Text-to-image DDPMs

Giovani Tavares
giovanitavares@usp.br

University of São Paulo — December 11, 2025

1 Goal

The goals of the present work are to first document the Deriving the training and sampling algorithms of Denoising Diffusion Probabilistic Models (DDPMs) and also use the insights obtained in such derivations to modify the DDPM implementation in [1] so that it becomes a *text-to-image* generative model, instead of a model that generates random images. In summary, [1] implements a DDPM that generates random images., and what I did was to modify it so that the images are not random, but instead it is possible to pass the class parameter to the sampling algorithm so it outputs an image from such class.

With the knowledge and insights obtained in the derivation of the training and sampling algorithms of DDPMs described in the first section, we will implement a conditional DDPM with FiLM layers [3] attatched to this report. FiLM layers are what lets one marginalize the generated images distribution learned by the model on their classes, so that the result is a *text-to-image* DDPM.

The sections of the present work are:

- **What Are DDPMs**: Deriving the training and sampling algorithms of Denoising Diffusion Probabilistic Models (DDPMs) using the several references shown in the end of this work
- **Conditional DDPMs - A text-to-image Generative Model**: Deriving the training and sampling algorithms of a *conditional* DDPMs that lets one guide the generation of images of specific classes instead of random ones as in the original DDPM

2 What are DDPMs?

Denoising Diffusion Probabilistic Models (DDPMs) are models capable of predicting *noise* from a noisy input. By using such prediction, a sampling algorithm can be used to remove the noise from the input which results in a denoised output.

DDPMs are made of two processes: a **forward** and a **reversion** process. The former is responsible for gradually adding noise to a image by sampling from a normal distribution according to a Markov Chain. The latter removes added noise by sampling from another normal distribution. In simple terms, the training of DDPMs involve learning the reversion process' distribution's parameters.

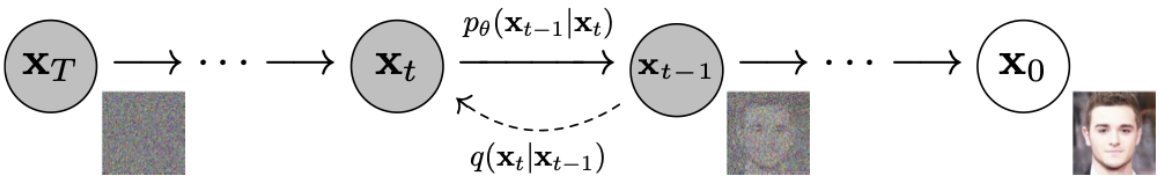

Figure 1: Image extracted from the original DDPM paperl: "*Denoising Diffusion Probabilistic Models*" [2]

2.1 Forward Process q

The process of adding noise to an input image (x_0) is a Markov Chain that generates a noisier image x_t from a less noisy image x_{t-1}. Hence, x_t represents the result of adding noise to x_{t-1} by transitioning it **once** in the Markov Chain.

From the original DDPM paper, we know that in the forward process, a sample x_t is produced by adding noise to a sample x_{t-1} according to a normal distribution defined below:

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

Ideally, one would need a single transition from x_0 to get to x_t, with $t > 0$.

The authors of [2] achieves such ideal scenario by defining a cumulative noise α_t presented below:

Definition 2.1 (Cumulative Noise)**.** *The Cumulative Noise (α_t) added to an input x_0 up to the t-th step is defined as:*

$$\alpha_t := 1 - \beta_t \tag{2}$$

$$\bar{\alpha_t} := \prod_{s=1}^{t} a_s \tag{3}$$

which leads to (4)

$$q(x_t|x_0) := \mathcal{N}(x_t; \sqrt{\bar{\alpha_t}} \times x_0; (1-\bar{\alpha_t})I) \tag{5}$$

Definition 2.1 permits the sampling of x_t to be done in a single transition from x_0, instead of having to produce every intermediate image.

According to the original DDPM paper, the **Variance Schedule** $\beta_1, ..., \beta_T$ sequence that defines the noisy images distribution are held constant.

2.2 Reverse Process p_θ

The Reverse Process is a Markov Chain with a Standard Gaussian initial state and Gaussian transition distribution p_θ parametrized by mean μ_θ and variance Σ_θ. **The core idea of the chain is that the transitions remove each a little bit of the noise from the initial state up until the noise-free state $\mathbf{x_0}$.**

Definition 2.2 (Reverse Process Transitions)**.** *The Reverse Process is a Markov Chain with the following transitions:*

$$p(x_T) = \mathcal{N}(x_T; 0; 1) \tag{6}$$

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{7}$$

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t); \Sigma_\theta(x_t, t)) \tag{8}$$

What we ultimately want is to have a $\mathbf{x}_0$ that is as likely as possible. We can use the standard rule of probability to obtain a marginalization of $p(\mathbf{x}_0)$ using the latent variables $\mathbf{x_{1:T}}$

Definition 2.3 (Reverse Process Prior)**.** *Ideally, the Reverse Process would let us sample from:*

$$p(\mathbf{x_0}) = \int p(\mathbf{x_0}, \mathbf{x_{1:T}}) \mathbf{d_{1:T}} \tag{9}$$

(10)

From the defition above, we see that $p(\mathbf{x}_0)$ is very complex due to its **multidimensionality**, which makes it intractable. That is why in DDPMs, $p(\mathbf{x}_0)$ is never computed directly, but instead its lower bound.

2.2.1 Evidence Lower Bound / ELBO

The Evidence Lower Bound is a tight lower bound that limits $log(p(\mathbf{x_0}))$ from below. Hence, when $p(\mathbf{x_0})$ is intractable as in the case of DDPMs, one can always maximize such lower bound as a means to ensure that $log(p(\mathbf{x_0}))$ is as large as possible. Such lower bound is often called **ELBO** and will be demonstrated using two different approaches: the **Jensen's Inequality** and the **KL Divergence**.

Definition 2.4 (Evidence Lower Bound - Jensen's Inequality)**.** *Let's use the Rule Of Total Probability to find a lower bound for the log-likelihood function.*

$$log[p_\theta(\mathbf{x_0})] = log \int_{\mathbf{x_{1:T}}} p(\mathbf{x_0}, \mathbf{x_{1:T}}) d\mathbf{x_{1:T}} \tag{11}$$

$$log[p_\theta(\mathbf{x_0})] = log \int_{\mathbf{x_{1:T}}} p(\mathbf{x_0}, \mathbf{x_{1:T}}) \frac{q(\mathbf{x_{1:T}}|\mathbf{x_0})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})} d\mathbf{x_{1:T}} \tag{12}$$

$$log[p_\theta(\mathbf{x_0})] = log(\mathbb{E}_q \Big[\frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})} \Big]) \tag{13}$$

the Jensen's inequality tells us (14)

$$f(\mathbb{E}(\mathbf{X})) \geq \mathbb{E}(f(\mathbf{X}))) \tag{15}$$

for any concave function f. (16)

log is concave, hence: (17)

$$log[p_\theta(\mathbf{x_0})] \geq \mathbb{E}_q \Big[log \frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})} \Big] \tag{18}$$

If we define the Evidence Lower Bound L as: (19)

$$L := \mathbb{E}_q \Big[-log \frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})} \Big] \tag{20}$$

$$\implies -log[p_\theta(\mathbf{x_0})] \leq L \tag{21}$$

Definition 2.5 (Evidence Lower Bound - KL Divergence)**.** *In order to reverse the forward process, we need that the forward process' distribution $q(\mathbf{x_{1:T}}|\mathbf{x_0})$ is as close to $p(\mathbf{x_{1:T}}|\mathbf{x_0})$ as possible. We can use the Kullback-Leibler (KL) divergence between q and p ($\mathbf{D_{KL}}$) to evaluate their difference as find a bound to $log[p_\theta(\mathbf{x}_0)]$.*

$$\mathbf{D_{KL}}\big[q(\mathbf{x_{1:T}}|\mathbf{x_0})||p(\mathbf{x_{1:T}}|\mathbf{x_0})\big] := \mathbb{E}_q\big[log(q(\mathbf{x_{1:T}}|\mathbf{x_0}) - log(p(\mathbf{x_{1:T}}|\mathbf{x_0}))\big] \tag{22}$$

using the Bayes' Rule we can write (23)

$$p(\mathbf{x_{1:T}}|\mathbf{x_0}) = \frac{p(\mathbf{x_{1:T}}, \mathbf{x_0})}{p(\mathbf{x_0})} \tag{24}$$

$$\implies \mathbf{D_{KL}}\big[q(\mathbf{x_{1:T}}|\mathbf{x_0})||p(\mathbf{x_{1:T}}|\mathbf{x_0})\big] = \mathbb{E}_q\big[log(q(\mathbf{x_{1:T}}|\mathbf{x_0}) - log(p(\mathbf{x_{1:T}}|\mathbf{x_0})) + log(p(\mathbf{x_0}))\big] \tag{25}$$

the prior of the latent variables does not depend on q (26)

$$\mathbf{D_{KL}}\big[q(\mathbf{x_{1:T}}|\mathbf{x_0})||p(\mathbf{x_{1:T}}|\mathbf{x_0})\big] = \mathbb{E}_q\big[log(q(\mathbf{x_{1:T}}|\mathbf{x_0}) - log(p(\mathbf{x_{1:T}}, \mathbf{x_0}))\big] + log(p(\mathbf{x_0})) \tag{27}$$

$$\implies log(p(\mathbf{x_0})) = \mathbf{D_{KL}}\big[q(\mathbf{x_{1:T}}|\mathbf{x_0})||p(\mathbf{x_{1:T}}|\mathbf{x_0})\big] - \mathbb{E}_q\big[log(q(\mathbf{x_{1:T}}|\mathbf{x_0}) - log(p(\mathbf{x_{1:T}}, \mathbf{x_0}))\big] \tag{28}$$

$$log(p(\mathbf{x_0})) = \mathbf{D_{KL}}\big[q(\mathbf{x_{1:T}}|\mathbf{x_0})||p(\mathbf{x_{1:T}}|\mathbf{x_0})\big] + \mathbb{E}_q\big[log(p(\mathbf{x_{1:T}}, \mathbf{x_0}) - log(q(\mathbf{x_{1:T}}|\mathbf{x_0}))\big] \tag{29}$$

but $\mathbf{D_{KL}}\big[q(\mathbf{x_{1:T}}|\mathbf{x_0})||p(\mathbf{x_{1:T}}|\mathbf{x_0})\big] \geq 0$ (30)

$$\implies -log(p(\mathbf{x_0})) \leq \mathbb{E}_q\big[-log \frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})}\big] \tag{31}$$

If we define the Evidence Lower Bound L as: (32)

$$L := \mathbb{E}_q \Big[-log \frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})} \Big] \tag{33}$$

$$\implies -log[p(\mathbf{x_0})] \leq L \tag{34}$$

We have found upper bound L for the negative log-likelihood function that can be maximided in the forward process' training.

2.2.2 Noise Predictor Training

In order to use L as the loss function in our training, further algebraic manipulation must be performed

Definition 2.6 (Noise Predictor's Loss Derivation)**.** *In order to build the Noise Predictor's loss function, we need to remember that both forward and reverse processes are Markov Chains and use this fact to manipulate L.*

$$L = \mathbb{E}_q\Big[- log\frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})}\Big] \tag{35}$$

The forward and reverse processes are Markov Chains, so (36)

$$\mathbb{E}_q\Big[- log\frac{p(\mathbf{x_{0:T}})}{q(\mathbf{x_{1:T}}|\mathbf{x_0})}\Big] = \mathbb{E}_q\Big[log\frac{p(\mathbf{x_T})\prod_{t=1}^{T} p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})}{q(\mathbf{x_1}|\mathbf{x_0})\prod_{t=2}^{T} q(\mathbf{x_t}|\mathbf{x_{t-1}},\mathbf{x_0})}\Big] \tag{37}$$

$$= \mathbb{E}_q\Big[- log\frac{p(\mathbf{x_T})\prod_{t=1}^{T} p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})}{q(\mathbf{x_T}|\mathbf{x_0})\prod_{t=2}^{T} q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})}\Big] \tag{38}$$

$$= \mathbb{E}_q\Big[- log\frac{p(\mathbf{x_T})p_\theta(\mathbf{x_0}|\mathbf{x_1})\prod_{t=2}^{T} p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})}{q(\mathbf{x_T}|\mathbf{x_0})\prod_{t=2}^{T} q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})}\Big] \tag{39}$$

$$= \mathbb{E}_q\Big[- log\frac{p(\mathbf{x_T})}{q(\mathbf{x_T}|\mathbf{x_0})}\Big] - logp_\theta(\mathbf{x_0}|\mathbf{x_1}) + \sum_{t=2}^{T}\mathbb{E}_q\Big[- log\frac{p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})}{q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})}\Big] \tag{40}$$

$$= \mathbb{E}_q\Big[- log\frac{p(\mathbf{x_T})}{q(\mathbf{x_T}|\mathbf{x_0})}\Big] - logp_\theta(\mathbf{x_0}|\mathbf{x_1}) - \mathbb{E}_q\Big[\sum_{t=2}^{T}\mathbf{D_{KL}}\big[q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})||p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t}))\big]\Big] \tag{41}$$

We see that the first term is parameter free, because $p(\mathbf{x_T})$ is fixed and defined as a Gaussian, while $q(\mathbf{x_T}|\mathbf{x_0})$ is also Gaussian from the definition of the forward process. Hence, we are left with the second and third terms from L.

More specifically, we can conclude that maximizing ELBO (L) is equivalent to minimizing the KL-Divergence between $q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})$ and $p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})$

We know that both distributions are Gaussians, which makes computing the KL Divergence between them easier if we know their mean and variance. We will begin by calculating such moments for $q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})$.

$$q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0}) = \frac{q(\mathbf{x_t}|\mathbf{x_{t-1}},\mathbf{x_0})q(\mathbf{x_{t-1}}|\mathbf{x_0})}{q(\mathbf{x_t}|\mathbf{x_0})} \tag{42}$$

we know the q distribution from Definition 2.1, hence (43)

$q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})$ is a product of known Gaussians over another known Gaussian that lets us define (44)

$$\mu_q(\mathbf{x_t},\mathbf{x_0}) = \frac{(1-\bar{\alpha}_{t-1})\sqrt{\alpha_t}\mathbf{x_t} + (1-\alpha_t)\sqrt{\bar{\alpha}_{t-1}}\mathbf{x_0}}{(1-\bar{\alpha}_t)} \tag{45}$$

$$\Sigma_q(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{I} \tag{46}$$

$$\implies q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0}) = \mathcal{N}(\mathbf{x_{t-1}};\mu_q(\mathbf{x_t},\mathbf{x_0});\Sigma_q(t)) \tag{47}$$

We have just defined the $q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})$ distribution. Now let's move on to the $p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})$ distribution.

$$p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t}) = \mathcal{N}(\mathbf{x_{t-1}};\mu_\theta(\mathbf{x_t});\Sigma_\theta(t)) \tag{48}$$

the reverse process variance is defined as the ground truth variance of the forward process: (49)

$$\Sigma_\theta(t) = \Sigma_q(t) \tag{50}$$

we are only left with the distribution's mean μ_θ (51)

$$p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t}) = \mathcal{N}(\mathbf{x_{t-1}};\mu_\theta(\mathbf{x_t});\Sigma_q(t)) \tag{52}$$

Equation 50 makes it much easier to calculate $\mathbf{D_{KL}}\big[q(\mathbf{x_{t-1}}|\mathbf{x_t},\mathbf{x_0})||p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t}))\big]$:

Now we know we are trying to compute the KL Divergence between two Gaussians with the exact same variance.

For that, there is the following result that arives from the definition of such divergence

$$d_1(x) = \mathcal{N}(\mu_1, \sigma^2) \tag{53}$$

$$d_2(x) = \mathcal{N}(\mu_2, \sigma^2) \tag{54}$$

$$\text{The KL divergence } D_{KL}(d_1|d_2) \text{ is given by:} \tag{55}$$

$$D_{KL}(d_1|d_2) = \frac{(\mu_1 - \mu_2)^2}{2\sigma^2} \tag{56}$$

$$\text{Hence,} \tag{57}$$

$$\mathbf{D_{KL}}\left[q(\mathbf{x_{t-1}}|\mathbf{x_t}, \mathbf{x_0})||p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t}))\right] = \mathbf{D_{KL}}\big(\mathcal{N}(\mathbf{x_{t-1}}; \mu_q(\mathbf{x_t}, \mathbf{x_0}); \Sigma_q(t)), \mathcal{N}(\mathbf{x_{t-1}}; \mu_\theta(\mathbf{x_t}); \Sigma_q(t))\big) \tag{58}$$

$$= \frac{1 - \bar{\alpha}_t}{2(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}\big|\big|(\mu_q - \mu_\theta)_2^2\big|\big| \tag{59}$$

As our goal is to minimize the KL Divergence, from equation 59 we see that such goal comes down to basically minimizing the difference $\mu_q - \mu_\theta$, i.e.,

We just need to minimize the difference between the means of the reverse and forward processes' distributions. We need to define the reverse process' distribution mean (μ_θ) prediction by taking a look at the forward process' one (μ_q).

$$\mu_q(\mathbf{x_t}, \mathbf{x_0}) = \frac{(1 - \bar{\alpha}_{t-1})\sqrt{\alpha_t}\mathbf{x_t} + (1 - \alpha_t)\sqrt{\bar{\alpha}_{t-1}}\mathbf{x_0}}{(1 - \bar{\alpha}_t)} \tag{60}$$

$$\text{we can define the prediction} \tag{61}$$

$$\hat{\mu}_q(\mathbf{x_t}, \mathbf{x_0}) = \mu_\theta(\mathbf{x_t}) \tag{62}$$

$$= \frac{(1 - \bar{\alpha}_{t-1})\sqrt{\alpha_t}\mathbf{x_t} + (1 - \alpha_t)\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_\theta}{(1 - \bar{\alpha}_t)} \tag{63}$$

In equation 63 we see that we are using our reverse process model's prediction $\mathbf{x}_\theta$ in the prediction of its distribution's mean, which let's us rewrite $\big|\big|(\mu_\theta - \mu_q)_2^2\big|\big|$ in terms of $\mathbf{x_0}$ and $\mathbf{x}_\theta$ which leves us with the following for the KL Divergence:

$$\mathbf{D_{KL}}\big(\mathcal{N}(\mathbf{x_{t-1}}; \mu_\theta(\mathbf{x_t}); \Sigma_q(t)), \mathcal{N}(\mathbf{x_{t-1}}; \mu_q(\mathbf{x_t}, \mathbf{x_0}); \Sigma_q(t))\big) = \frac{(1 - \bar{\alpha}_t)(\bar{\alpha}_{t-1})}{2(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}\big|\big|(\mathbf{x}_\theta - \mathbf{x_0})_2^2\big|\big| \tag{64}$$

The author's of the DDPM paper mention that equation 64 can be used as the loss function to train the reverse process model. On the other hand, we now that the forward process actually predict the noise that was added to an input $\mathbf{x_t}$ intead of predicting $\mathbf{x}_\theta$ directly. This means that the loss function must account for the error prediction somehow. This is achieved by further analysing $\mathbf{x_0}$ and $\mathbf{x}_\theta$ and remembering how $\mathbf{x}_\theta$ was defined in the forward process.

$$q(x_t|x_0) := \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} \times x_0; (1 - \bar{\alpha}_t)\mathbb{I}) \tag{65}$$

$$\text{which let's us write} \tag{66}$$

$$\mathbf{x_t} = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon \tag{67}$$

$$\implies \mathbf{x_0} = \frac{\mathbf{x_t} - \sqrt{1 - \bar{\alpha}_t}\epsilon}{\sqrt{\bar{\alpha}_t}} \tag{68}$$

$$\text{for a Standard Gaussian Noise } \epsilon. \tag{69}$$

$$\text{We can now define our prediction } \hat{\epsilon} = \epsilon_\theta \tag{70}$$

$$\mathbf{x}_\theta = \frac{\mathbf{x_t} - \sqrt{1 - \bar{\alpha}_t}\epsilon_\theta}{\sqrt{\bar{\alpha}_t}} \tag{71}$$

$$\implies \frac{(1 - \bar{\alpha}_t)(\bar{\alpha}_{t-1})}{2(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}\big|\big|(\mathbf{x}_\theta - \mathbf{x_0})_2^2\big|\big| = \frac{(1 - \bar{\alpha}_t)(\bar{\alpha}_{t-1})}{2(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}\frac{(1 - \alpha_t)^2}{(1 - \bar{\alpha}_t)\alpha_t}\big|\big|(\epsilon_\theta - \epsilon)_2^2\big|\big| \tag{72}$$

Even though equation 72 could be used directly as the loss function for the noise predictor, the DDPM paper authors mention that optimizing $\left|\left|(\epsilon_\theta - \epsilon)_2^2\right|\right|$ without the scaling factor with the cumulative noise α_t is enough. Hence, we have finally defined a function to be minimized for the noise predictor training and hence write its algorithm.

Algorithm 1: Noise Predictor Training

1: **repeat**
2: $\mathbf{x_0} \sim \mathbf{q}(\mathbf{x_0})$ ▷ Sample image from training set
3: $\mathbf{x_0} \sim \mathbf{Uniform}(\{\mathbf{1}, \ldots, \mathbf{T}\})$ ▷ Sample the step of the Forward Process Markov Chain
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Sample standard gaussian noise to be added to the input
5: $\mathbf{x_t} = \sqrt{\bar{\alpha}_\mathbf{t}}\mathbf{x_0} + \sqrt{\mathbf{1} - \bar{\alpha}_\mathbf{t}}\epsilon$ ▷ Forward Process/ Generating Noisy Image
6: Take Gradient Descent Step on $\nabla_\theta(||\epsilon - \epsilon_\theta(\mathbf{x_t}, \mathbf{t})||)$
7: **until** converged

2.2.3 Sampling Algorithm Derivation

Now that we have defined a way to predict an input image $\mathbf{x_t}$'s noise, we need a way to use such prediction to reconstruct the original de-noised image $\mathbf{x_0}$, i.e., we need a way to sample from $p(\mathbf{x_0})$. To do so, let's recall how we have defined the $p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})$ distribution.

$$p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t}) = \mathcal{N}(\mathbf{x_{t-1}}; \mu_\theta(\mathbf{x_t}); \Sigma_q(t)) \tag{73}$$

$$\mu_\theta(\mathbf{x_t}) = \frac{(1 - \bar{\alpha}_{t-1})\sqrt{\alpha_t}\mathbf{x_t} + (1 - \alpha_t)\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_\theta}{(1 - \bar{\alpha}_t)} \tag{74}$$

$$\mathbf{x}_\theta = \frac{\mathbf{x_t} - \sqrt{1 - \bar{\alpha}_t}\epsilon_\theta}{\sqrt{\bar{\alpha}_t}} \tag{75}$$

$$\implies \mu_\theta(\mathbf{x_t}) = \frac{\mathbf{x_t}}{\sqrt{\alpha_t}} - \frac{(1 - \alpha_t)(\sqrt{1 - \bar{\alpha}_t})}{(1 - \bar{\alpha}_t)(\sqrt{\alpha_t})}\epsilon_\theta = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x_t} - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_\theta\right) \tag{76}$$

$$\Sigma_\theta(t) = \Sigma_q(t) = \frac{(1 - \alpha_t)(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{I} \tag{77}$$

We now have defined $\mathbf{x_{t-1}}$'s mean and variance given $\mathbf{x_t}$ which let's us write it as (78)

$$\mathbf{x_{t-1}} = \mu_\theta(\mathbf{x_t}) + \sqrt{\Sigma_\theta(t)}\mathbf{z} \tag{79}$$

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{80}$$

We have now a way to generate $\mathbf{x_{t-1}}$ from $\mathbf{x_t}$. This means that if we have $\mathbf{x_1}$ we can generate $\mathbf{x_0}$. This let's us finally define our sampling algorithm:

Algorithm 2: Sampling

1: **repeat**
2: $\mathbf{x_T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Sample random noisy image
3: $\mathbf{T} \sim \mathbf{Uniform}(\{\mathbf{1}, \ldots, \mathbf{1000}\})$ ▷ Sample random length of the Denoising Chain. Max chain size of 1000 was set arbitrarily
4: **for** $\mathbf{t} = \mathbf{T}, \ldots, \mathbf{1}$ **do**
5: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$ else $\mathbf{z} = \mathbf{0}$
6: $\mathbf{x_{t-1}} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x_t} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x_t}, \mathbf{t})\right) + \sqrt{\Sigma_\theta(t)}\mathbf{z}$ ▷ Sampling $\mathbf{x_{t-1}}$ from $p_\theta(\mathbf{x_{t-1}}|\mathbf{x_t})$
7: **end for**
8: **return** $\mathbf{x_0}$

3 Conditional DDPMs - A text-to-image Generative Model

In a standard Denoising Diffusion Probabilistic Model (DDPM), the neural network $\epsilon_\theta(x_t, t)$ receives a noisy sample x_t and a timestep t and learns to predict the noise ϵ that produced x_t from the clean image x_0. A *conditional* DDPM extends this parametrization to

$$\epsilon_\theta(x_t, t, c),$$

where c is some conditioning signal, in our case it is the class label turned into a text embedding. It is such that the denoising process is guided by it.

3.1 FiLM Conditioning

Feature-wise Linear Modulation (FiLM) [3] introduces conditioning by modulating the intermediate feature maps of the noise prediction (in our case it is implemented as a U-Net) using an affine transformation derived from the conditioning embedding. Given an intermediate activation h and a conditioning vector c, a FiLM layer produces channel-wise scale and shift parameters $\gamma(c)$ and $\beta(c)$, and applies the transformation:

$$\text{FiLM}(h \mid c) = \gamma(c) \odot h + \beta(c),$$

where $\odot$ denotes channel-wise multiplication. This modulation is applied after a the normalization layer, which in our case it is as GroupNorm (GN):

$$h' = \gamma(c) \odot \text{GN}(h) + \beta(c).$$

FiLM allows the conditioning signal to influence the forward and reverse processes *at every layer* of the U-Net, making the model significantly more expressive than simple concatenation of inputs or adding the embedding only at the input level. While the diffusion process itself (the forward noising and reverse denoising equations) does not change, the neural network that parameterizes the reverse distribution becomes condition-dependent through FiLM.

3.2 Effect on Training

The DDPM training objective remains the "simple" noise prediction loss:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0,t,\epsilon,c}\left[\|\epsilon - \epsilon_\theta(x_t, t, c)\|^2\right],$$

but the U-Net now incorporates FiLM in each residual block so that the predicted noise depends on the conditioning. The rest of the DDPM training algorithm is unchanged.

3.3 Effect on Sampling

During sampling, the reverse process uses the conditional model $\epsilon_\theta(x_t, t, c)$ inside each transition step. The DDPM equations for sampling are unaffected; only the network that predicts the mean of the reverse distribution depends on c.

3.4 Algorithms

Algorithm 3: FiLM-Conditional DDPM Training

1: **repeat**
2: $\mathbf{x_0} \sim \mathbf{q}(\mathbf{x_0})$ ▷ Sample clean image
3: $\mathbf{c} \sim \mathbf{q}(\mathbf{c}|\mathbf{x_0})$ ▷ Sample conditioning signal
4: $\mathbf{t} \sim \mathbf{Uniform}(\{\mathbf{1}, \ldots, \mathbf{T}\})$ ▷ Sample timestep
5: $\boldsymbol{\epsilon} \sim \boldsymbol{\mathcal{N}}(\mathbf{0}, \boldsymbol{I})$ ▷ Sample Gaussian noise
6: $\mathbf{x_t} = \sqrt{\bar{\alpha}_\mathbf{t}}\, \mathbf{x_0} + \sqrt{\mathbf{1} - \bar{\alpha}_\mathbf{t}}\, \epsilon$ ▷ Forward (noising) process
7: Take gradient step on

$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \epsilon_\theta(x_t, t, c) \right\|^2,$$

where ϵ_θ is a U-Net whose residual blocks apply FiLM modulation:

$$\mathrm{GN}(h) \xrightarrow{\mathrm{FiLM}(c)} \gamma(c) \odot \mathrm{GN}(h) + \beta(c).$$

8: **until** converged

Algorithm 4: FiLM-Conditional DDPM Sampling

1: $\mathbf{c}$ given ▷ Condition for generation
2: $\mathbf{x_T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ ▷ Initial noise sample
3: **for** $\mathbf{t} = \mathbf{T}, \ldots, \mathbf{1}$ **do**
4: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ **if** $t > 1$ **else** $\mathbf{z} = \mathbf{0}$
5: $\mathbf{x_{t-1}} = \frac{\mathbf{1}}{\sqrt{\alpha_\mathbf{t}}} \left(\mathbf{x_t} - \frac{\mathbf{1}-\alpha_\mathbf{t}}{\sqrt{\mathbf{1}-\bar{\alpha}_\mathbf{t}}}\, \epsilon_\theta(\mathbf{x_t}, \mathbf{t}, \mathbf{c}) \right) + \sqrt{\boldsymbol{\Sigma}_\theta(\mathbf{t})}\mathbf{z}$
6: **end for**
7: **return** $\mathbf{x_0}$

3.5 Results of the PyTorch Implementation of Conditional DDPM

The original DDPM implementation in [1] defines the noise prediction model as a U-Net with generic residual blocks. In order to turn this implementation into a *text-to-image* model, we change its residual layer to a FiLM residual layer.

In the notebook attatched to this work, there are more details about the implementation. Essentially, what was done was substituting the **ResBlock** class from [1] with the **FiLMResBlock** class defined in the section 4 of the notebook. Also, the training and sampling algorithms were modified to reproduced the algorithms defined in the previous section (3.4).

Below there are three samples of the images generated by the conditional DDPM.

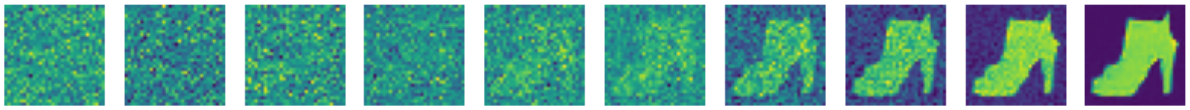

Figure 2: Heel generated by the text-to-image DDPM

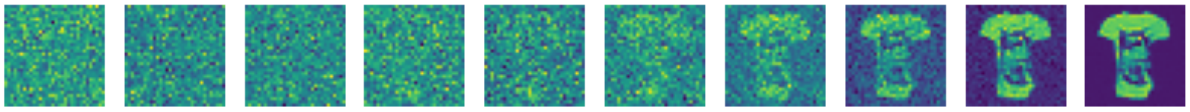

Figure 3: T-shirt generated by the text-to-image DDPM

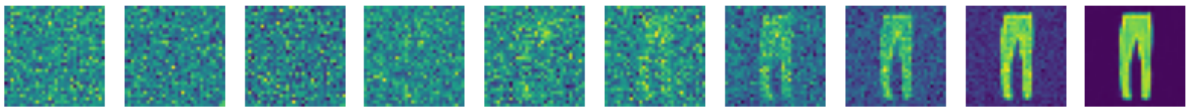

Figure 4: Pants generated by the text-to-image DDPM

4 Self evaluation

I did a deep study I did on the DDPM paper [2] and the resources that explain its Evidence Lower Bound, which resulted in my presentation of this work's first section to the Deep Learning class' students this semester along with my own implementation of [1].

Motivated by that presentation, I was not satisfied with simply reproducing [1]. I wanted to go further and implement a text-to-image generative model. This way, I studied about FiLM with [3] and was able to modify [1] to become a text-to-image generative model, which I consider one of my greatest achievements in deep learning up until today, given that it was an original idea.

This way, even though I think there is still room to improve the present work so that the trained text-to-image generated more diverse images from each class, I believe this work is a success.

References

[1] Nicholas DiSalvo. Diffusion model from scratch in pytorch (ddpm). `https://towardsdatascience.com/diffusion-model-from-scratch-in-pytorch-ddpm-9d9760528946/`, Jul 2024. Towards Data Science article, accessed 2025-12-11.

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020.

[3] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer, 2017.

[4] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015.

[5] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[6] Jake Tae. A step up with variational autoencoders. `https://jaketae.github.io/study/vae/`, 2019. Accessed: 2025-12-11.

[7] Jake Tae. From elbo to ddpm. `https://jaketae.github.io/study/elbo/`, 2021. Accessed: 2025-12-11.

[8] YouTube. Neural networks [7.8] : Deep learning – variational bound. `https://www.youtube.com/watch?v=pStDscJh2Wo`, n.d. Accessed: 2025-12-11.