Lecture Notes on Diffusion Models

Written by Srivatsa Kundurthy & Alex Kozik, based on Cornell CS 4782 Lecture on Diffusion Models Part II.

Recap: Generative Models

What should be clear at this point is that generative modeling is a fundamentally challenging task.

Generative Adversarial Networks (GANs) train a generator to synthesize data that fools a discriminator trained to distinguish between real and fake samples. This adversarial setup has led to remarkable sample quality, especially in high-resolution image generation. However, GANs come with their own challenges: mode collapse, unstable training, and the lack of an explicit density model make them difficult to train and evaluate probabilistically.

With Variational Autoencoders (VAEs), we approached this challenge as an encoder-decoder problem: mapping an input x to a latent representation $z = q_\phi(x)$ using a learned encoder, and then reconstructing the input via $x \approx p_\theta(z)$. While VAEs offer fast inference, they often struggle with generating high-quality, detailed samples.

So we're left with a tradeoff:

- VAEs: stable training, but blurry generations.
- GANs: sharp images, but unstable and non-probabilistic.

However, both are lightning fast. Perhaps this is part of the issue. For example, for the VAE, the one-shot nature of both encoding and decoding makes it so that we have to compress the input into a latent space in a single step and attempt to reconstruct it just as quickly. This abruptness can limit its expressivity.

So what if we slowed things down? What if we transformed data into noise—and noise back into data—*gradually*?

This is the key intuition behind diffusion models, which were introduced in the previous lecture.

Recap: Denoising Diffusion Models

The diffusion models we study consist of two parts, a *forward diffusion process* and a *reverse denoising process.*

- The forward diffusion process gradually adds noise to the input.
- The reverse denoising process learns to undo the noise at each step to reconstruct data.

Each process occurs over several time steps.

Forward Process

The forward process q defines a Markov chain that adds Gaussian noise to the original data x_0 over T steps to get increasingly noisy versions $x_1, x_2, \ldots, x_T$, until such a point that $x_T \approx \mathcal{N}(0, I)$.

While we can iteratively add noise $\epsilon \sim \mathcal{N}(0, I)$ to each x_t, it is advantageous to imagine a closed form. That is:

$$q(x_t \mid x_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\, x_0,\ (1 - \bar{\alpha}_t) I\big)$$

We can compute this as

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + (1 - \bar{\alpha}_t)\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, with $\alpha_t \in (0, 1)$ defines the noise schedule.

Noise Schedules

In diffusion models, noise schedules determine how much Gaussian noise is added to a clean data point (i.e. image) at each step of the forward diffusion process.

In doing so, the noise schedule defines **how quickly or slowly** this corruption happens.

A popular noise schedule is the cosine schedule, defined as

$$\bar{\alpha}_t = \cos^2\big(0.5\pi\, t/T\big)$$

Reverse Process

The forward process adds noise step-by-step to get x_T. It destroys the original image x_0. To be able to generate new data, we need to build a model that reconstructs data starting from pure noise.

The reverse process does exactly this: it learns to denoise from x_T to x_0 using a learned generative model.

We model the reverse process as a Markov chain of Gaussians.

Final timestep distribution:

$$p(x_T) = \mathcal{N}(x_T; 0, I)$$

This serves as the starting point for generation. Now, we want our model to learn to denoise each timestep, until we are back at a good reconstruction of the original input data x_0. Formally, this idea is captured by

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

where

- The mean $\mu_\theta(x_t, t)$ is predicted by a neural network.
- The variance σ_t^2 is often fixed or learned, depending on the implementation.

In other words, we are learning to model the conditional distribution of a cleaner image given a noisier one. The full generative model is a chain of these reverse steps:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

Of course, the challenge lies in learning p_θ, and learning it *well*.

Diffusion Learning Objective

Our objective is actually quite similar to the VAE. In an ideal world, we'd match the distribution $p(x)$ from which the data is drawn from. Of course, this is usually not tractable.

Instead, let's see if we can come up with an Evidence Lower Bound that *is tractable*. That is, we want to come up with some lower bound on $\log p(x)$ that we can seek to maximize with a model.

ELBO Derivation

We want the marginal likelihood of our data point x_0 given a latent noise sequence $x_{1:T}$.

$$\log p(x_0) = \log \int p(x_{0:T})\, dx_{1:T} \quad \text{where} \quad p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t).$$

Next, we introduce a variational posterior (an identity term here) and recognize the expression as an expectation over the forward process.

$$\log p(x_0) = \log \int p(x_{0:T}) \frac{q(x_{1:T} \mid x_0)}{q(x_{1:T} \mid x_0)}\, dx_{1:T} = \log \mathbb{E}_{q(x_{1:T}|x_0)}\Big[\tfrac{p(x_{0:T})}{q(x_{1:T}|x_0)}\Big].$$

Since log is concave, we can apply Jensen's Inequality

$$\log p(x_0) \;\geq\; \mathbb{E}_{q(x_{1:T}|x_0)}\Big[\log \tfrac{p(x_{0:T})}{q(x_{1:T}|x_0)}\Big] \;\equiv\; \mathcal{L}(x_0).$$

Next, we plug in the forward and reverse factors we derived.
$p(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$ for the forward process. and $q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$ for the reverse process.
With some algebra, we get

$$\mathcal{L}(x_0) = \mathbb{E}_q\Big[\log p(x_T) + \sum_{t=1}^{T} \log p_\theta(x_{t-1} \mid x_t) - \sum_{t=1}^{T} \log q(x_t \mid x_{t-1})\Big].$$

Subsequently, we re-index and collect terms into:

1. **Reconstruction term**
$$\mathbb{E}_{q(x_1|x_0)}\big[\log p_\theta(x_0 \mid x_1)\big].$$
This term assess to what degree our learned p_θ allows us to recover the original data x_0.

2. **Prior matching term**:
$$-\mathbb{E}_{q(x_{T-1}|x_0)}\big[D_{\mathrm{KL}}(q(x_T \mid x_{T-1}) \| p(x_T))\big].$$
This term assesses our ability to fully destroy the input image (that is, make it so that $p(x_T) = \mathcal{N}(0, I)$).

3. **Consistency/ de-noising matching term**:
$$-\sum_{t=1}^{T-1} \mathbb{E}_{q(x_{t-1}, x_{t+1}|x_0)}\big[D_{\mathrm{KL}}(q(x_t \mid x_{t-1}) \| p_\theta(x_t \mid x_{t+1}))\big].$$
Finally, the last term is the objective we strive to learn. It asks, "for all t up to T, if we start from x_0 and add noise for t steps using the forward process q, to what degree can we recover x_{t-1} using our learned reverse model p_θ?" In other words, it ensures that each reverse denoising step matches the corresponding forward noise step.

$$\log p(x_0) \;\geq\; \underbrace{\mathbb{E}_{q(x_1|x_0)}\left[\log p_\theta(x_0 \mid x_1)\right]}_{\text{reconstruction}}$$

$$- \underbrace{\mathbb{E}_{q(x_{T-1}|x_0)}\left[D_{\mathrm{KL}}(q(x_T \mid x_{T-1}) \parallel p(x_T))\right]}_{\text{prior matching}}$$

$$- \sum_{t=1}^{T-1} \underbrace{\mathbb{E}_{q(x_{t-1},x_{t+1}|x_0)}\left[D_{\mathrm{KL}}(q(x_t \mid x_{t-1}) \parallel p_\theta(x_t \mid x_{t+1}))\right]}_{\text{denoising matching term}}$$

We take the denoising matching term as our learning objective.

This is the only non-zero, parameterized term in our ELBO. Minimizing it forces the learned reverse kernels $p_\theta(x_{t-1} \mid x_t)$ to match the true posteriors $q(x_{t-1} \mid x_t, x_0)$ at every timestep.

In fact, observe that $q(x_{t-1} \mid x_t, x_0)$ is Gaussian and can be expressed as:

$$q(x_{t-1} \mid x_t, x_0) \;=\; \mathcal{N}\big(x_{t-1};\, \tilde{\mu}_t(x_t, x_0),\, \tilde{\beta}_t\, I\big),$$

Recall that we defined the noise-scheduling hyperparameters as

$$\begin{aligned}
\beta_t &\;:\; \text{preset variance at step } t,\\
\alpha_t &:= 1 - \beta_t,\\
\bar{\alpha}_t &:= \prod_{s=1}^{t} \alpha_s,\\
\bar{\alpha}_{t-1} &:= \prod_{s=1}^{t-1} \alpha_s,
\end{aligned}$$

and can then write the closed-form posterior variance and mean as

$$\begin{aligned}
\tilde{\beta}_t &:= \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\,\beta_t,\\
x_t &= \sqrt{\bar{\alpha}_t}\,x_0 \;+\; \sqrt{1 - \bar{\alpha}_t}\,\epsilon, \quad \epsilon \sim \mathcal{N}(0, I),\\
\tilde{\mu}_t(x_t, x_0) &:= \frac{1}{\sqrt{\alpha_t}}\Big(x_t \;-\; \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\,\epsilon\Big).
\end{aligned}$$

Parameterizing the Denoising Model

Since both

$$q(x_{t-1} \mid x_t, x_0) \quad \text{and} \quad p_\theta(x_{t-1} \mid x_t)$$

are Gaussian, their KL divergence reduces to a simple squared-difference between means:

$$\mathcal{L}_{t-1} \;=\; D_{KL}\big(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\big) \;=\; \mathbb{E}_q\Big[\frac{1}{2\sigma_t^2}\,\big\|\tilde{\mu}_t(x_t, x_0) \;-\; \mu_\theta(x_t, t)\big\|^2\Big] \;+\; C$$

Here, σ_t^2 is the (shared) variance of both Gaussians (often set to $\tilde{\beta}_t$). The constant C absorbs all terms independent of θ.

Recall from the forward process that we define the noised data at time t by

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

where

$$\alpha_t \;=\; 1-\beta_t, \quad \bar{\alpha}_t \;=\; \prod_{s=1}^{t} \alpha_s.$$

By Bayes' rule on two Gaussians, the true posterior $q(x_{t-1} \mid x_t, x_0)$ is also Gaussian with

$$\tilde{\mu}_t(x_t, x_0) \;=\; \frac{1}{\sqrt{1-\beta_t}}\Big(x_t \;-\; \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon\Big), \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\,\beta_t.$$

Thus, we can seek to predict this $\tilde{\mu}_t$. But instead of doing this directly Ho et al. proposed letting the network predict the noise ϵ instead. That is, training a network that would model

$$\epsilon_\theta(x_t, t) \;\approx\; \epsilon.$$

such that we could then model $\tilde{\mu}_t$ as

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

Finally, we can substitute this μ_θ into the KL to obtain a mean-squared error on the noise:

$$\mathcal{L}_{t-1} \;=\; \mathbb{E}_{x_0 \sim q(x_0),\, \epsilon \sim \mathcal{N}(0,I)}\left[\frac{\beta_t^2}{2\,\sigma_t^2\,(1-\beta_t)\,(1-\bar{\alpha}_t)}\big\|\epsilon \;-\; \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon}_{x_t},\, t)\big\|^2\right] \;+\; C.$$

In practice, the collected constants are often dropped from the training objective:

$$\mathcal{L}_{t-1} \;=\; \mathbb{E}_{x_0 \sim q(x_0),\, \epsilon \sim \mathcal{N}(0,I)}\left[\big\|\epsilon \;-\; \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon}_{x_t},\, t)\big\|^2\right] \;+\; C.$$

This is actually quite interpretable. We train by taking the mean squared error between the learned and true noise at each step!

Architecture

Diffusion models typically use a U-Net backbone with ResNet blocks and self-attention to implement the noise-prediction network $\epsilon_\theta(x_t, t)$. You may recall the U-Net architecture from our lecture on Modern Vision Networks.

U-Net Structure

Encoder (down-sampling path)

- Sequential ResNet blocks reduce spatial resolution
- Optionally interleave self-attention at intermediate resolutions to capture long-range dependencies

Bottleneck

- Lowest resolution features processed by ResNet + attention

Decoder (up-sampling path)

- Mirror of encoder: ResNet blocks + up-sampling (nearest or transpose-convolution)
- Skip-connections from encoder layers inject high-resolution detail. This is a critical features that enables high quality reconstructions.

The Time Factor

Diffusion models have the consideration of time t that needs to be preserved and represented. For this, we borrow sinusoidal positional embeddings from the transformer.

Self-Attention

Within the U-Net, we place self-attention layers at one or more resolutions (often in high level feature maps) to help model global correlations.

The aforementioned architecture is defined in **Ho et al. (NeurIPS 2020)** , who proposed the original DDPM U-Net, and **Dhariwal & Nichol (NeurIPS 2021)**, who made crucial improvements.

Training

We are now equipped to discuss the procedure for training a diffusion model. We'll be using the "one-step" forward process equation:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Recall that $\bar{\alpha}_t$ is a product of noise schedule terms and controls how much of the original image remains vs. how much noise is added and ϵ is sampled from a standard normal distribution (i.e., random Gaussian noise).

Training Algorithm (Repeat Until Convergence)

Step 1: Sample clean image

$$x_0 \sim q(x_0)$$

- Sample a clean image x_0 from the training data distribution q.
- This is the starting point of the forward process.

Step 2: Sample time step

$$t \sim U\{1, 2, \ldots, T\}$$

- Choose a random time step t uniformly from all possible time steps 1 to T.
- This ensures the model learns to denoise at every possible stage of the forward process, from slightly noisy to very noisy.

Step 3: Sample Gaussian noise

$$\epsilon \sim \mathcal{N}(0,1)$$

- Sample Gaussian noise to be added to the image.
- This noise is used in the forward process to get x_t using the "one-step" formula and later used to compute the loss by comparing it to the model's prediction.

Step 4: Optimizer step on loss As we previously derived, our objective is

$$\mathcal{L}(\theta) = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(x_t,t)\|^2\right]$$

- The model $\epsilon_\theta(x_t,t)$ is trained to predict the noise ϵ that was added to get from x_0 to x_t.
- This is a mean squared error (MSE) loss between the true noise and the predicted noise.
- The optimizer (e.g., Adam) updates the model parameters θ accordingly.

Inference Sampling

During inference time, we seek to generate new data using our trained diffusion model. We start with pure noise and iteratively denoise it using our ϵ_θ model to recover a coherent image.

Initialization

We begin with a sample of pure Gaussian noise:

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

- This is the most corrupted/noised version of the image — the starting point for sampling.
- Observe that we start at final time step T since inference is done over the reverse process, attempting to recover some x_0.

Iterative Denoising Process

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

1. Noise Sampling:

$$z \sim \mathcal{N}(0,I) \quad \text{if } t > 1, \quad \text{else } z = 0$$

Interestingly, at every single time step we apply a fresh noise sample. This may seem odd, but the reason we do this is to make the generated outputs diverse. If we didn't do this, the generations would follow similar trajectories.

2. Reverse Step:

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

- $\epsilon_\theta(x_t, t)$: predicted noise component in x_t.
- α_t: forward noise schedule coefficient at timestep t, which is pre-determined.
- $\bar{\alpha}_t$: cumulative product of α_s up to timestep t.
- σ_t: predefined variance schedule controlling how much randomness is added at each step.

Here, the first term is simply subtracting off the predicted noise from the current sample, while the second term is introducing a small amount of stochasticity by adding a little noise back after denoising.

3. Output At timestep $t = 1$, we return x_0.

Comparing Generative Models

While diffusion models have excellent mode coverage and very high quality samples, they are quite slow during execution since we have several steps of denoising, each of which is a full pass through the learned network.

On the other hand, GANs have quick sampling and high-quality outputs, but suffer from mode collapse.

And finally, VAEs share quick sampling with GANs and mode coverage with Diffusion models, but struggle with low-quality outputs.

All three of the generative models we learned make tradeoffs. But as of now, research efforts have centered around diffusion models because the two qualities that they promise are very desirable, and we are willing to deal with slow execution.

Alternate Perspective on Diffusion Models: Score-Based Models

Background

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:

$$s_\theta(x) = \nabla_x \log p_\theta(x) = -\nabla_x f_\theta(x) - \nabla_x \log Z_\theta = -\nabla_x f_\theta(x)$$

Therefore,

$$s_\theta(x) = -\nabla_x f_\theta(x)$$

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

$$s_\theta(x) = \nabla_x \log p(x)$$

- This points in the direction where the probability density increases the fastest.

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:

$$x_{t+1} \leftarrow x_i + \epsilon \nabla_x \log p(x) + \sqrt{2\epsilon}\, z_i, \quad i \in \{0, 1, 2, \ldots, K\}$$

In this equation, each 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 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:

$$\nabla_x \log p(x)$$

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

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

However, the true score $\nabla_x \log p(x)$ 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:

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

This objective tries to learn a score function s_θ that approximates the score of the noisy distribution $p_t(x_t)$.

Using denoising score matching, we can replace the unknown score with an expression involving the known conditional $q_t(x_t \mid x)$.

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

If

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

then

$$q_t(x_t \mid x)$$

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

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

Substituting this into the loss gives:

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

which is rewritten as:

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

Connecting Denoising Score Matching to DDPMs

We know that:

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

Assuming the forward noise process:

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

We substitute into the gradient:

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

This shows that the score function is:

$$s_\theta(x_t, t) = -\frac{\epsilon_\theta(x_t, t)}{\sigma_t}$$

Reformulating the Loss

Starting from the denoising score matching loss:

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

Substitute the score identity:

$$s_\theta(x_t,t)=-\frac{\epsilon_\theta(x_t,t)}{\sigma_t},\qquad \frac{x-x_t}{\sigma_t^2}=-\frac{\epsilon}{\sigma_t}$$

The loss becomes:

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

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

An incredibly useful feature of diffusion models is *control*.

Consider the following scenario: We want to generate images conditioned on a label (e.g., "a cat"), but we may not have (image, label) pairs to train a conditional diffusion model.

Instead, we *do* have access to:

- A pretrained unconditional diffusion model $p_t(x_t)$
- A classifier $p(y \mid x_t)$ that tells us how likely the label y is given a noisy image x_t. This is not *too* hard to obtain, but there may be some instabilities.

We want to sample from $p_t(x_t \mid y)$, the distribution of images conditioned on label y. We can't sample from this directly, but we can guide the sampling using the gradient of the log-probability (a score function):

$$\nabla_{x_t}\log p_t(x_t \mid y)$$

Using Bayes' rule, we can decompose this into:

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

Where:

- $\nabla_{x_t}\log p_t(x_t)$: score function from the unconditional diffusion model.
- $\nabla_{x_t}\log p_t(y \mid x_t)$: gradient from the classifier — tells you how to tweak x_t to make it more likely to be classified as label y.

This trick allows us to guide the diffusion process toward samples that are consistent with a given label y, even without retraining the diffusion model.

Classifier-Free Guidance

Traditional classifier guidance requires a separately trained classifier to guide the sampling process toward a desired condition (e.g., a class label or text prompt). However, training such a classifier on noisy data is non-trivial and can introduce instability. Classifier-free guidance eliminates this need by modifying the diffusion model itself to enable both conditional and unconditional behaviors.

Training Procedure

- A single neural network ϵ_θ is trained to predict noise from a noisy input x_t at timestep t.
- The model is trained on both:
  - **Conditional examples**: where conditioning information y (e.g., a class label or text embedding) is provided.
  - **Unconditional examples**: where the conditioning information is dropped (e.g., replaced with a null embedding).
- During training, the conditioning input is randomly omitted with a fixed probability (e.g., 10%–20%) to ensure the model learns to perform both tasks.
- Conditioning can be incorporated via input concatenation or cross-attention mechanisms.

Modified Sampling Distribution

At inference time, the model performs conditional generation by modifying the target distribution as follows:

$$\log \tilde{p}_t(x_t \mid y) \propto \log p_t(x_t \mid y) + w \log p_t(y \mid x_t)$$

This modification results in an adjusted sampling gradient:

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

Simplifying:

$$= (1-w)\nabla_{x_t} \log p_t(x_t) + w\nabla_{x_t} \log p_t(x_t \mid y)$$

This formulation interpolates between the unconditional and conditional scores, with $w \geq 0$ serving as the guidance scale.

With classifier-free guidance, we can simplify the model architecture and training pipeline, and avoid the gradient instability and additional computation overhead of classifier guidance. This method is widely adopted in virtually all text-to-image generators.

Latent Diffusion Models (LDMs)

Latent Diffusion Models (LDMs) are a class of generative models that apply diffusion processes in a compressed latent space, rather than directly on high-dimensional data such as images. This yields substantial computational benefits while preserving generative quality.

Traditional diffusion models operate directly on pixel-space data (e.g., 256x256 RGB images), which leads to high memory and compute requirements due to the dimensionality. Latent Diffusion Models mitigate this by:

- Encoding images into a lower-dimensional latent space using an autoencoder

- Applying diffusion in that latent space
- Decoding the denoised latent back into image space

Architecture

LDMs consist of three main components:

1. Autoencoder (Encoder–Decoder Pair)

- Encoder: Compresses an input image x into a latent representation $z = \mathcal{E}(x)$.
- Decoder: Reconstructs the image $\tilde{x} = \mathcal{D}(z)$
- The autoencoder is trained with a perceptual and pixel-wise reconstruction loss, ensuring high-quality reconstructions.

2. Latent Diffusion Model

- A standard diffusion model (e.g., U-Net architecture) is applied in the latent space to learn the distribution $p(z)$.
- Noise is added to the latent vector over T steps and the model learns to reverse this process.

3. Conditioning Mechanism (for Text-to-Image Tasks)

- Conditioning (e.g., on text prompts) is integrated using cross-attention layers between the latent diffusion U-Net and a text encoder (e.g., CLIP or OpenCLIP)
- This enables controllable generation from text prompts

4. Discriminator

- In some variants, a discriminator is used during training for adversarial regularization to improve the sharpness and realism of reconstructions.

Stable Diffusion

Stable Diffusion (by Stability AI, LMU Munich, and Runway) is a prominent implementation of latent diffusion, designed for text-to-image synthesis. Key design details include:

- **Training Dataset**: LAION-5B (large-scale dataset of image-text pairs scraped from the web)
- **Text Encoder**: CLIP ViT-L/14, trained to embed text prompts into conditioning vectors
- **Autoencoder**:
  - Uses a convolutional VAE-like structure
  - Latents are typically 1/8 or 1/16 the spatial resolution of the input image
- **Latent U-Net**: A UNet-based denoising model trained in latent space
- **Classifier-Free Guidance**: Used during inference to interpolate between conditional and unconditional predictions for stronger text-image alignment

Summary

Diffusion models have emerged as a powerful class of generative models, capable of producing high-quality samples across a variety of domains. These models were introduced from two complementary perspectives: the variational perspective, which frames diffusion as a latent variable model trained using variational inference, and the score-based generative modeling perspective, which focuses on estimating the gradient of the data distribution.

A key strength of diffusion models is their flexibility in conditional generation. They can be conditioned on various modalities, such as text or images, to produce outputs that align with a desired specification—enabling applications like text-to-image synthesis, image inpainting, or super-resolution.

Modern implementations, such as Stable Diffusion, further enhance efficiency by performing the generative process in the latent space of a pretrained autoencoder. This strategy significantly reduces computational cost while maintaining high fidelity, making diffusion models practical for large-scale deployment.

Bonus: Discrete Diffusion

LLaDA (Large Language Diffusion with mAsking) is a diffusion model for language generation, introduced by Nie et al. (2025).

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

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

Since each token is masked independently, we have that

$$q_{t|0}q(x_t \mid x_0) = \prod_{i=1}^{L} q_{t|0}(x_t^i \mid x_0^i)$$

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).

$$\mathbb{E}_{p_{\text{data}}(x_0)}[\log p_\theta(x_0)] \le -\mathbb{E}_{t,x_0,x_t}\left[\frac{1}{t}\sum_{i=1}^{L} \mathbb{1}\{x_t^i = \mathrm{M}\} \log p_\theta(x_0^i \mid x_t)\right]$$

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 x_t, we want to generate 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 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

References

Cornell University. (2025). *Diffusion Models II* [Lecture slides]. CS 4782: Deep Learning. https://www.cs.cornell.edu/courses/cs4782/2025sp/slides/pdf/week9_2_slides.pdf

Krishna, R. (2023). *Lecture 15: Vision and Language* [Lecture slides]. CS231n: Deep Learning for Computer Vision. Stanford University. https://cs231n.stanford.edu/slides/2023/lecture_15.pdf

Luo, C. (2022). *Understanding diffusion models: A unified perspective* (arXiv:2208.11970). arXiv. https://arxiv.org/pdf/2208.11970