Advanced Course on Deep Generative Models

Lecture 9: Diffusion Models

Adapted from Volodymyr Kuleshov and CVPR tutorial

Dan Rosenbaum, CS Haifa

Outline

- Score Based Models
  - Annealed Langevin Dynamics
- Diffusion Models as Latent Variable Models
  - ELBO
  - DDPM
- Connection to Continuous Normalizing Flows
  - ODEs and SDEs
  - Flow Matching

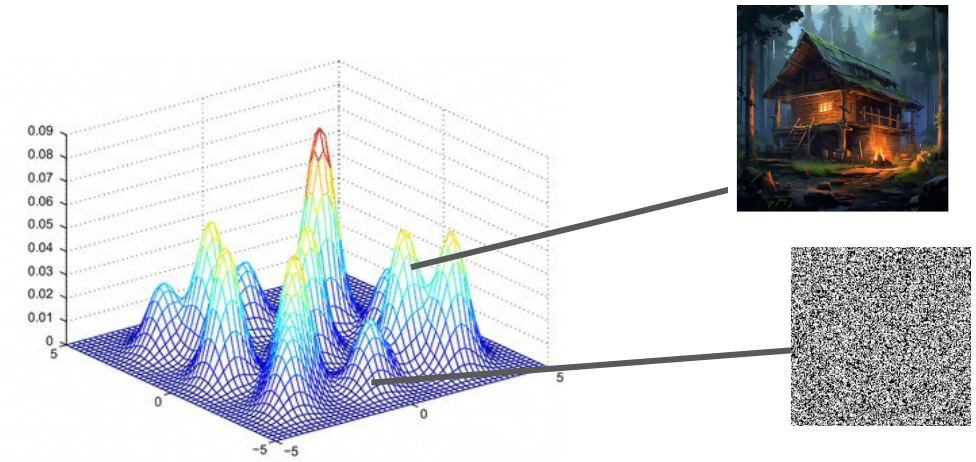
Generative models

Probabilistic model with high dimensional output.

- looking for the parameters θ such that p_θ is close to p_{data}

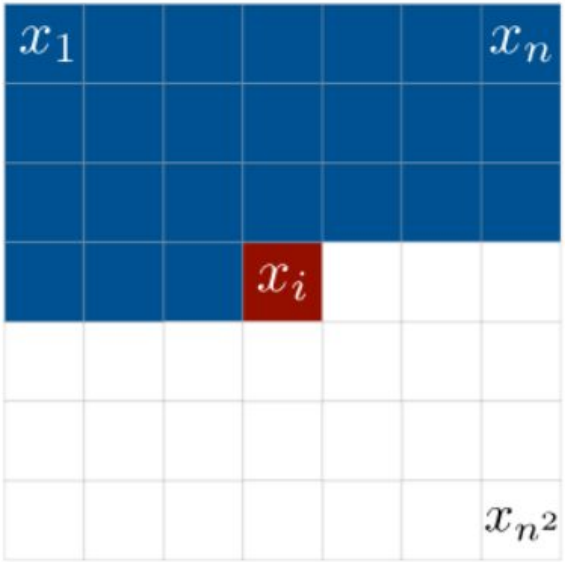
Usage:

- Generate data
- Representation learning
- Probabilistic inference

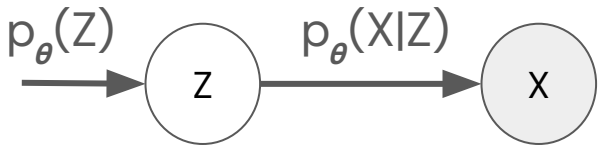


Recap - autoregressive models

- Autoregressive models: $p_\theta(x) = \Pi\, p_\theta(x_i | x_{<i})$
- Probability distributions factorize into a product of factors
- We can efficiently represent **p** via conditional independence and/or neural parameterizations
- Problem: Sampling is slow



Recap - Variational Autoencoders



- Combine simple models into more flexible ones:
  $p(x) = \int p_\theta(x|z)p(z)dz$ with "simple" models $p_\theta(x|z)$, $p(z)$
- *Train via amortized variational inference using an inference network to approximate the posterior* $q_\phi(z \mid x)$
- Directed model permits efficient generation: $z \sim p(z)$, $x \sim p_\theta(x|z)$
- Problem: samples are usually blurry

Recap - Energy Based Models

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

Pros:

1. extreme flexibility: can use pretty much any function $f_\theta(\mathbf{x})$ you want

Cons: Computing $Z(\theta)$ numerically is hard.

1. Sampling from $p_\theta(\mathbf{x})$ is in general hard
2. Evaluating and optimizing likelihood $p_\theta(\mathbf{x})$ is hard (learning is hard)

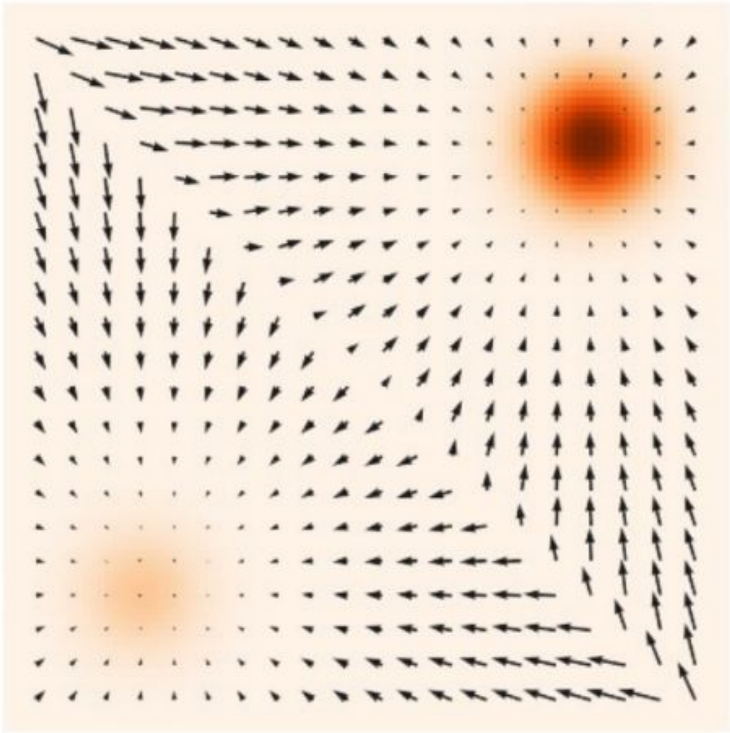
The (Stein) Score Function: Definition

Given a probability density $p_\theta(\mathbf{x})$, its (Stein) score function is defined as

$$\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$$

- This is the gradient of the density with respect to the input.
- Note that this is **different** from the gradient $\nabla_\theta \log p_\theta(\mathbf{x})$ of the log-density with respect to the parameters θ (which is what we use in gradient descent).
- The score function points in the direction of increasing model probability.

Score function: 2D



No Need For Normalization

Suppose we have an energy-based model of the form

$$p_\theta(\mathbf{x}) = \frac{1}{\int \exp(f_\theta(\mathbf{x}))d\mathbf{x}} \exp(f_\theta(\mathbf{x})) = \frac{1}{Z(\theta)} \exp(f_\theta(\mathbf{x}))$$

- Computing a score function of the model does not require $Z(\theta)$:

$$\nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = \nabla_\mathbf{x} f_\theta(\mathbf{x}) - \nabla_\mathbf{x} \log Z(\theta) = \nabla_\mathbf{x} f_\theta(\mathbf{x})$$

- **Idea**: Instead of learning $p_\theta(\mathbf{x})$, learn a model $\nabla_\mathbf{x} \log p_\theta(\mathbf{x})$ of the score function of the data $\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$!

How can we use a score-based model?

Langevin dynamics

Langevin dynamics is an MCMC sampling process that allows us to sample from $p_\theta(\mathbf{x})$ using only its score $\nabla_x \log p_\theta(\mathbf{x})$.

- Initialize $\mathbf{x}_0$ from a noise distribution
- For steps $t = 1, 2, \ldots, T$:
  - Sample a noise variable $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - $\mathbf{x}_t = \mathbf{x}_{t-1} + \frac{\alpha_t}{2} \nabla_x \log p_\theta(\mathbf{x}) + \sqrt{\alpha_t} \epsilon_t$

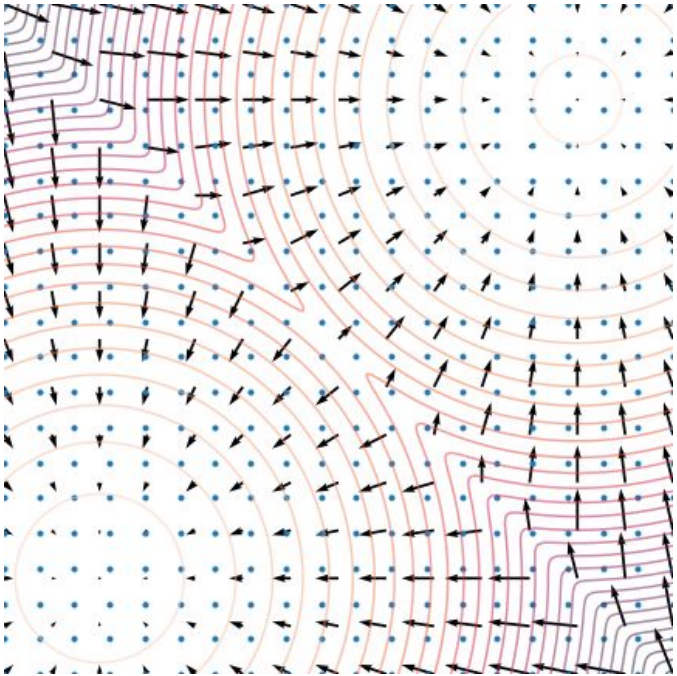
Note that:

- This process can be seen as gradient ascent on $\log p_\theta(\mathbf{x})$;
- It is guaranteed to produce samples from $p_\theta(\mathbf{x})$ when the step size $\alpha_t \to 0$ as $T \to \infty$ at the right annealing rate.

Sampling with score-based models

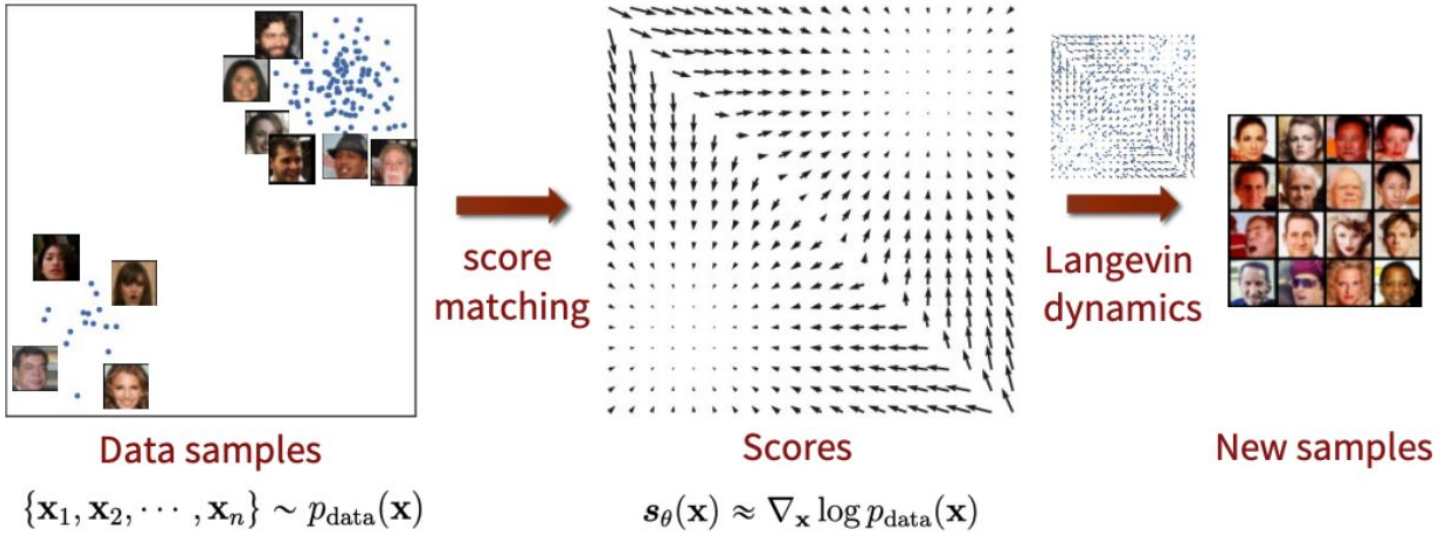
Langevin dynamics

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\, \mathbf{z}_i, \quad i = 0, 1, \cdots, K$$



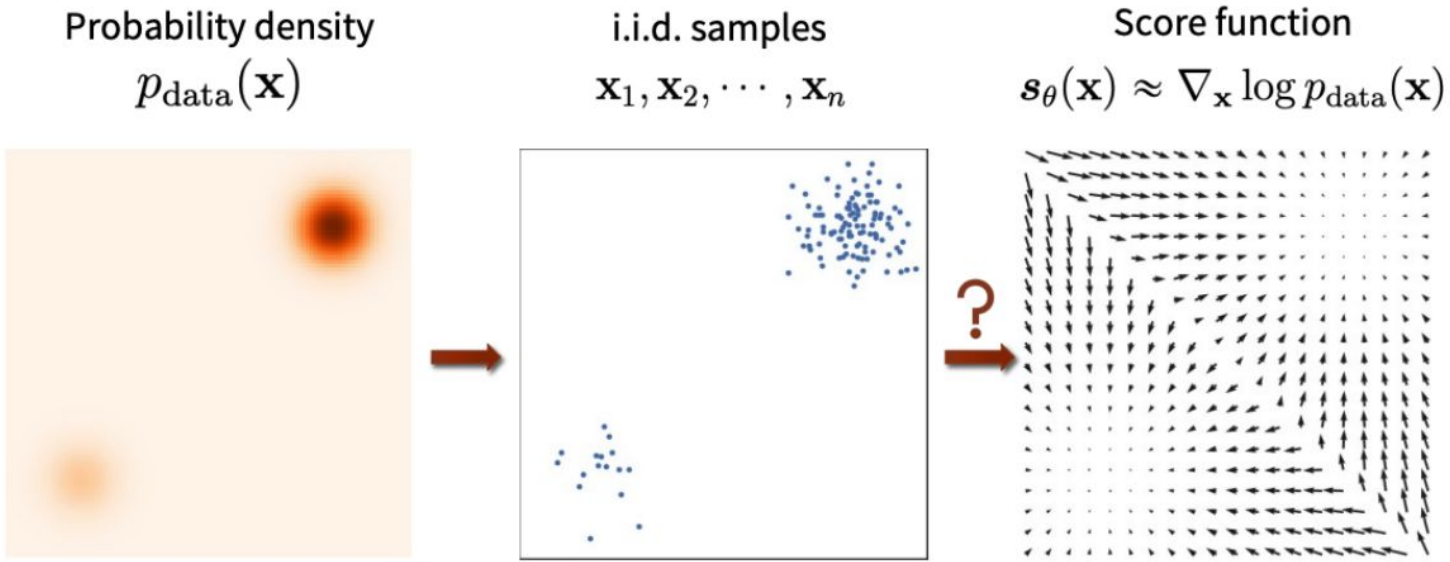
Score Function Modeling

Thus, our high-level strategy is to learn $s_\theta(\mathbf{x}) \approx \nabla_x \log p_\theta(\mathbf{x})$ from a set of samples, and then generate new $\mathbf{x}$ via Langevin dynamics.



Score Function Estimation

The idea of score function estimation is to learn a model $s_\theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$ of the score function from a dataset of sample points.

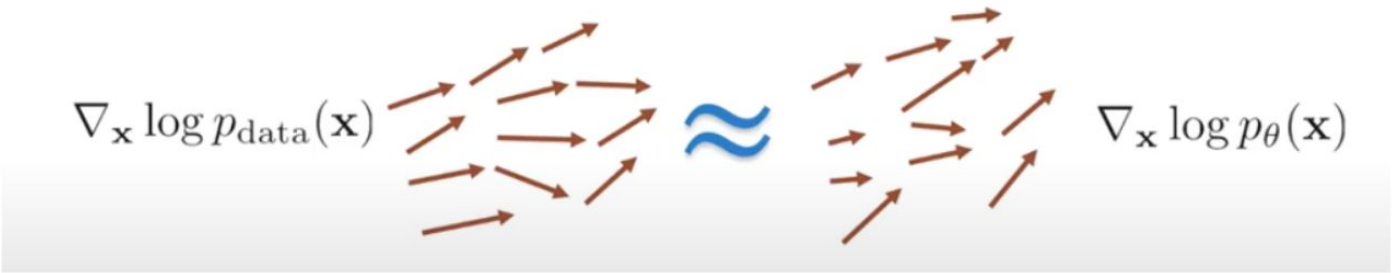


Score Function Estimation

- We are given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$
- Our task is to estimate $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$
- Our model is a parameterized function $s_\theta(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$
- Our objective is that $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \approx s_\theta(\mathbf{x})$

How do we fit score functions?

Data and model scores are vector fields; we want them to be aligned:



Score Matching Via Fisher Divergence

- Formally, we use the Fisher divergence:

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[||\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}) - s_\theta(\mathbf{x})||_2^2\right]$$

- It's the average Euclidean distance between two vectors over all $\mathbf{x}$
- The Fisher divergence is zero if and only if $\nabla_{\mathbf{x}}\log p_{\text{data}}(\mathbf{x}) \approx s_\theta(\mathbf{x})$

Score Matching

Problem: We don't know the ground truth score.

Solutions:

1. Score Matching: $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\frac{1}{2}\|s_\theta(\mathbf{x})\|_2^2 + \text{tr}\left(\nabla_{\mathbf{x}} s_\theta(\mathbf{x})\right)\right]$

2. Denoising Score Matching: $\mathbb{E}_{x\sim p(x),\epsilon=\mathcal{N}(0,I)}\left[\|\epsilon - s_\theta(x+\sigma\cdot\epsilon)\|_2^2\right]$

Denoising Score Matching

Tweedie's Formula:

Given a noisy data point **y** (using a clean data point **x** and adding Gaussian noise to it), and the MMSE estimate $\hat{x} = \mathbb{E}[x|y]$, then:

$$\nabla_y \log p_\sigma(y) = \frac{1}{\sigma^2}(\hat{x} - y)$$

$$where\ p_\sigma(y) = \int p_\sigma(y|x)\, p_{data}(x)\, dx.$$

Proof

$$\begin{aligned}
\nabla_y \log p_\sigma(y) &= \frac{1}{p_\sigma(y)} \frac{\partial}{\partial y} p_\sigma(y) \\
&= \frac{1}{p_\sigma(y)} \frac{\partial}{\partial y} \int p_\sigma(y|x)\, p(x)\, dx \\
&= \int \frac{p(x)}{p_\sigma(y)} \frac{\partial}{\partial y} \mathcal{N}\left(y \,|\, x, I\sigma^2\right) dx
\end{aligned}$$

The derivative of a Gaussian has quite a neat form, which will allow us to continue:

$$\begin{aligned}
\frac{\partial}{\partial y} \mathcal{N}\left(y \,|\, x, I\sigma^2\right) &= \frac{\partial}{\partial y} \frac{1}{Z} e^{-\frac{1}{2\sigma^2}\|x-y\|^2} \\
&= \frac{1}{Z} e^{-\frac{1}{2\sigma^2}\|x-y\|^2} \frac{\partial}{\partial y} \left(-\frac{1}{2\sigma^2}\|x-y\|^2\right) \\
&= \mathcal{N}\left(y \,|\, x, I\sigma^2\right) \frac{1}{\sigma^2}(x-y)
\end{aligned}$$

Proof

Substituting the derivative of a Gaussian:

$$\begin{aligned}
\nabla_y \log p_\sigma(y) &= \int \frac{p(x)\, p_\sigma(y|x)}{p_\sigma(y)} \frac{1}{\sigma^2}(x-y)\, dx \\
&= \frac{1}{\sigma^2} \int p(x|y)(x-y)\, dx \\
&= \frac{1}{\sigma^2} \left[\int x p(x|y)\, dx - y \int p(x|y)\, dx \right] \\
&= \frac{1}{\sigma^2} \left[\mathbb{E}_x[x|y] - y \right] = \frac{1}{\sigma^2}(\hat{x} - y)
\end{aligned}$$

Denoising Score Matching

This means we can compute the score using $\nabla_y \log p_\sigma(y) = \frac{1}{\sigma^2}(\hat{x} - y)$

And optimize the following objective function:

$$\begin{aligned}
\mathbb{E}_{x\sim p(x), y\sim p(y|x)=\mathcal{N}(x,\sigma^2 I)}\left[\|\nabla_y \log p_\sigma(y) - s_\theta(y)\|_2^2\right] &= \mathbb{E}_{x\sim p(x), y\sim p(y|x)=\mathcal{N}(x,\sigma^2 I)}\left[\|\hat{x} - y - s_\theta(y)\|_2^2\right] \\
&\approx \mathbb{E}_{x\sim p(x), y\sim p(y|x)=\mathcal{N}(x,\sigma^2 I)}\left[\|x - y - s_\theta(y)\|_2^2\right] \\
&= \mathbb{E}_{x\sim p(x), \epsilon=\mathcal{N}(0,I)}\left[\|-\epsilon - s_\theta(x + \sigma\cdot\epsilon)\|_2^2\right]
\end{aligned}$$

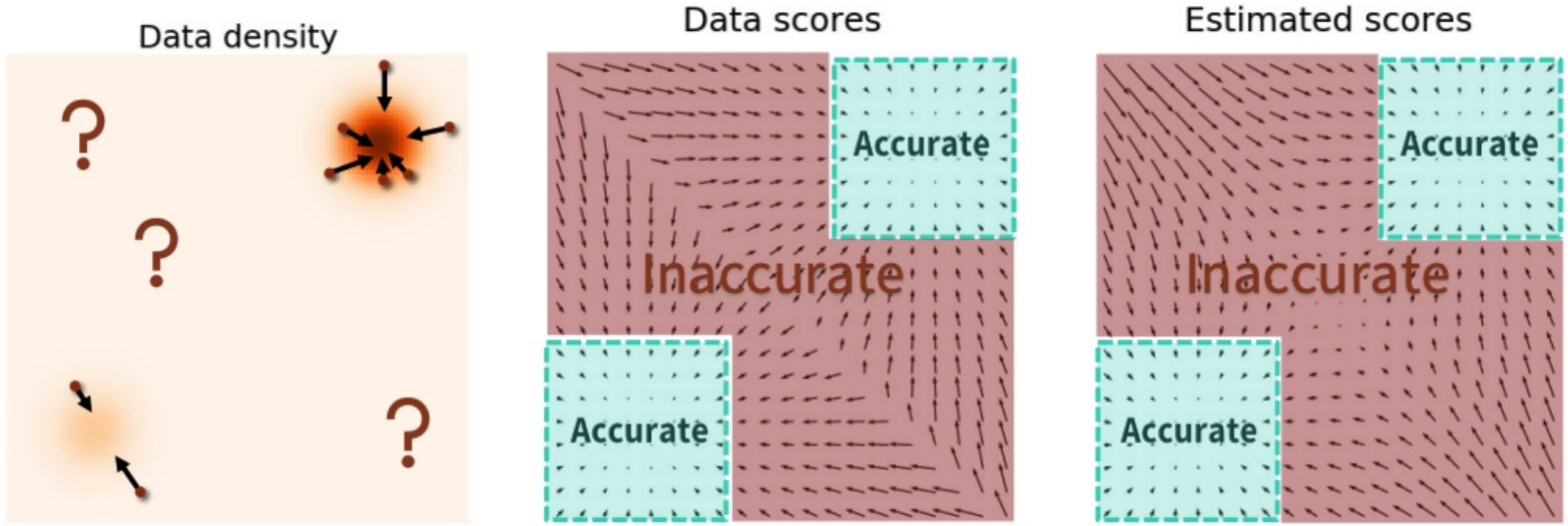
Manifold Hypothesis

- Images lie on a low dimensional manifold in space

- Intuition of denoising score matching:
  - adding noise takes us out of the manifold
  - Going back is the quickest direction to increase probability

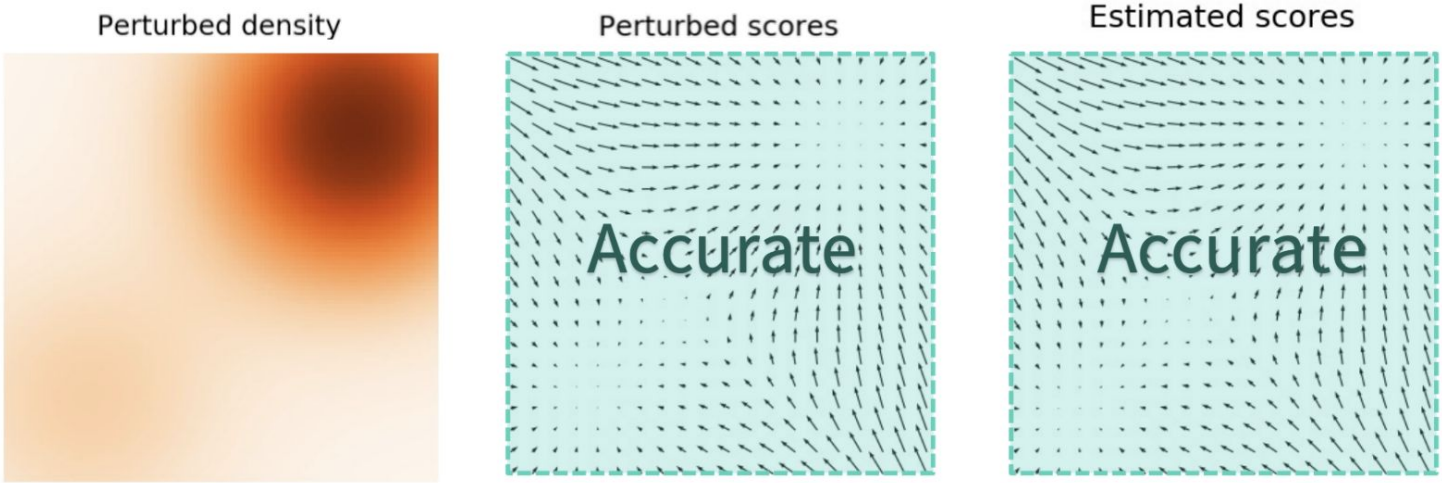
- Problem: how is the score defined outside the manifold?

Learning and Sampling in Low Density Regions



- Not enough samples to learn the score in low density regions
- Score is very small and unstable ⇒ to weak to generate samples

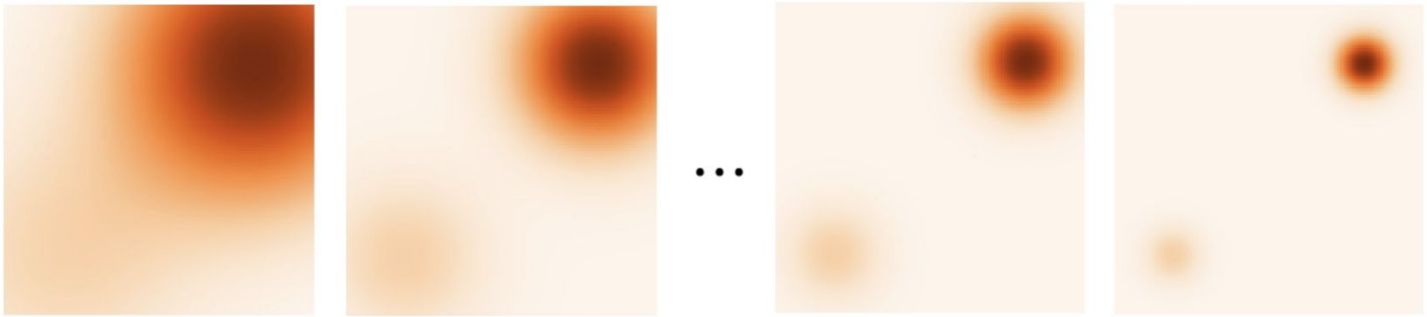
Adding Strong Noise



- Score is well defined everywhere
- Enough samples for learning, smoother distribution for sampling
- The distribution is different than the original - tradeoff

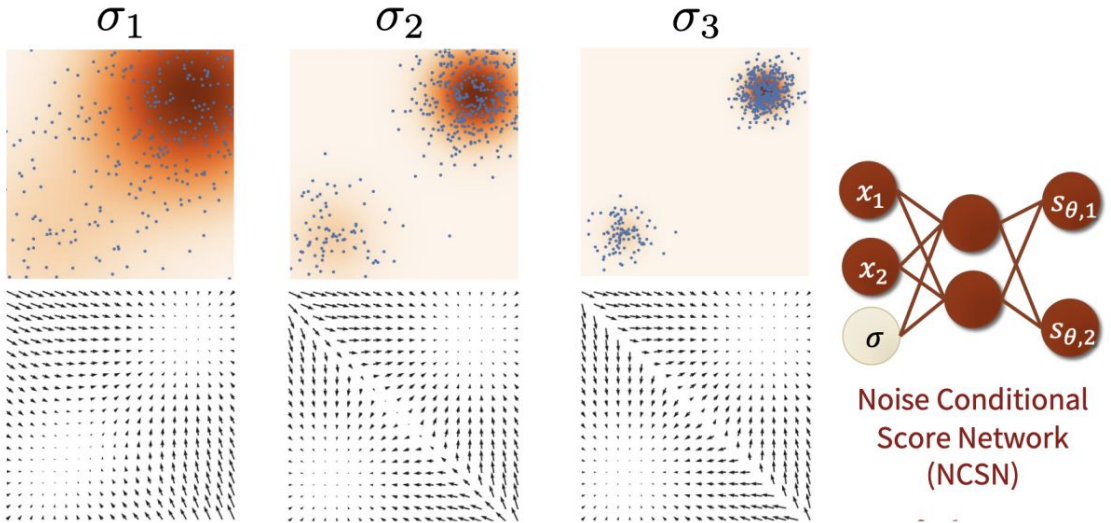
Multi-Scale Noise

$$\sigma_1 > \sigma_2 > \cdots > \sigma_{L-1} > \sigma_L$$



- Train using both small and large amounts of noise
- Start sampling using larger noise, gradually reduce the noise

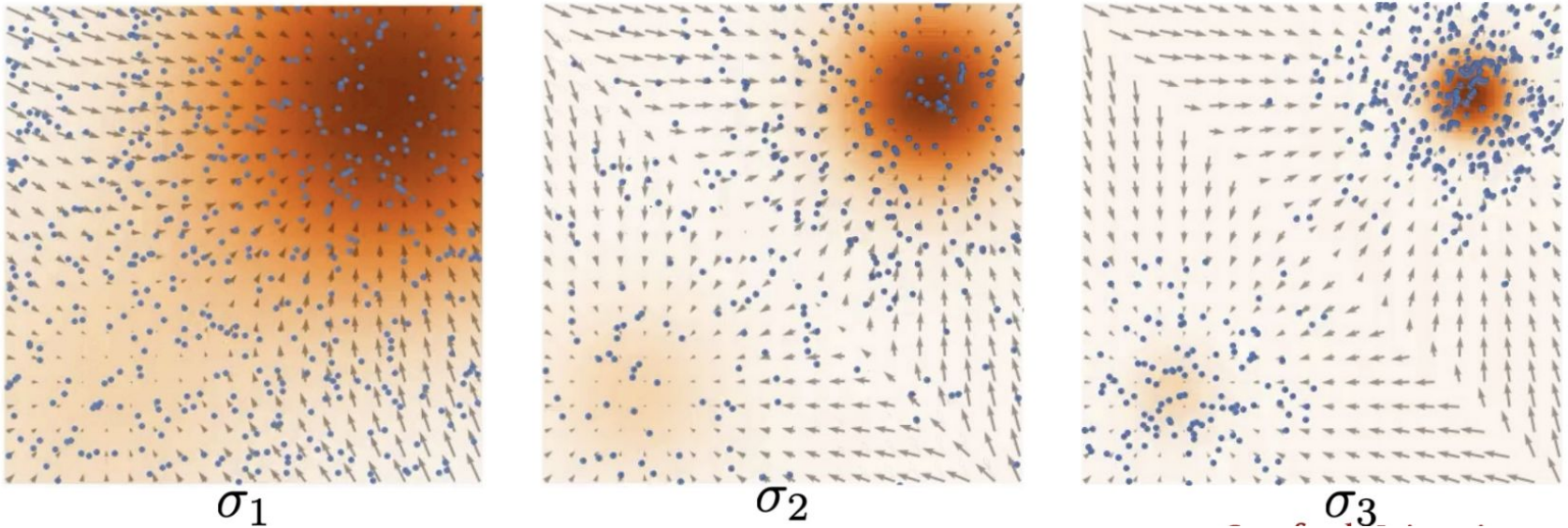
Noise Conditional Score Networks



Same model is trained for all σ. Trained using the following objective:

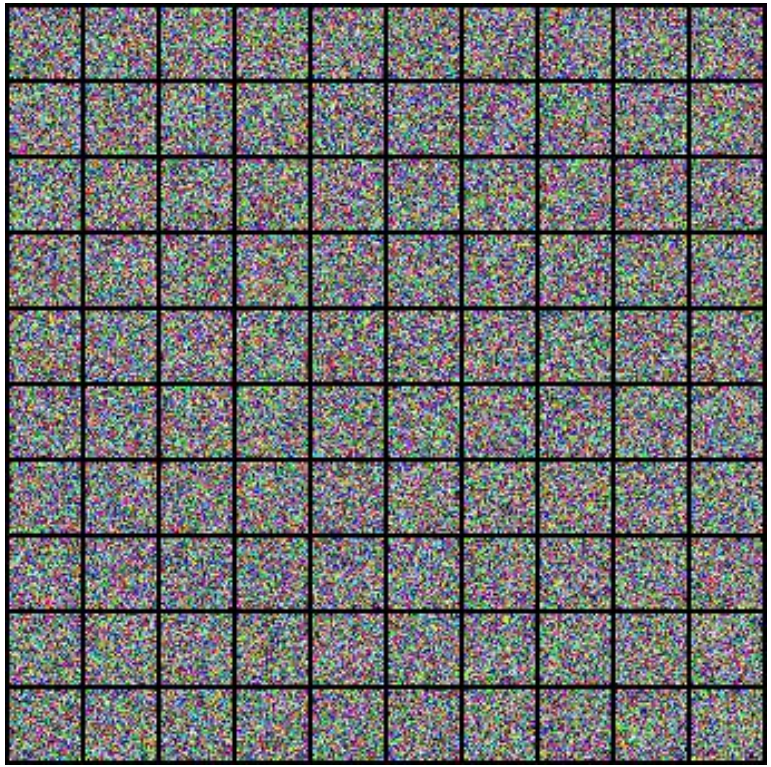
$$\mathbb{E}_{\sigma\sim\{\sigma_1,\sigma_2,\ldots,\sigma_L\}}\mathbb{E}_{x\sim p(x),\epsilon=\mathcal{N}(0,I)}\left[\|\epsilon - s_\theta(x+\sigma\cdot\epsilon,\sigma)\|_2^2\right]$$

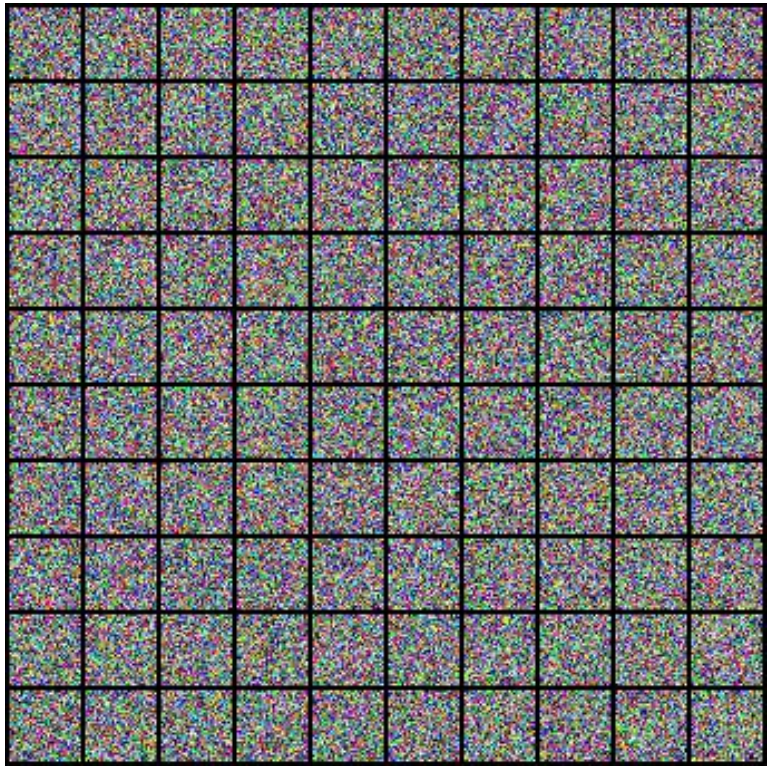
Annealed Langevin Dynamics



Annealed Langevin Dynamics

- Initialize $\mathbf{x}_0$ from a noise distribution
- For noise levels $\sigma_\ell \in \{\sigma_1, \sigma_2, ..., \sigma_L\}$:
  - Set step size to $\alpha_t = \beta \cdot \frac{\sigma_\ell}{\sigma_L}$ for some $\beta > 0$
  - For steps $t = 1, 2, \ldots, T$:
    - Sample a noise variable $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    - $\mathbf{x}_t = \mathbf{x}_{t-1} + \frac{\alpha_t}{2} s_\theta(\mathbf{x}, \sigma_\ell) + \sqrt{\alpha_t}\epsilon_t$
  - Set $\mathbf{x}_0 = \mathbf{x}_T$: start next run at the end of last run.





Outline

- Score Based Models
  - Annealed Langevin Dynamics
- Diffusion Models as Latent Variable Models
  - ELBO
  - DDPM
- Connection to Continuous Normalizing Flows
  - ODEs and SDEs
  - Flow Matching

Latent variable models with variational training

Sohl-Dickstein, Weiss, Maheswaranathan and Ganguli,
"Deep Unsupervised Learning using Nonequilibrium Thermodynamics", 2015.

Score based models

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

Denoising Diffusion Probabilistic Models

Jonathan Ho, Ajay Jain, Pieter Abbeel, 2020

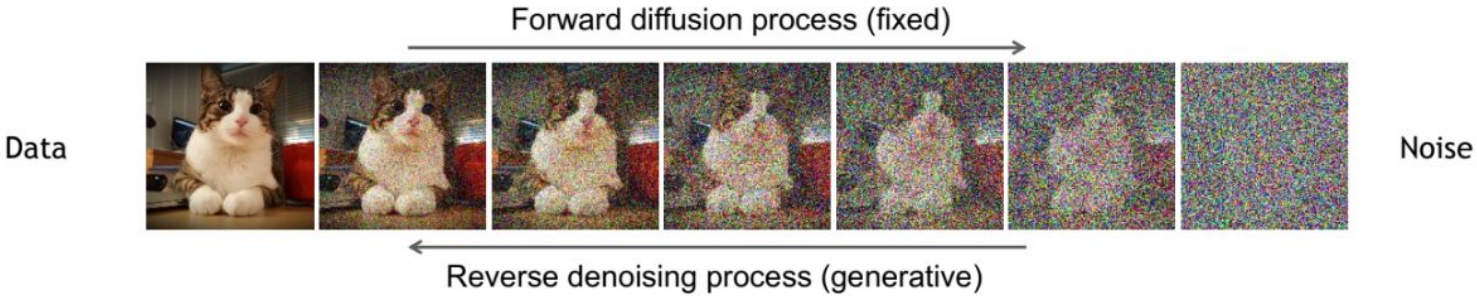
Score-Based Generative Modeling through Stochastic Differential Equations

Song, Sohl-Dickstein, Kingma, Kumar, Ermon and Poole 2021.

Denoising Diffusion Probabilistic Models

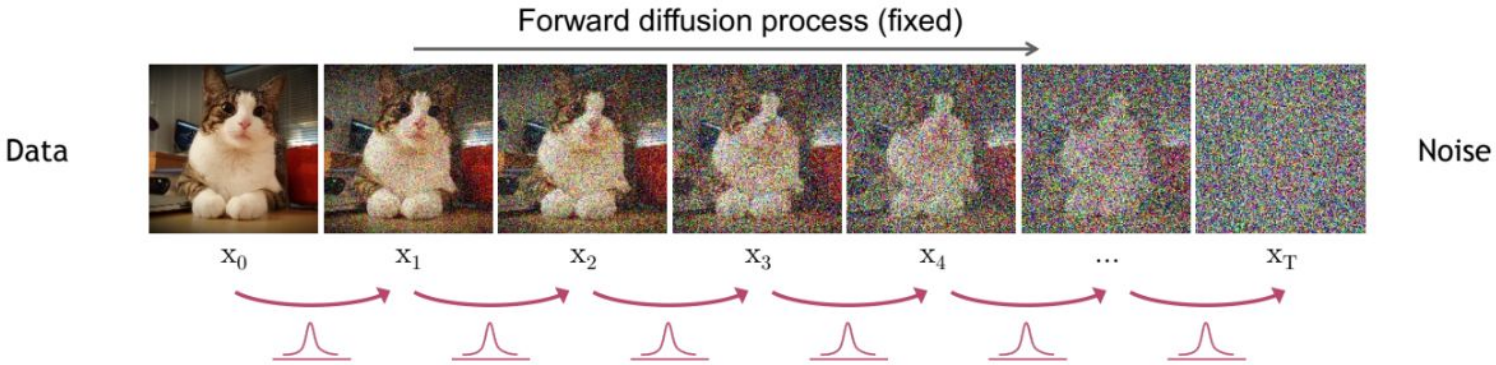
Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising



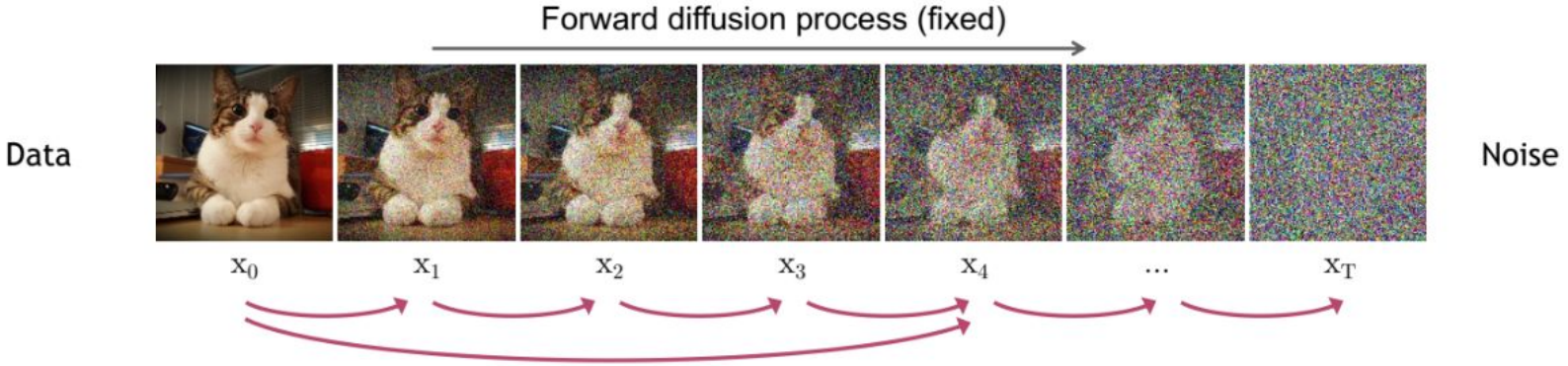
Forward diffusion process

The formal definition of the forward process in T steps:



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \qquad \text{(joint)}$$

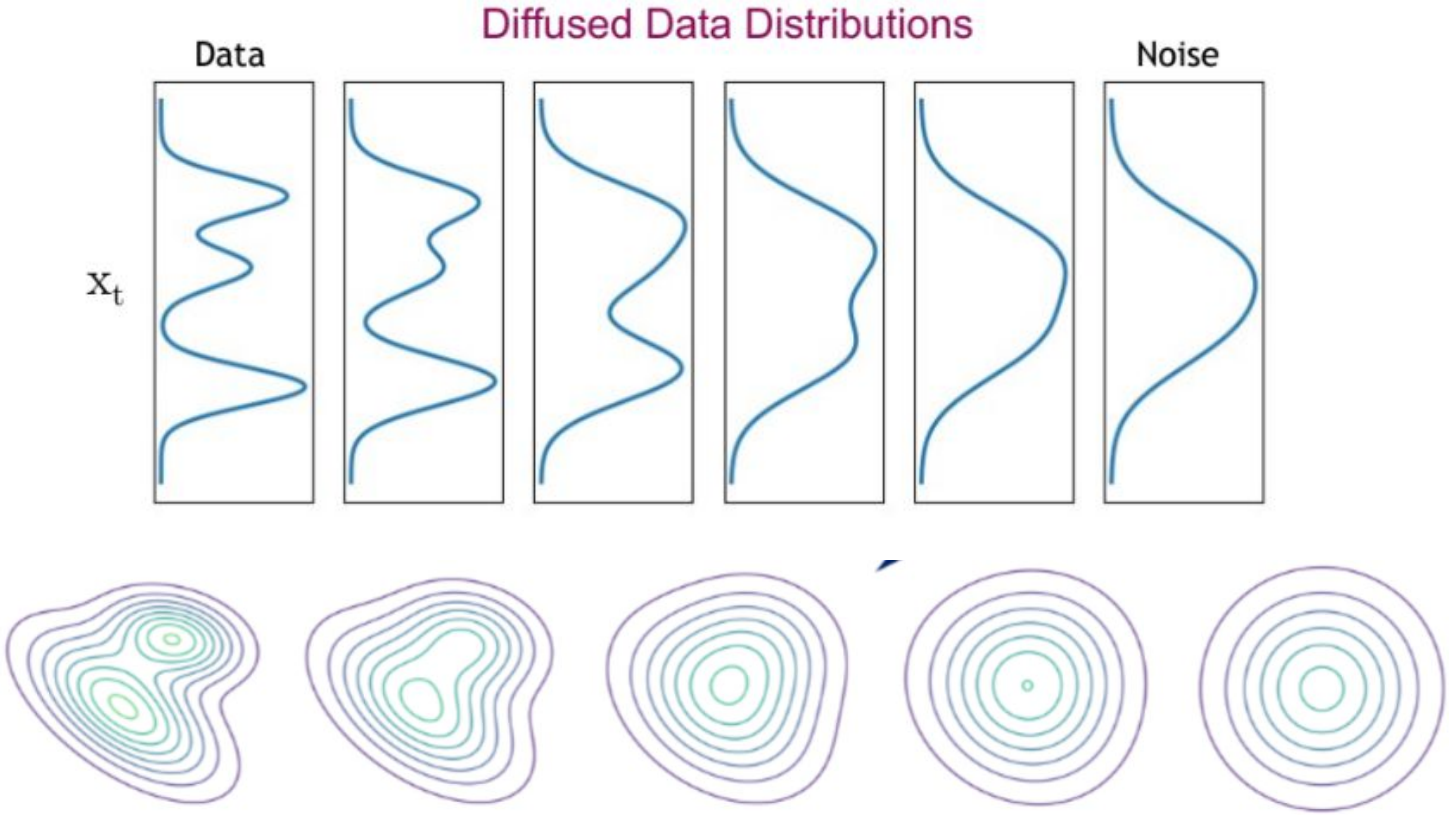
Forward diffusion process



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 $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

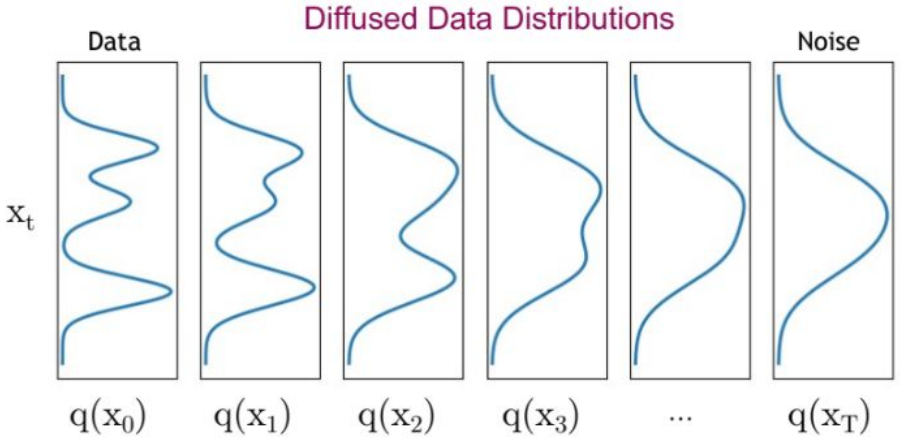

Diffused Data Distributions
Data
Noise
x_t

Marginals of q

So far, we discussed the diffusion kernel $q(\mathbf{x}_t|\mathbf{x}_0)$ but what about $q(\mathbf{x}_t)$?

$$\underbrace{q(\mathbf{x}_t)}_{\text{Diffused data dist.}} = \int \underbrace{q(\mathbf{x}_0, \mathbf{x}_t)}_{\text{Joint dist.}} d\mathbf{x}_0 = \int \underbrace{q(\mathbf{x}_0)}_{\text{Input data dist.}} \underbrace{q(\mathbf{x}_t|\mathbf{x}_0)}_{\text{Diffusion kernel}} d\mathbf{x}_0$$

The diffusion kernel is Gaussian convolution.



We can sample $\mathbf{x}_t \sim q(\mathbf{x}_t)$ by first sampling $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ and then sampling $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$ (i.e., ancestral sampling).

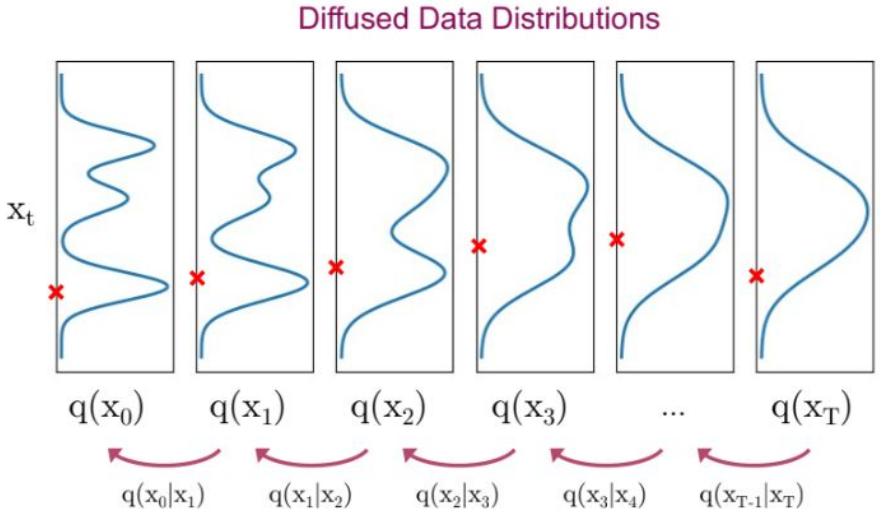
Reverse Process

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

Generation:

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

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$

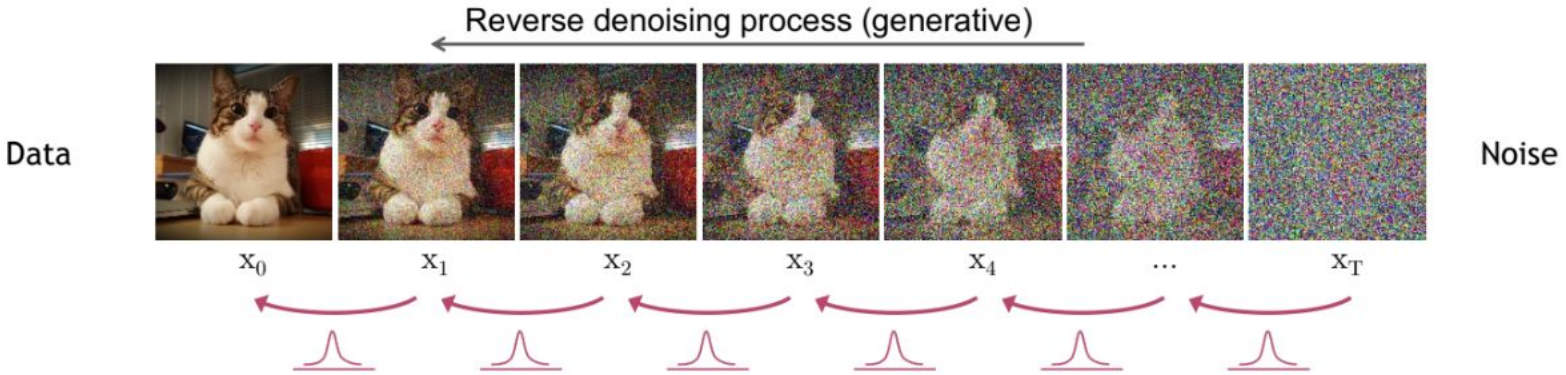


In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.

Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Normal distribution if β_t is small in each forward diffusion step.

Learning to denoise

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)

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

Training using ELBO

Training is performed by optimizing the usual variational bound on negative log likelihood:

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

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

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

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

$$
\begin{aligned}
L &= \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t > 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t > 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} - \log \frac{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}{q(\mathbf{x}_1|\mathbf{x}_0)} \right] \\
&= \mathbb{E}_q \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T|\mathbf{x}_0)} - \sum_{t > 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1) \right]
\end{aligned}
$$

Objective function

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

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

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$. Ho et al. NeurIPS 2020 observe that:

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

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

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

Objective function

With this parameterization

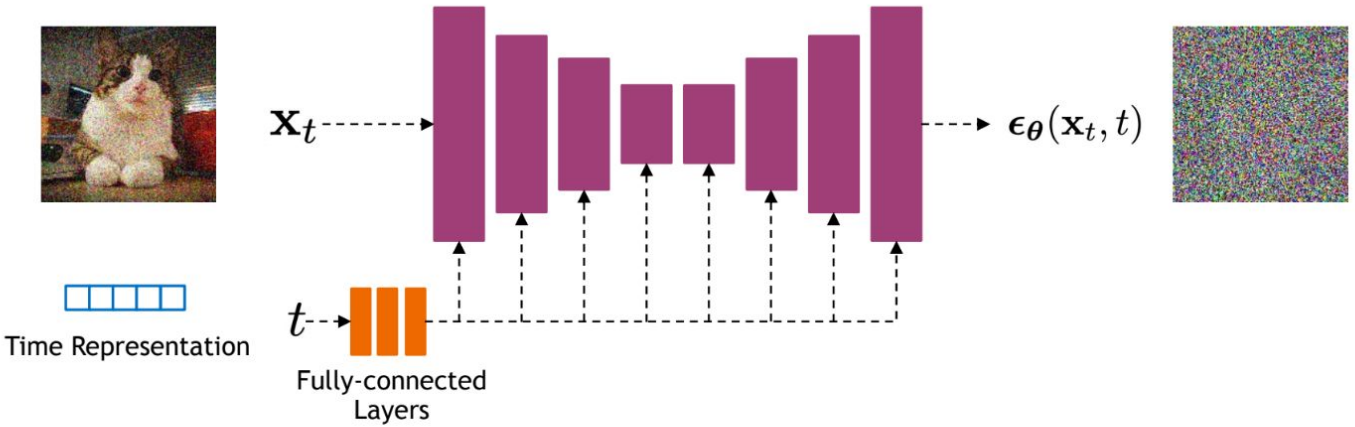
$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \Bigg[\underbrace{\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} ||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon}_{\mathbf{x}_t},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

Simplify by setting λ = 1

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

Implementation

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



Time representation: sinusoidal positional embeddings or random Fourier features.

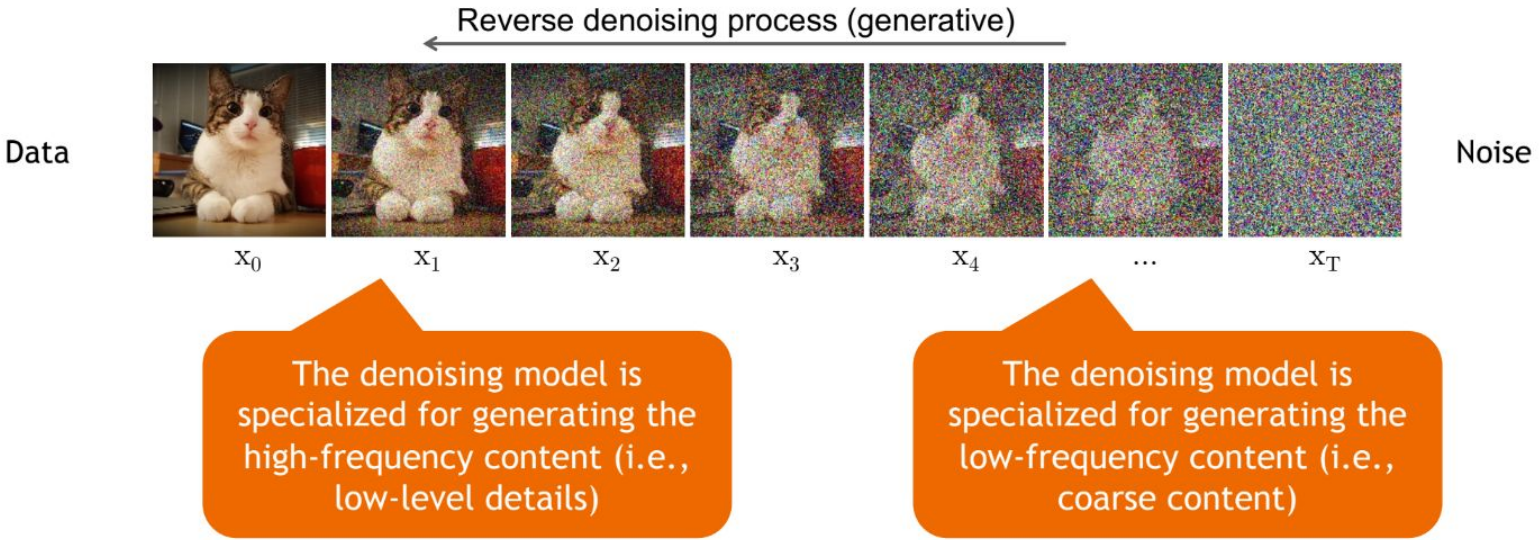
Algorithm 1 Training

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: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 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** $\mathbf{x}_0$

Content - Detail tradeoff



The weighting of the training objective for different timesteps is important!

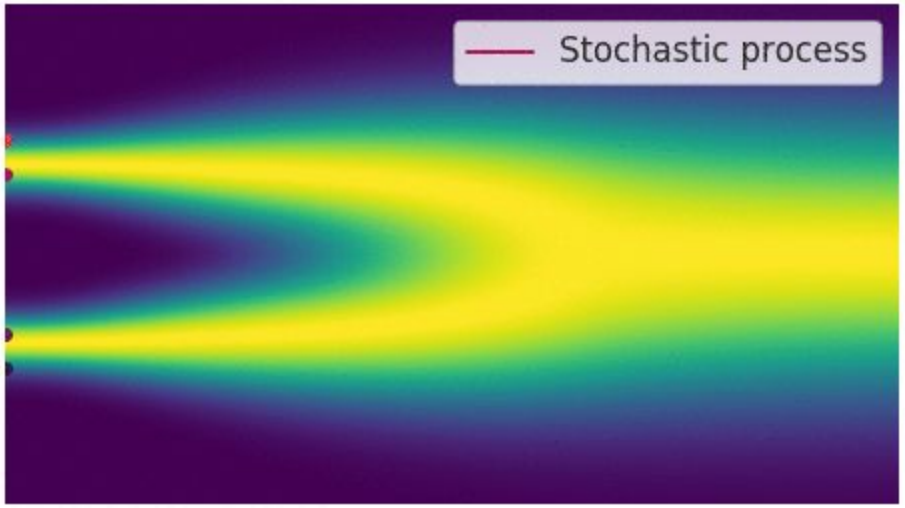
Table 1: CIFAR10 results. NLL measured in bits/dim.

Model	IS	FID	NLL Test (Train)
Conditional			
EBM [11]	8.30	37.9	
JEM [17]	8.76	38.4	
BigGAN [3]	9.22	14.73	
StyleGAN2 + ADA (v1) [29]	**10.06**	**2.67**	
Unconditional			
Diffusion (original) [53]			≤ 5.40
Gated PixelCNN [59]	4.60	65.93	3.03 (2.90)
Sparse Transformer [7]			**2.80**
PixelIQN [43]	5.29	49.46	
EBM [11]	6.78	38.2	
NCSNv2 [56]		31.75	
NCSN [55]	8.87 ± 0.12	25.32	
SNGAN [39]	8.22 ± 0.05	21.7	
SNGAN-DDLS [4]	9.09 ± 0.10	15.42	
StyleGAN2 + ADA (v1) [29]	$\mathbf{9.74} \pm 0.05$	3.26	
Ours (L, fixed isotropic $\mathbf{\Sigma}$)	7.67 ± 0.13	13.51	≤ 3.70 (3.69)
Ours (L_{simple})	9.46 ± 0.11	**3.17**	≤ 3.75 (3.72)

Outline

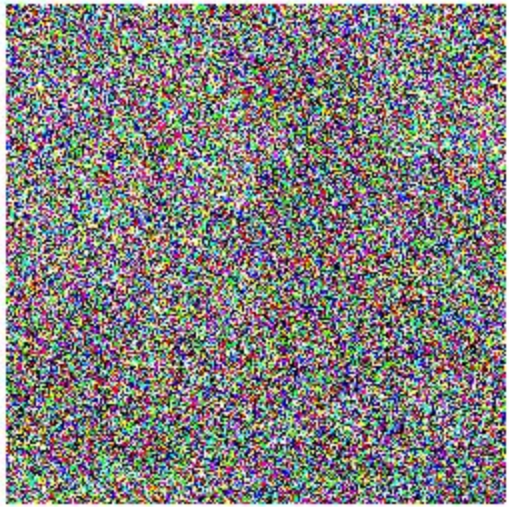
- Score Based Models
  - Annealed Langevin Dynamics
- Diffusion Models as Latent Variable Models
  - ELBO
  - DDPM
- Connection to Continuous Normalizing Flows
  - ODEs and SDEs
  - Flow Matching

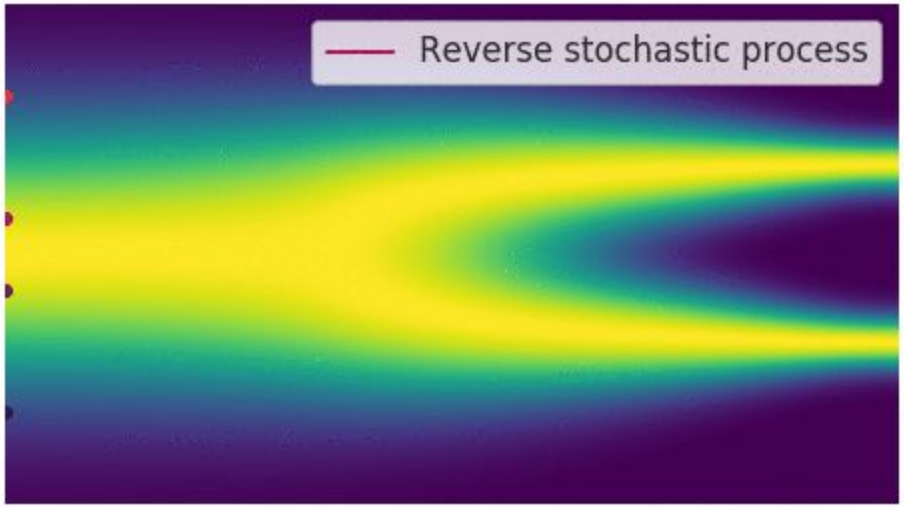
Stochastic differential equations (SDEs)



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

Reversing the SDE

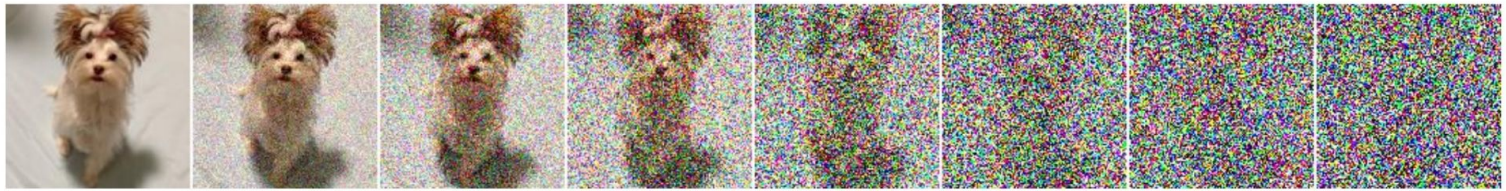




$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

Forward SDE (data → noise)

$\mathbf{x}(0)$ — $\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$ → $\mathbf{x}(T)$



score function

$\mathbf{x}(0)$ ← $\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$ — $\mathbf{x}(T)$

Reverse SDE (noise → data)

Probability Flow ODE

Consider reverse generative diffusion SDE:

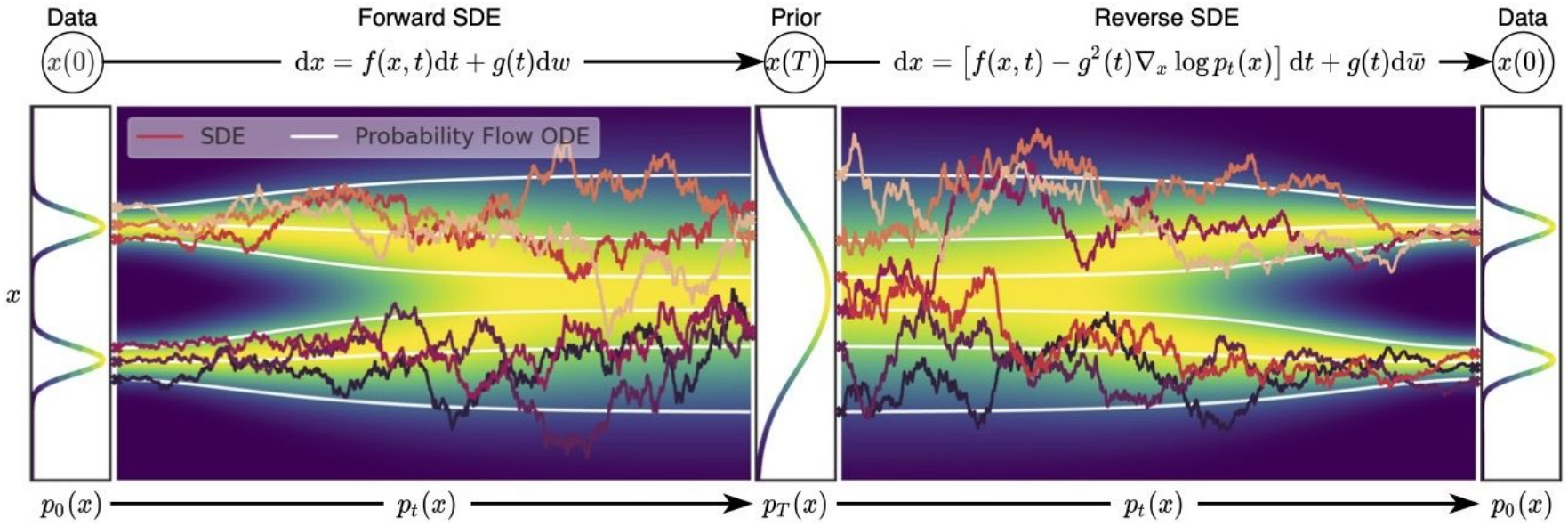
$$\mathrm{d}\mathbf{x}_t = -\frac{1}{2}\beta(t)\left[\mathbf{x}_t + 2\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t)\right]\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\bar{\boldsymbol{\omega}}_t$$

In distribution equivalent to **"Probability Flow ODE"**:

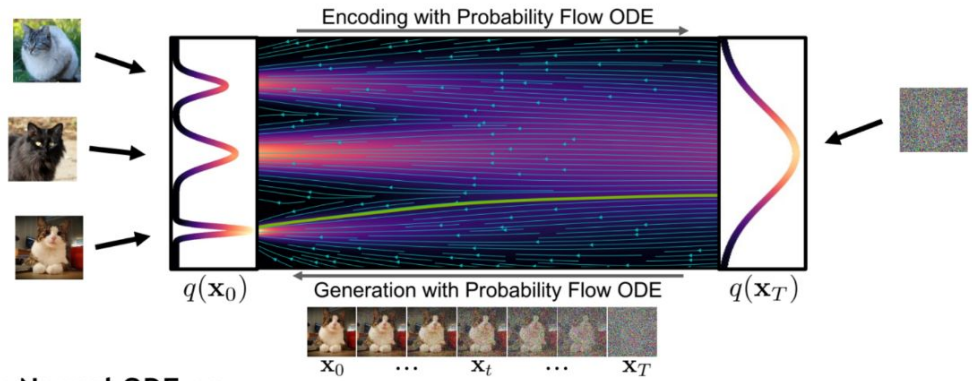
(solving this ODE results in the same $q_t(\mathbf{x}_t)$ when initializing $q_T(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$)

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

Probability Flow ODE



Probability Flow ODE



- **Probability Flow ODE as Neural ODE or Continuous Normalizing Flow (CNF):**

$$\mathrm{d}\mathbf{x}_t = -\frac{1}{2}\beta(t)\left[\mathbf{x}_t + \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]\mathrm{d}t$$

$$\left(\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = -\frac{1}{2}\beta(t)\left[\mathbf{x}_t + \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]\right)$$

- Enables use of **advanced ODE solvers**
- **Deterministic encoding and generation** (semantic image interpolation, etc.)
- **Log-likelihood computation** (instantaneous change of variables):

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) = \log p_T(\mathbf{x}_T) - \int_0^T \mathrm{Tr}\left(\frac{1}{2}\beta(t)\frac{\partial}{\partial \mathbf{x}_t}\left[\mathbf{x}_t + \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right]\right)\mathrm{d}t$$

Computing exact log-likelihood

Method	Negative log-likelihood (bits/dim) ↓ on CIFAR-10	Negative log-likelihood (bits/dim) ↓ on ImageNet 32x32
Sparse Transformer	**2.80**	-
Image Transformer	2.90	3.77
Ours	2.83	**3.76**

Why use Differential Equation framework?

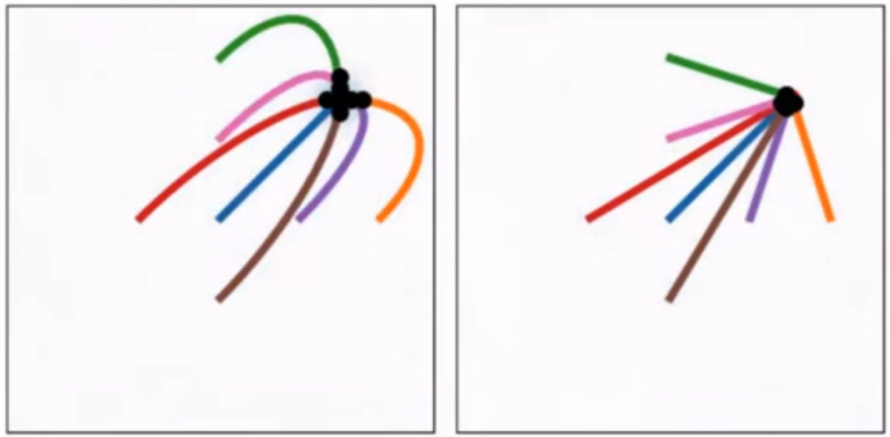
Advantages of the Differential Equation framework for Diffusion Models:

- Can leverage broad existing literature on **advanced and fast SDE and ODE solvers**
- Allows us to construct deterministic **Probability Flow ODE**
    - Deterministic Data Encodings
    - Log-likelihood Estimation
- **Clean mathematical framework** based on Diffusion Processes and Score Matching; connections to Neural ODEs, Continuous Normalizing Flows and Energy-based Models

Flow Matching

Generalization of diffusion models:

- Define a mapping from latent sample to data sample such that the marginal distributions agree with:
  $p(x_0)$, $p(x_T)$
- Train the mapping parameters by matching the velocity vector for all timesteps.
- Can result in more efficient trajectories



Diffusion (VP) Cond-OT

Diffusion Models: Summary

Pros:

- Simple objective function for training
- Good results
- Can be used for probabilistic reasoning
- Easy to extend to conditional modeling

Cons:

- Slow sampling
- No natural implementation for discrete data