EE367

Diffusion Project

Winter 2026

Sriram Ramanathan
3-13-2026

Contents

Introduction 2

Denoising with DDPM 2

Unconditional Generation with DDPM 4

SDEdit with DDPM 6

Score Annealed Langevin Dynamics 7

Diffusion Posterior Sampling 9

Derivations 10

Implementation Errors & Limitations 13

Introduction

Diffusion models are a class of generative models widely used in modern image processing and synthesis. They work by learning a two-stage process:

1. **Forward diffusion**, where random noise is gradually added to an image until it becomes nearly pure noise, and
2. **Reverse diffusion**, where a neural network is trained to iteratively remove that noise step by step, reconstructing a clean image from noise.

Denoising Diffusion Probabilistic Model [Jonathan Ho, Ajay Jain, Pieter Abbeel, 2020], DDPM in short, are class of generative models that achieve high quality image synthesis by modeling data generation as a gradual denoising process.

The core idea of DDPM is to define a forward diffusion process that incrementally adds Gaussian noise to a data sample over many time steps until it becomes pure noise. A neural network is then trained to learn the revers process by removing noise step by step to recover a clean image from noise. Key aspects of this model are –

- Forward process is fixed and not learned
- The reverse process is modeled as a parameterized Gaussian distribution
- Instead of predicting the clean (original) image, the model is trained to predict the noise added at each step which simplifies training
- The loss function is simple mean squared error loss

The drawback is the iterative/slow nature of image generation which requires 1000s of sequential denoising steps. However, a single shot denoising formulations also exist.

This project explores different applications and variations of DDPM.

Denoising with DDPM

Using a pretrained diffusion model, denoising of an image with different noise levels was attempted. Key elements of this solution are discussed below

Forward Process

An image $x_0$is gradually corrupted with Gaussian noise over Tsteps:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\mathbf{z}$$

where $\bar{\alpha}_t$is a fixed noise schedule.

Reverse Process
Given noisy image, the reverse process estimates clean image using score based single shot denoising provided by the relation

$$\hat{\boldsymbol{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t + (1-\bar{\alpha}_t)\boldsymbol{s}_\theta(x_t, t))$$

Evaluation

The denoising reverse process was invoked for different noise levels – as defined by the number of time steps in the forward and reverse process. The resulting denoised images were verified both visually and objectively using PSNR and LPIPS metrics.

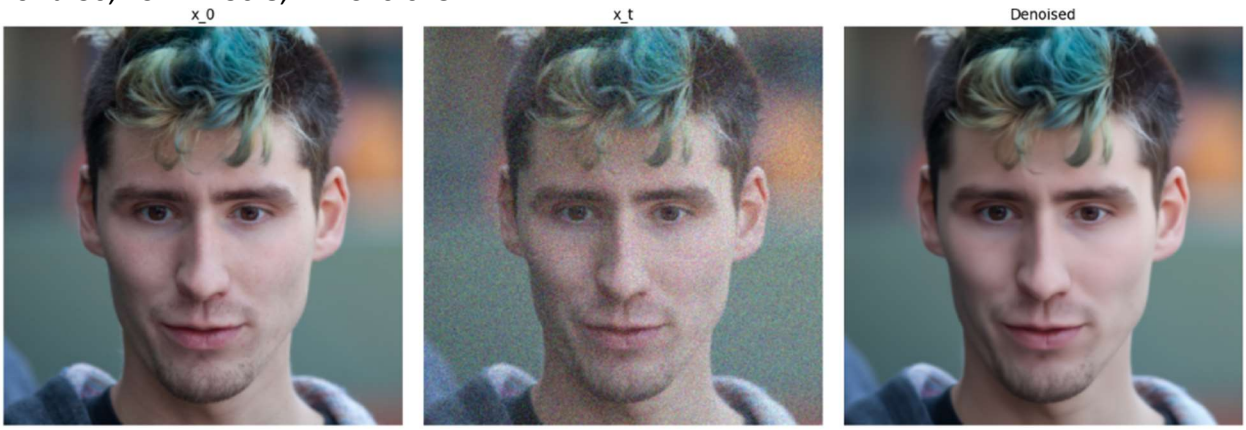
While PSNR measures how similar two images are at pixel level, Learned Perceptual Image Patch Similarity (LPIPS) is a perceptual metric that measures how similar two images look to a human.

Observation

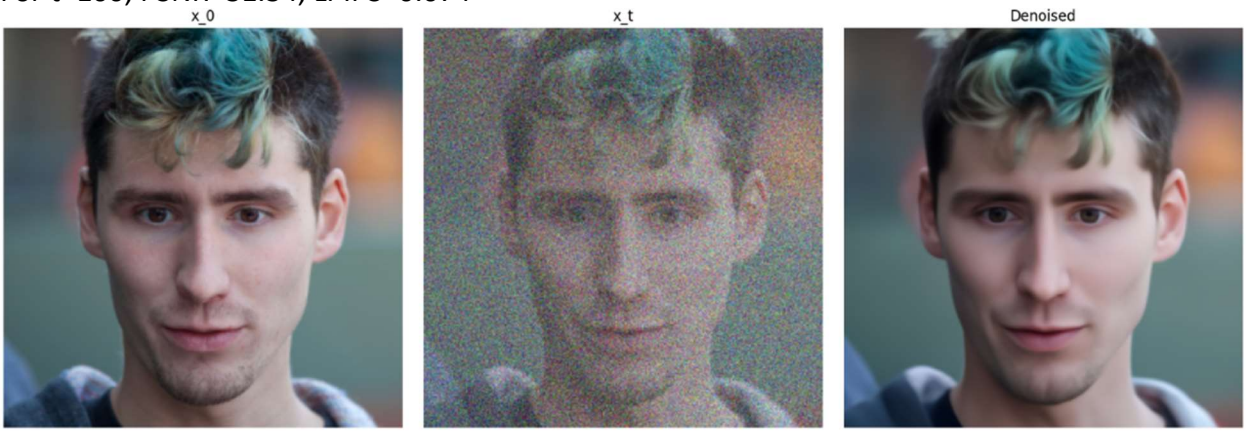
For lower noise levels, the solution is able to denoise and produce a clean image that appears similar to the original ground truth image. However, at higher noise levels, the though the reverse process produces images that appear clean and denoised, the features in the image are blurred.

Results

For t=30, PSNR = 36.8, LPIPS=0.025

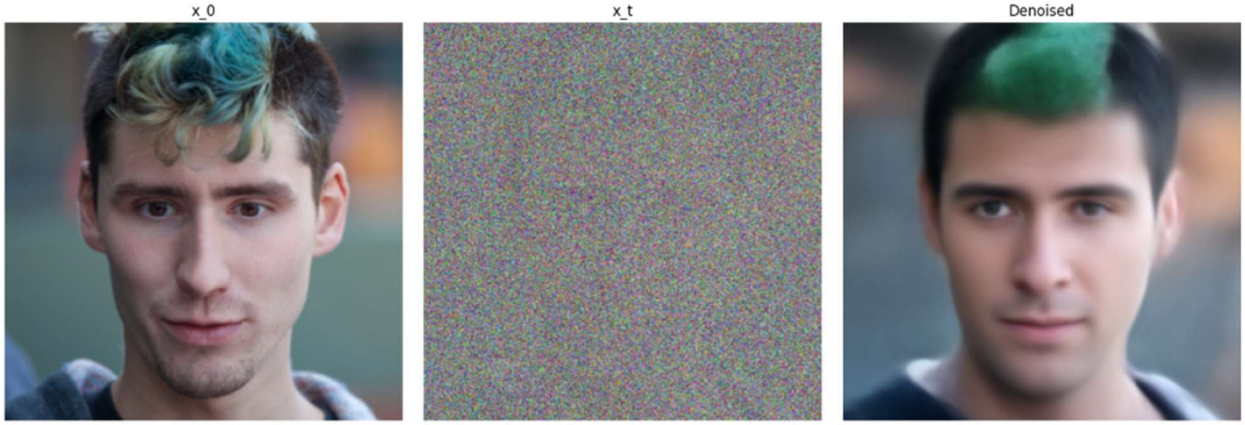


For t=100, PSNR=31.34, LPIPS=0.074



For t=300, PSNR=25.71, LPIPS=0.17

For t=500, PSNR=21.68, LPIPS=0.3



Unconditional Generation with DDPM

Using pre-trained DDPM, an image is generated from pure noise unconditionally. In addition to the single step denoising as discussed in the above section, the posterior mean i.e., x_{t-1} is calculated.

Reverse Process / Single step denoising

Given noisy image, the reverse process estimates clean image using score based single shot denoising provided by the relation

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t + (1 - \bar{\alpha}_t)s_\theta(x_t, t))$$

Posterior Mean Calculation

Given the noisy image at time t, and the noisy image at time 0, this computes the previous noisy image x_{t-1} using the relation

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t + (1-\alpha_t)s_\theta(x_t, t)) + \sqrt{1-\alpha_t}z$$

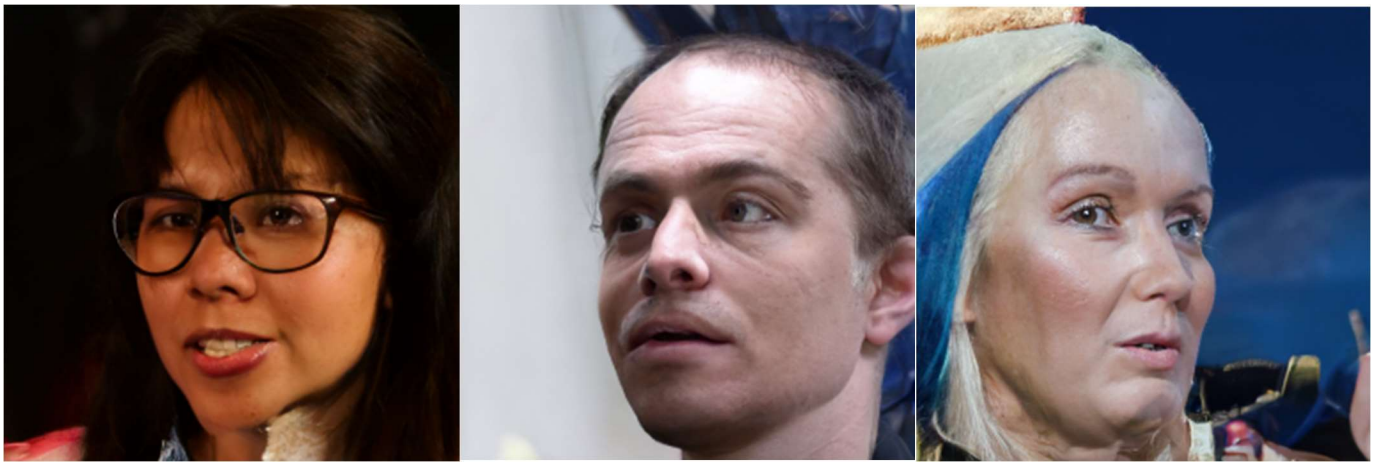
Sampling Loop

Starting with pure noise, the loop iteratively denoises the image through 1000 time steps to arrive at the x0. The algorithm flow is as below –

$$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\textbf{for } t = T, \ldots, 1 \textbf{ do}$$
$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ if } t > 1, \text{ else } \mathbf{z} = \mathbf{0}$$
$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t + (1-\bar{\alpha}_t)s_\theta(x_t, t))$$
$$x_{t-1} = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1-\alpha_t)}{1-\bar{\alpha}_t}\hat{x}_0 + \sqrt{1-\alpha_t}z$$
$$\textbf{end for}$$
$$\textbf{return } \mathbf{x}_0$$

Results

The sampling loop was run thrice to produce the below results.



SDEdit with DDPM

Instead of unconditional generation with pure noise, SDEdit starts from a partially noised version of an input image and does a controlled edit to result in a clear image that preserves key features from the noised version.

This builds on unconditional generation key changes outlined below –

1. Instead of noise, a clean image is chosen. This serves as the ground truth
2. The image is sent through a forward process that adds a box or gaussian noise
3. Further random gaussian noise is added to this image forming the *measurement*
4. The sampling loop for denoising consists of two stages
    a. Subject measurement through the t/2 of the DDPM forward process
    b. Subject the noised *measurement* through t/2 of the DDPM backward process

Unlike unconditional generation, since the denoising starts with a noised version of a known image, the denoising process produces an image that is broadly *like* the ground truth.

Evaluation

SDEdit was performed with ground truth image on which box inpainting or random noise kernel was applied. The PSNR and LPIPS metrics were calculated.

Results

Though the results appear noise free and the generated image preserve features of the ground truth, it is not faithful to the original face.

For Inpainting (with box mask applied), PSNR=20.02, LPIPS=0.18

For Deconvolution with random noisy kernel applied, PSNR=20.25, LPIPS=0.19

Score Annealed Langevin Dynamics

The solution discussed above do not account for image formation model and are not truly inverse problems. To use DDPM to solve inverse problems, two approaches were considered.

A **linear inverse problem** has the form

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}$$

where

- $\mathbf{x} \in \mathbb{R}^d$: unknown signal or image
- $\mathbf{A}$: known forward operator (blur, subsampling, CT, MRI, etc.)
- $\mathbf{y}$: observed measurements
- $\mathbf{n} \sim \mathcal{N}(0, \sigma_n^2 I)$: noise

These problems are typically **ill-posed**:

- $\mathbf{A}$may be non-invertible
- multiple $\mathbf{x}$explain the same $\mathbf{y}$

A Bayesian formulation of this can be of the form

Likelihood (Gaussian noise)

$$\log\, p(\mathbf{y} \mid \mathbf{x}) = -\frac{1}{2\sigma_n^2} \parallel \mathbf{y} - \mathbf{A}\mathbf{x} \parallel^2 + \text{const}$$

Assuming there is access to a learned prior $p(\mathbf{x})$, not explicitly, but via its **score function**

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

Combining the two, in order to sample from:

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

Using Bayes:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid \mathbf{y}) = \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x})}_{\text{prior score}} + \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{y} \mid \mathbf{x})}_{\text{data term}}$$

Score ALD approximates the data term as

$$\underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{y} \mid \mathbf{x})}_{\text{data term}} = \underbrace{\nabla_{\mathbf{x}} \log pt(\mathbf{y} \mid \mathbf{xt})}_{\text{data term}}$$

Building on SDEdit formulation, ScoreALD updates x_{t-1} with gradient of likelihood with respect to x_t as below

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

for $t = T, \ldots, 1$ **do**

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

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t + (1 - \bar{\alpha}_t) s_\theta(x_t, t))$$

$$x_{t-1} = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)}{1 - \bar{\alpha}_t} \hat{x}_0 + \sqrt{1 - \alpha_t} z$$

$$x_{t-1} = x_{t-1} - \frac{1}{2(\sigma^2 + \gamma_t^2)} \nabla_{x_t} \left\| \mathcal{A}(x_t) - y \right\|^2$$

end for

return $\mathbf{x}_0$

Result

Diffusion Posterior Sampling

Diffusion Prior Sampling uses a pretrained diffusion model as a learned prior to solve inverse problems (e.g., denoising, super-resolution, inpainting). Instead of explicitly modeling a prior $p(x)$, diffusion models provide the score (gradient of log-density) of natural data. During reconstruction, sampling is guided by both:

- the diffusion prior
- the measurement likelihood

The xt-1 update rule differs slightly from Score ALD in that it depends on $\hat{x}_0$ and not x_t. The algorithm flow is as below -

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

for $t = T, \ldots, 1$ **do**

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

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t + (1 - \bar{\alpha}_t)s_\theta(x_t, t))$$

$$x'_{t-1} = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)}{1 - \bar{\alpha}_t}\hat{x}_0 + \sqrt{1 - \alpha_t}z$$

$$x_{t-1} = x'_{t-1} - \zeta_t \nabla_{x_t} \left\| \mathcal{A}\left(\hat{x}_0\right) - y \right\|^2$$

end for

return $\mathbf{x}_0$

Result

Unavailable. Code was running at 38% at the time of submission.

Derivations

Given

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, z_{t-1} \quad -(1)$$

and, $\alpha_t = 1 - \beta_t \quad -(2)$

Expand x_{t-1} and z_{t-1} in (1)

$$x_t = \sqrt{1-\beta_t}\left[\sqrt{1-\beta_{t-1}}\, x_{t-2} + \sqrt{\beta_{t-1}}\, z_{t-2}\right] + \sqrt{\beta_t}\left[\sqrt{\beta_{t-1}}\, z_{t-2}\right]$$

Rewrite with αs

$$x_t = \sqrt{\alpha_t}\left[\sqrt{\alpha_{t-1}}\, x_{t-2}\right] + \sqrt{\alpha_t}\sqrt{1-\alpha_{t-1}}\, z_{t-2} + \sqrt{1-\alpha_t}\sqrt{1-\alpha_{t-1}}\, z_{t-2}$$

$$= \sqrt{\alpha_t}\sqrt{\alpha_{t-1}}\, x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\, z_{t-2}$$

Repeated substitution of x_i and z_i with x_{i-1} and z_{i-1} gives

$$x_t = \sqrt{\alpha_t}\sqrt{\alpha_{t-1}} \cdots \sqrt{\alpha_0}\, x_0 + \sqrt{1 - \alpha_t \alpha_{t-1} \alpha_{t-2} \cdots \alpha_0}\; z$$

$$\boxed{x_t = \bar{\alpha}_t x_0 + \sqrt{1 - \bar{\alpha}_t}\; z.}$$

(2) Given

$$\hat{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t + (1 - \bar{\alpha}_t) S_\theta(x_t, t) \right) \quad - (1)$$

$$x_{t-1} = \frac{\sqrt{\alpha_t}\,(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\,(1 - \alpha_t)}{1 - \bar{\alpha}_t} \hat{x}_0 \quad - (2)$$

Substitute $\hat{x}_0$ from (1) in (2)

$$\frac{\sqrt{\bar{\alpha}_{t-1}}\,(1 - \alpha_t)}{(1 - \bar{\alpha}_t)} \times \frac{1}{\sqrt{\bar{\alpha}_t}} \left[x_t + (1 - \bar{\alpha}_t) S_\theta \right]$$

Since $\bar{\alpha}_t = \alpha_t \bar{\alpha}_{t-1}$ the term becomes

$$\frac{1-\alpha_t}{1-\bar{\alpha}_t} \times \frac{\sqrt{\alpha_{t-1}}}{\sqrt{\alpha_t}\sqrt{\alpha_{t-1}}} \left(x_t + (1-\bar{\alpha}_t) S_\theta(x,t)\right)$$

$$\frac{1}{\sqrt{\alpha_t}} \cdot \frac{1-\alpha_t}{1-\bar{\alpha}_t} \left(x_t + (1-\alpha_t) S_\theta(x,t)\right)$$

Equation (2) becomes

$$x_{t-1} = \left[\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} + \frac{1-\alpha_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\alpha_t}}\right] x_t$$

$$+ \frac{1}{\sqrt{\alpha_t}} \cdot \frac{1-\alpha_t}{1-\bar{\alpha}_t} \cdot (1-\alpha_t) S_\theta(x,t)$$

x_t term

$$= \frac{\alpha_t - \bar{\alpha}_{t-1}\sqrt{\alpha_t} + 1 - \alpha_t}{1-\bar{\alpha}_t} x_t$$

Multiply numerator & denominator by $\sqrt{\alpha_t}$

$$\frac{1 - \alpha_t \cdot \bar{\alpha}_{t-1}}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)} x_t = \frac{1-\bar{\alpha}_t}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)} x_t$$

$$= \frac{1}{\sqrt{\alpha_t}} x_t$$

Simplifying Score term

$$\frac{1}{\sqrt{\alpha_t}} \cdot \frac{1-\alpha_t}{1-\bar{\alpha}_t} (1-\bar{\alpha}_t) s_\theta$$

$$= \frac{1}{\sqrt{\alpha_t}} (1-\alpha_t) s_\theta$$

Combining both

$$\boxed{x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t + (1-\alpha_t) s_\theta(x,t) \right)}$$

Implementation Errors & Limitations

The implementation for unconditional generation was originally done and tested on Intel CPU. With no apparent error in the implementation, it produced pure noisy image as a result. Since the runtime on CPU was 70+ minutes, further troubleshooting was not attempted and the code moved to Google Colab GPU. Unmodified code ran successfully on GPU and produced the desired results.

SDEdit, ScoreALD and DPS were developed on Google Colab. However due to idling, the allocated time on GPU ran out during ScoreALD development and the code was switched back to CPU. There is likely some part of the code which depends on GPU, causing relatively poor results for ScoreALD and DPS.