Denoising Diffusion Probabilistic Models

An extraordinary paper published by Jonathan Ho, Ajay Jain, Pieter Abbeel

Abstract

Denoising Diffusion Probabilistic Models (DDPMs) are a class of generative models that have recently gained prominence for their ability to produce high-quality samples across diverse domains such as images, audio, and molecular structures. This novel approach revolves around the concept of gradually adding noise to data until a simple, noise-dominant distribution is reached and then learning to reverse this process. The model effectively learns a sequence of denoising steps that incrementally reconstruct the data from the noise. Unlike traditional generative models like Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs), DDPMs operate through an iterative Markov chain techniques. This framework not only provides a robust theoretical grounding but also allows DDPMs to generate samples by converting noise into complex patterns through a series of learned transformations. The inherent stability of this process, absence of mode collapse, and the ability to produce diverse high-fidelity outputs have positioned DDPMs as a significant advancement in the field of generative modeling. DDPM goes above and beyond image synthesis, showing promising results in super-resolution, painting, and conditional generation tasks, marking a crucial step forward in the development of generative models that can learn from complex, high-dimensional data distributions.

Introduction

Deep generative models have shown remarkable success in generating high-quality samples across various data modalities. Prominent types such as Generative Adversarial Networks (GANs), autoregressive models, flows, and Variational Autoencoders (VAEs) have produced compelling image and audio samples.

Why do we need Generative Model? The primary function is to understand and capture the underlying patterns or distributions from a given set of data. Once these patterns are learned, the model can then generate new data that shares similar characteristics with the original dataset.

Why do we need Diffusion based Generative Models? Compared to traditional generative models, diffusion models have - ease of training with simple and efficient loss functions, generate highly realistic images, interpretable latent space, and robustness to overfitting.

A diffusion model is a parameterized Markov chain, trained using variational inference, designed to produce data-matching samples after a finite number of transitions. These transitions are specifically learned to reverse a diffusion process—a Markov chain that incrementally introduces noise into data until the original signal is obliterated. When this diffusion involves minor amounts of Gaussian noise, the sampling transitions can also be modeled as conditional Gaussians, facilitating a straightforward neural network parameterization[6][1].

DDPMs are responsible for making diffusion models practical. In this report, I will highlight the key concepts and techniques behind DDPMs and train DDPMs on some datasets for unconditional image generation. I will go over both theory (+ some math), the underlying code. I will also highlight conditional DDPM briefly.

Background

Diffusion models are composed of two opposite processes i.e., Forward & Reverse Diffusion Process. Figure 1 is a high level conceptual overview of the entire image space[10].

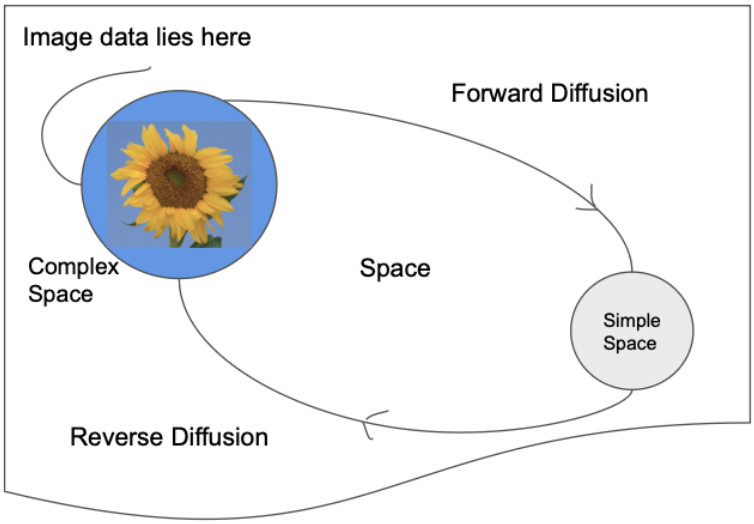

Figure 1: Conceptual Reference of Entire Image Space

Forward Diffusion Process:
In the Forward Diffusion process -

- We slowly and iteratively add noise to (corrupt) the images in our training set.
- What we are doing here is converting the unknown and complex distribution that our training set belongs to into one that is easy for us to sample
- At the end of the forward process, the images become entirely unrecognizable

Reverse Diffusion Process:
In the Reverse Diffusion process, the idea is to reverse the forward diffusion process.

- We gradually undo the changes made to images during the first part of the process.
- Our aim is to find a way back to the original subspace.
- In diffusion probabilistic models, this is done by referring to the small iterative steps taken during the forward diffusion process.

- The PDF that satisfies the corrupted images in the forward process differs slightly at each step.
- In the reversal phase, we use a deep learning model at each step to predict the parameters of these changes.
- After the model is trained, we can start from any point in the simple space and use the model to iteratively take steps to lead us back to the data subspace.

In reverse diffusion, we iteratively perform the denoising in small steps, starting from a noisy image. This approach for training and generating new samples is much more stable than GANs and better than previous approaches like variational autoencoders (VAE) and normalizing flows.

Mathematical Analysis behind DDPM

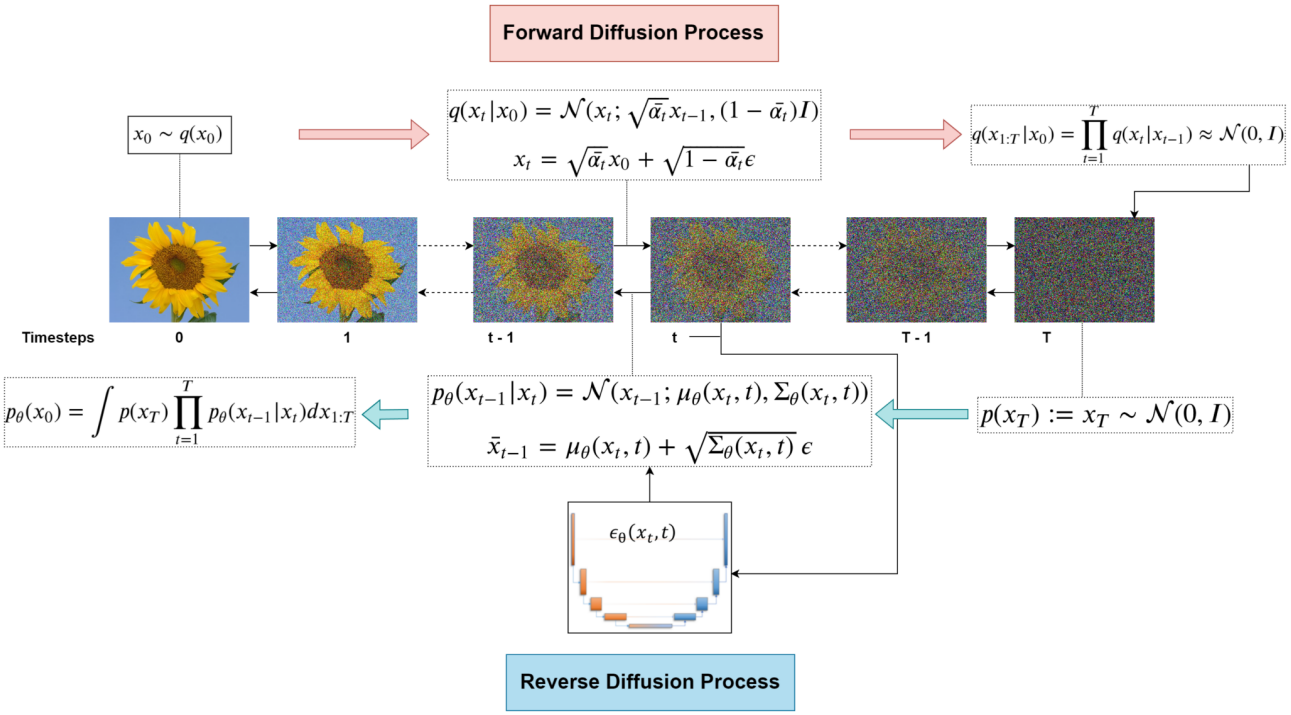

Figure 2: All forward and backward equations

Figure 2 shows all the equations regarding DDPM.

Forward Diffusion in short:
The distribution q in the forward diffusion process is defined as Markov Chain given by:

$$q(x_1, \ldots, x_T \mid x_0) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}) \tag{1}$$

$$q(x_t \mid x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{2}$$

At each timestep t, the parameters that define the distribution of image x_t are set as:

- Mean: $\sqrt{1-\beta_t}x_{t-1}$
- Covariance: $\beta_t I$

The term β is known as the "diffusion rate" and is precalculated using a "variance scheduler". Using the reparameterization trick and some more calculation regrading gaussian addition[13], we can conclude the forward process as:

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_0, (1-\alpha_t)I) \tag{3}$$

In practice, the authors of DDPMs use a "linear variance scheduler" and define β in the range [0.0001, 0.02] and set the total timesteps T = 1000.

Reverse Diffusion in short:
We should end up with a pure noise distribution by the end of the forward process, given we set the variance schedule appropriately i.e., the distribution we will end up with will be $\mathcal{N}(x_T; 0, I)$. For the reverse process, we will start with noise, and will try to undo the noise at each timestep to obtain back the original image. We can write this process as:

$$p_\theta(x_0 : x_T) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t) = p(x_T) \prod_{t=1}^{T} \mathcal{N}(x_{t-1}; \mu_0(x_t, t), \Sigma_0(x_t, t)) \tag{4}$$

where the parameters of the multivariate Gaussian are time-dependent and are to be learned. A few things to note about the equations:

- $p_\theta(x_0 : x_T)$ is the reverse distribution and $p(x_{t-1} \mid x_t)$ is known as the reverse diffusion kernel. $\mu_0(x_t, t), \Sigma_0(x_t, t)$ are the learnable parameters of the reverse distribution.
- $p(x_T)$ is nothing but $q(x_T)$ i.e., the point where the forward process ends is the starting point of the reverse process. There can be n number of pathways to arrive at a sample $p(x_0)$ starting from a noise sample. To obtain $p_\theta(x_0)$, we would then be required to integrate over all the possible pathways i.e., $p_\theta(x_0) = \int p_\theta(x_0 : x_T) dx_{1:T}$. Calculating density like in the above step is intractable. A neural network is sufficient to predict the mean μ_0 and the diagonal covariance matrix Σ_0 for the reverse process.

The Loss:
The authors take inspiration from VAEs and reformulate the training objective using a variational lower bound (VLB), also known as "Evidence lower bound" (ELBO).

The variational lower bound (VLB) loss is defined as:

$$\text{VLB loss} = E[-\log p_\theta(x_0)] \leq E_q[L_T + \sum_{t>1} D_{KL}(q(x_{t-1} \mid x_t, x_0) \parallel p_0(x_t \mid x_{t-1})) + L_0] \tag{5}$$

The denoising score matching (DSM) loss is expressed as:

$$\text{DSM loss} = \text{constant} \times \left\| \epsilon - \epsilon_\theta \left(\sqrt{\hat{\alpha}_t} x_0 + \sqrt{1 - \hat{\alpha}_t} \epsilon, t \right) \right\|^2 \tag{6}$$

Finally, the simple loss L_{simple} is given by:

$$L_{\text{simple}} = E_{t,x_0,\epsilon} \left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2 \right] \tag{7}$$

This is the final loss function we use to train DDPMs, which is just a "Mean Squared Error" between the noise added in the forward process and the noise predicted by the model. This is the most impactful contribution of the paper Denoising Diffusion Probabilistic Models.

Experimental Results

Experiment 1

I have run DDPM model inspired from this PyTorch implementation[12]. Due to memory limit, I reduced the step size from 700000 to 7000. Figure 3 is one of my samples from ./results folder after training step, input was from bedroom samples of the paper using **T4 GPU**.

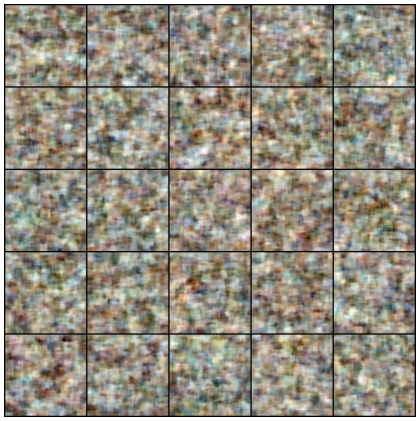

Figure 3: Sample after training using 7000 training steps

Model Installation:
$ pip install denoising_diffusion_pytorch

Then the following Python code sets up a denoising diffusion model, defines its parameters, and initiates the training process.

```
from denoising_diffusion_pytorch import Unet, GaussianDiffusion, Trainer

model = Unet(
    dim = 64,
    dim_mults = (1, 2, 4, 8),
    flash_attn = True
)

diffusion = GaussianDiffusion(
    model,
    image_size = 128,
    timesteps = 1000,
    sampling_timesteps = 250
)

trainer = Trainer(
    diffusion,
    '/content/drive/MyDrive/ese531/bedroom_samples', //path to my image dataset
    train_batch_size = 32,
    train_lr = 8e-5,
    train_num_steps = 700000,
    gradient_accumulate_every = 2,
    ema_decay = 0.995,
    amp = True,
    calculate_fid = True
)

trainer.train()
```

Experiment 2

For learning purpose, I have implemented forward diffusion process separately from scratch inspired from [7]. It doesn't necessarily need GPU. Figure 4 is my input and 5 is output.

Figure 4: Input used in Forward Process Implementation

Implementation is shared through my Colab Notebook.

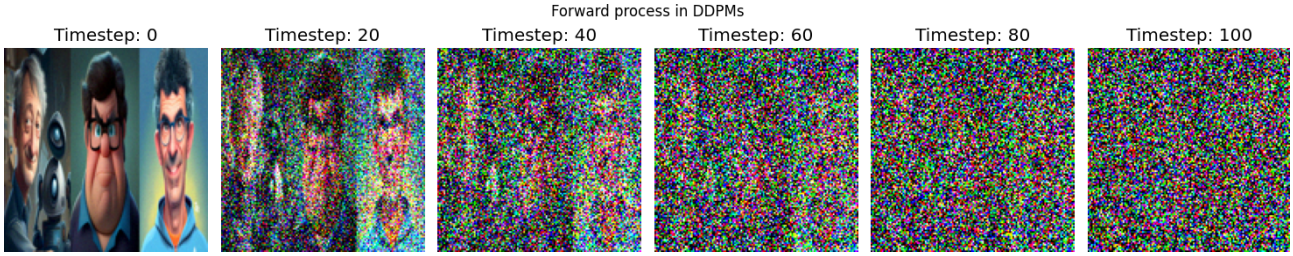

Figure 5: Output

Experiment 3

I have also tried using MNIST dataset inspired from here[4]. Two sample outputs are shared in figure 6. Implementation can be seen through my Colab Notebook. Code snippet is as follows:

```
from labml import experiment
from labml.configs import option
from labml_nn.diffusion.ddpm.experiment import Configs

experiment.create(name="diffuse", writers={'screen'})
configs = Configs()
experiment.configs(configs, {
    'dataset': 'MNIST',
    'image_channels': 1,
    'epochs': 15,
})
configs.init()
experiment.add_pytorch_models({'eps_model': configs.eps_model})

# Start the experiment
with experiment.start():
    configs.run()
```

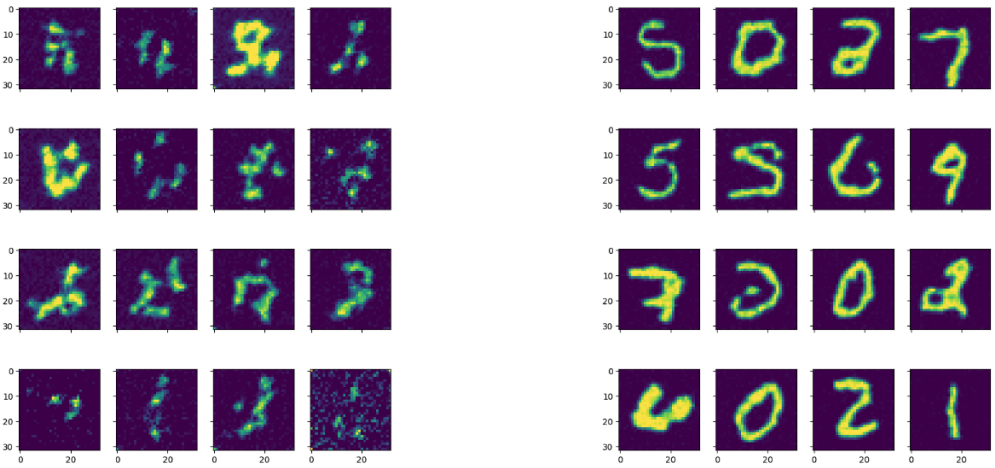

Figure 6: MNIST dataset outputs

Conditional DDPM

A conditional diffusion model is a modification of an unconditional diffusion model. In conditional diffusion models, an additional input, y (eg. a class label or a text sequence) is available and we try to model the conditional distribution $p(x|y)$ instead. This allows us to generate data given the conditioning signal [9].

Conditional Diffusion MNIST is also an implementation of conditional diffusion[11][14]. It is a self-contained implementation. It learns to generate MNIST digits, conditioned on a class label. The neural network architecture is a small U-Net. This looks like figure 7.

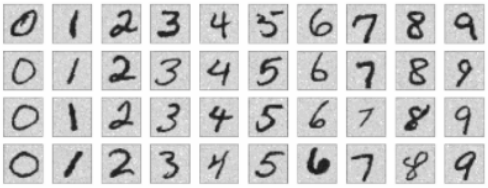

Figure 7: MNIST digit

I have run conditional image generator model from the *Hugging Face*[5][2]. Figure 8 and 9 are my outputs. I have used **L4 GPU** to run it.

First of all, imported the necessary module from the diffusers library and load the model:

```
from diffusers import DiffusionPipeline
generator = DiffusionPipeline.from_pretrained(``CompVis/ldm-text2im-large-256")
generator.to('cuda')
```

Applied text prompts using the generator:

```
image = generator('An image of a cat laughing').images[0]
```

The result can be saved as follows:

```
image.save('image_of_cat_laughing.png')
```

Official AI Art Generation Tools using Diffusion Models

The real wave of AI art generation was started by four organizations.

- DALL-E 2 by OpenAI
- Imagen by Google
- Stable Diffusion by StabilityAI
- Midjourney by an organization with the same name

Figure 8: Output image of a squirrel painting

Figure 9: Output image of a cat laughing

These are all based on DDPM. Except for Imagen, all other models are accessible either through APIs or GitHub repositories.
Figure 10 and 11 are two images generated using DALL-E 2[8]. Figure 12 and 13 are from Imagen[3].

Conclusion

Denoising Diffusion Probabilistic Models (DDPMs) have emerged as a potent class of generative models, enabling the generation of high-fidelity, diverse samples with a methodologically sound process. They provide a stable alternative to GANs, resolving common issues like mode collapse through a controlled and gradual denoising process. Their flexibility in sample generation makes them ideal for applications in areas requiring detailed output, such as art generation and scientific or medical imaging. Moreover, the ability to condition these models enhances their utility. DDPMs represent a promising direction in generative modeling, blending robust mathematical foundations with practical efficacy in image synthesis and beyond.

References

[1] Mickael Boillaud. *Denoising Diffusion Model from scratch using PyTorch*. URL: https://medium.com/@mickael.boillaud/denoising-diffusion-model-from-scratch-using-pytorch-658805d293b4.

[2] Thomas Capelle. *How To Train a Conditional Diffusion Model From Scratch*. URL: https://wandb.ai/capecape/train_sd/reports/How-To-Train-a-Conditional-Diffusion-Model-From-Scratch--VmlldzoyNzIzNTQ1.

Figure 10: Prompt: a dragon standing on top of a mountain, snowy mountains in the background, artistic

Figure 11: Prompt: a dragon standing on top of a mountain, snowy mountains in the background, artistic, oil painting

Figure 12: Prompt: A high contrast portrait of a very happy fuzzy panda, dressed as a chef in a high end kitchen making dough. There is a painting of flowers on the wall behind him

Figure 13: Prompt: A brain riding a rocketship heading towards the moon

[3] William Chan Chitwan Saharia et al. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". In: *arXiv preprint arXiv:2205.11487* (2022).

[4] *Denoising Diffusion Probabilistic Models (DDPM)*. URL: https://colab.research.google.com/github/labmlai/annotated_deep_learning_paper_implementations/blob/master/labml_nn/diffusion/ddpm/experiment.ipynb#scrollTo=AYV_dMVDxyc2.

[5] The Hugging Face. *Conditional Image Generation*. URL: https://huggingface.co/docs/diffusers/v0.3.0/en/using-diffusers/conditional_image_generation.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *arXiv preprint arxiv:2006.11239* (2020).

[7] Aakash Kumar Nain. *Deep Dive into DDPMs*. URL: https://github.com/AakashKumarNain/diffusion_models/blob/main/notebooks/deep_dive_into_ddpms.ipynb.

[8] Sovit Rath. URL: https://learnopencv.com/ai-art-generation-tools/.

[9] Aman Shrivastava. *Text-to-Image Synthesis with Conditional Diffusion Models*. URL: https://www.cs.rice.edu/~vo9/cv-seminar/slides/aman-diffusion.pdf.

[10] Vaibhav Singh. *An In-Depth Guide to Denoising Diffusion Probabilistic Models DDPM – Theory to Implementation*. URL: https://https://learnopencv.com/denoising-diffusion-probabilistic-models/.

[11] TeaPearce. *Conditional Diffusion MNIST*. URL: https://github.com/TeaPearce/Conditional_Diffusion_MNIST.

[12] Phil Wang. *DDPM in pytorch*. URL: https://github.com/lucidrains/denoising-diffusion-pytorch.

[13] Wei Ye. *Understanding the Denoising Diffusion Probabilistic Model (DDPMs), the Socratic Way*. URL: https://towardsdatascience.com/understanding-the-denoising-diffusion-probabilistic-model-the-socratic-way-445c1bdc5756.

[14] Jun Yu Zijian Zhang Zhou Zhao and Qi Tian. "ShiftDDPMs: Exploring Conditional Diffusion Models by Shifting Diffusion Trajectories". In: *arXiv preprint arXiv:2302.02373* (2023).