#lecture
previous lecture: 100 School/junior spring/CS 4782/lecture notes/lecture 16
next lecture:
class: CS 4782
slides: week9_2_slides.pdf

- topics covered

Overview

- Review of diffusion models
- Score-based generative modeling offers a different perspective on DDPM
- Conditional diffusion models can be used to generate samples conditioned on other text or image
- Stable diffusion is perfect in the latent space of pre-trained autoencoders

Denoising diffusion models

Denoising diffusion models consist of two processes

1. Forward diffusion process that gradually adds noise to input
2. Reverse denoising process that learns to generate data by denoising

Forward pass

Can sample x_t in closed-form as

$$q(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$
$$\implies x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \;\; \epsilon \sim \mathcal{N}(0, I), \bar{\alpha}_t \in (0,1)$$

Noise schedules

- Define the noise schedule in terms of $\bar{\alpha}_t \in (0,1)$
  - Some monotonically decreasing function from 1 to 0
- Cosine noise schedule

$$\bar{\alpha}_t = \cos\left(\frac{0.5\pi t}{T}\right)^2$$

Key idea

We introduce a generative model to approximate the reverse process

$$p(x_T) = \mathcal{N}(x_T; 0, I)$$
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \sigma_t^2 I)$$
$$\implies p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

The learning objective is

$$\mathbb{E}_{q(x_t|x_0)}[D_{\text{KL}}(q(x_{t-1}|x_t, x_0) \;\; || \;\; p_\theta(x_{t-1}|x_t)]$$

Training objective

- Bound the likelihood with the ELBO
  - Exactly like VAEs

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\left[\log \frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}\right]$$

Learning Objective!

$$= \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0) \parallel p(\boldsymbol{x}_T))}_{\text{prior matching term}} - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right]}_{\text{denoising matching term}}$$

where $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is the tractable posterior distribution:

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

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

Parameterizing the denoising model

Since both $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ are normal distributions, the KL divergence has a simple form

$$L_{t-1} = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}||\tilde{\mu}(x_t, x_0) - \mu_\theta(x_t, t)||^2\right] + C$$

- $\tilde{\mu}(x_t, x_0)$ is the true mean of the posterior $q(x_{t-1}|x_t, x_0)$
- $\mu_\theta(x_t, t)$ is the predicted mean, which is generated by the model $p_\theta(x_{t-1}, x_t)$
  - We train our model to predict this

Recall that

$$x_t = \sqrt{\bar{\alpha}_t} + \sqrt{(1-\bar{\alpha}_t)}\epsilon$$

Then, we have that

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{1-\beta_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\right)\epsilon$$

So, we represent the mean of the denoising model using a noise-prediction network

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

With this parameterization

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

:s231n.stanford.edu/slides/2023/lecture_15.pdf

Alternative diffusion parameterization: Data prediction

We can also view the diffusion network as learning to predict the original data

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

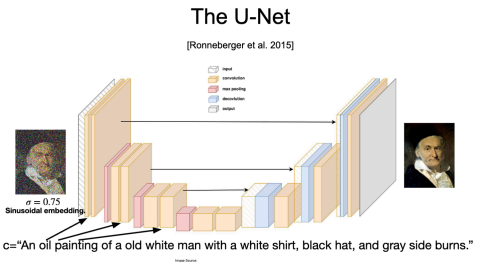
$$\implies x_0 = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon}{\sqrt{\alpha_t}}$$

$$\implies x_\theta(x_t, t) = \frac{x_t - \sqrt{1-\bar{\alpha}_t}\epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

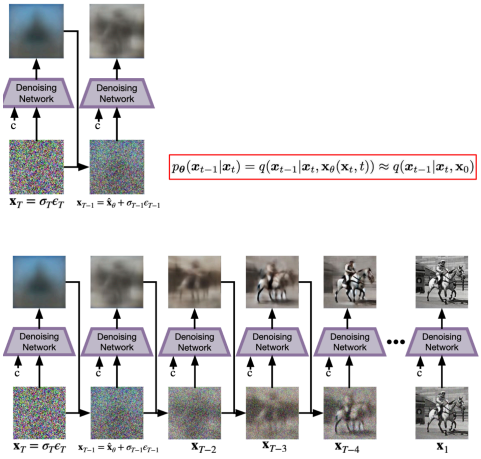
So, the loss becomes:

$$L_{t-1} = \mathbb{E}_{x_0 \sim q(x_0), \epsilon \sim \mathcal{N}(0,I)} \left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2 \right] + C$$

U-Net



Diffusion sampling



Training algorithm

Repeat until convergence:

1. $x_0 \sim q(x_0)$
   - Sample original image from image distribution
2. $t \sim \mathcal{U}(1, 2, \ldots, T)$
   - Sample random time step uniformly
3. $\epsilon \sim \mathcal{N}(0, 1)$
   - Sample Gaussian noise
4. Optimizer step on $L(\theta)$
   - Model predicts noise applied at time step t and calculate loss

Inference sampling algorithm

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ← Sample pure Gaussian noise

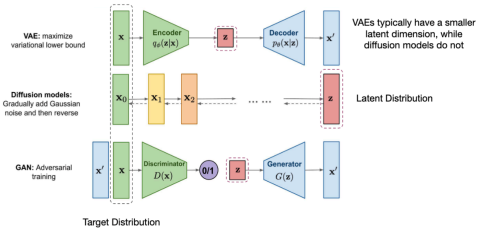
For $t = T, T-1 \ldots, 1$

  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$ ← Sample Gaussian noise to apply to image

  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ ← Predict noise applied to image and remove that noise

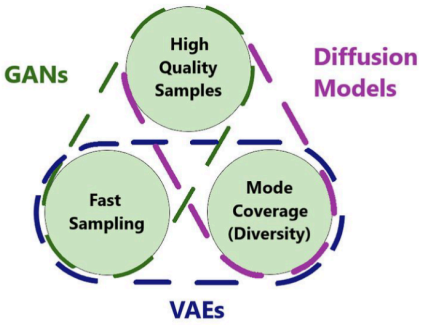
Return $\mathbf{x}_0$

Generative modeling



Diffusion models vs VAEs vs GAN

Source



Score-based models

We want to model the probability density function as follows:

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

Where $Z_\theta > 0$ is a normalizing constant such that

$$\int_x p_\theta(x)dx = 1$$

This will allow us to understand the distribution of images x and draw from it.

We want to maximize the log-likelihood of the data, which gives us the following objective:

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(x_i)$$

Problem: The normalization constant Z_θ is intractible, since we usually cannot compute it.

Solution: We will approximate the score function, which is given as follows:

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

Therefore,

$$s_\theta(\mathbf{x}) = -\nabla_\mathbf{x} f_\theta(\mathbf{x})$$

Then, we can use gradient ascent to view areas of high density in the PDF and draw from the distribution.

We train a model s_θ that approximates the real score function of the distribution.

Score function:

Let p_θ be a probability density function. Then, the score function is defined as

$$s_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

- It points in the direction where the probability density increases the fastest
- It is called the score function

Score-based models

Langevin dynamics provide a way to sample from a probability distribution, even when it is unnormalized. The sampling process is defined by the following update rule:

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

In this equation, each $\mathbf{x}_i$ represents a sample from the target distribution. The parameter ϵ is a step size that determines how large each update is, effectively controlling the speed of exploration. The term $\mathbf{z}_i$ is drawn from a standard Gaussian distribution $\mathcal{N}(0, 1)$ and adds randomness to the updates, which helps prevent the sampler from getting stuck in local modes of the distribution.

Score matching

Score matching is a method used to train models to match the *score function* of a probability distribution without needing access to the partition function (normalizing constant), which is often intractable.

The score function is the gradient of the log-probability with respect to the input:

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

In score matching, we define a loss function that penalizes the difference between a model's predicted score and the true score:

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

However, the true score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ is unknown, so direct minimization of this loss is not possible. Fortunately, score matching provides an alternative way to compute this loss without needing the true score function directly, using integration by parts.

Training

Denoising Score Matching and Training Objective

We begin with the score matching objective:

$$\sum_{t=1}^{T} \lambda(t)\, \mathbb{E}_{\mathbf{x}_t} \left[\| s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \|_2^2 \right]$$

This objective tries to learn a score function s_θ that approximates the score of the noisy distribution $p_t(\mathbf{x}_t)$.

Using **denoising score matching**, we can replace the unknown score with an expression involving the known conditional $q_t(\mathbf{x}_t|\mathbf{x})$.

$$\sum_{t=1}^{T} \lambda(t)\, \mathbb{E}_{\mathbf{x}_t} \left[\| s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}) \|_2^2 \right]$$

If

$$\mathbf{x}_t \sim \mathcal{N}(\mathbf{x}, \sigma_t^2 I)$$

then

$$q_t(\mathbf{x}_t \mid \mathbf{x})$$

Note that $\mathbf{x}_t \sim \mathcal{N}(\mathbf{x}, \sigma_t^2 I)$, therefore, $q_t(\mathbf{x}_t|\mathbf{x})$ is Gaussian. Therefore, we have:

$$\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}) = -\frac{\mathbf{x}_t - \mathbf{x}}{\sigma_t^2}$$

Substituting this into the loss gives:

$$\sum_{t=1}^{T} \lambda(t)\, \mathbb{E}_{\mathbf{x}_t} \left[\left\| s_\theta(\mathbf{x}_t, t) + \frac{\mathbf{x}_t - \mathbf{x}}{\sigma_t^2} \right\|_2^2 \right]$$

which is rewritten as:

$$\sum_{t=1}^{T} \lambda(t)\, \mathbb{E}_{\mathbf{x}_t} \left[\left\| s_\theta(\mathbf{x}_t, t) - \frac{\mathbf{x} - \mathbf{x}_t}{\sigma_t^2} \right\|_2^2 \right]$$

Connecting Denoising Score Matching to DDPMs

We know that:

$$\nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t \mid \mathbf{x}) = -\frac{\mathbf{x}_t - \mathbf{x}}{\sigma_t^2}$$

Assuming the forward noise process:

$$\mathbf{x}_t = \mathbf{x} + \sigma_t \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

We substitute into the gradient:

$$\begin{aligned} \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t \mid \mathbf{x}) &= -\frac{\mathbf{x} + \sigma_t \boldsymbol{\epsilon} - \mathbf{x}}{\sigma_t^2} \\ &= -\frac{\sigma_t \boldsymbol{\epsilon}}{\sigma_t^2} \\ &= -\frac{\boldsymbol{\epsilon}}{\sigma_t} \end{aligned}$$

This shows that the score function is:

$$s_\theta(\mathbf{x}_t, t) = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sigma_t}$$

Reformulating the Loss

Starting from the denoising score matching loss:

$$\lambda(t)\, \mathbb{E}_{\mathbf{x},t} \left[\left\| s_\theta(\mathbf{x}_t, t) - \frac{\mathbf{x} - \mathbf{x}_t}{\sigma_t^2} \right\|_2^2 \right]$$

Substitute the score identity:

$$s_\theta(\mathbf{x}_t, t) = -\frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sigma_t}, \quad \frac{\mathbf{x} - \mathbf{x}_t}{\sigma_t^2} = -\frac{\boldsymbol{\epsilon}}{\sigma_t}$$

The loss becomes:

$$\lambda(t)\, \mathbb{E}_{\mathbf{x},t} \left[\left\| \frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}{\sigma_t} - \frac{\boldsymbol{\epsilon}}{\sigma_t} \right\|_2^2 \right] = \frac{\lambda(t)}{\sigma_t^2}\, \mathbb{E}_{\mathbf{x},t} \left[\| \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon} \|_2^2 \right]$$

From this, we see that this loss function is the mean squared error loss used in DDPMs. We train the model to predict the noise using the ground truth noise as supervision.

So, DDPM training can be interpreted as a special case of denoising score matching when the corruption distribution is Gaussian.

Conditional diffusion with classifier guidance

- May not have access to paired data for training
- Use Bayes' rule to decompose the conditional score into the unconditional score and a likelihood term

$$\nabla_{x_t} \log p_t(x_t|y) = \nabla_{x_t} \log p_t(x_t) + \nabla_{x_t} \log p_t(y|x_t)$$

- Only need to train a classifier on noised data

Controlling diffusion models

- Use a classifier to guide noise

$$x_{t-1} = \underbrace{\hat{x_\theta}(x_t)}_{\text{denoise}} + \underbrace{\sigma_{t-1}\epsilon_{t-1}}_{\text{add random Gaussian noise}} + \underbrace{\alpha \nabla_{x_t} \log(p(y|x_t))}_{\text{add guide noise}}$$

- $\hat{x_\theta}$ is a model that de-noises the input x_t
- σ_{t-1} controls how much Gaussian noise we add at time step $t-1$
- ϵ_{t-1} is the Gaussian noise
- α controls the strength of the guidance
  - How much you bias your sample towards y
- $\nabla_{x_t} \log(p(y|x_t))$ is the score function
  - It points in the direction of steepest ascent of the probability density function $p(y|x_t)$
  - y is a label that we have selected
  - This term will point in the direction that will bring us closer to points x_i that are more likely to match the label y

Classifier-free guidance

- Train a joint conditional and unconditional diffusion model
- Conditioning information is added by concatenating to input or cross attending
- Modified conditional distribution

$$\log \tilde{p}_t(\mathbf{x}_t|\mathbf{y}) \propto p_t(\mathbf{x}_t|\mathbf{y}) p_t(\mathbf{y}|\mathbf{x}_t)^w$$

- Conditional sampling$$
  \nabla{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{x}_t \mid \mathbf{y})
  = \nabla{\mathbf{x}_t} \log p_t(\mathbf{x}_t)

- w \left(\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{y})

- \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \right)
  $$

Classifier-free guidance

- Significantly improves quality of conditional models
- Used by practically every conditional diffusion model

Modified conditional distribution:

$$\log \tilde{p}_t(x_t|y) \propto p_t(x_t|y) p_t(y|x_t)^w$$

- $\log \tilde{p}_t(x_t|y)$ is the modified conditional distribution that we sample from
- $p_t(x_t|y)$ is the original conditional distribution
- $p_t(y|x_t)$ is the likelihood term
- w
  - Higher w represents stronger influence from the condition y, leading to more faithful samples, but possibly less diverse

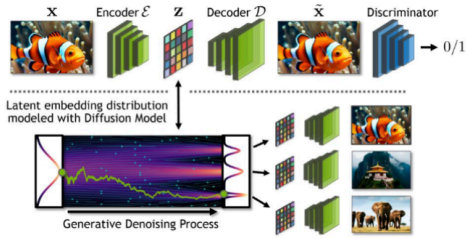
Conditional sampling:

$$\nabla_{x_t} \log \tilde{p}_t(x_t|y) = \nabla_{x_t} \log p_t(x_t) + w(\nabla_{x_t} \log p_t(x_t|y) - \nabla_{x_t} \log p_t(x_t))$$

Classifier-free guidance

- Significantly improves quality of conditional models
- Used by practically every conditional diffusion model

Latent diffusion



- Latent diffusion models offer excellent trade-off between performance and compute demands
- LDM with appropriate regularization, compression, downsampling ratio and strong autoencoder reconstruction
  - Computationally efficient diffusion model in latent space (compression and lower resolution)
  - Yet very high-performance latent diffusion + autoencoder + discriminator)
  - Highly flexible (can adjust autoencoder for diffusion tasks and data

Latent diffusion

Many state-of-the-art large-scale text-to-image models

- Stability AI's stable diffusion
- Meta's emu
- OpenAI's Dall-E 3

Review

- Diffusion models can be used to generate high quality samples
- Diffusion was introduced simultaneously from two different perspectives
  - Variational perspective
  - Score-based generative modeling
- **Conditional diffusion** models can be used to generate samples conditioned on other text or image
- **Stable diffusion** is performed in the latent spaces of pre-trained autoencoders

LLaDA (Large language diffusion with mAsking)

- Nie et al. (2025) - is the first paper to scale up discrete diffusion models to 8 billion parameters

Generative modeling principle

Key claim: There is nothing inherently better about left-to-right text generation. The key to scalability is maximizing the likelihood of the training data:

$$\max_{\theta} \mathbb{E}_{x\sim p_{\text{data}}}[\log p_\theta(x)] \iff \min_{\theta} \mathrm{KL}(p_{\text{data}}(x) \,||\, p_\theta(x))$$

Autoregressive models factorize the joint probability into product of conditional probabilities

Discrete diffusion models optimize a lower-bound on likelihood (ELBO)

Challenging the autoregressive framework

Beyond left-to-right text generation: Traditional autoregressive models generate text one token at a time, predicting the next token based on a left-to-right context using causal attention. This sequential nature limits flexibility and imposes a strict generation order.

In contrast, discrete diffusion models offer a fundamentally different approach. They allow for any-order token prediction by employing bi-directional attention and perform generation through a series of denoising steps—typically TTT steps for the entire

sequence—rather than one step per token.

Noising text

- The most popular way to noise text is to mask tokens independently at random
- Given a text sequence and $t \in (0, 1)$, we can noise it to any intermediate x_t

$$q_{t|0}(x_t^i|x_0^i) = \begin{cases} 1-t\ ;\ x_t^i = x_0^i \\ t\ ;\ x_t^i = M \end{cases}$$

Since each token is masked independently, we have that

$$q_{t|0}q(\mathbf{x}_t|\mathbf{x}_0) = \prod_{i=1}^{L} q_{t|0}(\mathbf{x}_t^i|\mathbf{x}_0^i)$$

Denoising text

The key idea when denoising text is to optimize the likelihood of the training data. In diffusion models, we don't have access to the likelihood. So, instead we optimize the likelihood lower-bound (ELBO).

$$\mathbb{E}_{p_{\text{data}}(\mathbf{x}_0)}[\log p_\theta(\mathbf{x}_0)] \leq -\mathbb{E}_{t,\mathbf{x}_0,\mathbf{x}_t}\left[\frac{1}{t}\sum_{i=1}^{L} \mathbb{1}\{\mathbf{x}_t^i = \mathrm{M}\} \log p_\theta(\mathbf{x}_0^i \mid \mathbf{x}_t)\right]$$

For masked diffusion models, the ELBO has a very intuitive form:

- Given a partially masked sequence, the model predicts all marked tokens simultaneously
- The ELBO computes the likelihood only on the masked tokens

Generating text

The key idea is to predict the original text, then add back "noise".

Given a partially/fully masked sequence $\mathbf{x}_t$, we want to generate $\mathbf{x}_s$.

1. Run the discrete diffusion model to simultaneously predict all of the masked tokens
2. For each of the masked tokens, remark them with probability $\frac{s}{t}$.

Repeat this process until you generate $\mathbf{x}_0$, the unmasked text.

Implementation details: Inference

LLaDA applies semi-autoregressive sampling, which works as follows:

1. The sequence is divided into blocks, generated from left-to-right
2. Within each block, apply LLaDA to unmask/generate

Inference

Semi-autoregressive sampling:

- LLaDA applies semi-autoregressive sampling
  - Sequence is divided into blocks, generated from left-to-right
  - Within each block, apply LLaDA to unmask/generate