UNIVERSITY OF TOKYO

# Implementation of a Tiny Denoising Diffusion Probabilistic Model

Lukas Kunz

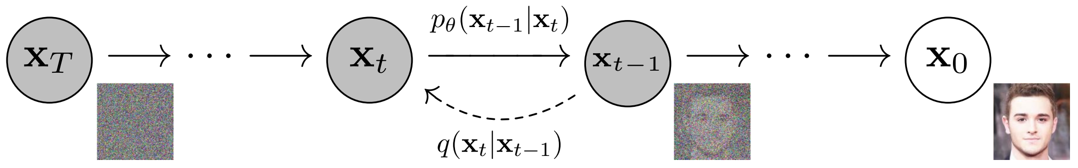
August 1, 2024

## 1 INTRODUCTION

The aim of this project was to implement a small but functional version of a diffusion model as proposed in the 2020 paper Denoising Diffusion Probabilistic Models by Jonathan Ho, Ajay Jain, and Pieter Abbeel. The paper introduces a novel approach to probabilistic diffusion models (diffusion models in short) that are for the first time capable of producing high quality images comparable or even superior to those produced by other types of generative models of that time (such as generative adversarial networks for example). In this report, I will first explain the general idea of denoising diffusion models with particular attention to the loss objective, then I will discuss the authors implementation and what makes it unique, and lastly, I will explain how I implemented my version of the model by referencing the code that can be found on the github repository accompanying this report.

## 2 WHAT ARE DENOISING DIFFUSION MODELS

Denoising diffusion models are a type of generative model that first corrupt a sample by step wise introducing more and more noise acording to a variance schedule $\beta_1, ..., \beta_T$ until the sample is close to pure noise. It then tries to learn to remove the added noise at each step. After training, such a model has learned a mapping from pure noise to the distribution of the training data and can thus be used to sample data like the training data from pure noise. Figure 2.1 illustrates this process and Figure 4.3 shows the actual sampling process of my model after training.



$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 2.1: Diffusion Graph from the Paper

## 2.1 Forward Process

Gradually corrupting the data with noise is called the forward process (or diffusion process) and is fixed to a Markov chain. At each step, Gaussian noise is added according to a variance schedule $\beta_1, ..., \beta_T$ :

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{2.1}$$

The complete distribution is then given by:

$$q(\mathbf{x}_{0:T}) = q(\mathbf{x}_0) \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{2.2}$$

## 2.2 Reverse Process

Similarely, the reverse process is also defined as a Markov chain. Here we try to learn the Gaussian transitions at each step starting at $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$ to gradually remove noise and finally arrive at a sample from the data distribution:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{2.3}$$

The full reverse process is then given by:

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \tag{2.4}$$

## 2.3 Training Objective

The full probability distribution of the model is given by:

$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \tag{2.5}$$

Since we want the model to output samples like in the training data, we want to maximize the expectation of $p_\theta(\mathbf{x}_0)$, where $\mathbf{x}_0$ are uncorrupted samples from the data distribution. This is the same as minimizing the expectation of the negative log likelihood of $p_\theta(\mathbf{x}_0)$. Thus, our loss objective is:

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \tag{2.6}$$

However, the integral of the model probability seems intractable. Thus, we need to simplify a little. First, we notice that if we multiply by one in the form of the approximate posterior of the forward process, we get the following expectation over the posterior:

$$
\begin{aligned}
p_\theta(\mathbf{x}_0) &= \int p_\theta(\mathbf{x}_{0:T}) \frac{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} d\mathbf{x}_{1:T} && (2.7)\\
&= \int q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) \frac{p(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} d\mathbf{x}_{1:T} && (2.8)\\
&= \mathbb{E}_q \left[ \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} \right] && (2.9)
\end{aligned}
$$

Now we can use Jensen's inequality to take the log inside the expectation and we get the following approximate loss objective, which is also known as the variational bound on negative log likelihood:

$$
\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q \left[ -\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)} \right] = \mathbb{E}_q \left[ -\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] =: \mathscr{L} \quad (2.10)
$$

Optimizing this objective works well in practice.

## 3 Authors Implementation

The above discussed description of denoising diffusion models still leaves a lot of freedom regarding how a particular model can be implemented. The authors main objective was to achieve a high sample quality, i.e. generate pictures of impressive quality. To that end, they came up with a particular implementation that includes the following:

- Fixed variance schedule.
- A particular parametrization of the Gaussian noise with a fixed scaled identity matrix as covariance matrix.
- Simplified loss objective to predict the noise added at each step (rather than predicting an image with less noise).
- A U-Net architecture with self attention, convolutional layers, time step conditioning and more.
- Exponential moving average during training
- etc.

Given that I am working with very limited computing resources, I only implemented a very small model based on the authors main ideas. As a result, my network architecture is much smaller and I haven't implemented everything the authors did. Thus, in the following I will only discuss what I consider the main achievement of the authors paper, which is their novel and simple loss objective.

## 3.1 Simple Loss

In appendix A of the paper, the authors show that the variational bound on negative log likelihood can be further rewritten as:

$$\mathscr{L} = \mathbb{E}_q \left[ \mathscr{L}_T + \sum_{t>1} \mathscr{L}_{t-1} + \mathscr{L}_0 \right] \tag{3.1}$$

where

$$\mathscr{L}_T = D_{KL}(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T)) \tag{3.2}$$

$$\mathscr{L}_{t-1} = D_{KL}(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)) \tag{3.3}$$

$$\mathscr{L}_0 = -\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \tag{3.4}$$

- $\mathscr{L}_T$ measures the KL divergence between the approximate posterior $q(\mathbf{x}_T \mid \mathbf{x}_0)$ and the prior $p(\mathbf{x}_T)$.
- $\mathscr{L}_{t-1}$ measures the KL divergence between the approximate posterior $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and the learned reverse process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$.
- $\mathscr{L}_0$ measures the negative log likelihood of the data under the learned distribution.

The authors then simplify this objective by first choosing a fixed variance schedule (instead of learning the parameters $\beta_1, ..., \beta_T$). By doing so the approximate posterior $q$ has no learnable parameters and thus, $\mathscr{L}_T$ is constant during training and can be omitted.

Secondly, for the reverse process $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$ the authors choose a fixed covariance $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$. Since now both $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ are Gaussian distributions with the same variance, the KL divergence simplifies to:

$$\mathscr{L}_{t-1} = \frac{1}{2\tilde{\sigma}_t^2} \left\| \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t) \right\|^2 + C \tag{3.5}$$

Now with the notation $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$, by reparametrizing $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and choosing the parametrization $\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{x}_t, t) \right)$ and after some rearranging, $\mathscr{L}_{t-1}$ further simplifies to:

$$\mathscr{L}_{t-1} = \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2 \tag{3.6}$$

Finally, empirically, the authors determined that leaving out the complicated scaling factor in front of $\mathscr{L}_{t-1}$ as well as omitting $\mathscr{L}_0$ leads to better sample quality. Thus we are left with the following simple loss objective:

$$\mathscr{L}_{\text{simple}}(\theta) = \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[ \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2 \right] \tag{3.7}$$

Thus, the simplified loss function $\mathcal{L}_{\text{simple}}$ focuses on the mean squared error between the true noise $\epsilon$ and the predicted noise $\epsilon_\theta$. This approach bypasses the need to directly minimize the entire variational bound and instead concentrates on the noise prediction aspect, which is the core task of denoising. This simplification significantly stabilizes training and yields high-quality generative performance.

## 4 My Implementation

My implementation can be found in the jupyter notebook in the github repository. Here, I will simply mention two things before presenting some results: first, a few words on my neural network architecture and second, I present my implementation of the simplified loss objective.

### 4.1 Neural Network Architecture

First, for the neural network architecture I adapted a very simple UNet implementation from Hugging Face shown in Figure 4.1. Since the neural net by Hugging Face is intended to predict the denoised image directly, I made the following two important changes to be able to predict the noise at any given time step:

1. First, I added a time step embedding and concatenated it with all layers, so the model would always be aware of where in the diffusion process it was.

2. Second, I added a linear layer to the end for the model to predict values in the range of the noise it is trained to predict.

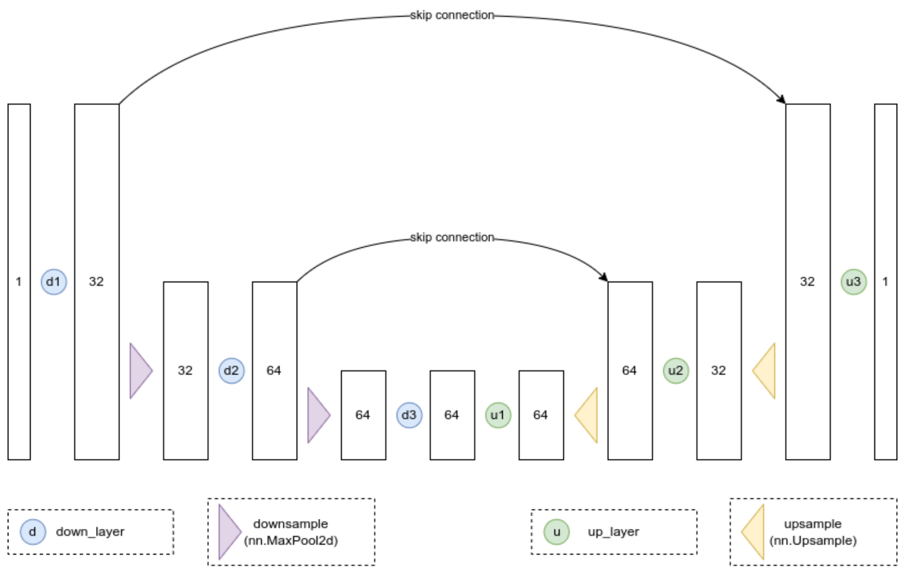


Figure 4.1: UNet from Hugging Face

### 4.2 Loss

As discussed above, the loss function the authors propose is the following:

$$\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2\right]$$

The implementation is quite straight forward as the code snippet 1 shows.

Listing 1: My Loss Implementation

```python
def noise_estimation_loss(self, x_0):
    # Noise
    epsilon = torch.randn_like(x_0).to(device)
    # get random timesteps t for each datapoint in the x_0 batch
    t = torch.randint(0, self.n_steps, (x_0.size(0),)).to(device)
    # Get coefficients
    sqrt_a = self.get_coefficients(self.alpha_bars_sqrt, t, x_0).
        to(device)
    sqrt_1ma = self.get_coefficients(self.
        one_minus_alpha_bars_sqrt, t, x_0).to(device)
    # Get model prediction
    out = self.forward(sqrt_a*x_0 + sqrt_1ma*epsilon, t.float())
    # Calculate Loss
    loss = (epsilon - out).square().mean()
    return loss
```

### 4.3 Results

Since my model is very small and simple, I trained it on the fashion MNIST dataset. I used a free google colab GPU to train the model for roughly 400 epochs or around 2h. The image quality did not significantly improve after 200 epochs.

First, Figure 4.2 shows excerpts of the training samples. Figure 4.3 shows the sampling process of the trained model. It starts with pure noise and gradually refines it over 100 time steps (the paper used 1000 steps). Lastly, Figure 4.4 shows 25 uncurated samples generated by my model after training for around 400 epochs or about 2h on a free google colab GPU. The image quality isn't very good and quite bright compared to the training samples, however, the model is clearly able to generate images like the ones in the training set.

## 5 Conclusion

Seeing that today the most impressive image generation results are achieved by diffusion models, the 2020 Denoising Diffusion Probabilistic Models paper has clearly had a major impact. Despite all the challenges that AI poses to society, I am excited to see what else is to come.

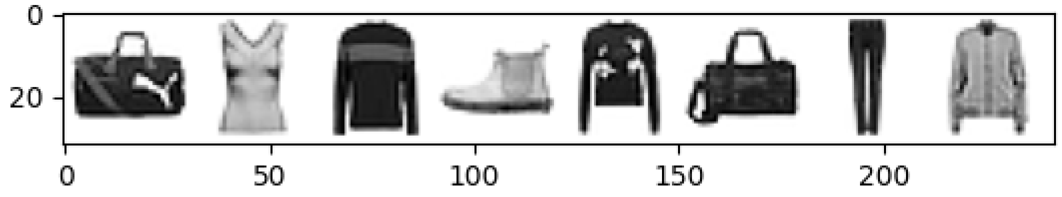


Figure 4.2: Training Data

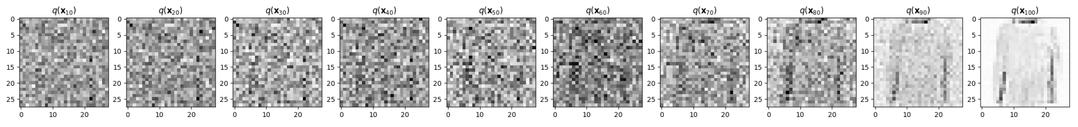


Figure 4.3: Sampling Process

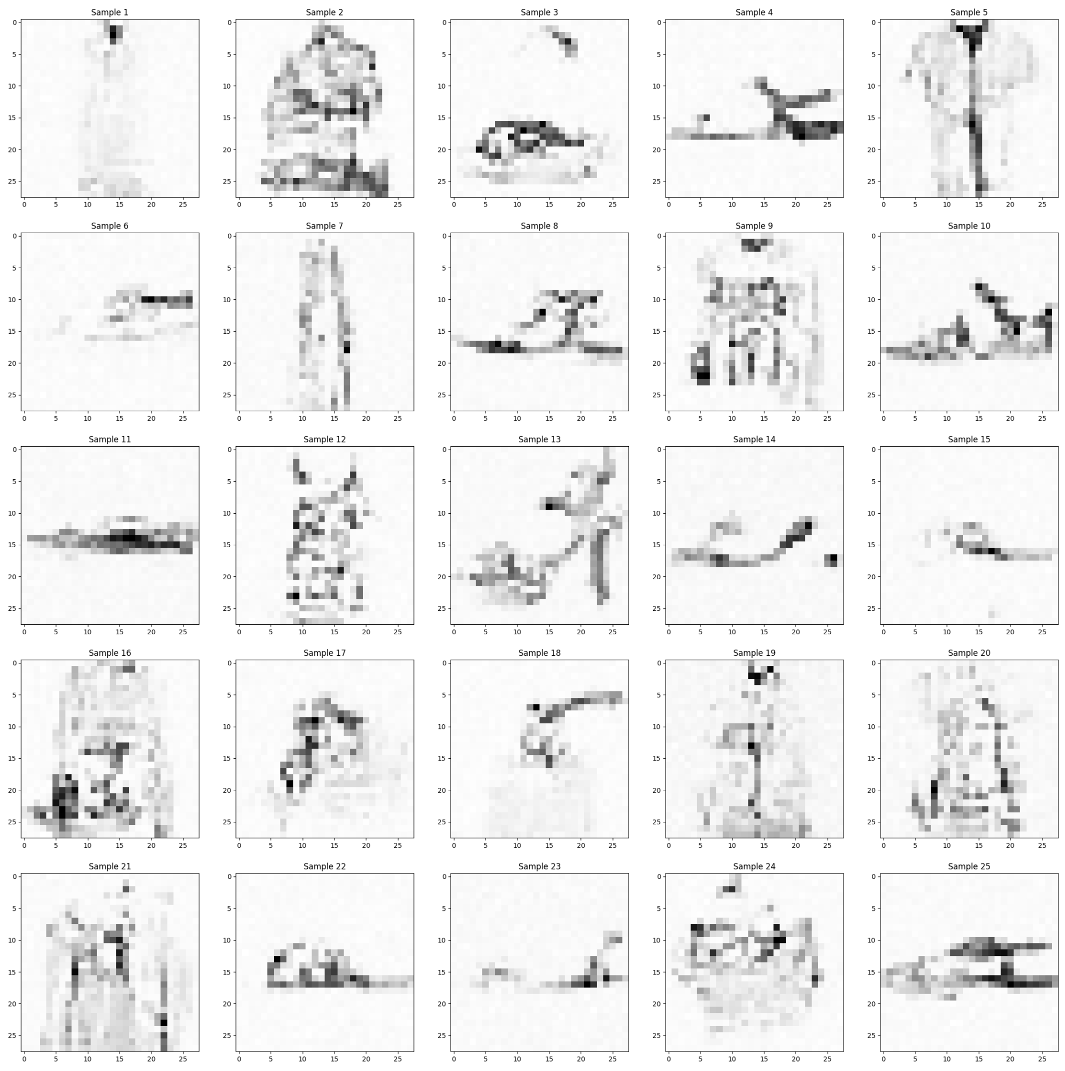


Figure 4.4: Uncurated Samples from the Model