Diffusion Models for Machine Learning
CS 584 Final Report

1st Isaias Rivera
College of Computing
Illinois Institute of Technology
Chicago, Illinois, USA
irivera3@hawk.iit.edu

I. Introduction

Diffusion models are a type of generative model that utilizes the principles of diffusion to generate realistic data. There are different types of diffusion models that have the same underlying idea including, diffusion probabilistic models, noise-conditioned score network, and denoising diffusion probabilistic models [1].

The basic idea behind diffusion-based generative models is that noise is first added to an initial input, where the resulting noisy image is then processed to subsequently remove that noise, while preserving the underlying structure of the image. This process is done multiple times where, at each step of this process, the noise level is decreased, and the resulting image becomes more and more refined. After this training, the model can then be used to generate data by passing random noise through the learned denoising process.

Diffusion-based generative models can function either unconditionally or guided such as with GLIDE's text-guided diffusion model, trading "diversity for fidelity" [2]. There are other examples of this online such as with Google's Imagen, and OpenAI's DALL-E 2.

There are many variations [3] [4] [5] [6] and different improvements [7] [8] to diffusion models that are being actively researched and have already been implemented, however, as in my proposal I mainly have wanted to explore implementing my own in an easily digestible format.

II. Focus

The main focus of my project has been with Denoising diffusion probabilistic models, or DDPM for short, as I have found the most literature revolving around this variation on diffusion models. I have not gone further than DDPMs, however, there are plenty of different perspectives and variations on DDPMs that have been made [6]. Regardless, a good part of my research has been with understanding both the math and implementation of every step with a DDPM [9] [10] [11]. However, the major component has been focused on compiling a very simple DDPM in jupyter that attempts to prevent delegating parts of the process off to an API, both to learn about the entire process and reach a point where images can be generated.

III. Implementation

The model that I have compiled consists of a very simple DDPM, with limited complexity. This allows it to be easier to understand, however, it is also very poor in terms of performance. My main notebook uses torch to implement everything, however, I have found many implementations that use a variety of APIs.

A. The Forward Process

The forward process is the step that involves adding noise to data. Given a number of timesteps, gaussian noise is added to data at each step, where subsequent steps are dependent on the output of the previous step. The main equation that describes this, is as follows

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1})$$

$$:= \prod_{t=1}^{T} \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1},\ \beta_t\mathbf{I})$$

Where T is the total number of timesteps, β_t is the variance schedule, where $\beta_t \in (0, 1)$ and $\beta_1 < \beta_2...$, it describes the amount of noise we want to add to each timestep. This schedule can simply be set as linear, however, a cosine schedule has been proven to improve the performance of the model [7] as it smooths out the rate at which the data is destroyed. Figure 1 shows this.

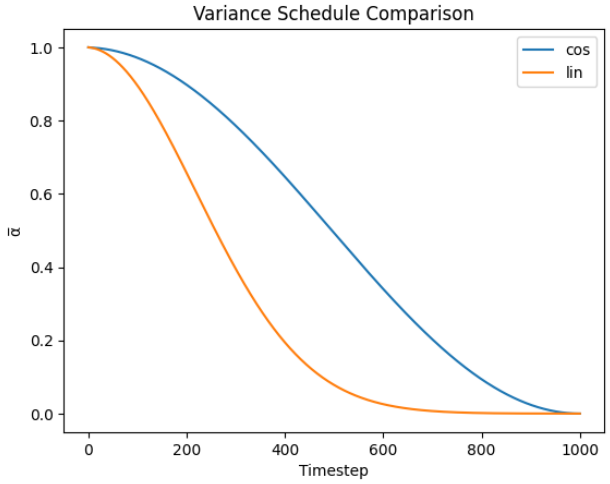

Fig. 1. Variance Schedule Comparison

1) Reparameterization: Reparameterization of the forward process allows for the forward process to be sampled at any given timestep. What this mainly involves is taking the fact that, given two gaussian distributions, the sum these variables are the same as the sum of their mean and variances $\mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$ [10]

What this allows us to do is,instead of just using the beta schedule as is, we use $\bar{\alpha}$, where $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$

This speeds up the forward diffusion process, which is especially helpful, as it is later randomly sampled in order to train the model.

B. The Reverse Process

The reverse process involves sampling the forward process using a random timestep and predicting the noise in that sample. Where a neural network is trained to predict this noise.

The following equation describes this process.

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

$$:= p(x_T) \prod_{t=1}^{T} \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Where X_{t-1} is predicted using the current timestep.

C. Timestep Embedding

Timestep Embedding, or positional embedding, is used to ensure that the model captures dynamics of the data over time. The original paper employs sinusoidal position embeddings to encode t. This is achieved by adding an additional channel to the input of each layer of a u-net. However, results are still relatively similar when this embedding is placed within the first input, which is what is done with the simple implementation.

D. Loss Function

The most common form of loss used is Huber loss, which is a hybrid of Huber loss balances the strengths of Mean Squared Error and Mean Absolute Error loss [4].

E. U-Net

A U-Net is a type of neural network architecture primarily in image segmentation tasks. It consists of an encoder and a decoder network with skip connections between them, which allows it to perform highly accurate segmentation of objects in images with relatively few training examples. Figure 2

A U-Net is the model used by the original DDPM paper, however, it is not the only model that can be used. But for this example, I used a very simplified version of the

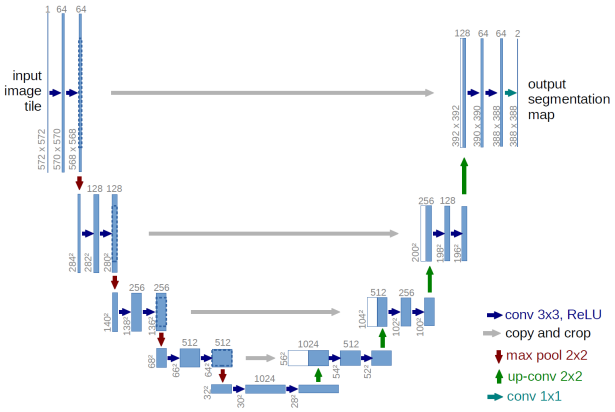

Fig. 2. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [12]

model [13] to aid in understanding the entire process.

F. Training

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t) \right\|^2$$
6: **until** converged

Fig. 3. Training algorithm

Figure 3 is the training algorithm as described in the original DDPM paper. However, simply put, we take a sample at a random timestep of the forward diffusion process and use a model to learn the noise that is added on. Additionally, it should be noted that, because this is typically done in batches, something like stochastic gradient descent is in use to optimize models. In the case of the simplified model, Adam is in use, as many implementations default to using it.

G. Sampling

Figure 4 is the sampling algorithm described in the original DDPM paper. To explain the process at a high level, it first begins with T, where pure noise is generated, where

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: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Fig. 4. Sample algorithm

the trained model is then used to gradually denoise it. This process is continued until t=0.

H. Simplified Example

Beyond the forward diffusion process, a majority of the structure for sampling and the u-net is compiled from various sources.

Unfortunately, due to time constraints, I had to reduce timestep and batch size as to allow the model to converge faster and ensure it did not get stuck at some local optima. Naturally, this affected the final output and, with an already simplified model, had major effects on the output. Regardless, a few samples managed to generate, as shown below. I also switched between cosine and linear beta schedules, as I found that linear schedules, although proven to produce worse results, converged faster.

Figure 5, 6, and 7 show samples of the datasets used for the following examples. FashionMNIST consists of 70,000 grayscale images of size 28x28 pixels, divided into 60,000 images for training and 10,000 images for testing while MNIST is a dataset of handwritten digits used for training and testing computer vision models. It consists of 70,000 grayscale images of size 28x28 pixels, divided into 60,000 images for training and 10,000 images for testing. StandfordCars is a large-scale image recognition dataset that consists of 16,185 images of 196 different car models. The images were collected from various online sources, such as auction websites and dealerships. The samples shown here, are after being processed to fit into tensors for

Fig. 5. Sample of FashionMNIST

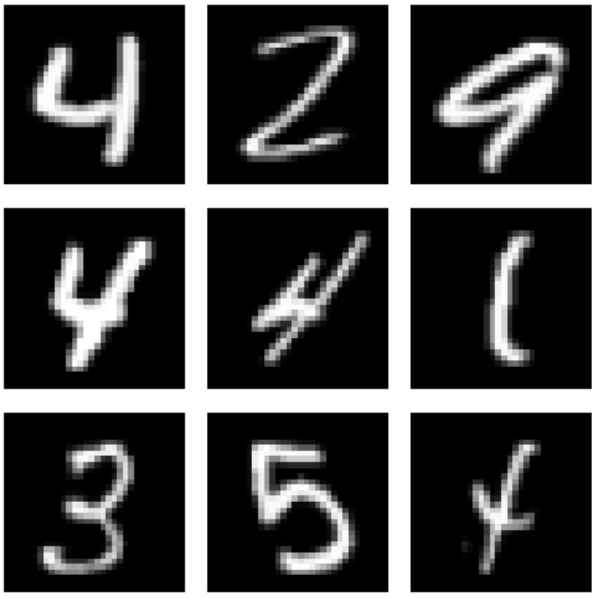

Fig. 6. Sample of MNIST

Fig. 7. Sample of StandfordCars

Fig. 8. FashionMNIST @ epoch 5 — linear schedule

terms of color, does not resemble a number as well. However, various articles and papers also show how, on the long run, cosine is always the better choice.

Overall, the results of the simplified model are not great, however, this was to be expected. Regardless, the fact that images are still able to converge with an un-optimized model on a consumer machine, shows how much DDPMs are capable of.

Another quick run using the StandfordCars data set is shown with figure 11. Due to time, I did have to cut it short, however, it is clear

further use, and then reversed back into viewable images, hence why, they are all uniform in size.

A comparison between the linear, figure 9 and cosine schedule, figure 10, for the MNIST examples exemplifies how the linear schedule was able to produce a clear attempt at generating a number at epoch 9, while with cosine at epoch 12, although looks better in

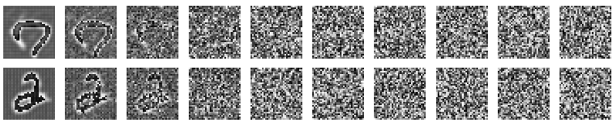

Fig. 9. MNIST @ epoch 9 — linear schedule

Fig. 10. MNIST @ epoch 12 — cosine schedule

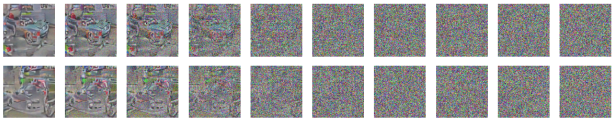

Fig. 11. StandfordCars @ epoch 27 — linear schedule

that some form of image was generating. This examples was using a 128x128 image to start with, where the previous examples were using only 32. This explains why at epoch 27 it is still very noisy and scattered.

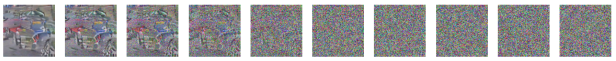

Fig. 12. StandfordCars @ epoch 63 — linear schedule

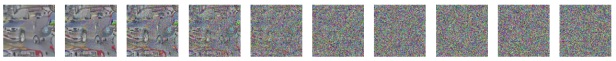

Fig. 13. StandfordCars @ epoch 69 — linear schedule

One final example of the model running the StandfordCars dataset in figure 12 13, just of an example where running it for longer does begin to generate more sensible features of a car. However, it still is not necessarily a single subject, at this point, yet.

The notebook made for this report can be found hosted on github here.

I. Conclusion

Overall, the Denoising Diffusion Probabilistic Model is a very interesting model, which has recently gained a lot of traction in terms of improvements and variations. Unfortunately, it is not as well documented or accessible, in terms of understanding, as something like GAN models, which have been around for longer. Regardless, I personally have learned a lot with the process of diffusion models and how they are implemented. Additionally, I have also gained a more hands on understanding on how applying and using models and techniques in general is done. Beyond this, DDPMs have already seen incredible improvements and variations. Some include training for both variance and mean, versus just mean [7], including atttention and residual blocks in the u-net [6], and Text to image generation [14].

REFERENCES

[1] L. Weng, "What are diffusion models?," *lilianweng.github.io*, Jul 2021.

[2] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2021.

[3] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022.

[4] A. Nain, "Denoising diffusion probabilistic model," 2022.

[5] A. Q. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever, and M. Chen, "GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models," in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 16784–16804, PMLR, 17–23 Jul 2022.

[6] N. R. Kashif Rasul, "The annotated diffusion model," 2022.

[7] A. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," 2021.

[8] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," 2022.

[9] A. Nain, "All you need to know about gaussian distribution."

[10] A. Nain, "A deep dive into ddpms."

[11] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020.

[12] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351 of *LNCS*, pp. 234–241, Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[13] A. Arora, "U-net a pytorch implementation in 60 lines of code," 2020.

[14] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, "Photorealistic text-to-image diffusion models with deep language understanding," 2022.