Generative Diffusion Models

Shervin Khalafi *and Alejandro Ribeiro

1 Project Description

In this project you will first learn a bit about Diffusion models. You will then implement a basic Diffusion Model and train it on the MNIST dataset so that it learns to generate new images of handwritten digits that are similar (but not identical) to those found in the dataset.

2 What Is a Diffusion Model?

In this section we will go over everything you need to know for implementing a diffusion model. For a thorough explanation of a Diffusion model including all the math, see "Understanding Diffusion Models: A Unified Perspective" by Calvin Luo.

2.1 Generative Models

A generative model is a type of statistical model that is capable of generating new data instances that resemble the training data. These models learn the underlying distribution of the data in order to produce new instances that could have plausibly come from the same dataset.

*If you have any questions regarding the project you can reach me at shervink@seas.upenn.edu

To be a bit more precise consider $q(\boldsymbol{x})$ to be the probability density assigned to the point $\boldsymbol{x} \in \mathbb{R}^d$ by the underlying distribution of the data. But it often is the case that we don't know the underlying distribution $q(\cdot)$. We only have access to some data points sampled from it in the form of a dataset $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(M)}\}$. The goal of a generative model is to learn a distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, parameterized by $\boldsymbol{\theta}$ such that $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is "as close as possible" to $q(\boldsymbol{x})$. In other words, the goal is to learn to generate new samples that "could have been from the same dataset".

2.2 Latent Representation of the Data

For many of the types of data we typically encounter, we can think of the data we observe, $\boldsymbol{x}$ as represented or generated by an associated unseen *latent* variable, which we can denote by random variable $\boldsymbol{z}$. The best intuition for expressing this idea is through Plato's Allegory of the Cave. In the allegory, a group of people are chained inside a cave their entire life and can only see the two-dimensional shadows projected onto a wall in front of them, which are generated by unseen three-dimensional objects passed before a fire. To such people, everything they observe is actually determined by higher-dimensional abstract concepts that they can never behold.

A diffusion model is one if many ways to model the relationship between the observed data $\boldsymbol{x}$ and the corresponding latents $\boldsymbol{z}$.

2.3 Diffusion Models

In a Diffusion model, we call the true data observation $\boldsymbol{x}_0$ (for example a natural image from our dataset) and we add small amounts of noise to it in multiple time steps to get a series of noisy versions of the original sample $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T$. Here, our latent $\boldsymbol{z}$ is essentially all of the noisy images $\boldsymbol{x}_t$ for $t = 1, \cdots, T$.

Each noising step can described as a conditional distribution:

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\alpha_t}\boldsymbol{x}_{t-1}, (1-\alpha_t)\mathbf{I}) \tag{1}$$

For those of you that are less familiar with probability, it suffices to know

that:

$$\boldsymbol{x}_t = \sqrt{\alpha_t} \cdot \boldsymbol{x}_{t-1} + (1 - \alpha_t) \cdot \boldsymbol{\epsilon} \tag{2}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$ is just a noise sample from the standard Gaussian distribution. The parameters α_t determine the size of the noise added in each time step and in this project are chosen as hyperparameters and are fixed during training. This choice of α_t is called the "noise schedule".

In a nutshell we start from initial image $\boldsymbol{x}_0$ and add a small amount of noise to it in each step until we reach an image $\boldsymbol{x}_T$ in the final step that's essentially just pure Gaussian noise (see Figure (1)). This is called the forward process. It is relevant that the properties of the Gaussian distribution lets us sample an $\boldsymbol{x}_t$ directly from $\boldsymbol{x}_0$ without having to go through all the steps as follows:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t} \cdot \boldsymbol{x}_{t-1} + (1 - \bar{\alpha}_t) \cdot \boldsymbol{\epsilon}_0 \tag{3}$$

where $\bar{\alpha} = \prod_{i=1}^{t} \alpha_i$ and $\boldsymbol{\epsilon}_0 \sim \mathcal{N}(\boldsymbol{\epsilon}_0; \mathbf{0}, \mathbf{I})$. Now comes the hard part which is: How can we start from pure noise i.e. $\boldsymbol{x}_T$ and iteratively denoise the image until we reach $\boldsymbol{x}_0$? We call this the backward process and this is what we will train a model to do.

Assume we have $\boldsymbol{x}_t$ and we want to find the most likely $\boldsymbol{x}_{t-1}$ that could have generated this $\boldsymbol{x}_t$. Well, if we also knew the $\boldsymbol{x}_0$ that was the starting point for generating both $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-1}$ we could compute this most likely $\boldsymbol{x}_{t-1}$ since we know all the transition distributions $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$. We call this "most likely" prediction $\mu_q(\boldsymbol{x}_t, \boldsymbol{x}_0, t)$ and we can derive it as:

$$\mu_q(\boldsymbol{x}_t, \boldsymbol{x}_0, t) = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})\boldsymbol{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)\boldsymbol{x}_0}{1 - \bar{\alpha}_t} \tag{4}$$

But the problem is, during the backward process when we start from random noise $\boldsymbol{x}_T$ and denoise it step by step, we don't know what $\boldsymbol{x}_0$ is. This is where the neural network finally comes into play. We will call our best prediction for $\boldsymbol{x}_{t-1}$ when we only have access to the current noisy image, $\mu_\theta(\boldsymbol{x}_t, t)$. In order to make this prediction as close as we can to $\mu_q(\boldsymbol{x}_t, \boldsymbol{x}_0)$, we choose it to be the following:

$$\mu_\theta(\boldsymbol{x}_t, t) = \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})\boldsymbol{x}_t + \sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t) \cdot \hat{x}_\theta(\boldsymbol{x}_t, t)}{1 - \bar{\alpha}_t} \tag{5}$$

Note that the only difference between (4) and (5) is that in (5) we have replaced $\boldsymbol{x}_0$,which we don't have access to in the backwards process, with $\hat{x}_\theta(\boldsymbol{x}_t, t)$. Here, $\hat{x}_\theta(\boldsymbol{x}_t, t)$ represents a neural network parameterized by θ that seeks to predict the original image $\boldsymbol{x}_0$ from noisy image $\boldsymbol{x}_t$ and time index t. So all we have to do is to train the neural network $\hat{x}_\theta(\boldsymbol{x}_t, t)$ to be good at predicting the original image $\boldsymbol{x}_0$ given noisy image $\boldsymbol{x}_t$ and time index t. With all this in mind, it makes sense to choose our loss function evaluated for a single image $\boldsymbol{x}_0$ to be the average error of our network in predicting $\boldsymbol{x}_0$:

$$\hat{\ell}(\theta; \boldsymbol{x}_0) = \mathbb{E}_{t \sim U\{2,T\}} \left[\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} \left[||\hat{x}_\theta(\boldsymbol{x}_t, t) - \boldsymbol{x}_0||_2^2 \right] \right] \tag{6}$$

Note that there are two averages being computed in the loss above.

- The first one $\mathbb{E}_{t \sim U\{2,T\}}$ is a uniform average over all time steps. This makes sense because we want our network to learn to predict the original image well at *all* time steps not just for any single time step. You can estimate this by sampling some time indices uniformly from 2 to T. [1]
- The second one $\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}$ tells us that for any given t, we should average the loss over different possible noisy images $\boldsymbol{x}_t$ that could be generated by the forward process. We do this because since the forward process is stochastic, even when starting from the same $\boldsymbol{x}_0$ you could get a different $\boldsymbol{x}_t$ each time you run the forward process. Our model should learn to denoise all of these potential $\boldsymbol{x}_t$ images well.

In the implementation of our diffusion model, we will train a neural network $\hat{x}_\theta(\boldsymbol{x}_t, t)$ to minimize the loss $\hat{\ell}(\theta; \boldsymbol{x}_0)$, averaged on samples $\boldsymbol{x}_0$ drawn from the MNIST dataset.

[1] The fact that we don't consider $t = 1$ is a bit technical. Don't worry about it in this project

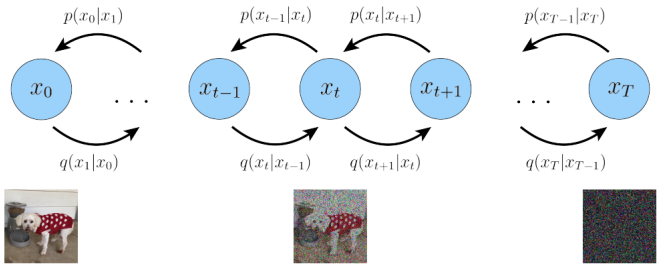

Figure 1. A Graphical representation of a Diffusion Model.

3 Notes on Implementation

To make your life a little easier, keep the following in mind when you are implementing the project:

- While you are free to use any neural network for $\hat{x}_\theta(\boldsymbol{x}_t, t)$ that takes as input a noisy image $\boldsymbol{x}_t$ and time index t and outputs an image with the same dimensions as $\boldsymbol{x}_t$, we suggest that you use the "UNet2DModel" from the "diffusers" python library. This network is a time-conditioned UNet. For the purposes of this project you can consider a UNet to be similar to a CNN where the dimensions of the output and input image are the same.
- If training on the entire MNIST dataset is taking too long on your computer, you can train on a randomly selected subset of the dataset. Try to make this subset as large as you can given the limitations of your hardware.
- Regardless of the previous point, training might take a long time so account for that in your time management.
- For generating new samples using our models there are two different ways you could do it (Known as DDPM and DDIM sampling). You could either sample a standard Gaussian random noise $\boldsymbol{x}_T$ and then iteratively apply

$$\boldsymbol{x}_{t-1} = \mu_\theta(\boldsymbol{x}_t, t) \tag{7}$$

  for $t = T, T-1, \cdots, 1$ until you reach $\boldsymbol{x}_0$. Alternatively, you could do the stochastic version of (7) and add small mean-zero Gaussian

noise $\boldsymbol{\epsilon}_t$ at each denoising step:

$$\boldsymbol{x}_{t-1} = \mu_\theta(\boldsymbol{x}_t, t) + \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t \tag{8}$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{\epsilon}_t; \mathbf{0}, \mathbf{I})$. You can try both. What advantage/disadvantage do you think each method has?

- For the noise schedule, feel free to play around with different schedules. You can try the one used in the paper "Denoising Diffusion Probabilistic Models" by Ho et al. as a good baseline (note that β_t in the paper is equivalent to our $1 - \alpha_t$).

4 Your Report

Your report should include the following for a full mark:

- Clear and concise explanations of each part of your implementation (comments in the code do not count).
- Attached code for each part of the implementation.
- Clear denoting of all the hyperparamters that you chose for your final model and the reasoning behind your choices.
- Comprehensive evaluation of the training process, including but not limited to multiple images generated by your model at different epochs/steps during the training process.