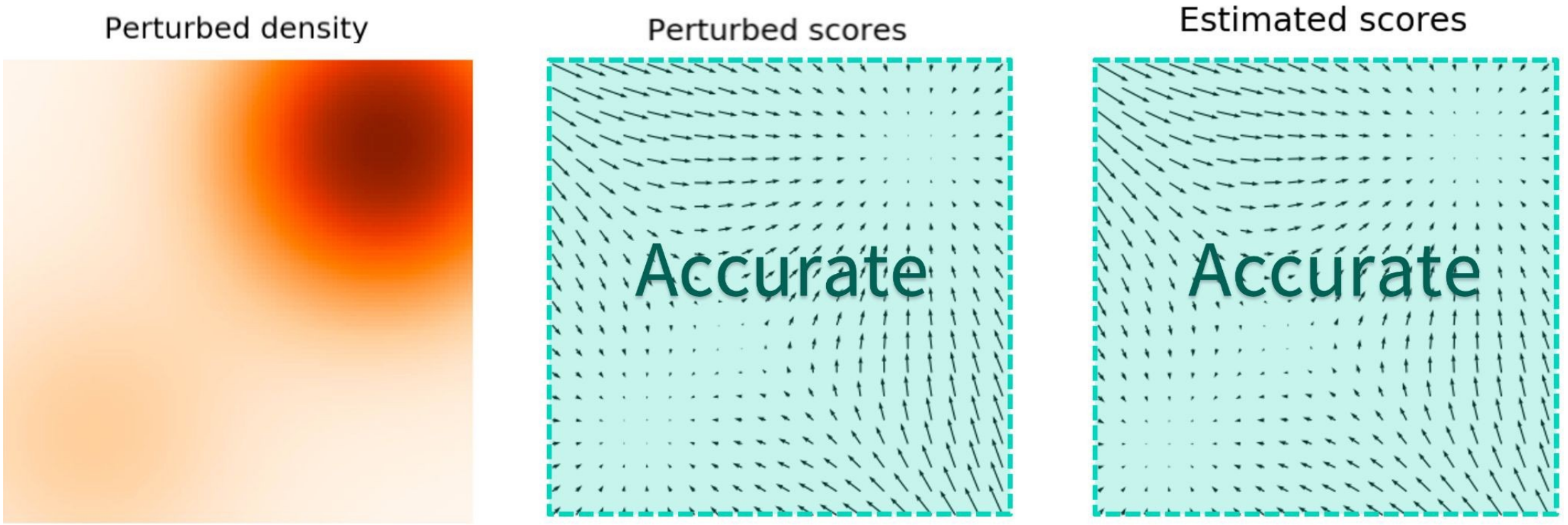


Song and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", *NeurIPS*, 2019

Score-based Generative Modeling

What is the solution for vanilla score-matching?

- Adding Gaussian noise!



source: Generative Modeling by Estimating Gradients of the Data Distribution

Score-based Generative Modeling

How do we choose an appropriate noise scale for the perturbation process?

Larger noise can covers more regions for better score estimation

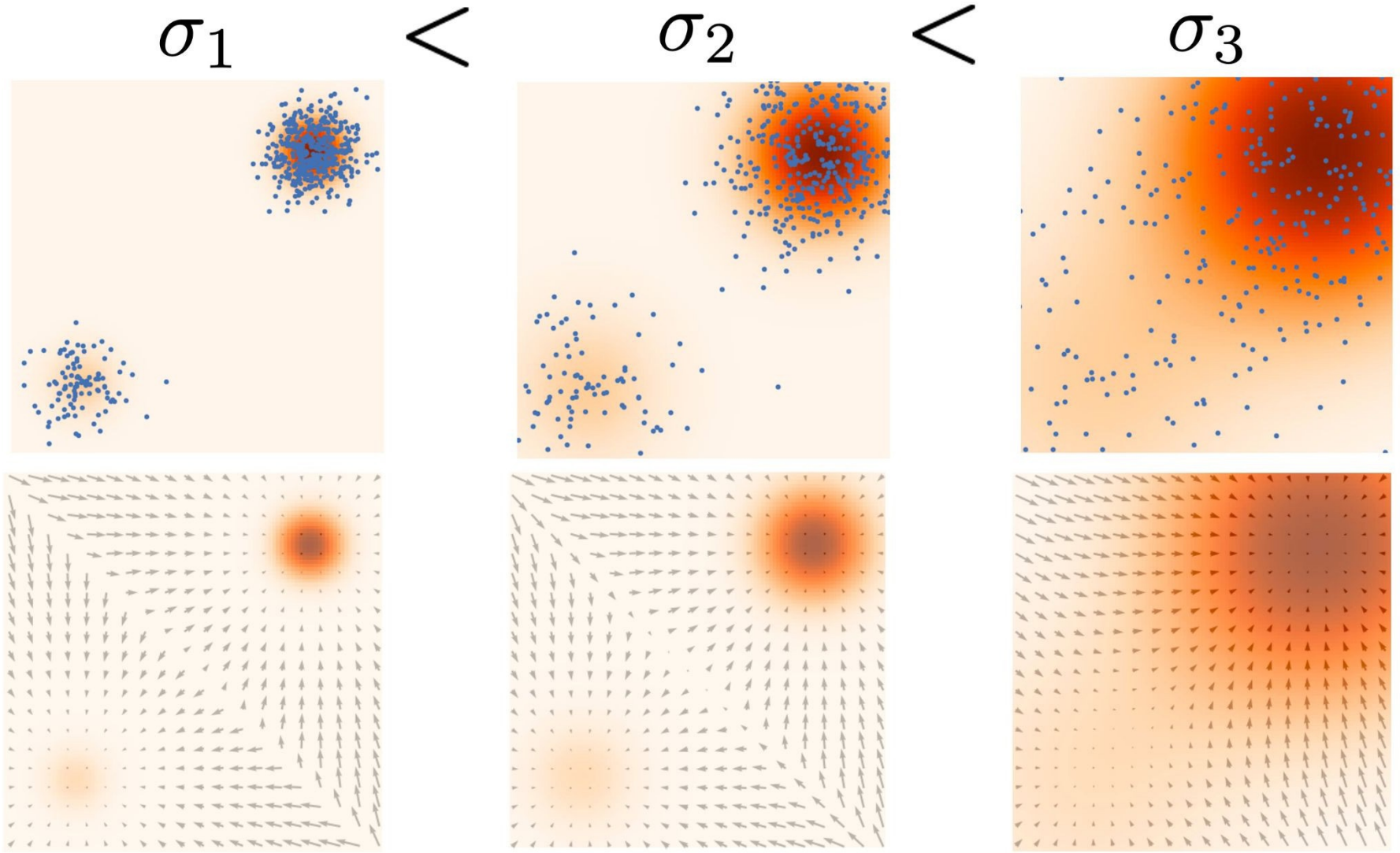
- Over-corrupts data and alters it significantly from the original distribution.

Smaller noise, corrupts original data distribution less

- Does not cover the low density regions as well as we would like.

To achieve the best of both worlds, we use multiple scales of noise perturbations simultaneously!

Score-based Generative Modeling

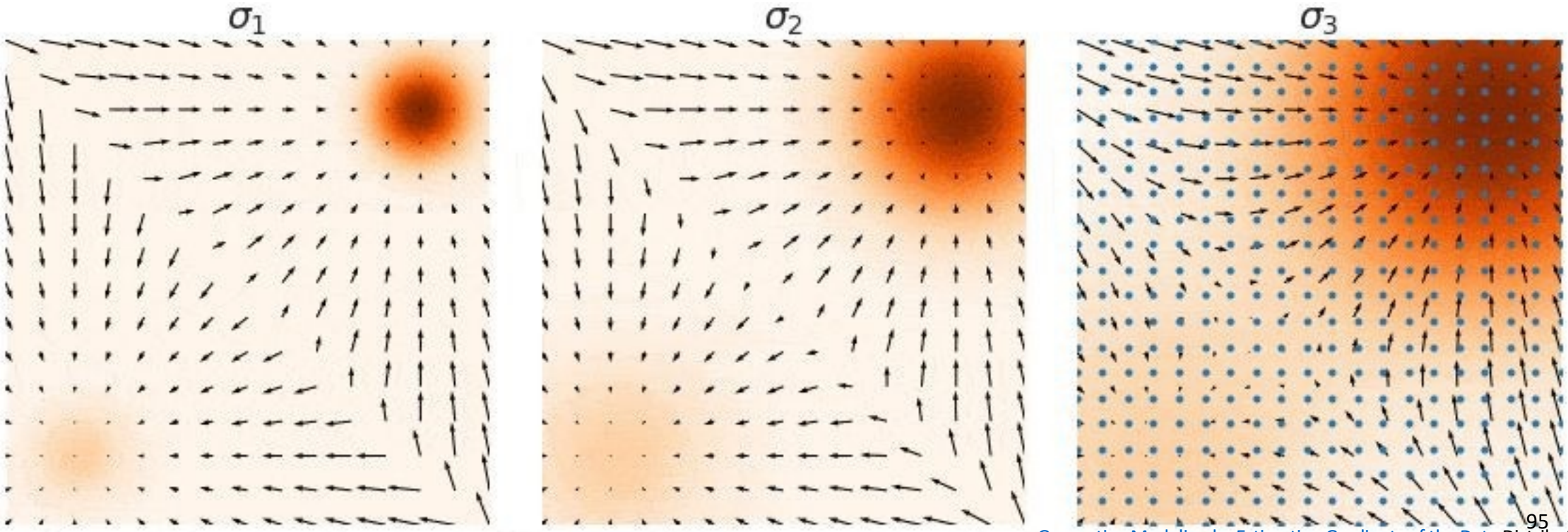


source: Generative Modeling by Estimating Gradients of the Data Distribution

Score-based Generative Modeling

Sampling is done using annealed Langevin Dynamics

- Running Langevin dynamics for each noise level in sequence, initializing the next noise level with the results of the previous one.

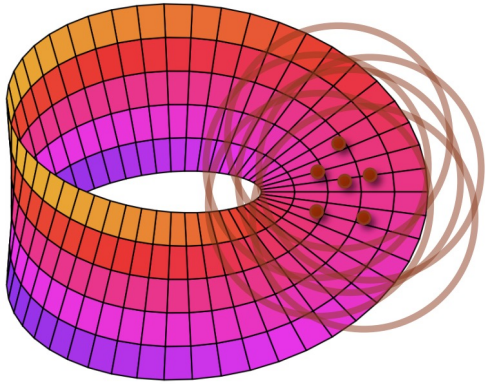


source: Generative Modeling by Estimating Gradients of the Data Distribution

Denoising score matching

- How do SGLD work when we encounter low dimensional manifold of data?!

- Adding random noise ensures the resulting distribution does not collapse
  - Manifold + noise
  - Score matching on noisy data.



- Given $p_{data}(x) = q(x_0)$, employs score matching to estimate the score of the perturbed data distribution $q(x_t) = \int q(x_0)q(x_t|x_0)\,dx$

Song and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", *NeurIPS*, 2019

Noise Conditional Score Networks

- Perturbing the data using various levels of noise

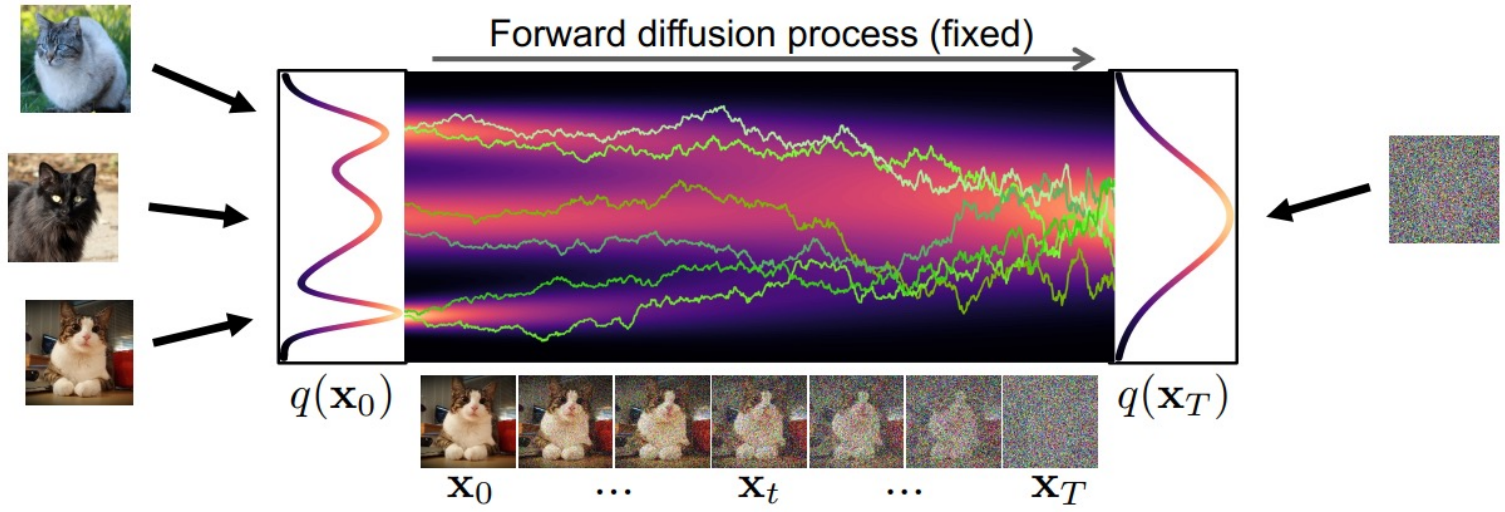
- Simultaneously estimating scores corresponding to all noise levels by training a **noise-conditioned score network** $s_\theta(x, t)$
  - Crucially, we train a single score network conditioned on the noise level and estimate the scores at all noise magnitudes.

$$s_\theta(x_t, t) \approx \nabla_{x_t} \log q(x_t)$$

- *Jointly* estimate scores of all the perturbed data at different noise levels.

Song and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", *NeurIPS*, 2019

Denoising Score Matching



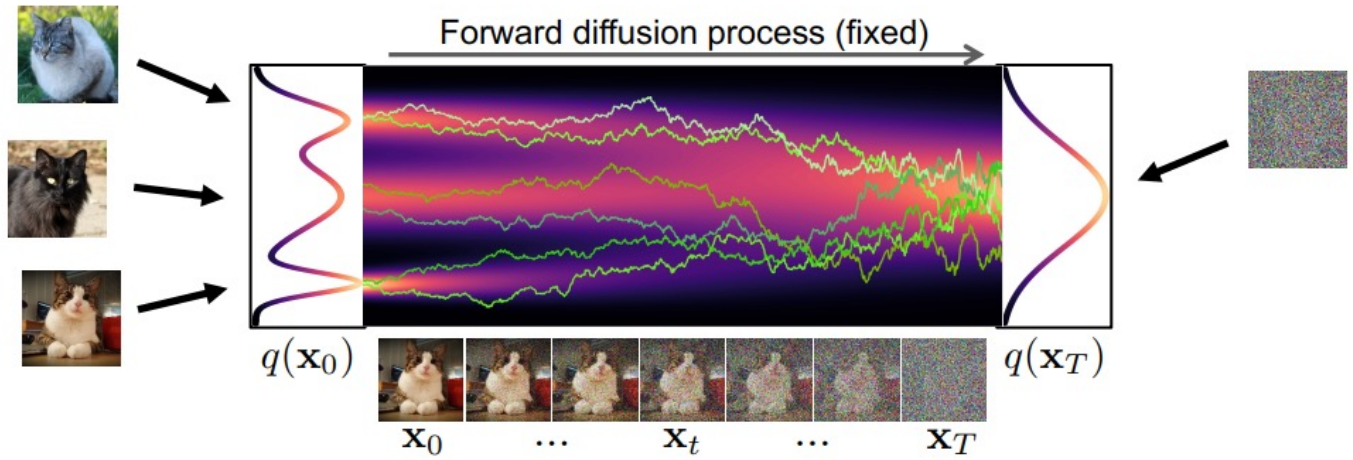
Naïve idea, learn model for the score function by direct regression?

$$\min_{\boldsymbol{\theta}} \underbrace{\mathbb{E}_{t\sim\mathcal{U}(0,T)}}_{\text{diffusion time } t} \underbrace{\mathbb{E}_{\mathbf{x}_t\sim q_t(\mathbf{x}_t)}}_{\text{diffused data } \mathbf{x}_t} \big|\big| \underbrace{\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}_{\text{neural network}} - \underbrace{\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)}_{\text{score of diffused data (marginal)}} \big|\big|_2^2$$

➡ **But $\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$ (score of the *marginal diffused density* $q_t(\mathbf{x}_t)$) is not tractable!**

Vincent, "A Connection Between Score Matching and Denoising Autoencoders", *Neural Computation*, 2011

Denoising Score Matching



- Instead, diffuse individual data points $\mathbf{x}_0$. Diffused $q_t(\mathbf{x}_t|\mathbf{x}_0)$ ***is*** tractable!
- **Denoising Score Matching:**

$$\min_{\boldsymbol{\theta}} \underbrace{\mathbb{E}_{t\sim\mathcal{U}(0,T)}}_{\text{diffusion time } t} \underbrace{\mathbb{E}_{\mathbf{x}_0\sim q_0(\mathbf{x}_0)}}_{\text{data sample } \mathbf{x}_0} \underbrace{\mathbb{E}_{\mathbf{x}_t\sim q_t(\mathbf{x}_t|\mathbf{x}_0)}}_{\text{diffused data sample } \mathbf{x}_t} \Big|\Big| \underbrace{\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}_{\text{neural network}} - \underbrace{\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0)}_{\text{score of diffused data sample}} \Big|\Big|_2^2$$

➡ **After expectations,** $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$**!**

Vincent, in *Neural Computation,* 2011
Song and Ermon, *NeurIPS,* 2019
Song et al. *ICLR*, 2021

$$J_{ESM_{q_\sigma}}(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\| s(\tilde{\mathbf{x}};\theta) - \frac{\partial \log q_\sigma(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}\right\|^2\right]$$

which we can develop as

$$J_{ESM_{q_\sigma}}(\theta) \quad = \quad \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\| s(\tilde{\mathbf{x}};\theta)\right\|^2\right] - g(\theta) + C_2 \qquad ($$

where $C_2 = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\|\frac{\partial \log q_\sigma(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}\right\|^2\right]$ is a constant that does not depend on θ, and

Vincent, in *Neural Computation*, 2011

$$J_{ESM_{q_\sigma}}(\theta) = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\| s\left(\tilde{\mathbf{x}};\theta\right) - \frac{\partial \log q_\sigma(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}\right\|^2\right]$$

which we can develop as

$$J_{ESM_{q_\sigma}}(\theta) \;=\; \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\| s\left(\tilde{\mathbf{x}};\theta\right)\right\|^2\right] - g(\theta) + C_2 \qquad (1)$$

where $C_2 = \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\frac{1}{2}\left\|\frac{\partial \log q_\sigma(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}\right\|^2\right]$ is a constant that does not depend on θ, and

$$\begin{aligned}
g(\theta) &= \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}})}\left[\left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial \log q_\sigma(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}\right\rangle\right] \\
&= \int_{\tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}}) \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial \log q_\sigma(\tilde{\mathbf{x}})}{\partial \tilde{\mathbf{x}}}\right\rangle d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}}) \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\frac{\partial}{\partial \tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}})}{q_\sigma(\tilde{\mathbf{x}})}\right\rangle d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial}{\partial \tilde{\mathbf{x}}} q_\sigma(\tilde{\mathbf{x}})\right\rangle d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial}{\partial \tilde{\mathbf{x}}} \int_{\mathbf{x}} q_0(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) d\mathbf{x}\right\rangle d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \int_{\mathbf{x}} q_0(\mathbf{x}) \frac{\partial q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}{\partial \tilde{\mathbf{x}}} d\mathbf{x}\right\rangle d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \int_{\mathbf{x}} q_0(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) \frac{\partial \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}{\partial \tilde{\mathbf{x}}} d\mathbf{x}\right\rangle d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} \int_{\mathbf{x}} q_0(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}{\partial \tilde{\mathbf{x}}}\right\rangle d\mathbf{x} d\tilde{\mathbf{x}} \\
&= \int_{\tilde{\mathbf{x}}} \int_{\mathbf{x}} q_\sigma(\tilde{\mathbf{x}}, \mathbf{x}) \left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}{\partial \tilde{\mathbf{x}}}\right\rangle d\mathbf{x} d\tilde{\mathbf{x}} \\
&= \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}},\mathbf{x})}\left[\left\langle s\left(\tilde{\mathbf{x}};\theta\right), \frac{\partial \log q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}{\partial \tilde{\mathbf{x}}}\right\rangle\right].
\end{aligned}$$

Vincent, in *Neural Computation*, 2011

Gradient of log density of a Gaussian variable

$$s_\theta(x_t, t) \approx \nabla_{x_t} \log q(x_t)$$

- Given a Gaussian distribution $x \sim N(\mu, \sigma^2 I)$:

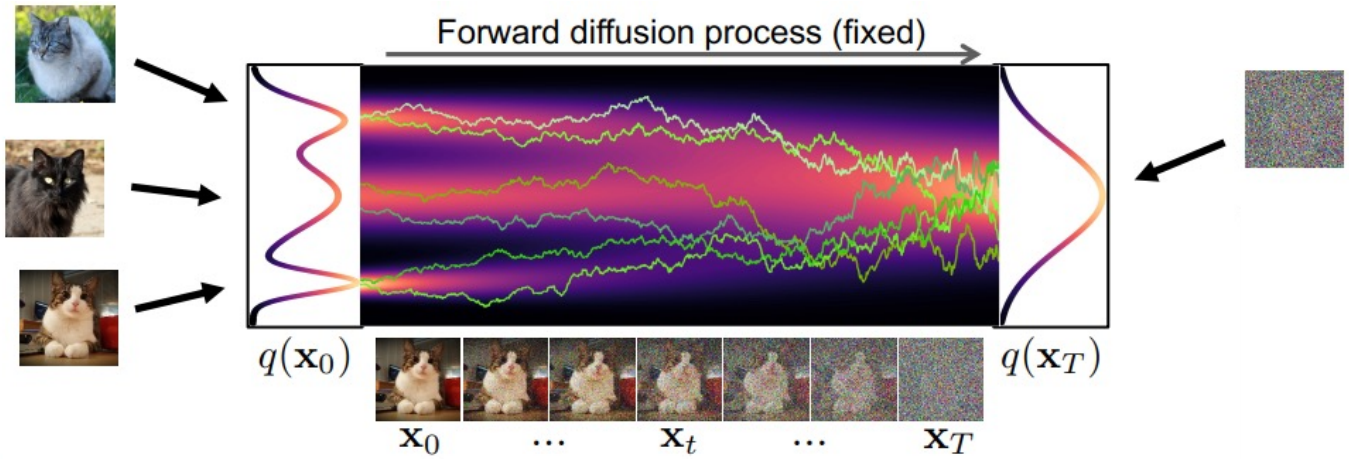
$$\nabla_x \log q(x) = -\frac{x - \mu}{\sigma^2} = -\frac{\epsilon}{\sigma} \qquad \epsilon \sim N(0, I)$$

- For $q(x_t)$, we can approximate its score as:

$$s_\theta(x_t, t) \approx \nabla_{x_t} \log q(x_t) = \mathbb{E}_{q(x_0)}\left[\nabla_{x_t} \log q(x_t|x_0)\right] = \mathbb{E}_{q(x_0)}\left[-\frac{\epsilon_\theta(x_t, t)}{\sqrt{1 - \bar{\alpha}_t}}\right]$$

Denoising Score Matching

- Implementation 1: Noise Prediction



- **Denoising Score Matching:**

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}(0,T)}\mathbb{E}_{\mathbf{x}_0\sim q_0(\mathbf{x}_0)}\mathbb{E}_{\mathbf{x}_t\sim q_t(\mathbf{x}_t|\mathbf{x}_0)}||\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)-\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t|\mathbf{x}_0)||_2^2$$

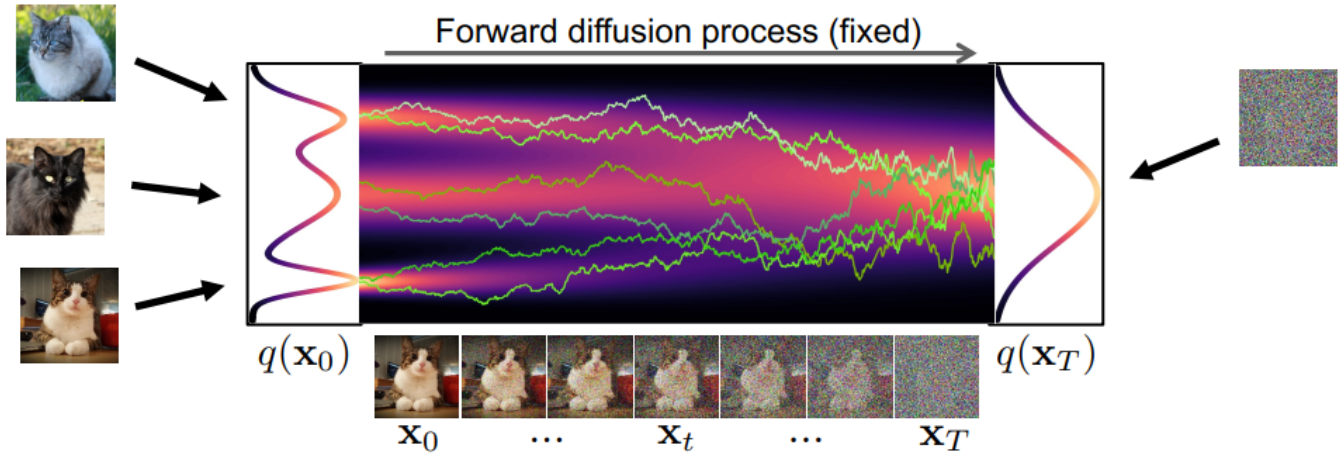
- Re-parametrized sampling: $\mathbf{x}_t=\gamma_t\mathbf{x}_0+\sigma_t\boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$
- Score function: $\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t|\mathbf{x}_0)=-\nabla_{\mathbf{x}_t}\frac{(\mathbf{x}_t-\gamma_t\mathbf{x}_0)^2}{2\sigma_t^2}=-\frac{\mathbf{x}_t-\gamma_t\mathbf{x}_0}{\sigma_t^2}=-\frac{\gamma_t\mathbf{x}_0+\sigma_t\boldsymbol{\epsilon}-\gamma_t\mathbf{x}_0}{\sigma_t^2}=-\frac{\boldsymbol{\epsilon}}{\sigma_t}$
- Neural network model: $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,t):=-\frac{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)}{\sigma_t}$

Vincent, in *Neural Computation*, 2011
Song and Ermon, *NeurIPS*, 2019
Song et al. *ICLR*, 2021

➡ $$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}(0,T)}\mathbb{E}_{\mathbf{x}_0\sim q_0(\mathbf{x}_0)}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\frac{1}{\sigma_t^2}||\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)||_2^2$$

Denoising Score Matching

- Implementation 2: Loss Weightings



- **Denoising Score Matching** objective with loss weighting $\lambda(t)$:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{t\sim\mathcal{U}(0,T)}\mathbb{E}_{\mathbf{x}_0\sim q_0(\mathbf{x}_0)}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\frac{\lambda(t)}{\sigma_t^2}||\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,t)||_2^2$$

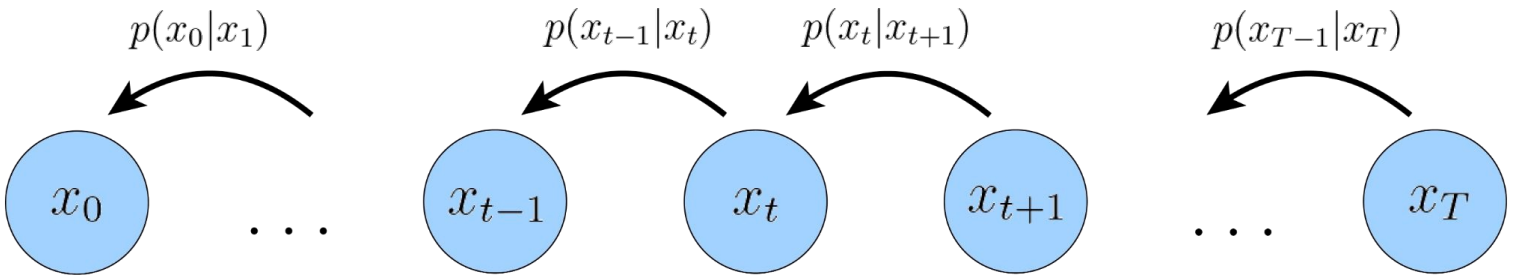
Different loss weightings trade off between model with good perceptual quality vs. high log-likelihood

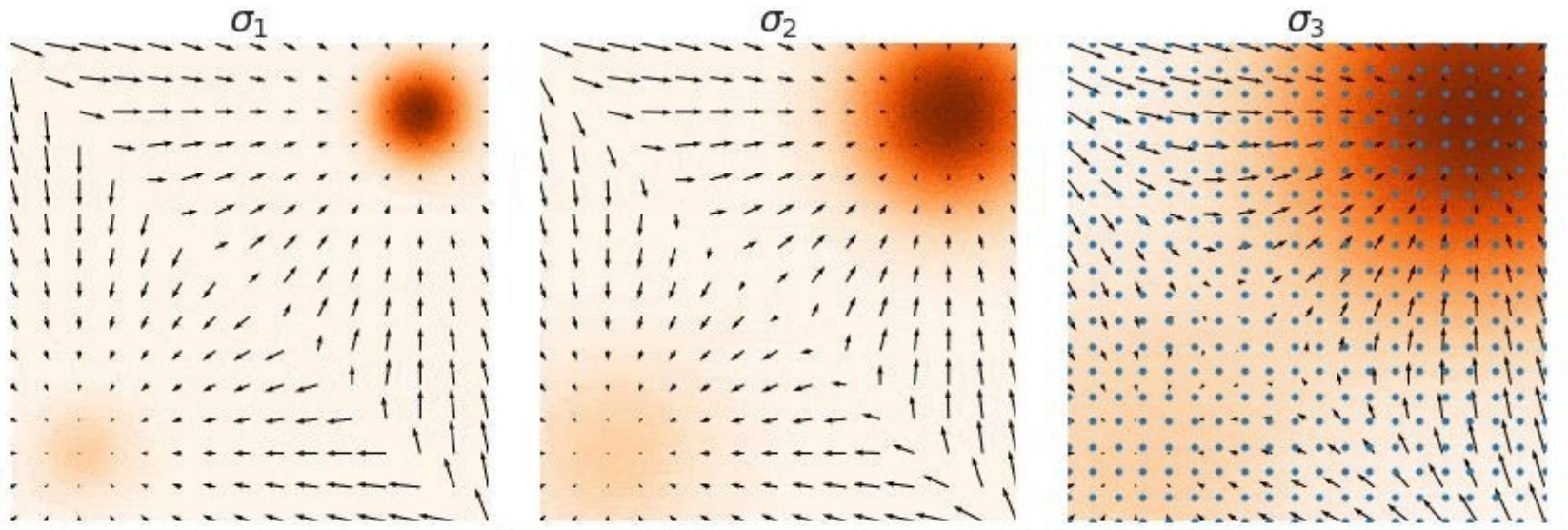
- *Perceptual quality:* $\lambda(t)=\sigma_t^2$
- *Maximum log-likelihood:* $\lambda(t)=\beta(t)$ *(negative ELBO)*

➡ **Same objectives as derived with variational approach in Part (1)!**

[Ho et al, NeurIPS, 2020]
[Song et al., NeurIPS, 2021]
[Kingma et al., NeurIPS, 2021]
[Vahdat et al., NeurIPS, 2021]
[Huang et al., NeurIPS, 2021]
[Karras et al., arXiv, 2022]

Sampling





NCSNs

Sampling

- **Annealed version of Langevin dynamics**: After training, during sampling, initially use scores corresponding to large noise, and gradually anneal down the noise level.
  - This helps smoothly transfer the benefits of large noise levels to low noise levels where the perturbed data are almost indistinguishable from the original ones

Unifying Two Interpretations

We have shown two equivalently valid ways to describe a diffusion model!

- One takes the perspective of a Markovian diffusion process, where samples are steadily denoised in the reverse process

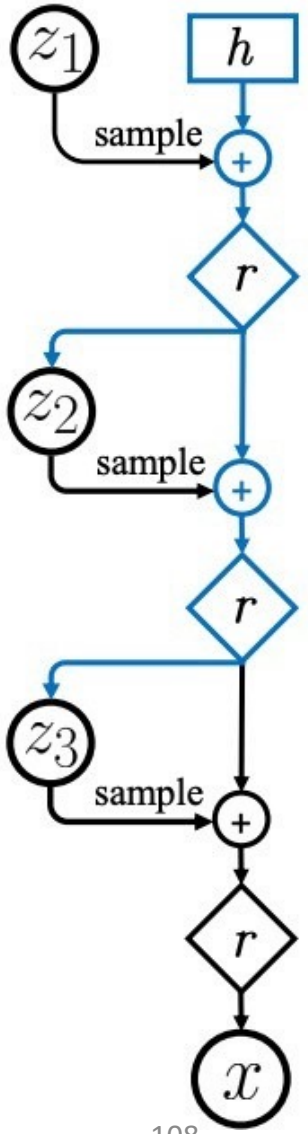
- Another takes the perspective of energy-based models and score matching where we iteratively refine an input through noise levels.

they are two sides of the same coin!

Connection to VAEs

Diffusion models can be considered as a special form of hierarchical VAEs. However, in diffusion models:

- The encoder is fixed
- The latent variables have the same dimension as the data
- The denoising model is shared across different timestep
- The model is trained with some reweighting of the variational bound.



Vahdat and Kautz, NVAE: A Deep Hierarchical Variational Autoencoder, NeurIPS 2020
Sønderby, et al.. Ladder variational autoencoders, NeurIPS 2016.

Advanced Techniques

- Questions to address with advanced techniques
- How to do high-resolution (conditional) generation?
  - Conditional diffusion models
  - Classifier(-free) guidance
  - Cascaded generation
- How to accelerate the sampling process?
  - Advanced forward diffusion process
  - Advanced reverse process
  - Hybrid models

Conditional Diffusion Models

So far we have been learning an unconditional diffusion model $p_\theta(\boldsymbol{x})$

Conditional Diffusion Models

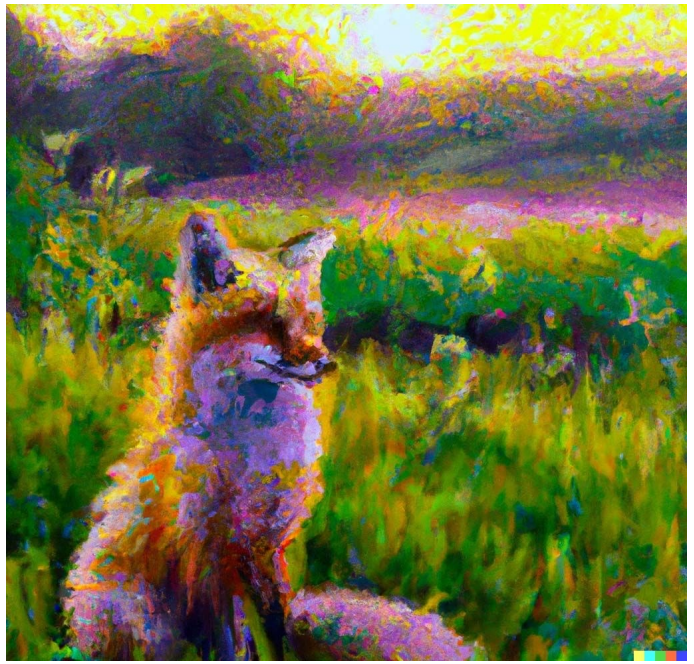
$$p(image \mid text\_caption)$$

How do we incorporate conditional information, to control data generation?

"a painting of a fox sitting in a field at sunrise in the style of Claude Monet"

Parti (but pretend it is ImageN)

StableDiffusion

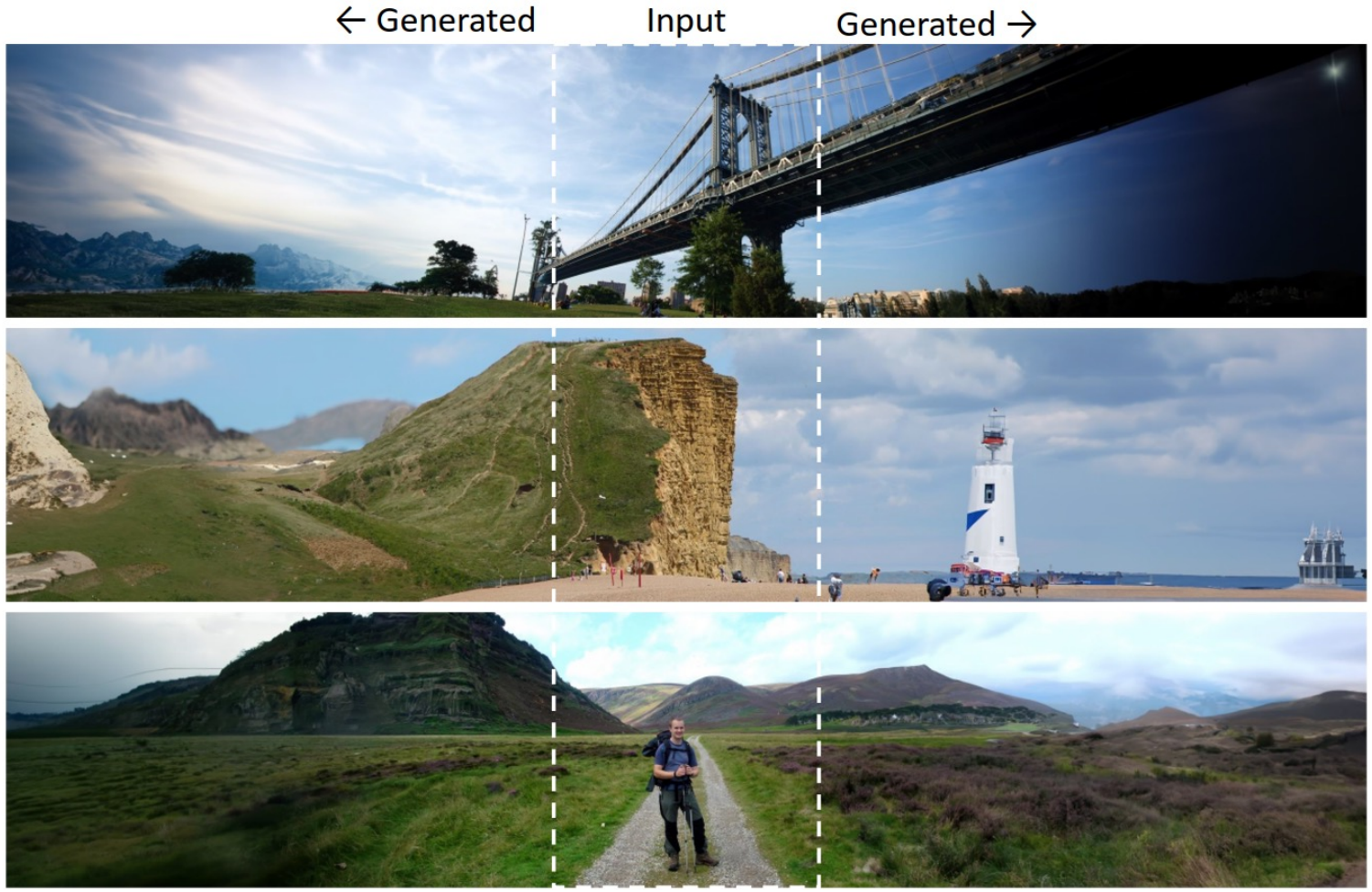


Dall-E 2.0

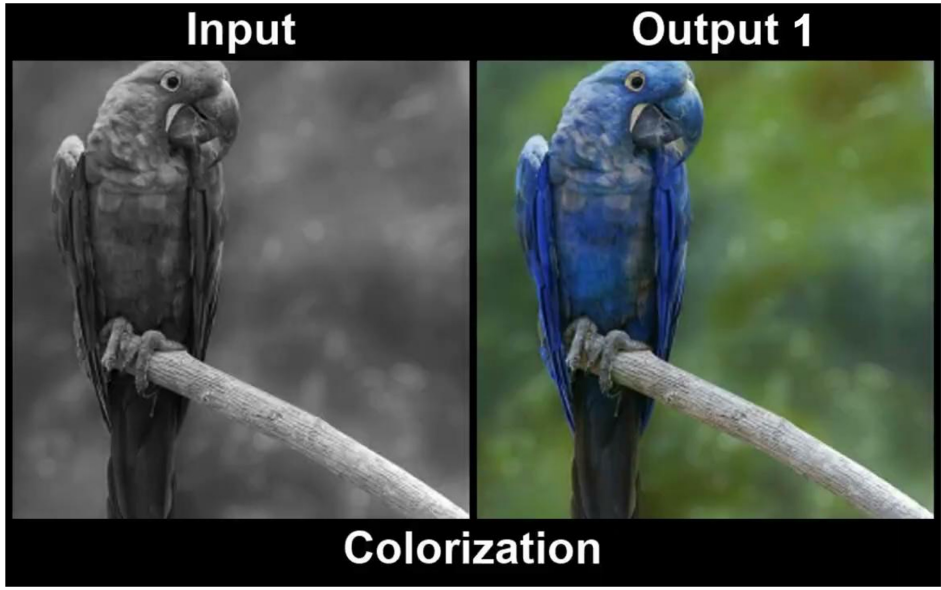
source: ImageN, StableDiffusion, Dall-E 2.0

Impressive conditional diffusion models

Panorama generation



Colorization



Saharia et al., “Palette: Image-to-Image Diffusion Models”, arXiv 2021.

Conditional diffusion models

- Include condition as input to reverse process

Reverse process: $p_\theta(\mathbf{x}_{0:T}|\mathbf{c}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}), \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t, \mathbf{c}), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t, \mathbf{c}))$

Variational upper bound: $L_\theta(\mathbf{x}_0|\mathbf{c}) = \mathbb{E}_q\left[L_T(\mathbf{x}_0) + \sum_{t>1} D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c})) - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1, \mathbf{c})\right].$

Incorporate conditions into U-Net

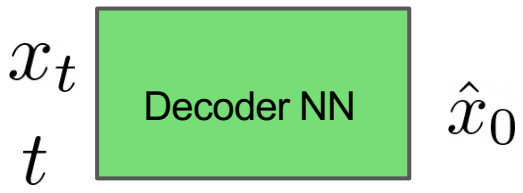
- **Scalar conditioning**: encode scalar as a vector embedding, simple spatial addition or adaptive group normalization layers.
- **Image conditioning**: channel-wise concatenation of the conditional image.
- **Text conditioning**: single vector embedding – spatial addition or adaptive group norm / a seq of vector embeddings - cross-attention.

Conditional Diffusion Models

How do we incorporate conditional information, to control data generation?

Suppose we have conditioning information y and now want to learn $p_\theta(\boldsymbol{x} \mid y)$

Well an unconditional diffusion model $p_\theta(\boldsymbol{x})$ really is just:



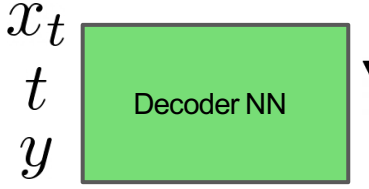
Three Different Interpretations

It turns out, training a DiffModel can be implemented as a neural net that:

- 🖼️ Predicts original image $\hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{x}_0 \longrightarrow \hat{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \boldsymbol{x}_0$

- 🔊 Predicts noise epsilon $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \boldsymbol{\epsilon}_0 \longrightarrow \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \boldsymbol{\epsilon}_0$

- 💯 Predicts score function $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t) \longrightarrow \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$



Caveat: A conditional diffusion model trained by this simple conditioning may ignore or downplay the given conditioning information.

Guidance

Guidance provides more explicit control on the amount of weight the model gives to the conditioning information, at the cost of sample diversity.

- How much should our generated x match y?

Let us take the score-based perspective of diffusion models.

Now, we are interested in learning $\nabla \log p(\boldsymbol{x}_t \mid y)$ rather than unconditional score function $\nabla \log p(\boldsymbol{x}_t)$

Classifier Guidance

- Using the gradient of a trained classifier as guidance

- Main Idea:
  - For class-conditional modeling of $p(x_t|c)$, train an extra classifier $p(c|x_t)$
  - Mix its gradients with the diffusion/score model during sampling

Dhariwal and Nichol, “Diffusion models beat GANs on image synthesis”, NeurIPS 2021

Classifier Guidance

- Using the gradient of a trained classifier as guidance
- Recap: What is a score function?

$$\min_{\boldsymbol{\theta}} \underbrace{\mathbb{E}_{t\sim\mathcal{U}(0,T)}}_{\text{diffusion time } t} \underbrace{\mathbb{E}_{\mathbf{x}_t\sim q_t(\mathbf{x}_t)}}_{\text{diffused data } \mathbf{x}_t} || \underbrace{\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}_{\text{neural network}} - \underbrace{\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)}_{\text{score of diffused data (marginal)}} ||_2^2$$

Classifier Guidance

- Applying Bayes rule to obtain conditional score function

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

$$\implies \log p(x \mid y) = \log p(y \mid x) + \log p(x) - \log p(y)$$

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

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

← Classifier

Guidance scale: value >1 amplifies the influence of classifier signal

$$p_\gamma(x \mid y) \propto p(x) \cdot p(y \mid x)^\gamma.$$

Slide Credits of guidance:
https://benanne.github.io/2022/05/26/guidance.html

Classifier Guidance

- Applying Bayes rule to obtain conditional score function

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

$$\implies \log p(x \mid y) = \log p(y \mid x) + \log p(x) - \log p(y)$$

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

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

← Classifier

Guidance scale: value >1 amplifies the influence of classifier signal

$$p_\gamma(x \mid y) \propto p(x) \cdot p(y \mid x)^\gamma.$$

Slide Credits of guidance:
https://benanne.github.io/2022/05/26/guidance.html

Classifier Guidance

Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

Score model

Classifier gradient

- Train unconditional Diffusion model
- Take your favorite classifier, depending on the conditioning type
- During inference / sampling mix the gradients of the classifier with the predicted score function of the unconditional diffusion model.

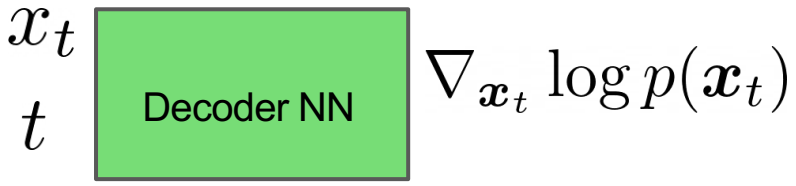
Classifier Guidance

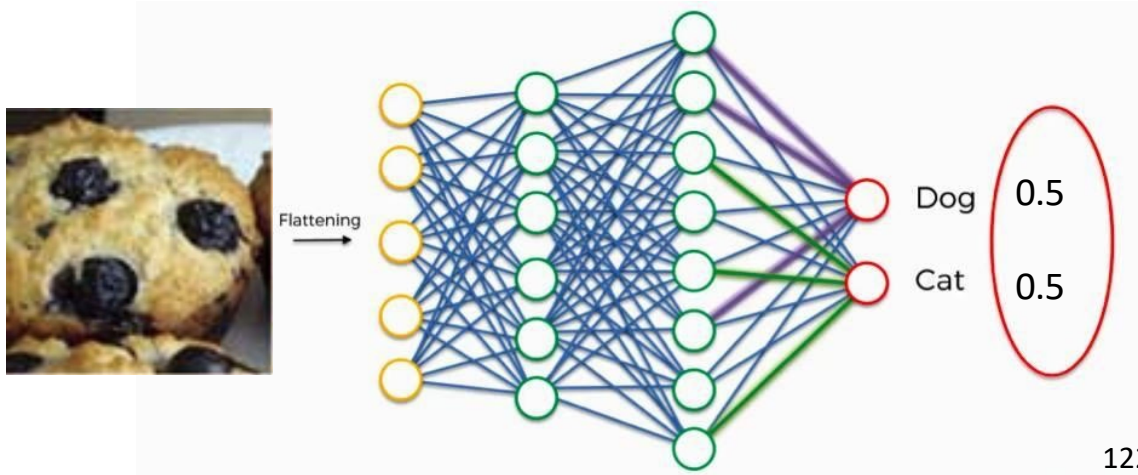
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}
\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left(\frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\
&= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\
&= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}
\end{aligned}$$

It turns out that training a conditional diffusion model is as simple as training an unconditional diffusion model (as before) along with a classifier $p(y|\boldsymbol{x}_t)$!





Classifier Guidance

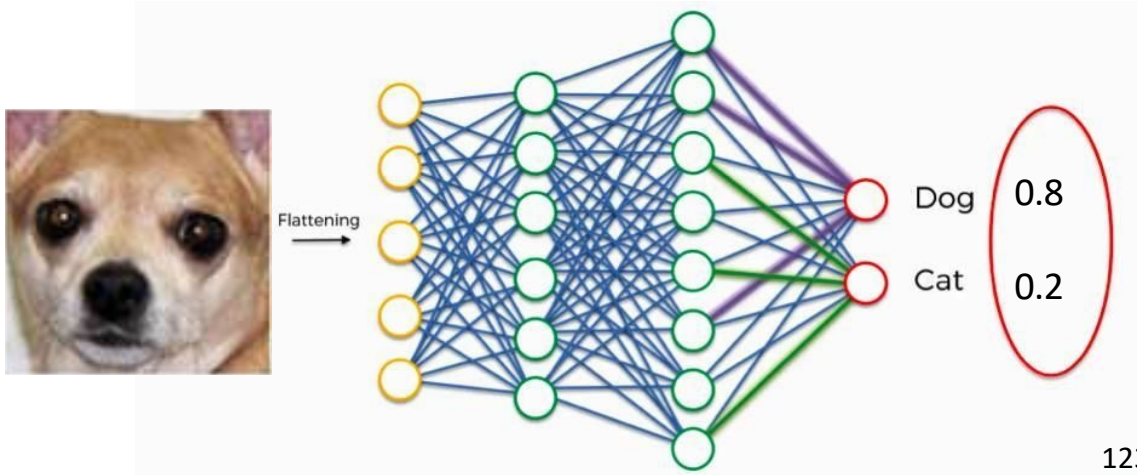
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Score of a conditional diffusion model:

$$\begin{aligned}\nabla \log p(\boldsymbol{x}_t \mid y) &= \nabla \log \left(\frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right) \\ &= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y) \\ &= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}\end{aligned}$$

It turns out that training a conditional diffusion model is as simple as training an unconditional diffusion model (as before) along with a classifier $p(y|\boldsymbol{x}_t)$!

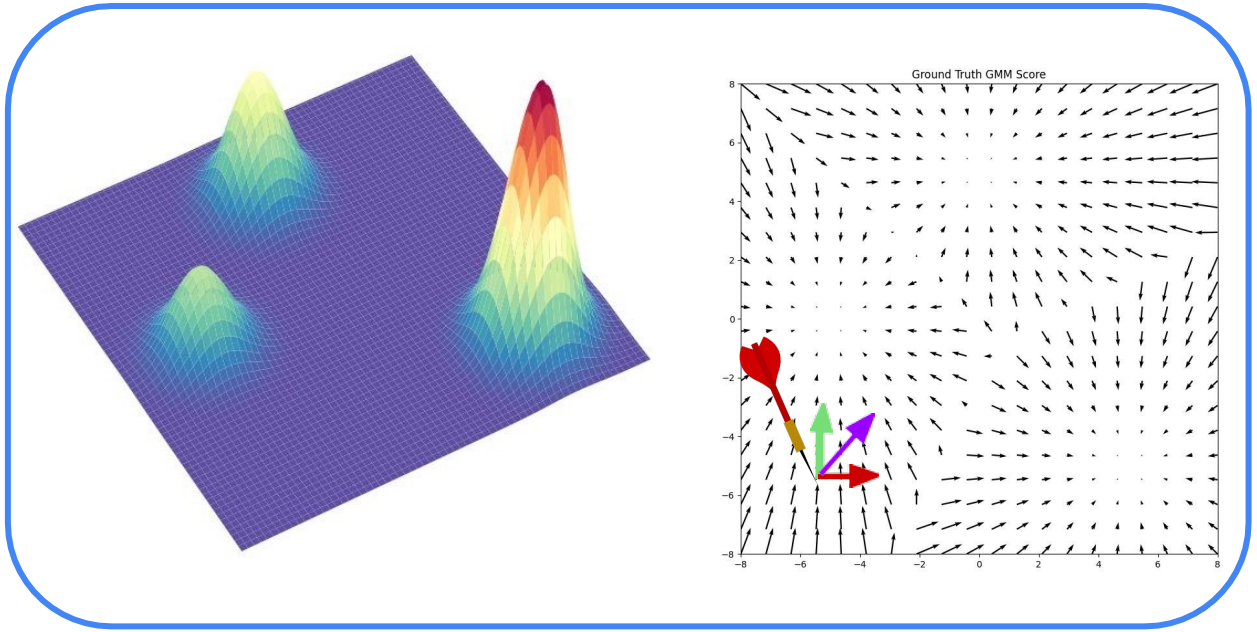
x_t
t
Decoder NN
$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t)$



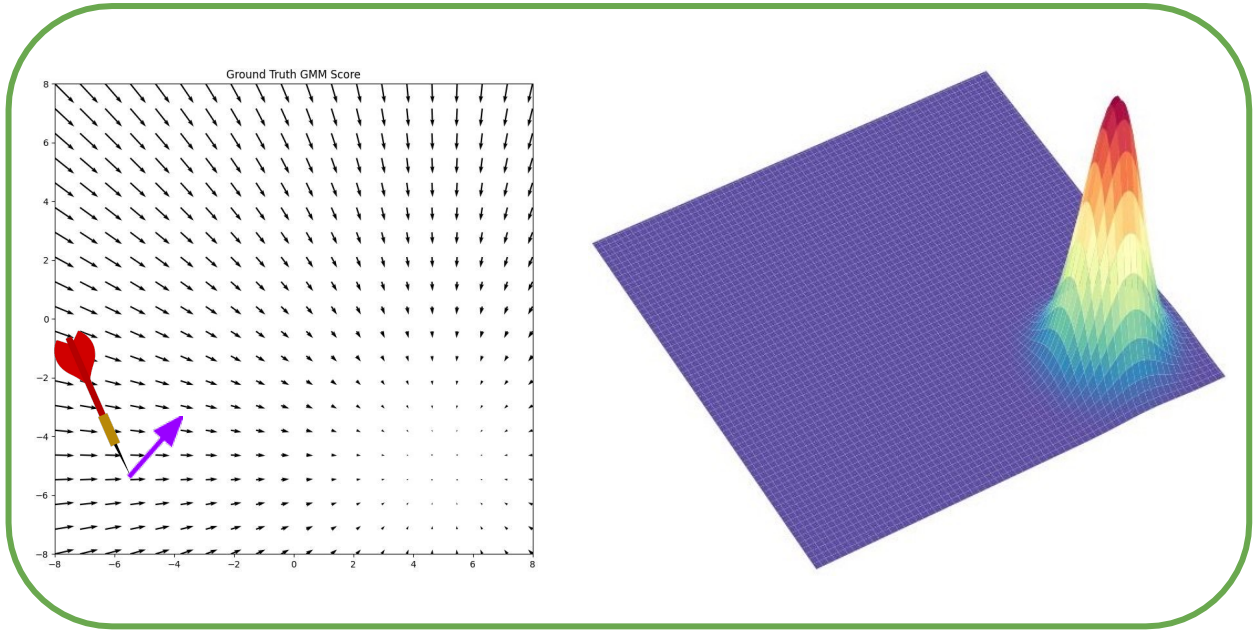
Classifier Guidance

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

$$\nabla \log p(\boldsymbol{x}_t \mid y) = \nabla \log p(\boldsymbol{x}_t) + \gamma \nabla \log p(y \mid \boldsymbol{x}_t)$$



$p(\boldsymbol{x})$

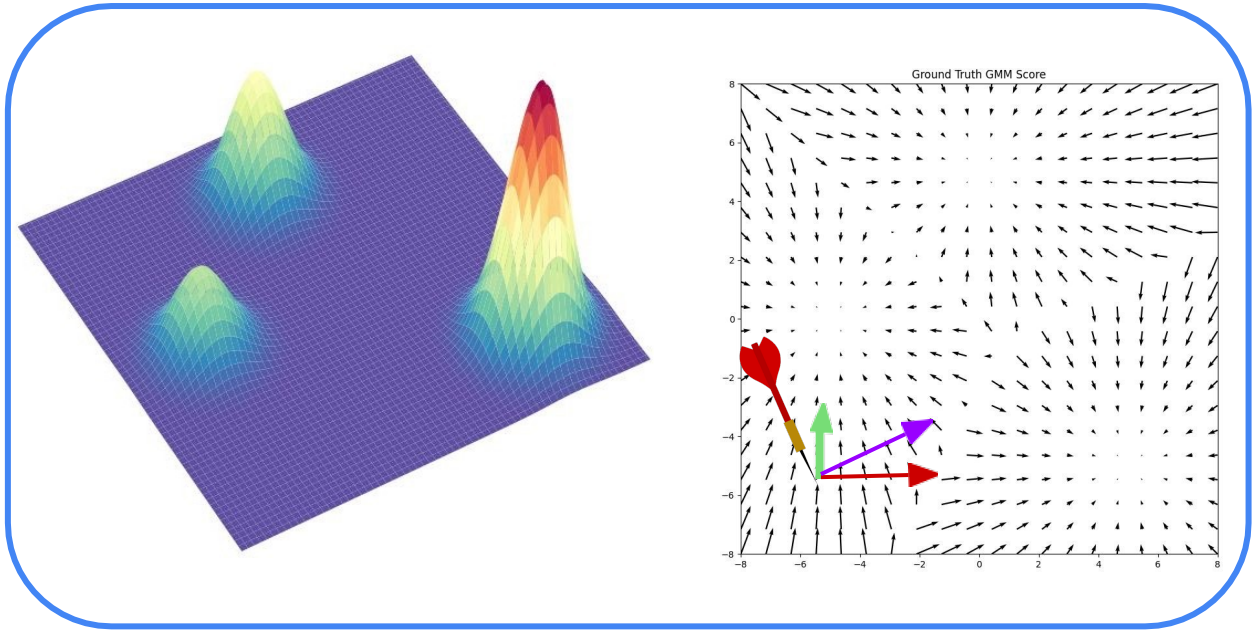


$p(\boldsymbol{x} \mid y = \quad)$

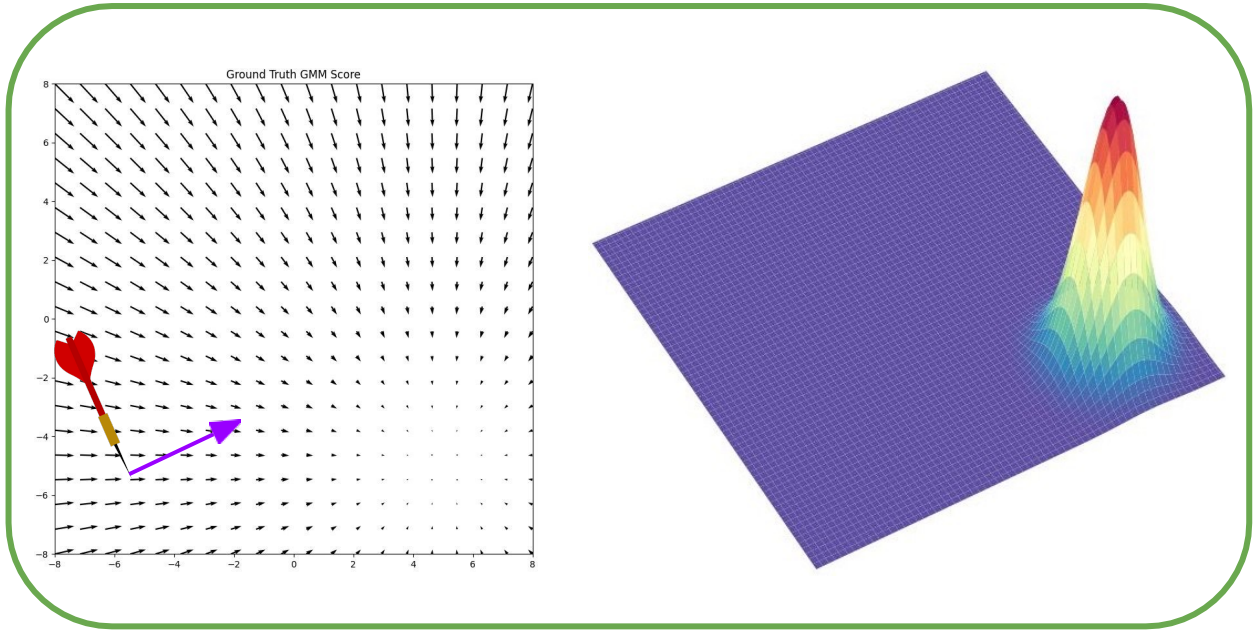
Classifier Guidance

Sampling is then done by querying the learned unconditional score function as well as the adversarial gradient of a classifier.

$$\nabla \log p(\boldsymbol{x}_t \mid y) = \nabla \log p(\boldsymbol{x}_t) + \gamma \nabla \log p(y \mid \boldsymbol{x}_t)$$



$p(\boldsymbol{x})$



$p(\boldsymbol{x} \mid y = \quad)$

Classifier Guidance

- Problems of classifier guidance

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

← Classifier

Guidance scale: value >1 amplifies the influence of classifier signal

- Re-train classifier on noisy images! Can't use existing pre-trained classifiers.
  - At each step of denoising the input to the classifier is a noisy image x_t
  - Classifier is never trained on noisy image.

- Most of the information in x is not relevant to predicting y, and as a result, taking the gradient of the classifier w.r.t. x can yield arbitrary (and even adversarial) directions in input space.

Toward Classifier-free

- Get guidance by Bayes' rule on conditional diffusion models

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

$$\implies \log p(y \mid x) = \log p(x \mid y) + \log p(y) - \log p(x)$$

$$\implies \boxed{\nabla_x \log p(y \mid x) = \nabla_x \log p(x \mid y) - \nabla_x \log p(x).}$$

Toward Classifier-free

- Get guidance by Bayes' rule on conditional diffusion models

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

$$\implies \log p(y \mid x) = \log p(x \mid y) + \log p(y) - \log p(x)$$

$$\implies \boxed{\nabla_x \log p(y \mid x) = \nabla_x \log p(x \mid y) - \nabla_x \log p(x).}$$

We proved this in classifier guidance.

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

Toward Classifier-free

- Get guidance by Bayes' rule on conditional diffusion models

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

$$\implies \log p(y \mid x) = \log p(x \mid y) + \log p(y) - \log p(x)$$

$$\implies \boxed{\nabla_x \log p(y \mid x) = \nabla_x \log p(x \mid y) - \nabla_x \log p(x).}$$

We proved this in classifier guidance.

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

$$\nabla_x \log p_\gamma(x \mid y) = \nabla_x \log p(x) + \gamma \left(\nabla_x \log p(x \mid y) - \nabla_x \log p(x)\right),$$

$$\nabla_x \log p_\gamma(x \mid y) = (1-\gamma) \nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y).$$

Score function for unconditional diffusion model

Score function for conditional diffusion model

Classifier-free Guidance

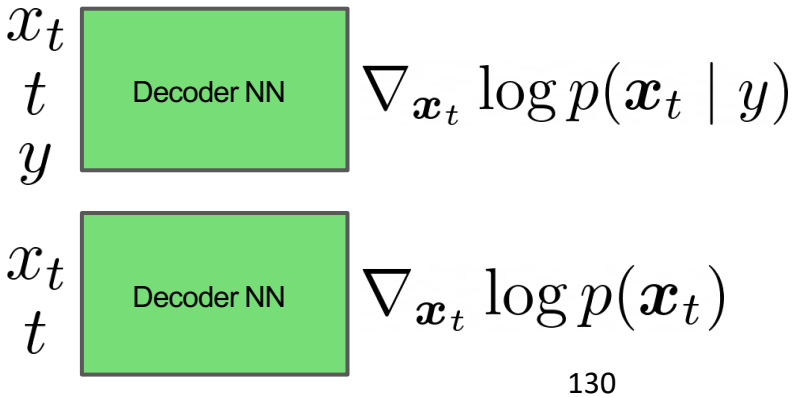
$$\nabla \log p(\boldsymbol{x}_t \mid y) = \underbrace{\gamma \nabla \log p(\boldsymbol{x}_t \mid y)}_{\text{conditional score}} + \underbrace{(1-\gamma) \nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}}$$

Questions:

- What happens if γ is 1?
- What happens if γ is larger than 1?
- How many diffusion models do we need to train?

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$$

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla \log p(\boldsymbol{x}_t)$$



Classifier-free Guidance

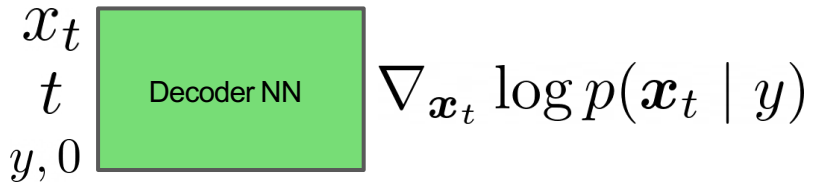
$$\nabla \log p(\boldsymbol{x}_t \mid y) = \underbrace{\gamma \nabla \log p(\boldsymbol{x}_t \mid y)}_{\text{conditional score}} + \underbrace{(1-\gamma) \nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}}$$

Questions:

- What happens if γ is 1?
- What happens if γ is larger than 1?
- How many diffusion models do we need to train?

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$$

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, 0) \approx \nabla \log p(\boldsymbol{x}_t)$$



Just one - as long as we train it with dropout on the conditioning info!

Classifier-free Guidance

- Get guidance by Bayes' rule on conditional diffusion models

$$\nabla_x \log p_\gamma(x \mid y) = (1-\gamma)\nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y).$$

Score function for unconditional diffusion model

Score function for conditional diffusion model

In practice:

- Train a conditional diffusion model $p(x|y)$, with *conditioning dropout*: some percentage of the time, the conditioning information y is removed (10-20% tends to work well).
- The conditioning is often replaced with a special input value representing the absence of conditioning information.
- The resulting model is now able to function both as a conditional model $p(x|y)$, and as an unconditional model $p(x)$, depending on whether the conditioning signal is provided.
- During inference / sampling simply mix the score function of conditional and unconditional diffusion model based on guidance scale.

Classifier-free Guidance

- Trade-off for sample quality and sample diversity

Non-guidance

Guidance scale = 1

Guidance scale = 3

Large guidance weight (λ) usually leads to better individual sample quality but less sample diversity.

Ho & Salimans, "Classifier-Free Diffusion Guidance", 2021

Classifier-free Guidance

- Get guidance by Bayes' rule on conditional diffusion models

$$\nabla_x \log p_\gamma(x \mid y) = (1-\gamma)\nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y).$$

This is a barycentric combination of the conditional and the unconditional score function. For $\gamma = 0$, we recover the unconditional model, and for $\gamma = 1$ we get the standard conditional model. But $\gamma > 1$ is where the magic happens. Below are some examples from OpenAI's GLIDE model[8], obtained using classifier-free guidance.

Score function for unconditional diffusion model

Score function for conditional diffusion model

Two sets of samples from OpenAI's GLIDE model, for the prompt *'A stained glass window of a panda eating bamboo.'*, taken from their paper. Guidance scale 1 (no guidance) on the left, guidance scale 3 on the right.

Classifier guidance

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

Guidance scale — Classifier

X Need to train a separate “noise-robust” classifier + unconditional diffusion model.
X Gradient of the classifier w.r.t. input yields arbitrary values.

Classifier-free guidance

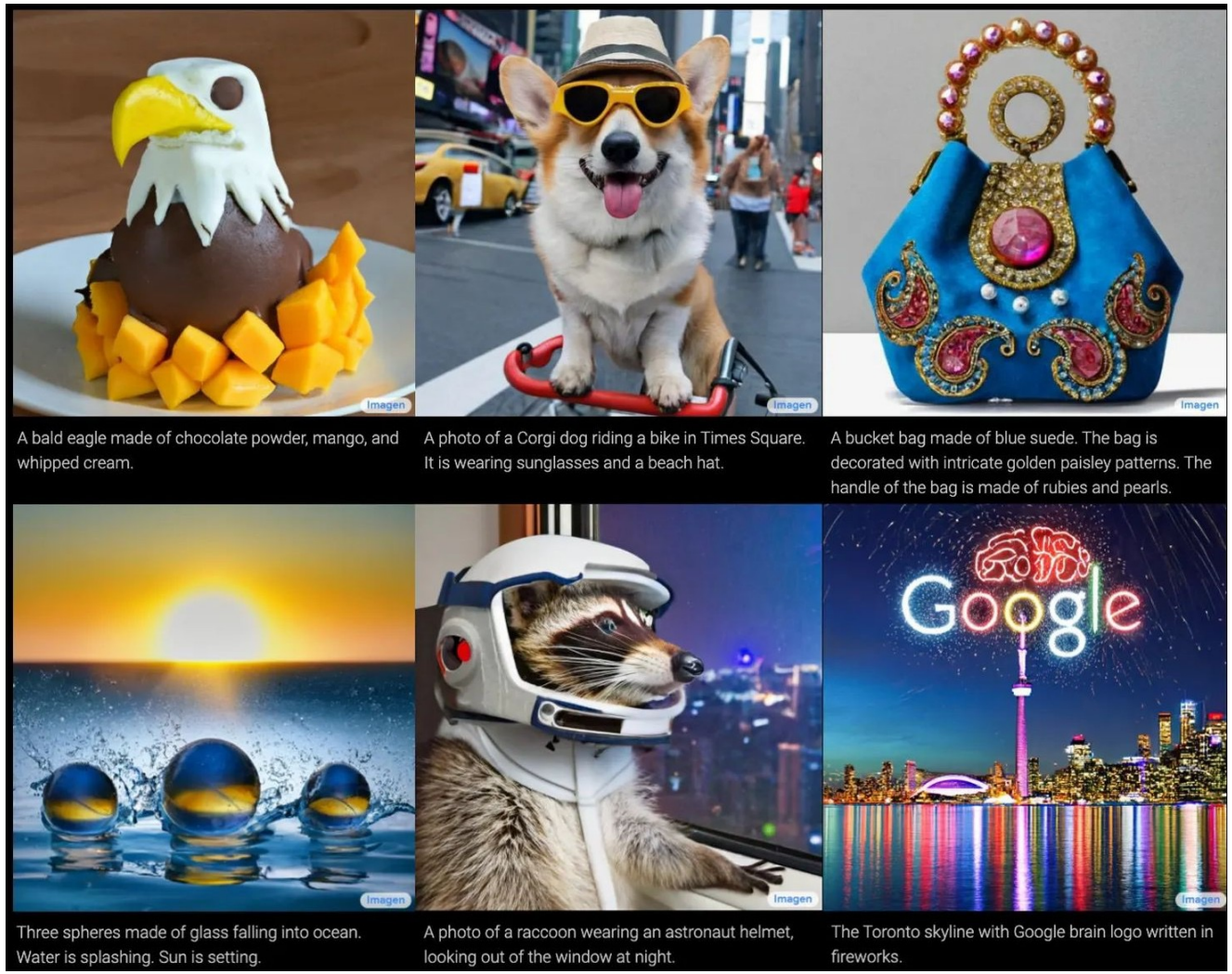
$$\nabla_x \log p_\gamma(x \mid y) = (1-\gamma) \nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y).$$

Score function for unconditional diffusion model — Score function for conditional diffusion model

+ Train conditional & unconditional diffusion model jointly via drop-out.
+ All pixels in input receive equally ‘good’ gradients.

Rather than constructing a generative model from classifier, we construct a classifier from a generative model!
Most recent papers use classifier-free guidance! Very simple yet very powerful idea!

Classifier-Free Guidance



source: ImageN

Models in Practice

Let’s explore some of the state-of-the-art diffusion models in practice:

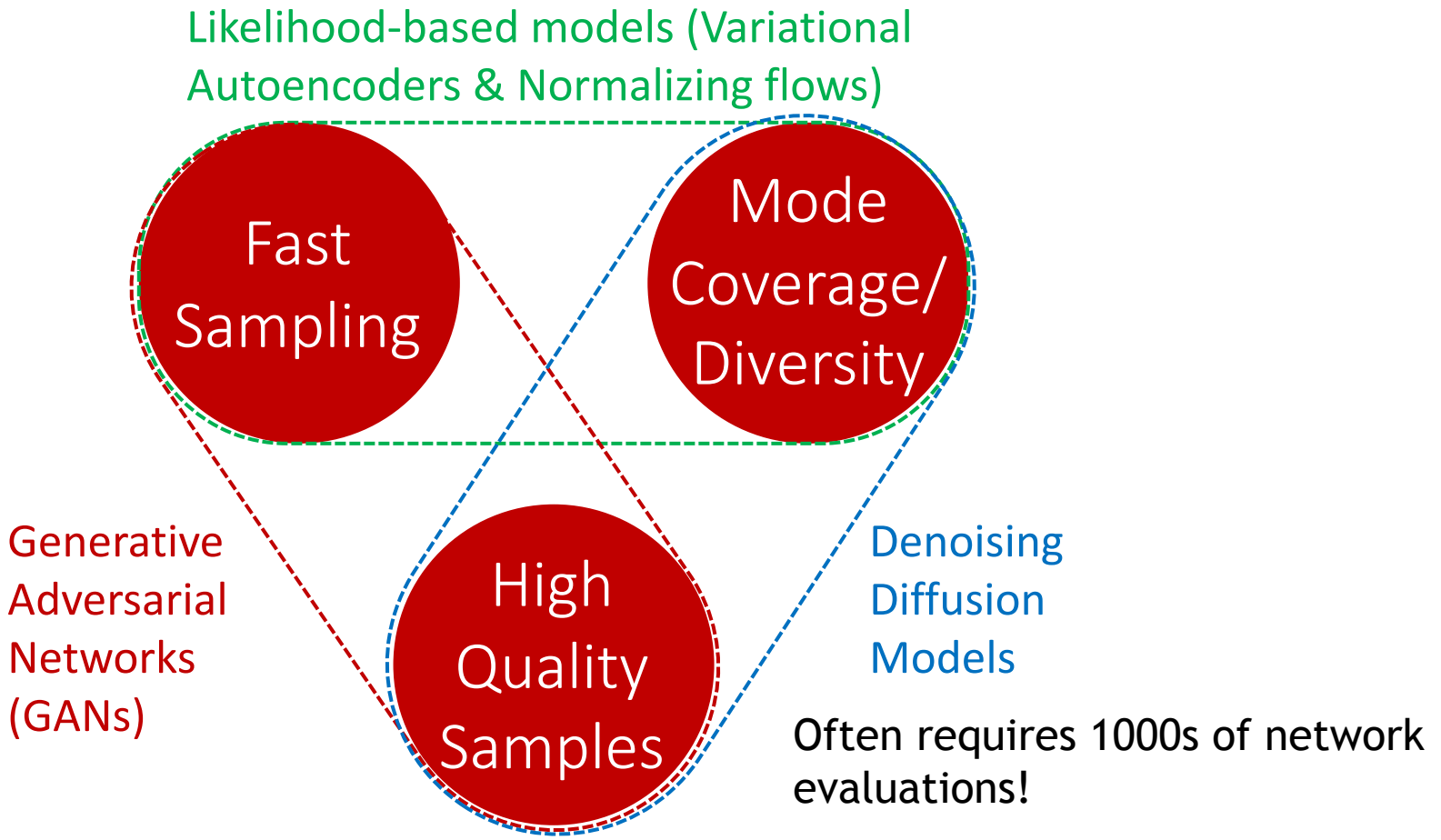
- DALL-E 3 

- ImageN 

- StableDiffusion 

What makes a good generative model?

- The generative learning trilemma

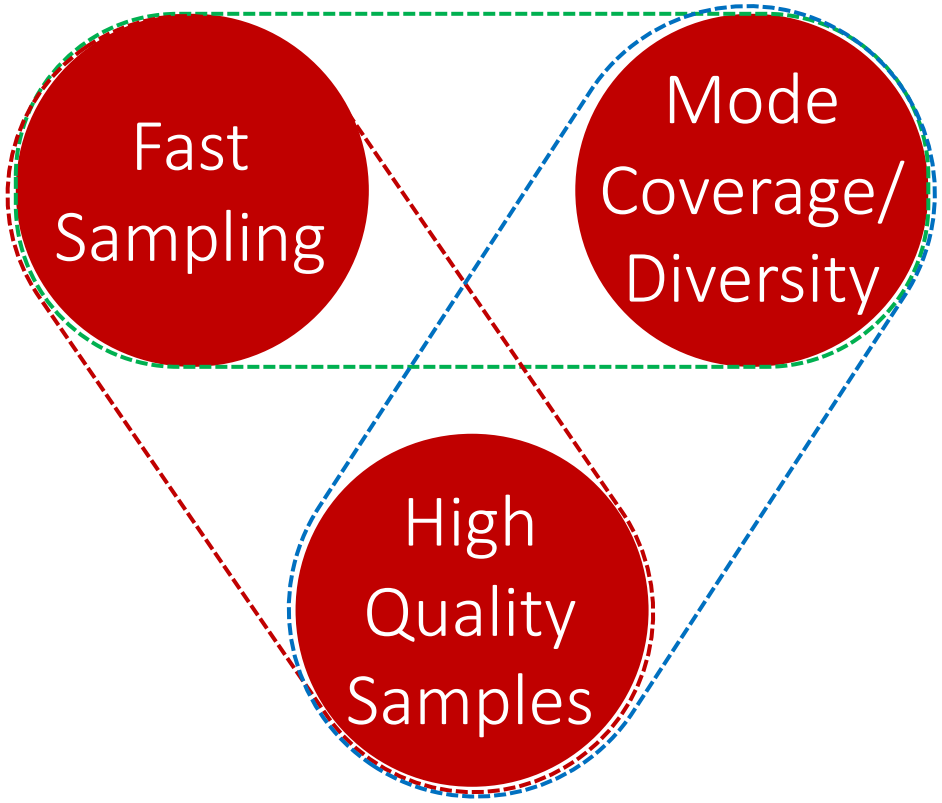


[Tackling the Generative Learning Trilemma with Denoising Diffusion GANs, ICLR 2022]

What makes a good generative model?

- The generative learning trilemma

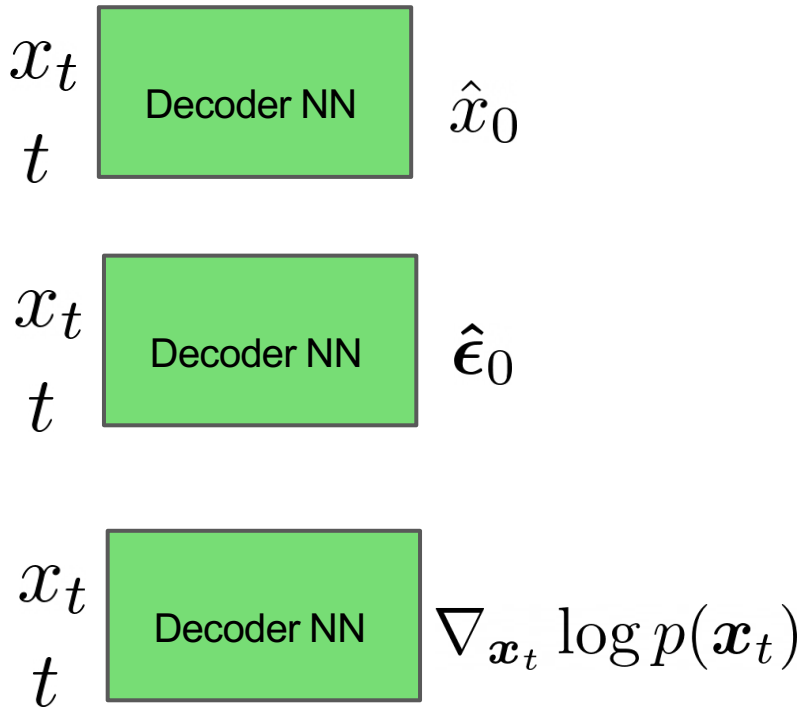
Tackle the trilemma by accelerating diffusion models



[Tackling the Generative Learning Trilemma with Denoising Diffusion GANs, ICLR 2022]

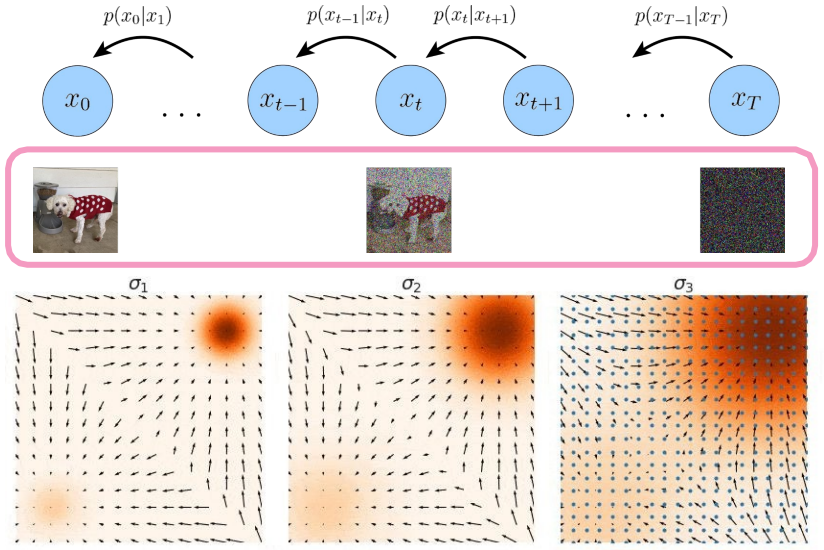
Summary

- Diffusion models rely on a long Markov chain of diffusion steps to generate samples
- We show that optimization boils down to learning a network to either predict the original image, the source noise, or the score function at arbitrary Gaussian noise corruption levels.



Summary

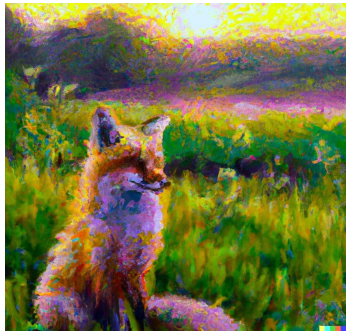
- We show that optimization boils down to learning a network to either predict the original image, the source noise, or the score function at arbitrary Gaussian noise corruption levels.
- We draw an explicit connection to score-based generative modeling and show they are equivalent in what they model, their objective, and sampling process.



Summary

- We show that optimization boils down to learning a network to either predict the original image, the source noise, or the score function at arbitrary Gaussian noise corruption levels.
- We draw an explicit connection to score-based generative modeling and show they are equivalent in what they model, their objective, and sampling process.
- We showcase how to build a conditional diffusion model, and apply guidance.

$p(image \mid text\_caption)$

x_t
t
$y, 0$

Decoder NN

$\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t \mid y)$

source: ImageN, StableDiffusion, Dall-E 2.0

Summary

Diffusion models have amazing generative performance!

- Probably state of the art generative model right now
- Absolutely incredible at learning conditional distributions

However, there is more work to be done:

- It is unlikely that this is how our brain works in modeling and generating data
- Latents themselves are not interpretable; they are just noise
- Sampling is an expensive procedure; both formulations require running multiple iterations of denoising.