Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Masterarbeit

Deep-learning: diffusion models for image construction.

Luis Enrique Martinez Rojas

luis.martinez.rojas@studium.uni-hamburg.de
Studiengang Mathematical Modelling in Engineering
Matr.-Nr. 7587418

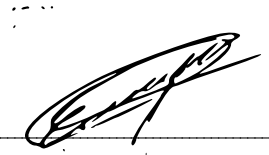
Erstgutachter: Dr. Ivan Yaroslavtsev
Zweitgutachter: Professor Dr. Thomas Schmidt

Abgabe: 10.2023

The worthwhile problems are the ones you can really solve or help solve, the ones you can really contribute something to.
– *Richard P. Feynman*

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.
Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den 24.10.2023 Unterschrift: 

Acknowledgments

I extend my heartfelt gratitude to Dr. Ivan Yaroslavtsev and Dr. Rüdiger Brecht for their invaluable contributions to the code and ongoing support. Their unwavering support has made this work possible. I am especially grateful to Ivan for his invaluable guidance and the generous time he has devoted to me. With their support, this journey has been even more enriching.

To my dear Mathmods friends, your encouragement has been a constant source of motivation. Thank you.

My loving family, your unwavering support mean the world to me.

A special thanks to the coordinators, Prof. Dr. Gasser and Astrid Benz, for their comprehensive support. Thank you for your assistance in making this journey possible.

Contents

1 Introduction

Diffusion models are a class of generative models used in machine learning and statistics to model complex data distributions. These models are particularly effective at generating realistic samples and performing tasks such as image denoising.

In a diffusion model, the data generation process involves two main steps: the forward diffusion process and the reverse diffusion process.

The forward diffusion process models the generation of data from a simple distribution to a more complex distribution. It starts with a simple initial distribution, often a known noise distribution such as Gaussian or uniform. Then, it gradually transforms the data by applying a series of diffusion steps. At each step, the data is updated based on a diffusion equation or a stochastic differential equation (SDE). This equation incorporates both the previous state of the data and a source of randomness. The diffusion process is designed in a way that the data becomes more complex and closer to the target distribution at each step. By applying a sufficient number of diffusion steps, the data distribution is expected to converge to the desired target distribution.

The reverse diffusion process, also known as the generative process or the sampling process, is the counterpart of the forward diffusion process. It aims to generate new samples by starting from a complex distribution and gradually transforming them back to the initial simple distribution. This process involves reversing the steps of the forward diffusion process. At each step, instead of updating the data, the reverse diffusion process reconstructs the data using a carefully designed update rule. The reverse diffusion process is crucial for generating new samples from the learned diffusion model. By conditioning the model on a target sample from the desired distribution, the reverse diffusion process allows us to generate new samples by iteratively applying the reverse updates.

Together, the forward and reverse diffusion processes form a cycle that allows us to model complex data distributions and perform various tasks such as denoising. During training, the model learns the parameters that control the diffusion steps and updates rules, optimizing them to match the target distribution.

Denoising diffusion probabilistic models leverage these forward and reverse diffusion processes to model the underlying data distribution and remove noise from observed data.

It's worth noting that diffusion models have gained significant attention in recent years, especially with the development of denoising diffusion probabilistic models and their successful applications in image generation and manipulation tasks. [den22]

1.1 Motivation

In the realm of computer vision and image processing, image reconstruction plays a fundamental role in various applications, ranging from medical imaging to surveillance systems and digital photography. The ability to restore high-quality images from noisy or degraded versions is a significant challenge that researchers and practitioners continue to tackle. This thesis finds the motivation in the possibility of reconstructing images following 'Denoising Diffusion Probabilistic Models' [HJA20].

Advancements in Denoising Techniques. The paper introduces an innovative approach to image denoising through the application of diffusion probabilistic models. By studying this paper, researchers can gain valuable insights into cutting-edge denoising techniques that go beyond traditional methods. The integration of deep learning and variational inference allows for the development of powerful models capable of effectively reducing noise and preserving important image details.

Improved Image Quality. One of the primary motivations for studying the paper is the pursuit of enhanced image quality. Image reconstruction using denoising diffusion probabilistic models presents an opportunity to restore images to a higher level of fidelity. By understanding the principles and techniques presented in the paper, researchers can acquire the necessary knowledge to apply advanced denoising algorithms and overcome the limitations of conventional approaches. This can significantly improve image reconstruction tasks, resulting in visually appealing and more accurate representations of the original scene.

Noise Reduction and Artifact Removal In addition to reducing noise, the paper's focus on denoising diffusion probabilistic models also addresses the removal of artifacts that often accompany noisy images. Artifacts can stem from various sources such as compression, sensor limitations, or image acquisition processes. By studying the paper, researchers gain access to novel methods and strategies for effectively reducing artifacts and preserving the reconstructed images' essential structural and textural features. The resulting images are visually appealing and provide more reliable and accurate information for subsequent analysis or interpretation.

Real-World Applications The potential applications of image reconstruction using denoising diffusion probabilistic models are vast and impactful. Medical imaging, for instance, can greatly benefit from improved image reconstruction techniques. By studying the paper, researchers can explore how the proposed methods can enhance the clarity and accuracy of medical images, leading to more accurate diagnoses and treatment planning. Similarly, surveillance systems can benefit from advanced denoising techniques to improve the visual quality of captured images or videos, thereby enabling more effective security measures and reliable event analysis.

Further Research and Innovation The paper serves as a foundation for further research and innovation in the field of image reconstruction. Researchers can identify potential avenues for improvement and extension by studying the presented techniques and

methodologies. This may include exploring variations of the denoising diffusion probabilistic models, incorporating additional priors or constraints, or adapting the methods to specific domains or imaging modalities. Through continued study and exploration, researchers can contribute to the advancement of image reconstruction techniques, pushing the boundaries of what is currently possible.

1.2 Objectives

The primary objectives of this thesis are as follows:

Theoretical Investigation: To provide a thorough theoretical analysis of diffusion models, specifically focusing on the forward and reverse processes. This investigation will explore the underlying mathematics, probabilistic concepts, and statistical principles that govern these processes, shedding light on their relationship to the overall model architecture.

Practical Implementation: To delve into practical implementation aspects of diffusion models, including the design of diffusion equations, stochastic differential equations, and update rules for the forward and reverse processes. This objective aims to equip researchers and practitioners with the necessary knowledge and tools to effectively apply diffusion models in various domains.

Performance Evaluation: To assess the performance of diffusion models in terms of generative capabilities, data reconstruction accuracy, and other relevant metrics.

2 Preliminaries

2.1 Probability theory and entropy

A measurable space $(\Omega, \mathcal{B})$ of state space Ω with a σ-field $\mathcal{B}$ of subsets of Ω. A σ-field or σ-algebra $\mathcal{B}$ is a nonempty collection of subsets of Ω with the following properties:

- $\Omega \in \mathcal{B}$.
- if $A \in \mathcal{B}$ then its complement $A^c \in \mathcal{B}$.
- if $A_n \in \mathcal{B}$, for $n = 1, 2, \cdots$ then $\bigcup_n A_n \in \mathcal{B}$.

An element A of Ω is called a *sample*. Elements of $\mathcal{B}$ are called *events*. A map given by $\mathbb{P} : \mathcal{B} \longrightarrow [0, 1]$ is called a measure if full fill

- *Non Negativity:* $\mathbb{P}(A) \geq 0$ for any $A \in \mathcal{B}$.
- *Countable Additivity:* if A_n is disjoint then $\mathbb{P}\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mathbb{P}\left(A_n\right)$.
- *Normalization:* $\mathbb{P}\left(\Omega\right) = 1$

If these three conditions are filled, then $(\Omega, \mathcal{B}, \mathbb{P})$ is called probability space.

Definition 1. *The probability that a random variable $X \in \Omega$ takes values in a finite interval (a, b) is given by the integral of a function called the* probability density function $p(x_{0:T}) : \mathbb{R}^{(T+1)d} \longrightarrow \mathbb{R}$ *[Blo04].*

$$\mathbb{P}\left(a \leq X \leq b\right) = \int_a^b p(x)dx \tag{2.1}$$

Satisfies the following conditions:

- $p(x_{0:T}) \geq 0 \qquad \forall x_i \in \mathbb{R}^d$
- $\int_{\mathbb{R}^{(T+1)d}} p(x_{0:T})dx_{0:T} = 1$
- $\mathbb{P}\left((x_t, x_s) \in A\right) = \int_A p(x_t, x_s) \qquad A \in \mathbb{R}^{2(d+1)}$

A crucial concept in this thesis is the idea of latent variables. These are unobservable variables that represent the hidden patterns or characteristics within the data. In the context of denoising diffusion models (DDMs), these latent variables are of paramount

importance. They play a central role in modeling and understanding how data is generated in cleaning noisy data and generating pristine information from observations.

Let's denote these latent variables as $x_{(0,\dots,T)}$, where $x_i \in \mathbb{R}^d$ for all i in [0, T]. The core concept is that we can effectively remove or 'integrate out' these latent variables using a specific mathematical operation, as expressed by the following equation:

$$p(x_0, x_2, \dots, x_T) := \int_{\mathbb{R}^{T\cdot d}} p(x_1)\, dx_1. \tag{2.2}$$

In simpler terms, this equation allows us to summarize or account for the influence of these hidden variables on our data, which is a key step in understanding and working with denoising diffusion models. It's a mathematical technique central to unraveling the underlying structures within the data and making it more manageable for analysis and modeling.
Then from Eq. (2.2) doing the same process $n-1$ times left as a result:

$$p(x_0, x_1) = \int_{\mathbb{R}^{(T-1)d}} p(x_0, x_1, ..., x_T) dx_2 dx_3 \cdots dx_T \tag{2.3}$$

From now in advance we set the notation $x_0, x_1, \cdots,$

Definition 2 ([Shi01])**.** *Let $\mu \in \mathbb{R}^d$ is the location and $\Sigma \in \mathbb{R}^{d\times d}$ is the covariance (positive semi-definite matrix). The probability density function of a* Multivariate normal distribution *(Gaussian) is given by :*

$$p(x) = \frac{1}{|\Sigma|^{\frac{1}{2}}\sqrt{2\pi}} \exp\left[-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right], \quad x \in \mathbb{R}^d. \tag{2.4}$$

The corresponding distribution is denoted by $\mathcal{N}(\mu, \Sigma)$.

Let $\mu \in \mathbb{R}$ is the location and $\sigma > 0$. The probability density function of a (one-dimensional) normal distribution *is given by :*

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right], \quad x \in R. \tag{2.5}$$

The corresponding distribution is denoted by $\mathcal{N}(\mu, \sigma^2)$.

Definition 3. *For an event $A \in \Omega$ given another $B \in \Omega$ with $\mathbb{P}(B) \neq 0$ we define the* conditional probability *by:*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} = \frac{\mathbb{P}(A, B)}{\mathbb{P}(B)} \tag{2.6}$$

Definition 4. *Let $A, B \in \mathcal{B}$ be two events with Eq.* (2.6)*, using that $\mathbb{P}(A \cap B) = \mathbb{P}(B \cap A)$ then we have the* Bayes' rule*:*

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)} \tag{2.7}$$

Definition 5. *For two events A and B in $\mathcal{B}$ the* chain rule *states that*

$$\mathbb{P}(A \cap B) = \mathbb{P}(B|A)\,\mathbb{P}(A) \tag{2.8}$$

and for random real variables

$$p(x_n, x_{n-1}) = p(x_{n-1}|x_n)\,p(x_n) \tag{2.9}$$

Definition 6. *A* stochastic process *is a collection of random variables $X_t : t \in T$ parameterized by a set T, called parametric spaces, and with values in a set S called state space. [Rin12]*

Let random variables $\Omega = \{x_o, x_1, ..., x_i, ..., x_T : x_i \in \mathbb{R}^d \;\; \forall 0 \leq i \leq T\}$.

Definition 7. *The* Markov property *can be expressed as follows: for any states $x_0, x_1, ..., x_{n-1}$ (past),x_n (present),x_{n+1}(future), the equality is filled:*

$$p(x_{n+1}|x_0, ..., x_n) = p(x_{n+1}|x_n) \tag{2.10}$$

this means that the probability of the event (x_n) only depends on the previous event (x_{n-1}), while the previous event $x_0, x_1, ..x_{n-2}$ are irrelevant.
The diffusing models are based on the use of Markov chains due to the diffusion process depending only on the last step, and the previous of this is irrelevant; we define:

Definition 8. *A* Markov Chain *is a stochastic process on discrete time $\{x_n : n = 0, 1, ...\}$, with discrete state space, and full fill the Markov property, for any integer $n \geq 0$, and any states $x_0, ..., x_{n+1}$, then[Rin12]*

$$p(x_{n+1}|x_0, ..., x_n) = p(x_{n+1}|x_n) \tag{2.11}$$

Let's consider the time $n+1$ as the future, n as the present and the times $0, 1, ..., n-1$ as the past of events, then the condition Eq. (2.11) could be interpreted as the process at the future time $n+1$ depend only of process in the present n, not on the past processes $0, 1, ..., n-1$.
Let's consider a discrete-time stochastic process $\{X_n\}$ that fulfills the Markov property. Then, to describe this property and equivalent conditions, the probability density function will be written as $p(x_n)$, i.e., the sub-index indicates the variable. Then the definition of the conditional probability Eq. (2.6) is giving by:

$$p(x_{n+1}|x_n) = \frac{p(x_{n+1}, x_n)}{p(x_n)} \tag{2.12}$$

In both the forward and reverse processes of the model, we come across transition probabilities at each time step. It proves beneficial to express the total probability in a

manner that is contingent on the conditional probabilities at each step. This approach allows us to break down the overall probability into a series of conditional probabilities, making the analysis and understanding of the process more tractable. Then using Eq. (2.12) it follows the recurrence:

$$\begin{aligned} p(x_1, x_0) &= p(x_1|x_0)p(x_0) \\ p(x_2, (x_1, x_0)) &= p(x_2|(x_1, x_0))p(x_1, x_0) = p(x_2|x_1)p(x_1|x_0)p(x_0) \\ &\vdots \\ p(x_n, ..., x_1, x_0) &= p(x_0)p(x_1|x_2)\cdots p(x_n|x_{n-1}) \end{aligned}$$

where we have used the Markov property, then the *joint distribution* $p : \mathbb{R}^{(T+1)d} \longrightarrow \mathbb{R}^d$ of the variables $x_0, x_1, ..., x_n$ can be written as:

$$p(x_0, x_1, ..., x_n) = p(x_0)\prod_{i=1}^{n} p(x_i|x_{i-1}) \tag{2.13}$$

This form is highly advantageous in the context of probabilistic models and serves as a cornerstone for their development.
The term following the product symbol in Eq. (2.13) represents a transition or conditional probability that relies solely on the immediate past, a concept known as the Markov property. To elucidate this transition term, we will introduce the concept of a kernel transition. Transition kernels are associated with Markov processes, stochastic processes where future states depend only on the current state and are independent of past states. In a Markov process, the transition kernel defines the probabilities of moving from the current state to various possible future states.

Definition 9. *Let S be a set and $\mathcal{S}$ a σ-field on S. The set S is called the state space.* A transition kernel K *is a function from $S \times \mathcal{S} \longrightarrow [0, 1]$ such that [Ken16]*

- *For all $x \in S$, $K(x, \cdot)$ is a probability measure on $(S, \mathcal{S})$.*
- *For all $A \in \mathcal{S}$, $K(\cdot, A)$ is a measurable function on S.*

In order to have an explicit expression to develop the model [RC04]. A σ-finite measure π is called *invariant* for a Markov chain with transition kernel $K(\cdot, \cdot)$ such that:

$$\pi(B) = \int_S K(x, B)\pi(x)dx, \qquad \forall B \in \mathcal{S} \tag{2.14}$$

Definition 10. *The expectation of a real random variable $\{X\}$ is defined as Lebesgue integral by its law μ; $\mathbb{E}[X] = \int x\mu(dx)$ if the integral is well defined its equal to the Lebesgue integral of X by the measure $\mathbb{P}$ [GS18]:*

$$\mathbb{E}[X] = \int_\Omega X(\omega)\mathbb{P}(d\omega) \tag{2.15}$$

If $\mathbb{E}[X]$ is finite, X is said to be an integrable random variable.

Lemma 2.1.1. *(*Jensen's Inequality*). If f is a convex function on (a, b) and X is a random variable taking values in (a, b) , then: [Rao11]*

$$f\left(\mathbb{E}[X]\right) \leq \mathbb{E}[f(X)] \tag{2.16}$$

In DDMs, entropy is used to measure the uncertainty in the data distribution. The main idea in the diffusion models is to decrease the uncertainty (entropy) with the goal of preserving the key information. As you progress through the diffusion process (generating data from noisy to clean), entropy tends to decrease. The goal is to model this process effectively, capturing the data distribution at different stages and, ideally, preserving the key information while reducing uncertainty or entropy.

Definition 11. *The* entropy *of a discrete random variable X on the probability space $(\Omega, \mathcal{B}, \mathbb{P})$ is defined by: [Gra90]*

$$H(X) = -\sum_{i=1}^{T} \mathbb{P}(X = x) \log \mathbb{P}(X = x) \tag{2.17}$$

Let's define a notation for a $x_i \in \mathbb{R}^d$ the probability distribution $p(x_i) = p_i$ with d the dimension. In the following lemma, we will work with a finite probability distribution $\{p_1, p_2, \cdots, p_T\}$ i.e. a set of nonnegative numbers that satisfies $p_1 + p_2 + \cdots + p_T = 1$.

Lemma 2.1.2. *Given two probability distribution$\{p_i\}$ and $\{q_i\}$, countable i.e. a sequence of nonnegative numbers that the sum is one, then*

$$\sum_i p_i \ln \frac{p_i}{q_i} \geq 0 \tag{2.18}$$

and the equality when $p_i = q_i \forall i$.

Proof. As $\ln(x)$ is a concave function, then for a real number, the inequality holds:

$$\ln(x) \leq x - 1$$

Then it follows

$$-\sum_i p_i \ln \frac{p_i}{q_i} = \sum_i p_i \ln \frac{q_i}{p_i} \leq \sum_i p_i \left(\frac{q_i}{p_i} - 1\right) = \sum_i q_i - \sum_i p_i = 1 - 1 = 0$$

By using Lemma 2.1.1 as ln is a convex function then

$$\sum_i p_i \ln \frac{q_i}{p_i} \leq \ln \left(\sum_i p_i \frac{p_i}{q_i}\right)$$

□

Definition 12. *Given a probability space $(\Omega, \mathbb{B}, P)$, with Ω a finite space, and another measure Q on the same space, we define the divergence of P with respect to Q as the relative entropy of the*

identity mapping with respect to the two measures: [WGBS23]

$$D_{KL}\left(P||Q\right) := \sum_{\omega\in\Omega} P(\omega)\ln\frac{P(\omega)}{Q(\omega)} \tag{2.19}$$

Measures P, Q on $\mathbb{R}^d$ with densities p and q, we have:

$$D_{KL}\left(P||Q\right) = \int_{\mathbb{R}^d} p(x)\log\frac{p(x)}{q(x)}dx$$

Theorem 2.1.3. *Given two probability measure P and Q on a common finite probability space, then*

$$D_{KL}\left(P||Q\right) \geq 0 \tag{2.20}$$

with equality if and only if $P = Q$. [Gra90]

From Definitions 10 and 12 we have:

$$\mathop{\mathbb{E}}_{x\sim p}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] = \int_{\mathbb{R}^d} p(x)\log\frac{p(x)}{q(x)}dx = D_{KL}\left(P(X)||Q(X)\right) \tag{2.21}$$

Utilizing Definition 12 to compute the Kullback-Leibler (KL) divergence between two Gaussian distributions as defined in Eq. (2.4) and incorporating Eq. (2.21), we can derive the following result: [[ZLC+21]]

$$\begin{aligned}
D_{KL}\left(\mathcal{N}\left(\mu_1,\Sigma_1\right)||\mathcal{N}\left(\mu_2,\Sigma_2\right)\right) &= \mathbb{E}_{x\sim p}\left[\log\frac{\mathcal{N}\left(\mu_1,\Sigma_1\right)}{\mathcal{N}\left(\mu_2,\Sigma_2\right)}\right] && (2.22)\\
&= \mathbb{E}_{x\sim p}\left[\log\left(\mathcal{N}\left(\mu_1,\Sigma_1\right)\right) - \log\left(\mathcal{N}\left(\mu_2,\Sigma_2\right)\right)\right] && (2.23)\\
&= \mathbb{E}_{x\sim p}\left[\log\frac{1}{|\Sigma_p|^{\frac{1}{2}}\sqrt{2\pi}} - \frac{1}{2}\left(x-\mu_p\right)^\top\Sigma_p^{-1}\left(x-\mu_p\right)\right. && (2.24)\\
&\quad \left. - \log\frac{1}{|\Sigma_q|^{\frac{1}{2}}\sqrt{2\pi}} + \frac{1}{2}\left(x-\mu_q\right)^\top\Sigma_q^{-1}\left(x-\mu_q\right)\right]\\
&= \mathbb{E}_{x\sim p}\left[\frac{1}{2}\log\frac{|\Sigma_q|}{|\Sigma_p|} - \frac{1}{2}\left(x-\mu_p\right)^\top\Sigma_p^{-1}\left(x-\mu_p\right)\right. && (2.25)\\
&\quad \left. + \frac{1}{2}\left(x-\mu_q\right)^\top\Sigma_q^{-1}\left(x-\mu_q\right)\right]
\end{aligned}$$

In order to address the second and third terms in Eq. (2.25), we invoke specific algebraic properties:

1. Let $M \in \mathbb{R}^{d\times d}$ be a matrix, then we have for $x \in \mathbb{R}^d$ that $x^\top Mx = \mathrm{tr}\{xx^\top M\}$.

2. Let ℓ be a linear mapping then $\mathbb{E}[\ell(\mathbf{x})] = \ell(\mathbb{E}[\mathbf{x}])$.

3. $\mathbb{E}[(\mathbf{x}-\mu_p)(\mathbf{x}-\mu_p)^T] = \Sigma_p$

Using these properties we can rewrite the second term as:

$$\mathbb{E}\left[(\mathbf{x}-\mu_p)^\top \Sigma_p^{-1}(\mathbf{x}-\mu_p)\right] = \mathbb{E}\left[\mathrm{Tr}\left((\mathbf{x}-\mu_p)(\mathbf{x}-\mu_1)^\top \Sigma_p^{-1}\right)\right] \tag{2.26}$$

$$= \mathrm{Tr}\left(\mathbb{E}\left[(\mathbf{x}-\mu_p)(\mathbf{x}-\mu_p)^\top\right]\Sigma_p^{-1}\right) \tag{2.27}$$

$$= \mathrm{Tr}\left(\Sigma_p \Sigma_p^{-1}\right) \tag{2.28}$$

$$= d \tag{2.29}$$

where d is the dimension.
For the third term it is useful the Eq. (380) in [PP] :

$$\left(x-\mu_q\right)^\top \Sigma_q^{-1}\left(x-\mu_q\right) = (\mu_q-\mu_p)^T \boldsymbol{\Sigma_q^{-1}} (\mu_q-\mu_p) + \mathrm{Tr}(\Sigma_p^{-1}\Sigma_q) \tag{2.30}$$

Then using Eqs. (2.29) and (2.30) in Eq. (2.25) we have:

$$D_{KL}\left(\mathcal{N}\left(\mu_1,\Sigma_1\right)||\mathcal{N}\left(\mu_2,\Sigma_2\right)\right)$$
$$= \frac{1}{2}\left(\log\frac{|\Sigma_2|}{|\Sigma_1|} + Tr\left(\Sigma_2^{-1}\Sigma_1\right) + \left(\mu_2-\mu_1\right)^{tr}\Sigma_2^{-1}\left(\mu_2-\mu_1\right) - d\right) \tag{2.31}$$

2.2 Forward Process

The forward trajectory in diffusion models serves as the initial step in the training process of a neural network. This essential phase involves the intentional introduction of noise to the data. The general objective is twofold: first, to acquire an understanding of how to effectively denoise data, and second, to gain the capability to generate novel data in the subsequent reverse process.

During this forward trajectory, the network encounters data that is intentionally corrupted by noise, simulating real-world scenarios where data can be imperfect or noisy. The network's task is to learn to distinguish between the informative signal within the data and the undesirable noise. By undergoing this training phase, the network becomes equipped with the knowledge and mechanisms required to denoise data effectively.

Once this denoising proficiency is established in the forward process, it can be harnessed in the reverse process. The reverse process involves generating entirely new and meaningful data. It is in this phase that the neural network leverages its learned denoising capabilities to craft data that adheres to the desired patterns and structures, essentially creating novel, high-quality data.

The forward trajectory begins with an initial state x^0, situated in the d-dimensional real space $\mathbb{R}^d$. This state represents the starting point of the data distribution, where $q(x^0) \sim x^0$ characterizes the raw data. Over the course of this trajectory, we proceed through a sequence of T diffusion steps. Each stage of the evolving distribution is denoted as $q(x^i)$. Referring to Eq. (2.13), we can formulate the total distribution $q : \mathbb{R}^{(T+1)d} \longrightarrow \mathbb{R}^d$ as

follows:

$$q(x_0, x_1, ..., x_T) = q(x_0) \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.32}$$

then using Eq. (2.12) the conditional probability given as:

$$q(x_0, x_1, ..., x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.33}$$

In the illustration Fig. 2.1, at step T, the data distribution $q(x_T)$ takes on a specific form, where it closely resembles pure Gaussian noise. In other words, it resembles a noise pattern characterized by $\mathcal{N}(x_T; \mathbf{0}, \boldsymbol{I})$. This means that the data distribution primarily consists of uncorrelated and normally distributed noise at this stage. We employ a prac-

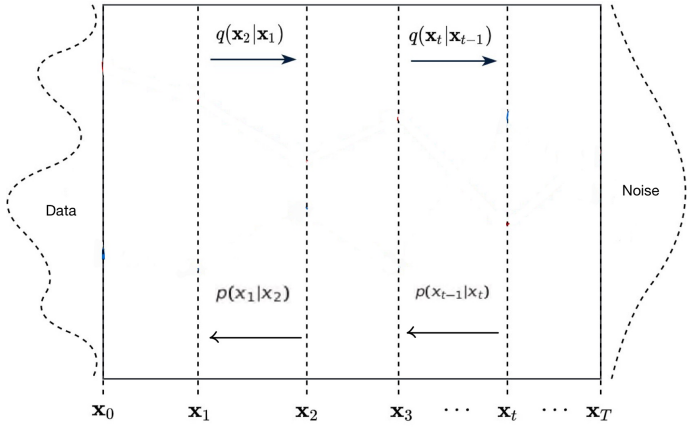


Figure 2.1: The curve on the left hand side represents the data (original image), and the right-hand side the pure noise

tical strategy to transform the probability distribution into a more well-behaved form π. This approach allows the tractability of training in our neural network. As it involves the iterative application of a Markov transition kernel denoted as $K(\cdot, \cdot; \beta)$.

$$\pi(x) = \int K(x, \overline{x}; \beta)\pi(\overline{x})d\overline{x} \tag{2.34}$$

Then we assume the conditional probability as:

$$q(x_t|x_{t-1}) := K(x_t, x_{t-1}; \beta_t) \sim \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \boldsymbol{I}) \tag{2.35}$$

The parameter β_t represents the covariance, and to streamline the process and reduce complexity, it is not treated as a trainable parameter. Instead, it remains fixed or 'frozen' during training, as elaborated in [KW22]. The values of this parameter may vary according to predefined schedules $[\beta_0, \beta_1, \cdots, \beta_T]$.
As demonstrated in [KW22], we can employ a clever technique known as the re-parameterization trick.:

$$q(x_t|x_{t-1}) = x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1} \tag{2.36}$$

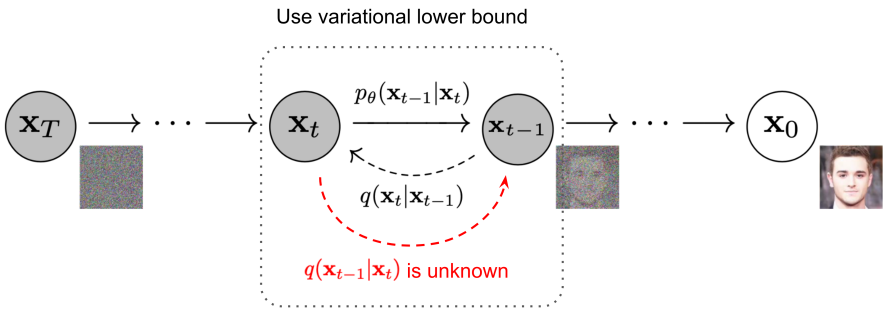


Figure 2.2: Illustration of the Markov chain in the forward and reverse process (taken from[HJA20]).

Then, using Eqs. (2.35) and (2.36), we have:

$$\mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \boldsymbol{I}) \sim \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1} \tag{2.37}$$

In Eq. (2.37) the final point x_t is given by x_{t-1} re-scaled by the mean value, and adding a Gaussian distribution $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ re-scaled by the covariance. In the study presented in [HJA20], a significant insight emerged. Rather than computationally executing the forward process step by step at each time increment, a transformation in variables facilitated the ability to encapsulate all the steps from time 0 to time 't' into a single, straightforward equation.
In order to achieve it, let's introduce the change of variable:

$$\alpha_t := 1 - \beta_t \tag{2.38}$$

defining:

$$\overline{\alpha_t} := \prod_{s=1}^{t} \alpha_s \tag{2.39}$$

By employing an iterative approach, we can utilize Eq. (2.36) in the following manner:

$$\begin{aligned} x_t &= \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} && (2.40)\\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{\alpha_t - \alpha_t\alpha_{t-1}}\epsilon_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-1} && (2.41)\\ &= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\epsilon}_{t-2} \\ &\vdots && (2.42)\\ &= \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\epsilon && (2.43) \end{aligned}$$

in Eq. (2.41), where $\epsilon_{t-1}, \epsilon_{t-1}, \epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $\bar{\epsilon}_{t-2}$ is the term after merging two Gaussian distributions, the explicit computation is given as:

$$\mathcal{N}(\mathbf{0}, \sigma_1^2\boldsymbol{I}) + \mathcal{N}(\mathbf{0}, \sigma_2^2\boldsymbol{I}) = \mathcal{N}(\mathbf{0}, (\sigma_1^2 + \sigma_2^2)\boldsymbol{I}) \tag{2.44}$$

$$\sigma_T^2 = \sigma_1^2 + \sigma_2^2 = \alpha_t - \alpha_t\alpha_{t-1} + 1 - \alpha_t = 1 - \alpha_t\alpha_{t-1}$$

Subsequently, we proceed to execute all the diffusion steps, starting from the original data and continuing up to time step 't', which can be succinctly expressed as:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}x_0, (1 - \overline{\alpha_t})\boldsymbol{I}) \tag{2.45}$$

2.3 Reverse process

In denoising diffusion models, the reverse process aims to remove noise added during the forward process using a neural network. We use a Gaussian distribution for both processes. The reverse process, a Markov chain, employs a neural network to predict parameters for the reverse diffusion kernel at each time step. During training, these parameters should resemble those of the forward diffusion kernel's posterior to return to the original data distribution. This approach facilitates data reconstruction and the generation of new samples from pure Gaussian noise, even during inference. Using again Eq. (2.13) we have:

$$p(x_0, x_1, ..., x_T) = p(x_T)\prod_{t=1}^{T} p(x_{t-1}|x_t) \tag{2.46}$$

In Fig. 2.1 it is illustrated the time step x_T, i.e. the first step for the reverse process, it will be pure noise by construction as follows:

$$p(x_T) = \mathcal{N}(x_T; \mathbf{0}, \boldsymbol{I}) \tag{2.47}$$

Where x_T is the ending point of the probability distribution after the semicolon we find the standard parameters of a normal probability distribution, namely the mean value and covariance.

The proposal of this work is denoise a Gaussian distribution, a natural question from Fig. 2.2 is to use the posterior distribution of the forward process $q(x_{t-1}|x_t)$ instead. The reverse process can be more stable and numerically precise, especially when you are dealing with very noisy data. In the forward process, you start with the true data and add noise at each step, which can sometimes lead to numerical instability or precision issues. In contrast, the reverse process starts with the noisy data and progressively reduces the noise level, which is more stable.

In order to get the relation between the reverse (p) and the posterior of the forward process we will use the Bayes' rule starting from Eq. (2.12):

$$p(x_{t-1}|x_t) = p(x_t|x_{t-1})\frac{p(x_{t-1})}{p(x_t)} \tag{2.48}$$

For the argument given above, we will not utilize the posterior of the forward transition, $q(x_t|x_{t-1})$. Instead, a solution presented in previous works like [HJA20] and [SDWMG15]

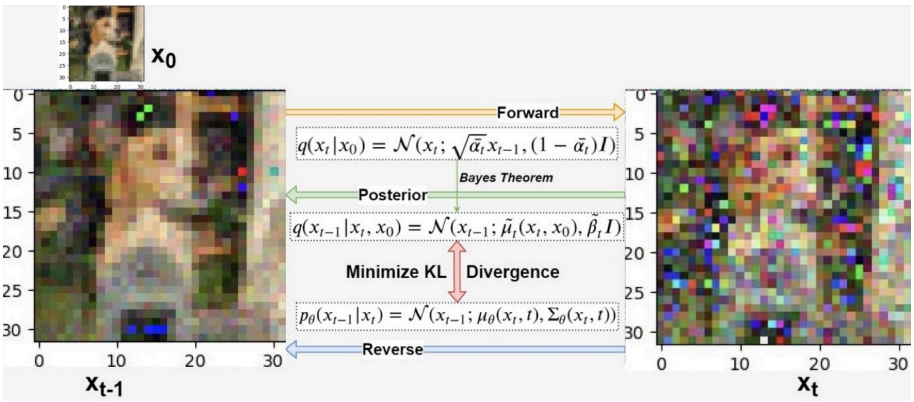


Figure 2.3: Illustration depicting the forward and reverse processes.

involves approximating the reverse process using a Gaussian distribution. In this approximation, we introduce trainable variables parametrized by θ following the notation of [HJA20]. This are given by the mean value ($\boldsymbol{\mu}_\theta(x_t, t)$) and the covariance ($\boldsymbol{\Sigma}\theta(x_t, t)$). Our objective is to minimize the Kullback-Leibler (KL) divergence between these two distributions. In Fig. 2.3 is illustrated how these transitions are related. These parameters are learned from the forward process based on the following:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t)\right) \tag{2.49}$$

As in [HJA20] we will not create a second neural network for the covariance, instead, we will use the schedule given for β_t, letting a constant scheduled covariance as:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \beta_t \boldsymbol{I}\right) \tag{2.50}$$

3 Diffusion models

In the realm of diffusion-based generative models, the fundamental training objective is centered around enhancing the model's capacity to generate samples that closely match the characteristics of the original data distribution. This objective is achieved by maximizing the log-likelihood, which essentially quantifies the likelihood of observing the generated sample 'x' at the culmination of the reverse diffusion process.

The aim of aligning the generated sample with the original data distribution is crucial for the model's performance. It implies that the model should have the ability to produce synthetic data points that are indistinguishable from real-world observations. In practical terms, this means that the generated data should exhibit the same statistical properties, patterns, and features as the authentic dataset, thereby enabling the model to capture and recreate the inherent complexity and diversity of the data.

By honing the log-likelihood of these generated samples, diffusion-based generative models strive to minimize the gap between synthetic and real data, facilitating applications such as image generation, data denoising, and data completion with remarkable fidelity to the underlying data distribution.

3.1 Variational bound on negative likelihood

Denoising Diffusion Models (DDMs) are latent variable models, as previously discussed, and they have the form, as follows:

$$p_\theta(x_0) := \int_{\mathbb{R}^{T \cdot d}} p_\theta(x_{0:T}) dx_{1:T} \tag{3.1}$$

In Equation 3.1, we define the likelihood of the data, denoted as $p_\theta(x_0)$ as an integral over a high-dimensional space $\mathbb{R}^d$, where T represents the number of time steps and d denotes the dimensionality of the data. This likelihood function is a central component of our generative model.
Inside the integral, it's important to note that the term x_0 is considered a fixed value within the probability density p. Therefore, we do not integrate over the variable x_0. It's worth emphasizing that if we were to illustrate this in terms of x_0, the overall joint probability would sum to 1, indicating that our focus is primarily on $p(x_0)$ the likelihood of the data. The Eq. (3.1) is harmful to compute as it is; to make it more manageable, it

will be necessary to rearrange the expression by adding a 1 as:

$$1 = \frac{q\left(x_{1\ldots T}|x_0\right)}{q\left(x_{1\ldots T}|x_0\right)}$$

then adding this identity into Eq. (3.1) yield:

$$\begin{aligned}
p_\theta(x_0) &:= \int p_\theta(x_{0:T}) dx_{1:T} \cdot 1 && (3.2)\\
&= \int p_\theta(x_{0:T}) dx_{1:T} \cdot \frac{q\left(x_{1\ldots T}|x_0\right)}{q\left(x_{1\ldots T}|x_0\right)} && (3.3)\\
&= \int dx_{1:T} q\left(x_{1\ldots T}|x_0\right) \frac{p_\theta(x_{0:T})}{q\left(x_{1\ldots T}|x_0\right)} && (3.4)\\
&= \int dx_{1:T} q\left(x_{1\ldots T}|x_0\right) p_\theta(x_T) \prod_{t=1}^{T} \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} && (3.5)
\end{aligned}$$

In Eq. (3.5) was used Eqs. (2.33) and (2.46). It will consider the ***log*** likelihood, for the following reasons: one, the *log* is a monotonic function and open the product in sums; second, it will be helpful in the KL-divergence used later in this model.
An important concept used is the entropy in information theory to characterize the 'unpredictability' of the random variable; if the entropy is low, then our variable is less unpredictable. Then using Definition 11:

$$\begin{aligned}
L &= -\sum_{x\in S} p_\theta(x_0) \log(p_\theta(x_0)) = \mathbb{E}\left(-\log p_\theta(x_0)\right) && (3.6)\\
&= -\mathbb{E}_{x_0\sim q(x_0)}\left[\log\left(p_\theta\left(x_0\right)\right)\right] && (3.7)\\
&= -\mathop{\mathbb{E}}_{q(x_0)}\left[\log\left(\int p_\theta\left(x_{0:T}\right) \mathrm{d}x_{1:T}\right)\right] && (3.8)\\
&= -\mathop{\mathbb{E}}_{q(x_0)}\left[\log\left(\int p_\theta\left(x_{0:T}\right) \cdot \frac{q\left(x_{1:T} \mid x_0\right)}{q\left(x_{1:T} \mid x_0\right)} \mathrm{d}x_{1:T}\right)\right] && (3.9)\\
&= -\mathop{\mathbb{E}}_{q(x_0)}\left[\log\left(\int \frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)} \cdot q\left(x_{1:T} \mid x_0\right) \mathrm{d}x_{1:T}\right)\right] && (3.10)\\
&= -\mathop{\mathbb{E}}_{q(x_0)}\left[\log\left(\mathop{\mathbb{E}}_{q(x_{1:T}|x_0)}\left[\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right]\right)\right] && (3.11)\\
&\leq -\mathop{\mathbb{E}}_{q(x_0)}\left[\mathop{\mathbb{E}}_{q(x_{1:T}|x_0)}\left[\log\left(\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right)\right]\right] && (3.12)\\
&= -\int_{x_0}\left(\int_{x_{1:T}} \log\left(\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right) q\left(x_{1:T} \mid x_0\right) \mathrm{d}x_{1:T}\right) q\left(x_0\right) \mathrm{d}x_0 && (3.13)\\
&= -\int_{x_0}\int_{x_{1:T}} \log\left(\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right) q\left(x_{1:T} \mid x_0\right) q\left(x_0\right) \mathrm{d}x_{1:T}\, \mathrm{d}x_0 && (3.14)\\
&= -\int_{x_{0:T}} \log\left(\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right) q\left(x_{0:T}\right) \mathrm{d}x_{0:T} && (3.15)\\
&= -\mathop{\mathbb{E}}_{q(x_{0:T})}\left[\log\left(\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right)\right] := L && (3.16)
\end{aligned}$$

A brief comment on Eq. (3.12): It employs Jensen's inequality as described in Lemma 2.1.1. Similarly, Equation 3.11 makes use of the definition provided in 10.

Equation 3.16 represents our upper bound. By applying Equation 3.5, we can deduce that:

$$L := -\mathop{\mathbb{E}}_{q}\left[\log\left(\frac{p_\theta\left(x_{0:T}\right)}{q\left(x_{1:T} \mid x_0\right)}\right)\right] \tag{3.17}$$

$$= \mathop{\mathbb{E}}_{q}\left[-\log\left(p_\theta\left(x_T\right)\prod_{t=1}^{T}\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_t|x_{t-1}\right)}\right)\right] \tag{3.18}$$

$$= \mathop{\mathbb{E}}_{q}\left[-\log\left(p_\theta\left(x_T\right)\right) - \log\left(\prod_{t=1}^{T}\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_t|x_{t-1}\right)}\right)\right] \tag{3.19}$$

$$= \mathop{\mathbb{E}}_{q}\left[-\log\left(p_\theta\left(x_T\right)\right) - \sum_{t=1}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_t|x_{t-1}\right)}\right] \tag{3.20}$$

$$= \mathop{\mathbb{E}}_{q}\left[-\log\left(p_\theta\left(x_T\right)\right) - \sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_t|x_{t-1}\right)} - \log\frac{p_\theta\left(x_0|x_1\right)}{q\left(x_1|x_0\right)}\right] \tag{3.21}$$

Here in Eq. (3.21) we use the Bayes rule Eq. (2.48). In order to complete the computation of Eq. (3.21), let's remark that q fulfill the Markov property, then is redundant to add x_0 as follows $q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0)$. To simplify Equation 3.21, we proceed by highlighting the following equality.

$$q(x_t|x_{t-1}) = q(x_t|x_{t-1}, x_0) \tag{3.22}$$

$$= q(x_{t-1}|x_t, x_0)\frac{q(x_t)}{q(x_{t-1})} \tag{3.23}$$

$$= q(x_{t-1}|x_t, x_0)\frac{q(x_t, x_0)q(x_0)}{q(x_{t-1}, x_0)q(x_0)} \tag{3.24}$$

$$= q(x_{t-1}|x_t, x_0)\frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} \tag{3.25}$$

Working on the second term of Eq. (3.21), we arrived at the following expression:

$$\sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_t|x_{t-1}\right)} = \sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_{t-1}|x_t, x_0\right)}\frac{q\left(x_{t-1}|x_0\right)}{q\left(x_t|x_0\right)} \tag{3.26}$$

$$= \sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_{t-1}|x_t, x_0\right)} + \underbrace{\sum_{t=2}^{T}\log\frac{q\left(x_{t-1}|x_0\right)}{q\left(x_t|x_0\right)}}_{:=\mathrm{A}} \tag{3.27}$$

Simplifying the A in Eq. (3.27),

$$A := \log\left(\prod_{t>1}\frac{q\left(x_{t-1}|x_0\right)}{q\left(x_t|x_0\right)}\right) \tag{3.28}$$

$$= \log\left(\frac{q\left(x_1|x_0\right)}{\cancel{q\left(x_2|x_0\right)}}\cdot\frac{\cancel{q\left(x_2|x_0\right)}}{\cancel{q\left(x_3|x_0\right)}}\cdots\frac{\cancel{q\left(x_{T-1}|x_0\right)}}{q\left(x_T|x_0\right)}\right) \tag{3.29}$$

$$= \log\left(\frac{q\left(x_1|x_0\right)}{q\left(x_T|x_0\right)}\right) \tag{3.30}$$

Merging Eqs. (3.21), (3.27) and (3.30) together gives as a result:

$$L = \mathop{\mathbb{E}}_{q}\left[-\log\left(p_\theta\left(x_T\right)\right) - \sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_t|x_{t-1}\right)} - \log\frac{p_\theta\left(x_0|x_1\right)}{q\left(x_1|x_0\right)}\right] \tag{3.31}$$

$$= \mathop{\mathbb{E}}_{q}\left[-\log\left(p_\theta\left(x_T\right)\right) - \sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_{t-1}|x_t,x_0\right)} - \log\frac{\cancel{q\left(x_1|x_0\right)}}{q\left(x_T|x_0\right)} - \log\frac{p_\theta\left(x_0|x_1\right)}{\cancel{q\left(x_1|x_0\right)}}\right] \tag{3.32}$$

$$= \mathop{\mathbb{E}}_{q}\left[\underbrace{-\log\frac{p_\theta\left(x_T\right)}{q\left(x_T|x_0\right)}}_{:=L_T} \underbrace{-\sum_{t=2}^{T}\log\frac{p_\theta\left(x_{t-1}|x_t\right)}{q\left(x_{t-1}|x_t,x_0\right)}}_{:=L_{t-1}} \underbrace{-\log p_\theta\left(x_0|x_1\right)}_{:=L_0}\right] \tag{3.33}$$

Equation 3.33 serves as the fundamental loss function pivotal in the optimization of our model's parameters.

3.2 Loss function and relative entropy

In denoising diffusion models (DDMs), choosing a suitable loss function is crucial for effectively training the model. The primary goal of the loss function in DDMs is to guide the model to generate synthetic data that closely matches the true data distribution.
Let's work out the term L^T from Eq. (3.33), from Eqs. (2.32) and (2.33) we have:

$$q\left(x_0, x_1, \cdots, x_T\right) = q\left(x_0\right) q\left(x_1, x_2, \cdots, x_T|x_0\right) \tag{3.34}$$

$$= q\left(x_0\right)\prod_{t=1}^{T} q\left(x_t|x_0\right) \tag{3.35}$$

in Eq. (3.34) we have to consider that for two independent random variables a and b the probability $p(a,b) = p(a)\cdot p(b)$, then the term L_T from Eq. (3.35) yields,

$$L_T = \mathop{\mathbb{E}}_{q(x_{0:T})}\left(-\log\frac{p_\theta\left(x_T\right)}{q\left(x_T|x_0\right)}\right) \tag{3.36}$$

$$= \mathop{\mathbb{E}}_{q(x_{0:T})}\left(\log\frac{q\left(x_T|x_0\right)}{p_\theta\left(x_T\right)}\right) \tag{3.37}$$

$$= \int_{\mathbb{R}^{(T+1)d}}\log\frac{q\left(x_T|x_0\right)}{p_\theta\left(x_T\right)} q\left(x_{0:T}\right) dx_{0:T} \tag{3.38}$$

$$= \int_{\mathbb{R}^{(T+1)d}}\log\frac{q\left(x_T|x_0\right)}{p_\theta\left(x_T\right)} q(x_0)\prod_{t=1}^{T} q\left(x_t|x_0\right) dx_{0:T} \tag{3.39}$$

$$= \int_{\mathbb{R}^{(T+1)d}}\log\frac{q\left(x_T|x_0\right)}{p_\theta\left(x_T\right)} q(x_0)q(x_T|x_0)\prod_{t=1}^{T-1} q\left(x_t|x_0\right) dx_{0:T} \tag{3.40}$$

$$= \int_{\mathbb{R}^{Td}}\left(\int_{\mathbb{R}^d} q(x_T|x_0)\log\frac{q\left(x_T|x_0\right)}{p_\theta\left(x_T\right)} dx_T\right) q(x_0)\prod_{t=1}^{T-1} q\left(x_t|x_0\right) dx_{0:T-1} \tag{3.41}$$

$$= \int_{\mathbb{R}^{Td}} D_{KL}\left(q(x_T|x_0)||p_\theta(x_t)\right) q\left(x_{0:T-1}\right) dx_{0:T-1} \tag{3.42}$$

$$= \mathop{\mathbb{E}}_{q(x_{0:T-1})}\left[D_{KL}\left(q(x_T|x_0)||p_\theta(x_t)\right)\right] \tag{3.43}$$

$$= \mathop{\mathbb{E}}_{x\sim q}\left[D_{KL}\left(q(x_T|x_0)||p_\theta(x_t)\right)\right] \tag{3.44}$$

In Eq. (3.42), we apply Eq. (2.21), enabling us to express L_T in terms of the relative entropy or Kullback-Leibler divergence. A similar approach can be employed for L_{t-1} in Eq. (3.33):

$$L_{t-1} = \int_{\mathbb{R}^{(T+1)d}} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} \underbrace{q(x_{t-1}|x_0)q(x_t|x_0)}_{:=\mathrm{C}} q(x_0) \prod_{i=1, i\neq t, t-1}^{T} q\left(x_i|x_0\right) dx_{0:T} \tag{3.45}$$

From Eq. (3.45), we will deduce the value of C by applying the Chain Rule, as defined in Definition 5, and leveraging Eqs. (2.12) and (3.35). This process aligns with the principles of the Chain Rule in probability.

$$C := q(x_{t-1}|x_0)q(x_t|x_0) \tag{3.46}$$

$$= \frac{q(x_{t-1}, x_t, x_0)}{q(x_0)} \tag{3.47}$$

$$= \frac{q(x_{t-1}, x_t, x_0)}{q(x_0)} \cdot \frac{\overbrace{q(x_t, x_0)}^{=1}}{q(x_t, x_0)} \tag{3.48}$$

$$= \frac{q(x_{t-1}, x_t, x_0)}{q(x_t, x_0)} \cdot \frac{q(x_t, x_0)}{q(x_0)} \tag{3.49}$$

$$= q(x_{t-1}|x_t, x_0) \cdot q(x_t|x_0) \tag{3.50}$$

Subsequently using Eq. (3.50), Eq. (3.45) produces the following result:

$$L_{t-1} = \int_{\mathbb{R}^{(T+1)d}} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} q(x_{t-1}|x_t, x_0) q(x_0) \prod_{i=1, i\neq t-1}^{T} q\left(x_i|x_0\right) dx_{0:T} \tag{3.51}$$

$$= \int_{\mathbb{R}^{Td}} \left(\int_{\mathbb{R}^d} \log \frac{q\left(x_{t-1}|x_t, x_0\right)}{p_\theta\left(x_{t-1}|x_t\right)} q(x_{t-1}|x_t, x_0) dx_{t-1} \right) q(x_0) \prod_{i=1, i\neq t-1}^{T} q\left(x_i|x_0\right) dx_{0:T} \tag{3.52}$$

$$= \mathop{\mathbb{E}}_{q(x_{0:T})}\left[D_{KL}\left(q\left(x_{t-1}|x_t, x_0\right)||p_\theta\left(x_{t-1}|x_t\right)\right)\right] \tag{3.53}$$

Then finally Eq. (3.33) yields as in [HJA20]:

$$H = \mathop{\mathbb{E}}_{x\sim q}\left[\underbrace{D_{KL}\left(q(x_T|x_0)||p_\theta(x_t)\right)}_{:=L_T} + \underbrace{\sum_{t=2}^{T} D_{KL}\left(q\left(x_{t-1}|x_t, x_0\right)||p_\theta\left(x_{t-1}|x_t\right)\right)}_{:=L_{t-1}} - \underbrace{\log p_\theta\left(x_0|x_1\right)}_{:=L_0} \right] \tag{3.54}$$

The point to articulate Eq. (3.54) as a KL divergence is because this measure the distance between 2 distributions for every time step $t \in [2 : T]$; Our objective is to reduce the Kullback-Leibler divergence between these two distributions. In other words, we aim for these two distributions to exhibit similarity at every time step within the range $t \in [2 : T]$. Ensuring similarity between the distributions guarantees that the images sampled from them will also be comparable at the corresponding time points.

The posterior of the forward process, $q(x_{t-1}|x_t, x_0)$, is static and lacks trainable parameters. It serves as a reference for the reverse process, $p_\theta(x_{t-1}|x_t)$, to approximate as closely as possible through adjustments in the neural network's parameter values facilitated by stochastic gradient descent optimization. Consequently, the reverse process, $p_\theta(x_{t-1}|x_t)$, generates images that resemble those from the static forward process, $q(x_{t-1}|x_t, x_0)$. In essence, the static reverse of the forward process provides the ground truth images or regression targets for the reverse process to align its generated images with at time steps $t = [2, T]$.

The objective is to minimize the discrepancy between the model's prediction and the ground truth noise. The KL-divergence loss function does not include any component that explicitly refers to the model's prediction. Instead, we minimize the discrepancy between the reverse process and the reverse of the forward process to ensure that the generated images from both processes are similar at each time step. This per-step similarity requirement between $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_t, x_0)$ establishes a gradual change in the generated images by the reverse process $p_\theta(x_{t-1}|x_t)$. As the static posterior of the forward process gradually removes noise from images step-by-step, the images generated by the reverse of the forward process exhibit a gradual denoising effect, becoming clearer over time. By regressing towards these progressively clearer images, the learned reverse process, with the time stamp t as input, is compelled to generate images that change gradually.

This per-step similarity requirement constrains the behavior of the neural network-based reverse process, forcing it to adhere to an already known and simpler process, namely the posterior of the forward process. The per-step KL-divergence prevents the learned neural network from producing unexpected results, such as generating an image of a cat at an early step and then transforming it into a human face.

It's important to note the timestamp range $t = [2, T]$ here. This range implies that the L_{t-1} terms only cover timestamps from 2 to T, leaving the first step at $t = 1$ unaccounted for. The initial step at $t = 1$, which ultimately generates the natural image, holds significance. Remember that three terms from L_v were left unanalyzed? We will later discover that these remaining terms account for the first timestamp.

3.3 Loss function for a Gaussian distribution

The objective in this section is to derive a clear and explicit expression for the loss term. We aim to compare the distributions of the posterior in the forward process with that of the reverse process. It's worth noting that at this stage, we have assumed an ansatz that treats both the forward and reverse processes as Gaussian distributions.

Our next step involves delving into Eq. (3.33) to meticulously work out each of the terms within it. This comprehensive analysis is critical for understanding the model's performance and its ability to effectively capture the relationship between the forward and reverse processes.

3.3.1 L_T loss term

In Eq. (3.54), derived in Section 3.1, the first term, denoted as L_T, is defined as follows:

$$D_{KL}\left(q(x_T|x_0)||p_\theta(x_t)\right) \tag{3.55}$$

It is important to emphasize that the covariance parameter, denoted as β, follows a schedule within the range of initial and final values, defined as $[10^{-4}, \ldots, 0.02]$. In the conditional probability q as given by Eq. (2.35), when setting $\beta_T = 0.02$, we have $q(x_T|x_0) = \mathcal{N}(x_T; \sqrt{1-\beta_T}x_0, \beta_T \boldsymbol{I}) = \mathcal{N}(x_T; \sqrt{0.98}x_0, 0.02\boldsymbol{I})$. Notably, in this case, there are no learnable parameters involved.

Conversely, Equation 2.47 represents pure noise, which implies that Eq. (3.55) remains constant throughout. Consequently, we do not consider it in the training process. This insight helps streamline the training and optimization of the model.

3.3.2 L_{t-1} loss term

The $D_{KL}\left(q\left(x_{t-1}|x_t, x_0\right)||p_\theta\left(x_{t-1}|x_t\right)\right)$ compare Eq. (2.49) with $q\left(x_{t-1}|x_t, x_0\right)$ (the posterior of the forward process) and it will be tractable when

$$q\left(x_{t-1}|x_t, x_0\right) = \mathcal{N}\left(x_{t-1}; \hat{\boldsymbol{\mu}}_t(x_t, t), \hat{\boldsymbol{\Sigma}}_t(x_t, t)\right) \tag{3.56}$$

In other words, when the conditional probability follows a normal distribution, the parameters $\hat{\boldsymbol{\mu}}_t(x_t, t)$ and $\hat{\boldsymbol{\Sigma}}_t(x_t, t)$ can be derived, in order to achieve that purpose lets recall Eq. (3.25) (Bayes's rule) yields:

$$q\left(x_{t-1}|x_t, x_0\right) = \frac{q\left(x_{t-1}|x_0\right) q\left(x_t|x_{t-1}\right)}{q\left(x_t|x_0\right)} \tag{3.57}$$

and using the Eqs. (2.39) and (2.45), then each of the three terms of Eq. (3.57) will looks like a Gaussian distribution for a random variable Eq. (2.5).

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right] = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\frac{\left(x^2 - x\mu + \mu^2\right)}{\sigma^2}\right] \tag{3.58}$$

In Eq. (3.58) is just expanded the squared term to highlight that the mean value (μ) will be multiplied for the endpoint (x), then Eq. (3.57):

$$\begin{aligned}
q\left(x_{t-1}|x_t, x_0\right) \sim \exp\Bigg[-\frac{1}{2}\Bigg(&-\frac{\left(x_t - \sqrt{\bar{\alpha}_t}x_0\right)^2}{1-\bar{\alpha}_t} \\
&+ \frac{\left(x_{t-1} - \sqrt{\bar{\alpha}_{t-1}}x_0\right)^2}{1-\bar{\alpha}_{t-1}} + \frac{\left(x_t - \sqrt{\alpha_t}x_{t-1}\right)^2}{1-\alpha_t}\Bigg)\Bigg] \\
= \exp\Bigg[-\frac{1}{2}\Bigg(&-\frac{x_t^2 - \sqrt{\bar{\alpha}_t}x_0x_t + \bar{\alpha}_t x_0^2}{1-\bar{\alpha}_t} \\
&+ \frac{x_{t-1}^2 - \sqrt{\bar{\alpha}_{t-1}}x_0x_{t-1} + \bar{\alpha}_{t-1}x_0^2}{1-\bar{\alpha}_{t-1}} + \frac{x_t^2 - \sqrt{\alpha_t}x_{t-1}x_t + \alpha_t x_{t-1}^2}{1-\alpha_t}\Bigg)\Bigg] \\
= \exp\Bigg[-\frac{1}{2}\Bigg(&x_{t-1}^2\left(\frac{1}{1-\bar{\alpha}_{t-1}} + \frac{\alpha_t}{1-\alpha_t}\right) \\
&+ x_{t-1}\left(\frac{\sqrt{\alpha_t}}{1-\alpha_t}x_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\right) + F(x_0, x_t)\Bigg)\Bigg]
\end{aligned}$$

Where F is a function depending only on (x_0, x_t) no one term depends on x_{t-1}, taking inspiration from the Eq. (3.58) the mean value is multiplied by the endpoint of the normal distribution, to be more clear, let's compute the normal distribution on Eq. (3.56) assuming $\hat{\boldsymbol{\mu}}_t$ is given:

$$\begin{aligned}
q\left(x_{t-1}|x_t, x_0\right) &= \mathcal{N}\left(x_{t-1}; \hat{\boldsymbol{\mu}}_t(x_t, t), \hat{\beta}_t(x_t, t)\boldsymbol{I}\right) \\
&\sim \exp\left[-\frac{1}{2}\frac{\left(x_{t-1} - \hat{\boldsymbol{\mu}}_t(x_t, x_0)\right)^2}{\hat{\beta}_t}\right] \\
&= \exp\left[-\frac{1}{2}\frac{x_{t-1}^2 - \hat{\boldsymbol{\mu}}_t(x_t, x_0)x_{t-1} + \hat{\boldsymbol{\mu}}_t(x_t, x_0)^2}{\hat{\beta}_t}\right]
\end{aligned}$$

Then from these two equations above, comparing blue with blue, red with red, and doing some algebra, we got that $\hat{\boldsymbol{\mu}}_t$ and $\hat{\beta}_t$ are given by:

$$\hat{\beta}_t\boldsymbol{I} := \hat{\boldsymbol{\Sigma}}_t(x_t, t) = \frac{(1-\bar{\alpha}_{t-1})\beta_t}{1 - -\bar{\alpha}_t}\boldsymbol{I} \tag{3.59}$$

$$\hat{\mu}_t\left(x_t, x_0\right) = \frac{\sqrt{\bar{\alpha}_{t-1}}\cdot\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}\left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t}x_t \tag{3.60}$$

The term L_{t-1} from Eq. (3.54) can be expressed using Eq. (2.31) then:

$$\begin{aligned} D_{KL}\left(q\left(x_{t-1} \mid x_t, x_0\right) \| p_\theta\left(x_{t-1} \mid x_t\right)\right) &= \frac{1}{2} \log\left(\frac{\det\left(\Sigma_\theta\left(x_t, t\right)\right)}{\det\left(\widetilde{\beta}_t \mathbf{I}\right)}\right) \\ &- \frac{1}{2} d \\ &+ \frac{1}{2} \operatorname{tr}\left\{\Sigma_\theta\left(x_t, t\right)^{-1} \cdot \tilde{\beta}_t \mathbf{I}\right\} \\ &+ \frac{1}{2}\left[\mu_\theta\left(x_t, t\right) - \hat{\mu}_t\left(x_t, x_0\right)\right]^T \cdot \Sigma_\theta\left(x_t, t\right)^{-1} \cdot \left[\mu_\theta\left(x_t, t\right) - \hat{\mu}_t\left(x_t, x_0\right)\right] \end{aligned} \tag{3.61}$$

In order to simplify the model, we will not consider two neural networks instead, it will depreciate the one related to the covariance according to [KW22], and this will be considered as a frozen parameter:

$$\Sigma_\theta\left(x_t, t\right) = \sigma_t^2 \mathbf{I} \tag{3.62}$$

with

$$\sigma_t^2 = \hat{\beta}_t \tag{3.63}$$

Then from Eq. (3.61), the only one term that is not constant will be the last one, and the rest will be considered as a C constant:

$$\begin{aligned} L_{t-1} - C &= \frac{1}{2}\left[\mu_\theta\left(x_t, t\right) - \hat{\mu}_t\left(x_t, x_0\right)\right]^T \cdot \Sigma_p\left(x_t, t\right)^{-1} \cdot \left[\mu_\theta\left(x_t, t\right) - \hat{\mu}_t\left(x_t, x_0\right)\right] \\ &= \frac{1}{2}\left[\mu_\theta\left(x_t, t\right) - \hat{\mu}_t\left(x_t, x_0\right)\right]^T \cdot \frac{1}{\sigma_t^2} \mathbf{I}^{-1} \cdot \left[\mu_\theta\left(x_t, t\right) - \hat{\mu}_t\left(x_t, x_0\right)\right] \\ &= \frac{1}{2\sigma_t^2}\left|\hat{\mu}_t\left(x_t, x_0\right) - \mu_\theta\left(x_t, t\right)\right|^2 \end{aligned} \tag{3.64}$$

To proceed with the calculations of the neural network $\hat{\mu}_t$ we have to use Eq. (2.40), given by:

$$x_0 = \frac{1}{\sqrt{\hat{\alpha}_t}}\left(x_t - \sqrt{1-\hat{\alpha}_t}\epsilon\right) \tag{3.65}$$

Then Eq. (3.60) yields:

$$\begin{aligned} \hat{\mu}_t\left(x_t, x_0\right) &= \left[\frac{\sqrt{\alpha_t} \cdot \left(1-\bar{\alpha}_{t-1}\right)}{1-\bar{\alpha}_t} + \frac{\sqrt{\bar{\alpha}_{t-1}} \cdot \beta_t}{\sqrt{\bar{\alpha}_t} \cdot \left(1-\bar{\alpha}_t\right)}\right] x_t - \frac{\sqrt{\bar{\alpha}_{t-1}} \cdot \beta_t \cdot \sqrt{1-\bar{\alpha}_t}}{\sqrt{\bar{\alpha}_t} \cdot \left(1-\bar{\alpha}_t\right)} \epsilon \\ &= \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\hat{\alpha}_t}}\epsilon\right) \end{aligned} \tag{3.66}$$

In Eq. (3.66) $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. Using this neural network gotten and designing a ϵ_θ such that given a x_t will predict noise ϵ, its plausible to consider for our neural network the parameter as:

$$\mu_\theta\left(x_t, t\right) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\hat{\alpha}_t}}\epsilon_\theta\right) \tag{3.67}$$

then from Eqs. (3.64), (3.66) and (3.67) we have the final version as a difference of the ground true noise and the predicted noise:

$$L_{t-1} - C = \frac{1}{2\sigma^2} \left\| \hat{\mu}_t \left(x_t, x_0 \right) - \mu_\theta \left(x_t, t \right) \right\|^2 \tag{3.68}$$

$$= \frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) - \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta \left(x_t, t \right) \right) \right\|^2 \tag{3.69}$$

$$= \frac{1}{2\sigma_t^2} \left\| \frac{1}{\sqrt{\alpha_t}} x_t - \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_t}} \epsilon - \frac{1}{\sqrt{\alpha_t}} x_t + \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta \left(x_t, t \right) \right\|^2 \tag{3.70}$$

$$= \frac{1}{2\sigma_t^2} \left\| - \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_t}} \epsilon + \frac{\beta_t}{\sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta \left(x_t, t \right) \right\|^2 \tag{3.71}$$

$$= \frac{1}{2\sigma_t^2} \frac{\beta_t^2}{\alpha_t \left(1 - \bar{\alpha}_t \right)} \left\| \epsilon - \epsilon_\theta \left(x_t, t \right) \right\|^2 \tag{3.72}$$

$$= \frac{1}{2\sigma_t^2} \frac{\beta_t^2}{\alpha_t \left(1 - \bar{\alpha}_t \right)} \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t} \epsilon, t \right) \right\|^2 \tag{3.73}$$

3.3.3 L_0 loss term

The last term of Eq. (3.54) will be

$$L_0 = -\mathop{\mathbb{E}}_{q} \left[\log \left(p \left(x_0 \mid x_1 \right) \right) \right] \tag{3.74}$$

$$= - \mathop{\mathbb{E}}_{q(x_0, x_1)} \left[\mathop{\mathbb{E}}_{q(x_0)} \left[\log \left(p \left(x_0 \mid x_1 \right) \right) \right] \right] \tag{3.75}$$

$$= \mathop{\mathbb{E}}_{q} \left[- \int \log \left(p \left(x_0 \mid x_1 \right) \mathrm{d}x_0 \right) \right] \tag{3.76}$$

$$= \mathop{\mathbb{E}}_{q} \left[\int \log \left(\frac{q \left(x_0 \mid x_1, x_0 \right)}{p \left(x_0 \mid x_1 \right)} \right) q \left(x_0 \mid x_1, x_0 \right) \mathrm{d}x_0 \right] \tag{3.77}$$

$$= \mathop{\mathbb{E}}_{q} \left[KL \left(q \left(x_0 \mid x_1, x_0 \right) || p \left(x_0 \mid x_1 \right) \right) \right] = L_0 \tag{3.78}$$

$$= \mathop{\mathbb{E}}_{q} \left[KL \left(q \left(x_{t-1} \mid x_t, x_0 \right) \| p \left(x_{t-1} \mid x_t \right) \right) \right] |_{t=1} = L_{t-1}|_{t=1} \tag{3.79}$$

We have used the property $q(x_0|x_1, x_0) = 1$ in Eq. (3.77), which reveals no KL divergence in the step $t = 1$. In similar way for Eq. (3.66) we have the equivalent:

$$\mu_1 \left(x_1, t \right) = \frac{1}{\sqrt{\alpha_1}} \left(x_1 - \frac{\beta_1}{\sqrt{1-\hat{\alpha}_1}} \epsilon \right) = \hat{\mu}_t \left(x_t, x_0 \right) |_{t=1} \tag{3.80}$$

as we said at the end of Section 3.1 in L_{t-1} we are considering $t \in [2, T]$ and now it will be considered $t \in [1, T]$ being for $t = 1$ the value for L_0, letting the expression to be optimized as follows:

$$L_{\text{simple}} = \mathop{\mathbb{E}}_{x_0, \epsilon, t} \left[\left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t} \epsilon, t \right) \right\|^2 \right] \tag{3.81}$$

Table 3.1: Algorithm 1 Training and Algorithm 2 Sampling [HJA20].

Algorithm 1 Training	Algorithm 2 Sampling
1: repeat	1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$	2: for $t = T, \ldots, 1$ do
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$	3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$	4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}} \epsilon_\theta (\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: Take gradient descent step on $\nabla_\theta \left\| \epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \epsilon, t \right) \right\|^2$	5: end for
6: until converged	6: return $\mathbf{x}_0$

3.4 Training and sampling

In a previous section, we calculated a quantitie called L_{simple} as shown in equation Eq. (3.81) this is an upper limit for the amount of uncertainty (entropy) in a probability distribution denoted as p in equation Eq. (3.16).

To improve the predictability of our distribution $p_\theta(x_t)$, we need to optimize the L_{simple} value. This optimization process is discussed in more detail in a previous chapter.

Let's focus on Algorithm 1, described in Table Section 3.4. This algorithm trains a neural network responsible for predicting noise, denoted as ϵ, which follows a normal distribution $\mathcal{N}(0, I)$.

During the training process of this neural network, we employ a gradient descent optimization method, explicitly using the Adam optimizer from the PyTorch library in Python. This optimizer helps adjust the neural network's parameters to minimize the discrepancy between predicted and actual noise, making our model better at generating accurate predictions.

In summary, we're optimizing a value called L_{simple} to reduce uncertainty in our distribution $p_\theta(x_t)$, and we're using Algorithm 1 along with the Adam optimizer to train a neural network to predict noise accurately.

After completing the training process of the neural network, the outcome is the learned parameters of that network. These parameters are used to compute $\mu_\theta(x_t, t)$.

In a similar manner to what we've seen in equation Eq. (2.40), we apply a reverse process. Here, we determine the mean value using the formula provided in Eq. (3.67).

Next, we introduce noise to this mean value. This noise follows a normal distribution with mean 0 and covariance matrix I, and we scale it by a factor of $\sqrt{\beta_t}$. This step is outlined in Algorithm 2 in Table Section 3.4.

To initiate this process, we start with an initial value x_T sampled from a normal distribution with mean 0 and covariance matrix I. Then, we iterate through the process until we reach the point where $t = 1$.

4 Architecture and code

In denoising diffusion models (DDMs), the U-Net architecture plays a pivotal role in noise removal, enhancing the quality of generated data, particularly in image reconstruction. Known for its image processing prowess, the U-Net's encoder-decoder structure and skip connections enable it to estimate and eliminate noise introduced during data degradation. It is trained on noisy-clean data pairs, gaining a deep understanding of the noise distribution and restoring data to its pristine state. Integrated into the reverse diffusion process, the U-Net efficiently denoises data, ensuring high-quality results at each step.

4.1 U-Net

The neural network it will approximate the $\epsilon_\theta(x, t) : \mathbb{R}^d \times \mathbb{N} \longrightarrow \mathbb{R}^d$. The architecture of the network (UNet) is illustrated in Fig. 4.1 and can be divided into two key components: the contracting path, located on the left-hand side, and the expansive path on the right of Fig. 4.1. The contracting path, as its name suggests, initiates the architecture and is responsible for the initial data compression. It comprises a sequence of 3×3 convolutions (unpadded convolutions), each of which is followed by a Rectified Linear Unit (ReLU) activation, and a 2x2 max pooling operation featuring a stride of 2. With each successive downsampling step, the number of feature channels is doubled, contributing to the reduction in spatial dimensions and the extraction of key features.[RFB15] In the expansive path, each step involves upsampling the feature map, followed by a 2x2 convolution

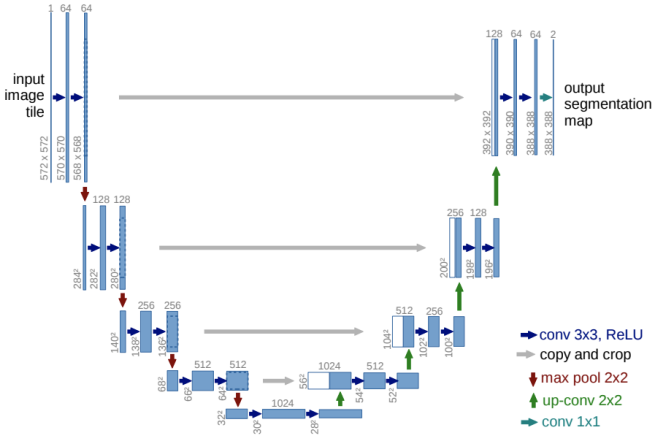


Figure 4.1: Unet architecture for a 32×32 pixel resolution.[RFB15]

(known as 'up-convolution') that reduces the number of feature channels by half. This is followed by concatenation with the correspondingly cropped feature map from the contracting path and two 3x3 convolutions, each with a ReLU activation. The cropping is essential to compensate for the loss of border pixels during convolution. Finally, a 1x1 convolution at the last layer maps each 64-component feature vector to the desired number of classes. In total, the network comprises 23 convolutional layers. "I have incorporated information from Ronneberger et. al. work on U-Net: Convolutional Networks for Biomedical Image Segmentation in my research". For further details [RFB15].
The code used to create the UNet is given by importing the ***nn*** package in python library, as is presented in the following code.

```
1  class UNet(nn.Module):
2      def __init__(self, img_channels = 1, time_embedding_dims = 128, labels =
                                            False, sequence_channels = (64, 128
                                            , 256, 512, 1024)):
3          super().__init__()
4          self.time_embedding_dims = time_embedding_dims
5          sequence_channels_rev = reversed(sequence_channels)
6
7          self.downsampling = nn.ModuleList([Block(channels_in, channels_out,
                                                time_embedding_dims, labels)
                                                for channels_in, channels_out
                                                in zip(sequence_channels,
                                                sequence_channels[1:])])
8          self.upsampling = nn.ModuleList([Block(channels_in, channels_out,
                                                time_embedding_dims, labels,
                                                downsample=False) for
                                                channels_in, channels_out in
                                                zip(sequence_channels[::-1],
                                                sequence_channels[::-1][1:])])
9          self.conv1 = nn.Conv2d(img_channels, sequence_channels[0], 3, padding=1
                                                )
10         self.conv2 = nn.Conv2d(sequence_channels[0], img_channels, 1)
11
12
13     def forward(self, x, t, **kwargs):
14         residuals = []
15         o = self.conv1(x)
16         for ds in self.downsampling:
17             o = ds(o, t, **kwargs)
18             residuals.append(o)
19         for us, res in zip(self.upsampling, reversed(residuals)):
20             o = us(torch.cat((o, res), dim=1), t, **kwargs)
21
22         return self.conv2(o)
```

4.2 Forward and reverse process routine

The algorithm in Section 3.4 outlines the training process focusing on optimizing noise prediction. We have chosen to run this process for 100 epochs in line with computational constraints. To facilitate this optimization, we employ the Adam optimizer method.

```
for epoch in range(NO_EPOCHS):
    for batch in trainloader:
        t = torch.randint(0, diffusion_model.timesteps, (BATCH_SIZE,)).long().
                                                    to(device)
        batch = batch.to(device)
        batch_noisy, noise = diffusion_model.Forward_Diffusion(batch, t, device
                                                    )
        predicted_noise = unet(batch_noisy, t)

        optimizer.zero_grad()
        loss = torch.nn.functional.mse_loss(noise, predicted_noise)
        mean_epoch_loss.append(loss.item())
        loss.backward()
        optimizer.step()

    if epoch > 70 and epoch % PRINT_FREQUENCY == 0:
        print('---')
        print(f"Epoch: {epoch} | Train Loss {np.mean(mean_epoch_loss)} | Val
                                                    Loss {np.mean(
                                                    mean_epoch_loss_val)}")
        if VERBOSE:
            with torch.no_grad():
                plot_noise_prediction(noise[0], predicted_noise[0])
                plot_noise_distribution(noise, predicted_noise)

        save_dir = f"size{BATCH_SIZE}/epoch{NO_EPOCHS}"
```

In this function, the forward diffusion process is at the heart of the training mechanism. It plays a pivotal role in introducing stochasticity and noise, a fundamental aspect in many training processes, particularly in deep learning models.

The function begins by generating random noise, following a standard normal distribution ($\mathcal{N}(\mathbf{0}, \boldsymbol{I})$), which is a common practice for introducing unpredictability in neural network training. This noise serves as a critical element in the training process.

The subsequent steps involve scaling the noise and the initial data. Here, $\sqrt{\alpha_t}$ and $\sqrt{1-\alpha_t}$ are computed, taking into account the specific time step t and the shape of the data. These values essentially control the magnitude of the noise and its influence on the data at this particular stage of the training process. The mean is then calculated as the element-wise product of $\sqrt{\alpha_t}$ and the initial data.

The final result is a tuple comprising the mean, which embodies the deterministic aspect of the data at this stage, and the noise, which encapsulates the stochastic component. The application of this function is a crucial step in optimizing the model's parameters,

making it an essential building block for the training process.

```
1         def Forward_Diffusion(self,x_0,t, device):
2         noise = torch.randn_like(x_0) #here sample as N(0,1)normal distribution
3         sqrt_alphas_hat_t = self.get_index_from_list(self.alphas_hat.sqrt(),t,
                                                       x_0.shape)
4         sqrt_one_minus_alphas_hat_t = self.get_index_from_list(torch.sqrt(1.-
                                                       self.alphas_hat),t,x_0.shape)
5
6         mean = sqrt_alphas_hat_t.to(device)*x_0.to(device)
7         variance = sqrt_one_minus_alphas_hat_t.to(device)*noise.to(device)
8
9         return mean + variance, noise.to(device)
```

The algorithm outlined in Section 3.4 for sampling begins with the initialization of pure noise. This is exemplified in the second line of the following code snippet:

```
1     with torch.no_grad():
2     img = torch.randn((1, 1) + IMG_SHAPE).to(device)
3     for i in reversed(range(diffusion_model.timesteps)):
4         t = torch.full((1,), i, dtype=torch.long, device=device)
5         img = diffusion_model.backward(img, t, unet.eval().to(device))
```

In this snippet, the process is initiated by creating random noise data, which serves as the starting point for the sampling procedure. This approach ensures that the generation process begins with an element of unpredictability, setting the stage for the diffusion model to progressively refine and generate the final output.
In the given Python code by Section 3.4, the backward function is responsible for modeling the reverse process and returning the noise component, p. Here's the code with annotations:

```
1     def backward(self, x, t, model, **kwargs):
2         betas_t = self.get_index_from_list(self.betas, t, x.shape)
3         sqrt_one_minus_alphas_hat_t = self.get_index_from_list(torch.sqrt(1. -
                                                       self.alphas_hat), t, x.shape)
4         sqrt_recip_alphas_t = self.get_index_from_list(torch.sqrt(1.0 / self.
                                                       alphas), t, x.shape)
5         mean = sqrt_recip_alphas_t * (x - betas_t * model(x, t, **kwargs) /
                                                       sqrt_one_minus_alphas_hat_t)
6         posterior_variance_t = betas_t
7
8         if t == 0:
9             return mean
10        else:
11            noise = torch.randn_like(x)
12            variance = torch.sqrt(posterior_variance_t) * noise
13            return mean + variance
```

At time $t = 0$, the function returns the mean without adding noise, as this is the initial step of the reverse process. The function generates random noise and computes the vari-

ance for subsequent time steps (when t is not zero). The sum of the mean and variance is returned, representing the data at the given time t. When added to the mean, this noise is a crucial element in the reverse process, introducing stochasticity and randomness to the data.

5 Experiments

The central goal of this thesis revolves around reconstructing images across the denoising diffusion models. To accomplish this objective, it is crucial to train the model effectively. As part of this training process, we will experiment with different configurations starting from the basis that we have generated a dataset consisting of ten thousand images, all of which depict triangles.

These triangular images adhere to two straightforward conditions. First, we begin by selecting three random points within a canvas of size nxn pixels. This canvas serves as the backdrop for our images. Next, we fill the region enclosed by these three randomly chosen points with black pixels, assigning a pixel value of 1 to those served areas. Importantly, these images are monochromatic, containing only one color channel.

In essence, the dataset we've created consists of a diverse set of triangle images that vary in size, shape, and orientation. These images will serve as the foundation for training our denoising diffusion model, allowing us to explore and develop techniques for reconstructing images that have been corrupted by noise.
Referring back to equation Eq. (2.35), the variable β_t plays a crucial role in introducing Gaussian noise to the data. A variance schedule guides this noise addition.

5.1 Schedule of β

A linear schedule is commonly used in most cases, as demonstrated in prior research such as [HJA20]. However, in this work, we explore additional noise schedules. One of these schedules is a sine wave-based schedule, the other schedule involves exponentiation with values of 1.5, 2, and 10.
These different schedules allow a more versatile and adaptable approach to introducing noise during data processing.
Certainly, when applying different noise schedules in the forward process, you are essentially modulating the level and pattern of noise added to the data. This variation in noise schedules can have several effects on the data processing and the resulting outcomes Fig. 5.2:

- Linear Schedule: The usual schedule for constant noise adding is used, for example, in [HJA20].
- Wave Schedule: This can create a dynamic and rhythmic pattern of noise.

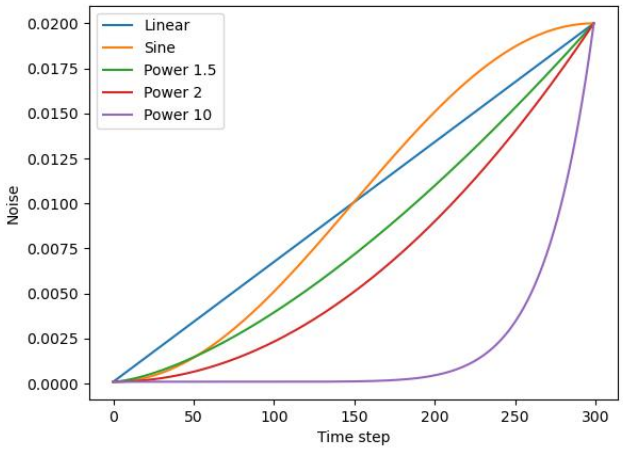


Figure 5.1: The schedule of the noise added.

- Exponential Schedules: Applying exponential schedules with different exponents (1.5, 2, and 10) can drastically change the noise profile. Lower exponents result in slower increases, while higher exponents lead to rapid amplification of noise.

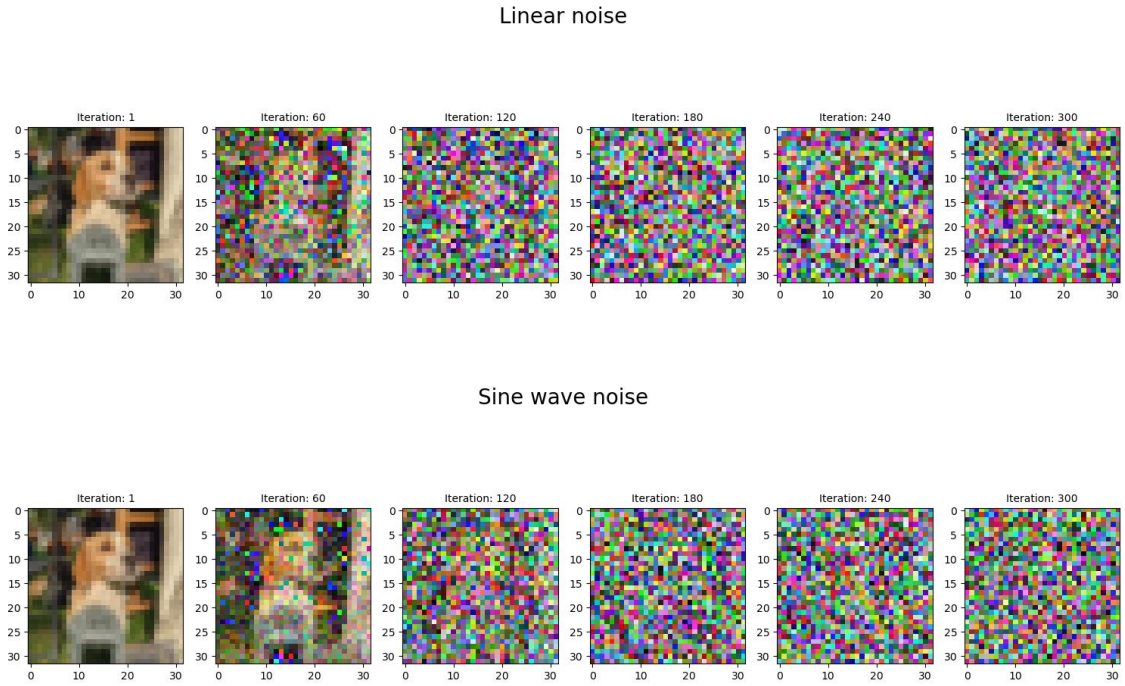


Figure 5.2: Different schedules of Gaussian noise are added to a personal image in order to visualize it.

These schedules are helpful when you need to model scenarios where noise grows or diminishes exponentially over time.

In Fig. 5.2, it is illustrated how the noise is added in every schedule, with the purpose of having a better understanding and in the way to determining the most optimal schedule for our training data.

5.2 Initial data with noise

To serve our specific purpose, we will create three datasets for training. The first dataset will consist of images with dimensions of $32x32$ pixels; the other two will be presented in the following chapter. However, these three datasets will ultimately be converted into torch images of the same size, specifically $32x32$ pixels.

When we generate these images and use the ImageDraw library in Python to fill the space

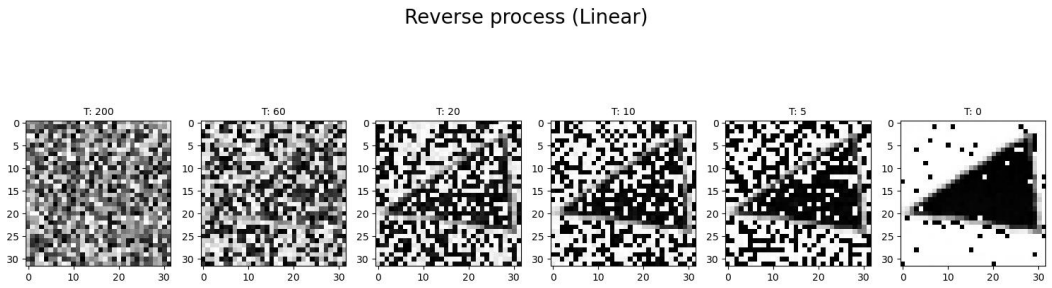


Figure 5.3: The reconstruction after using a training set where noise is present in the original data.

between three random points, some gray dots may be left around the curves. These gray dots contribute to the smoothness of the generated curves. It helps to draw the curves in a smoother way but is counterproductive in the training of images because this noise will be part of it and will be in the final result.

However, when we deal with the $32x32$ pixel images, these gray dots may become more noticeable and affect the quality of the data. Indeed, a closer view of the training set reveals that the images around the corners are noisy enough to affect the training process; the denoising result at different time steps is displayed in Fig. 5.3.

6 Results

This section focuses on the outcomes of our experiments and investigations. Here we present, analyze, and interpret the findings obtained through rigorous model training, experimentation, and analysis of parameters used. Our goal is to provide a comprehensive overview of the results, with a special focus on the implications and insights discovered throughout this study.
Here it will be presented the improvements made after the training process with the noise that caused distortion in the construction of the new samples

6.1 Training sets

Following the successful experimentation with the first training set and noise schedules to address the noise issue presented in the previous section. Low resolution made the noise around the figure more evident, as well as the process of converting data from an array to an image for creating the training set, and subsequently reading, resizing, and converting it back to a PyTorch array for training, had the potential to corrupt the training data.
As a solution, we opted for three options:

- **Low resolution:** This set is presented in the previous section when the direct resolution output is 32×32 pixels.

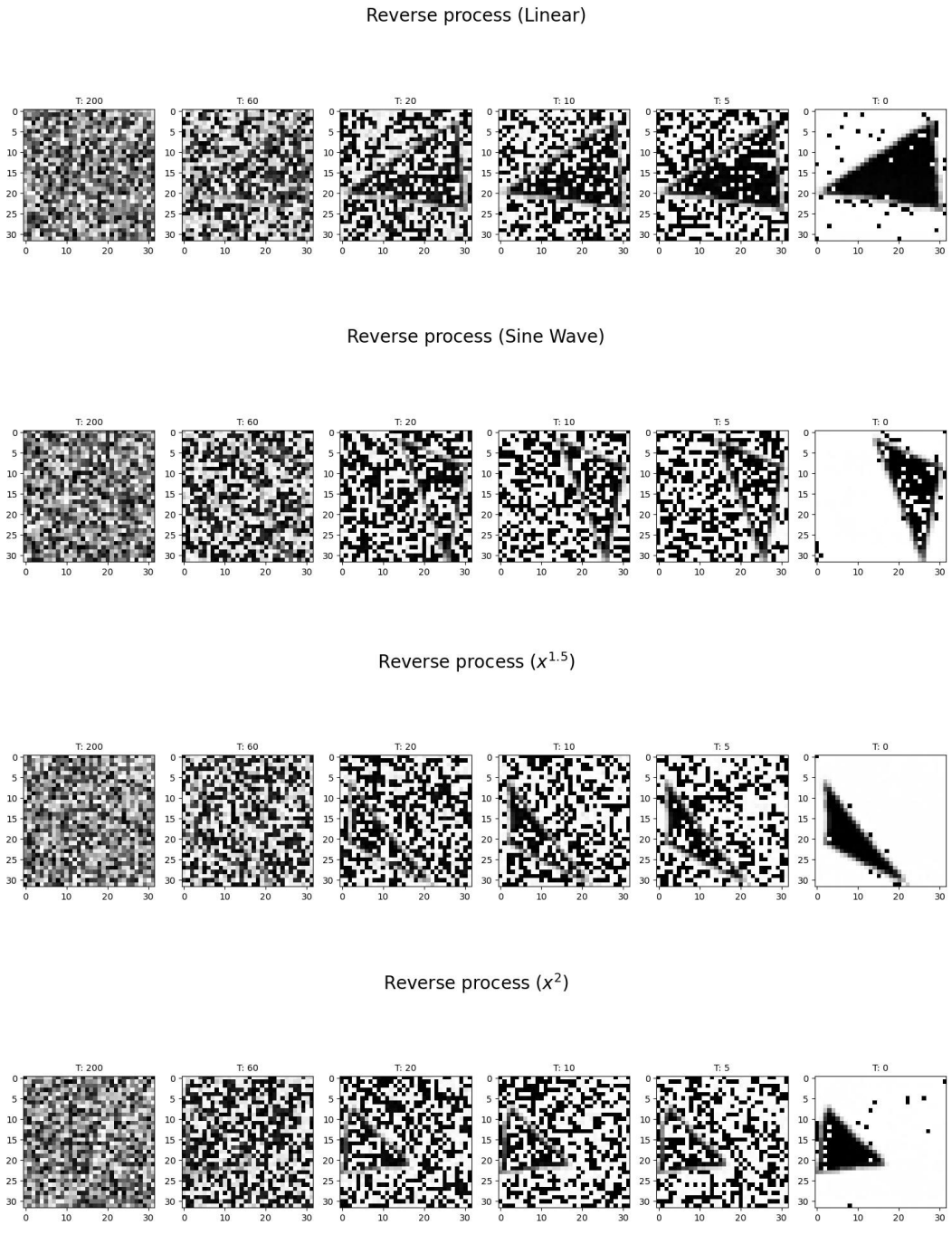
- **High resolution:** We opted for a larger training set of 720×720 pixel images, which are subsequently transformed into the smaller 32×32 format. This conversion process effectively mitigates the issue of excessive noise surrounding the main subject, rendering it nearly imperceptible.

- **Array format:** Save the training sets as '.npy' files, which essentially represent the arrays directly. By doing this, we eliminate any intermediaries that could introduce noise or corruption into the data, ensuring the integrity and reliability of our training sets.

Utilizing the same schedules as established in the previous section, we execute the code accordingly.

6.1.1 Low resolution

The training set of the low resolution presented in the previous section is used without the resizing process because it is already in the correct form; with all the considerations given above, the results in the construction of images for all the schedules plotted in Fig. 6.1. Where this noise present in the original training set is reflected in the results, around every triangle in Fig. 6.1 can be observed.



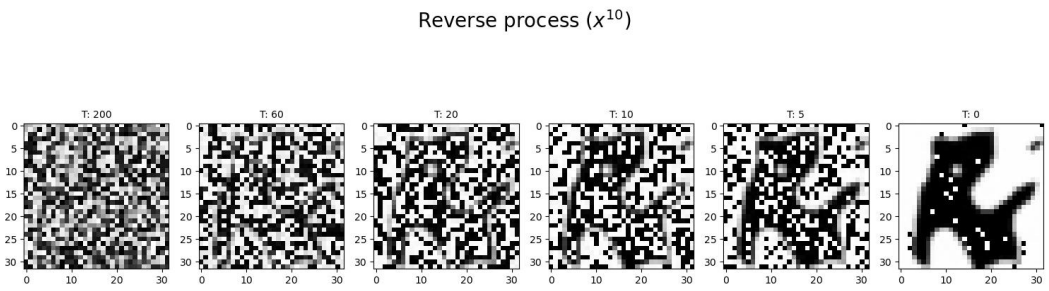
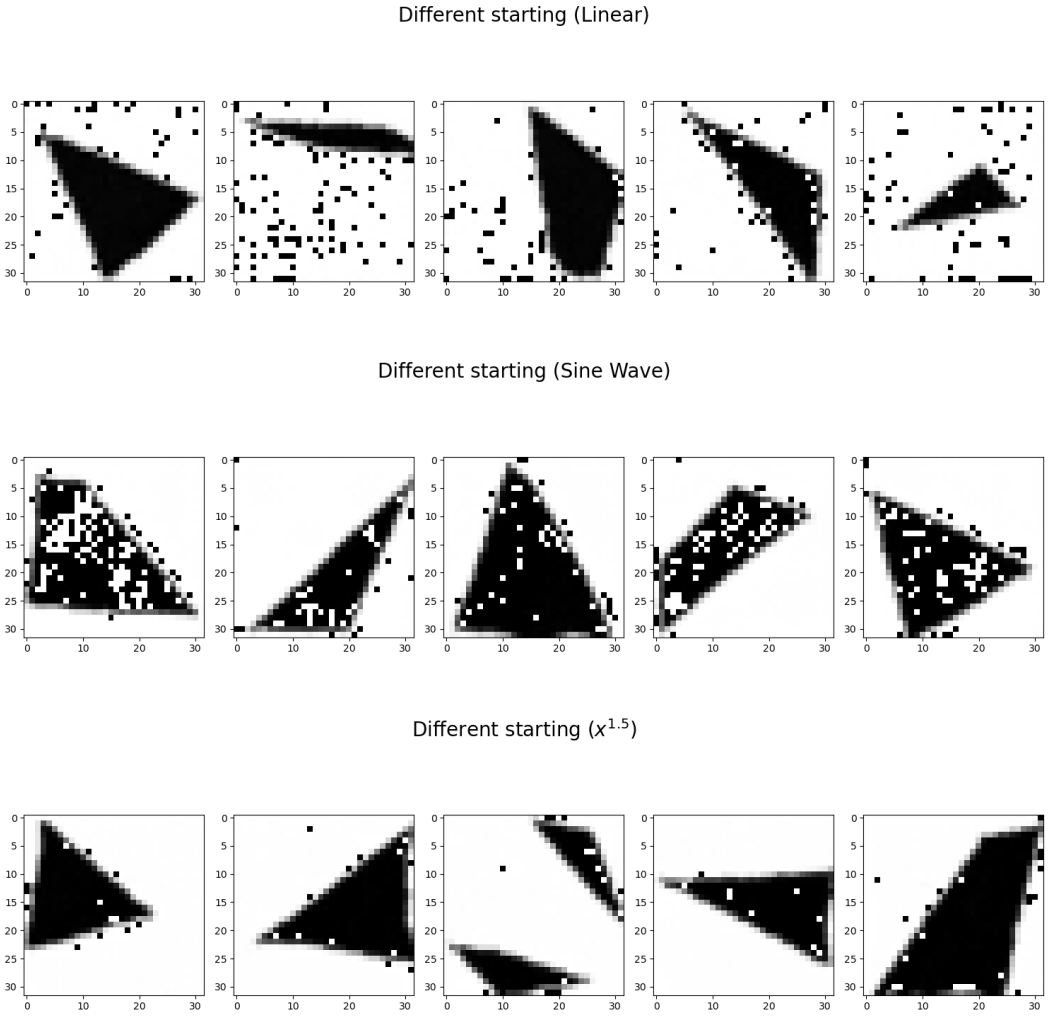


Figure 6.1: Denoising process after training with the set.

The results exhibit significant noise surrounding the generated triangles. To diversify our samples, in Fig. 6.2, we have plotted results using five distinct random noise patterns as starting points. This approach yields entirely new and distinctive outcomes each time, highlighting the influence of noise on the results. It's evident that the noise introduces variability and prevents a smooth, consistent result, indicating an opportunity for potential improvements.



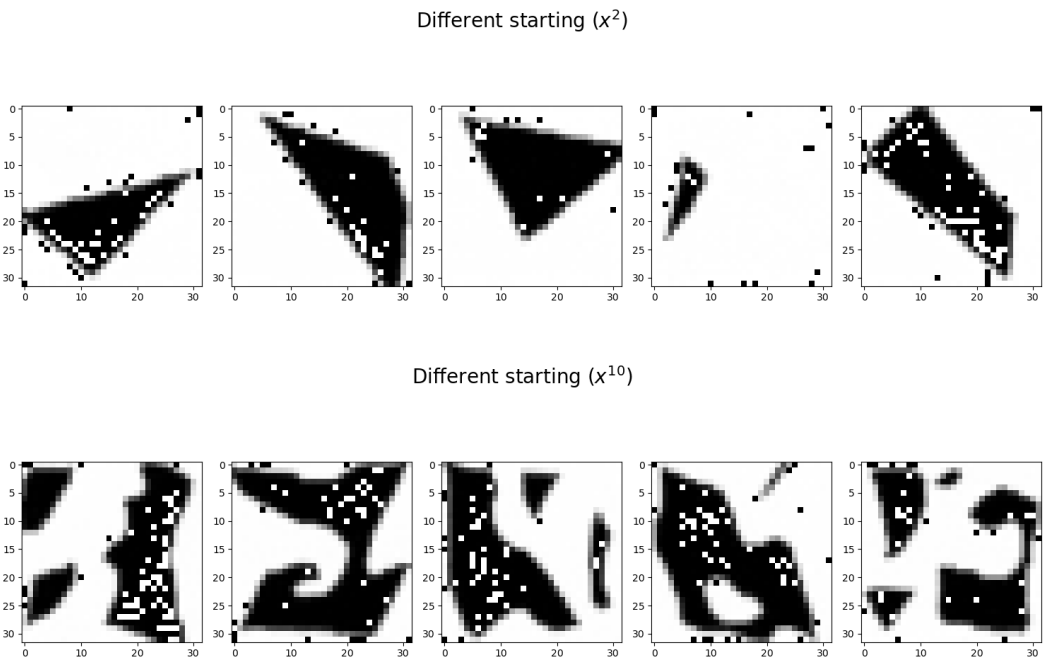
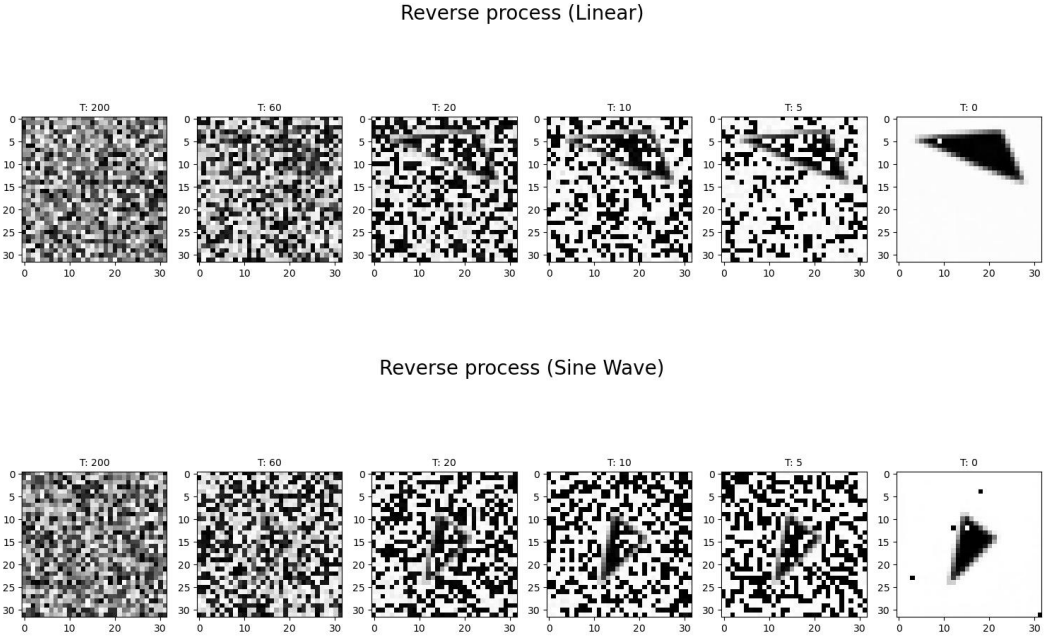


Figure 6.2: Denoised images for five different random noise as a starting point.

6.1.2 High resolution

By the final stage, the image has been transformed from a 720-pixel into a 32-pixel format, but with noise effectively mitigated. This transformation underscores a significant improvement in the results, particularly for the linear and wave schedules, where the improvements are most pronounced. The results after the denoising process are in Fig. 6.3.



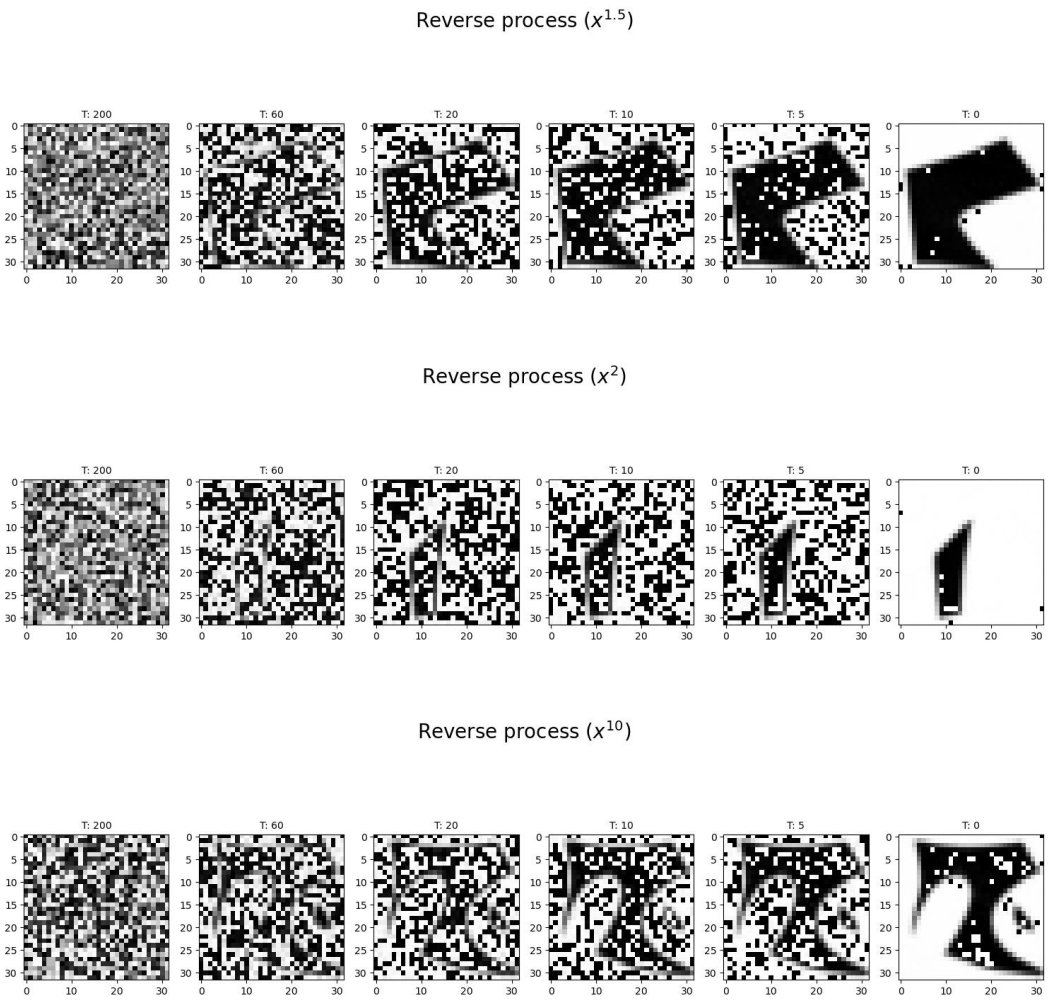
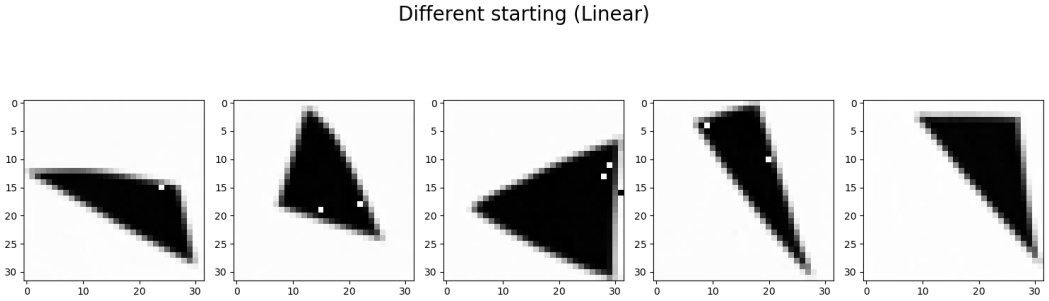


Figure 6.3: Reverse process using 720-pixel images aa training data.

Following the previous result, it was given five **randn** as a starting point, and from there, proceed with the reverse process, getting five different results as is shown in Fig. 6.4.



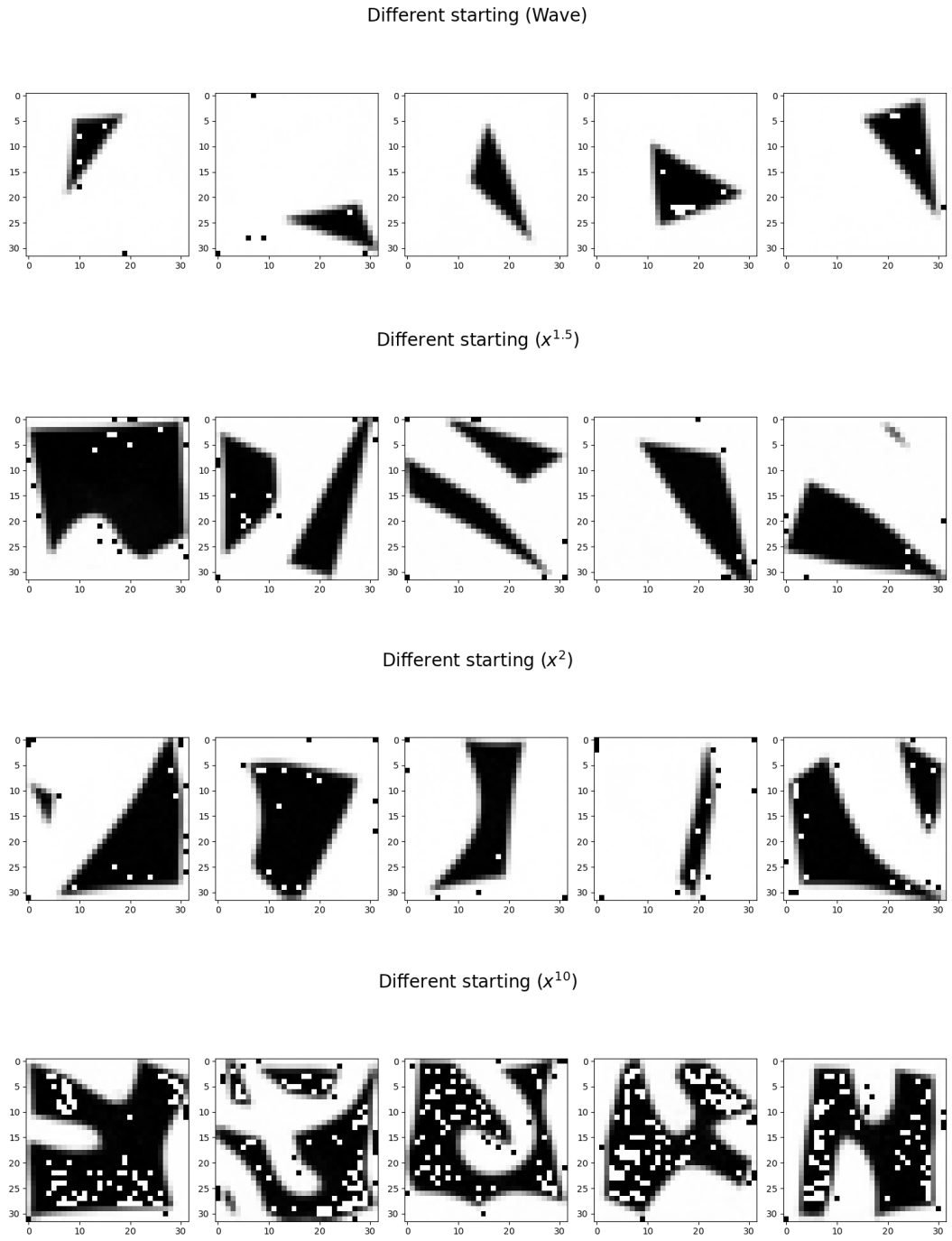


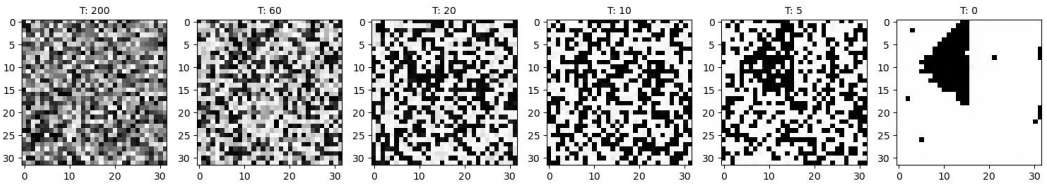
Figure 6.4: Denoised images for five different random noise as a starting point.

6.1.3 Array format

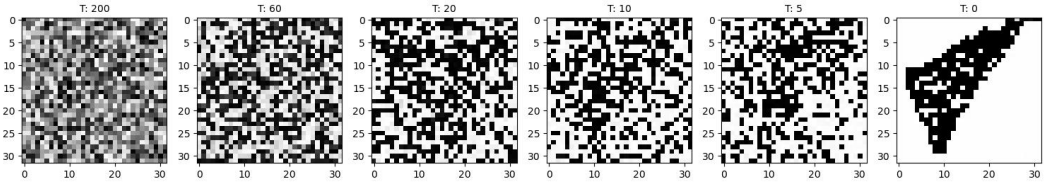
The last approach we thought to avoid the noise in the training set is to save the triangles in an array format, a 2-dimensional array that stores the points corresponding to black or white. A contra is we lose the smoothness of the figure, even though we did not have to resize the array to avoid the noise, the results for all the schedules are in Section 6.1.3.
In the high-power schedules, particularly in our case with a power of 10, the outcomes

were strikingly unexpected. This extreme schedule was employed to examine the effects when a substantial amount of noise is introduced towards the end of the diffusion process. The results serve as a clear illustration that the model lacks the necessary steps to effectively learn the denoising process.

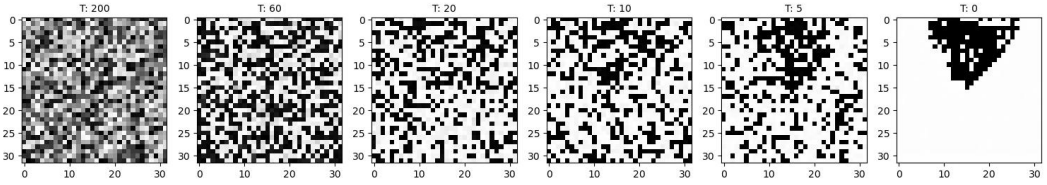
Reverse process (Linear)



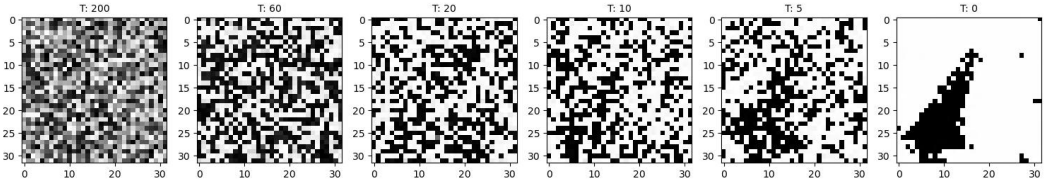
Reverse process (Wave)



Reverse process ($x^{1.5}$)



Reverse process (x^2)



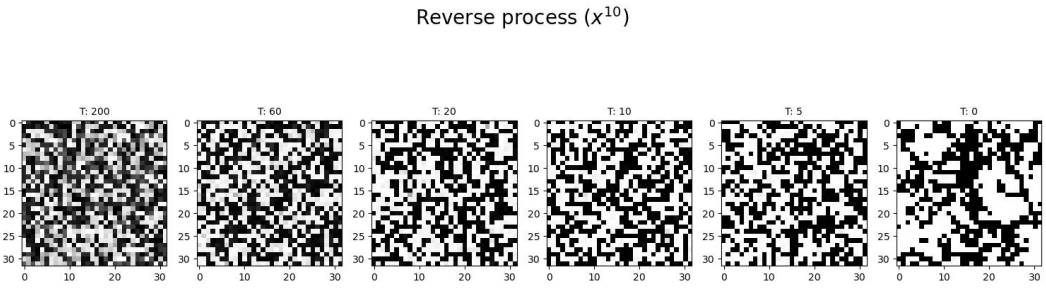
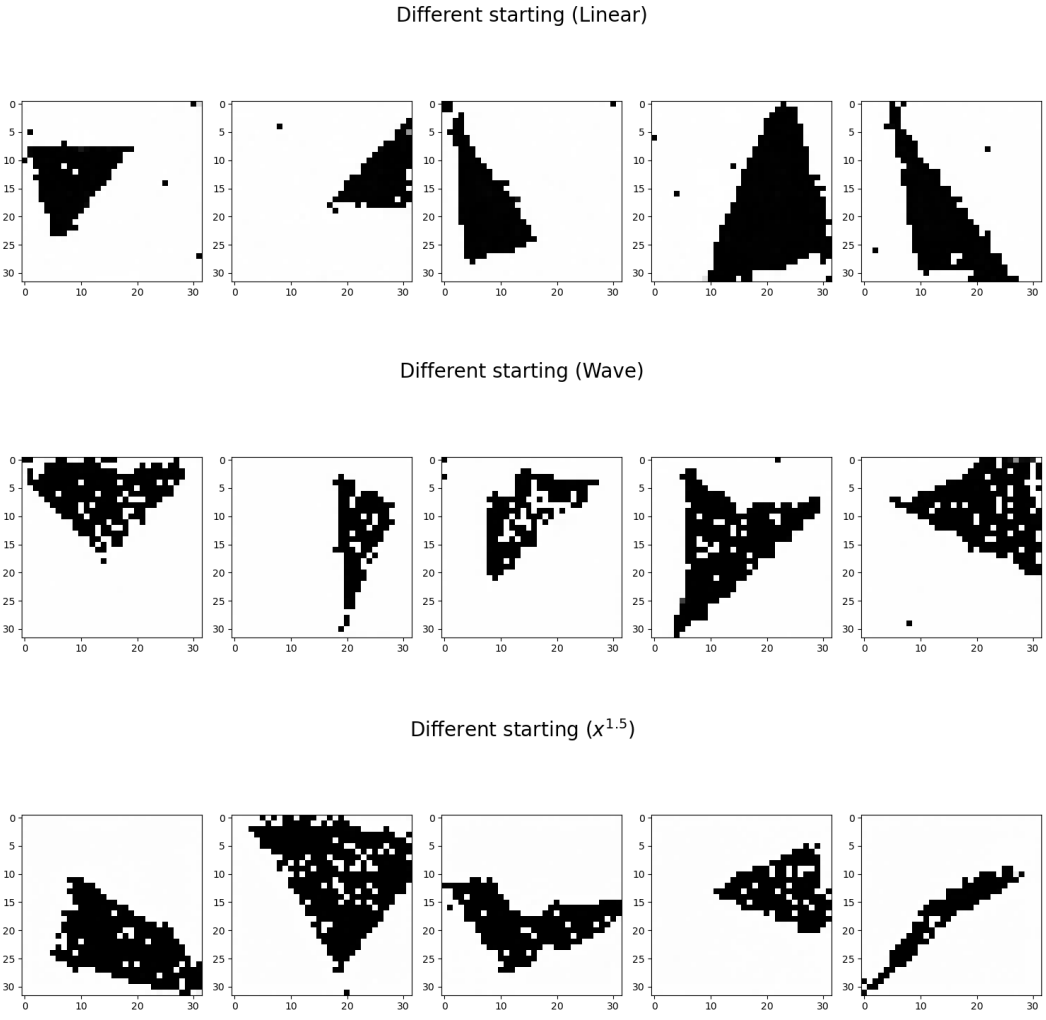


Figure 6.5: Denoised images for five different random noises as a starting point, using array as a training set.

Starting with five different random points, we got different results for each one Fig. 6.6; this result is unique for every new starting point.



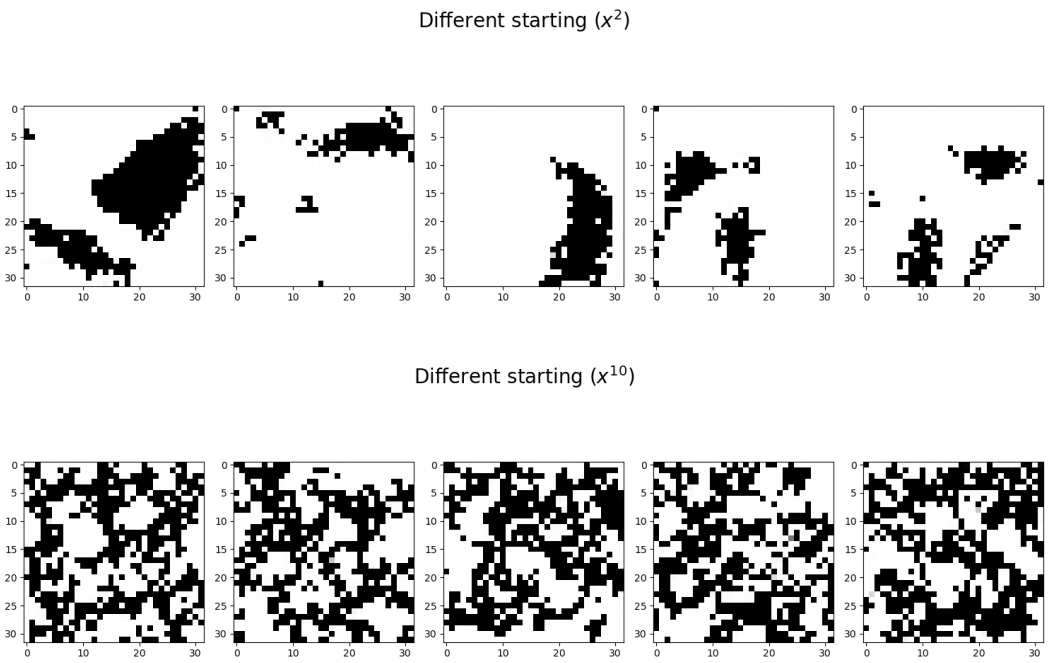


Figure 6.6: Denoised images for five different random noises as a starting point, with an array as a training set.

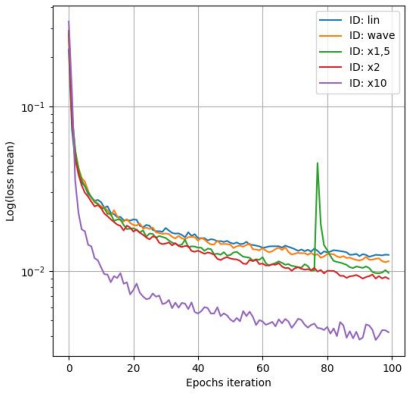
The results clearly demonstrate our successful efforts to mitigate noise in the source data. A compelling example is the comparison between the linear case in Fig. 6.1 and Section 6.1.3 and Fig. 6.3. It is evident that when we work directly with arrays, the training process yields a notably better resolution, but still, working with images without noise and in a higher resolution improves the results. This approach enhances the quality of the final results, emphasizing the advantages of working with raw arrays.

Working with high-resolution images, as we said before, helps to mitigate the noise produced by the ImageDraw package after the resizing to 32×32; this leads to a clean image for the training process such that the improvement can be visualized in Fig. 6.3, where the shape of the resulting triangles is pretty well defined for the **linear and wave** case. One crucial point to remark is that the training time taken in all three different sets is the same; then we would like to remark on the importance of choosing a good training set to improve the results.

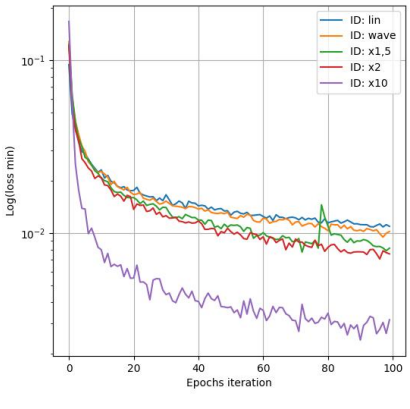
6.2 Loss function

The optimization process was governed by a loss function that predicted the noise through the Unet. This loss function was optimized using a specific learning rate, with the primary objective being stability. In other words, the aim was to achieve a point where further iterations would not significantly enhance the output quality.

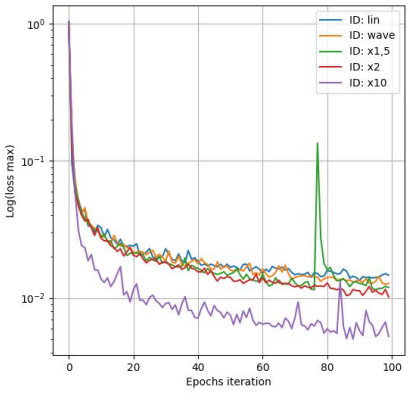
Fig. 6.7 provides an insight into the behavior of the loss function. It visually represents the mean, minimum, and maximum values of the loss function throughout the training



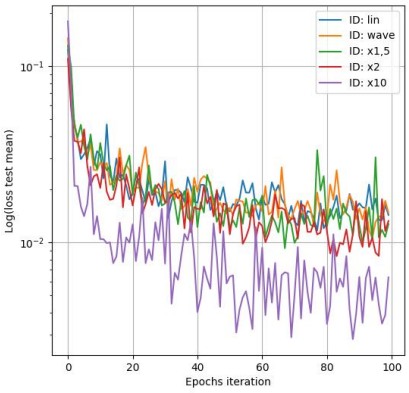
(a) Loss function mean value.



(b) Loss function min value.



(c) Loss function max value.



(d) Loss function with respect to the test set.

Figure 6.7: Loss function, mean, min, max and the comparison with a test set

process. The ultimate goal for optimal results is to stabilize the loss function. However, due to computational constraints, this stabilization process was performed for all schedules up to a maximum of 100 epochs of iterations. In the result gotten in the Fig. 6.7 is plotted the loss term of all the schedules, it is clear that the exponential ten times is the one in which the loss decreases faster, but this is not stable after the one hundred steps. On the other hand, the linear case stabilizes the noise, as seen in Fig. 6.7a. Then as the best result is the linear schedule, this will be extended to longer steps as is presented in Fig. 6.8.

If there is no limitation on the computational hardware this process could be extended for a longer time until it is below a fixed value. As a result, the data maintains visual coherence, making it well-suited for training our models.

As evident in Fig. 6.3 and Section 6.1.3, the outcome is notable. By the final stage, the image has been transformed back into a 32-pixel format, but with noise effectively mitigated. This transformation underscores a significant improvement in the results, partic-

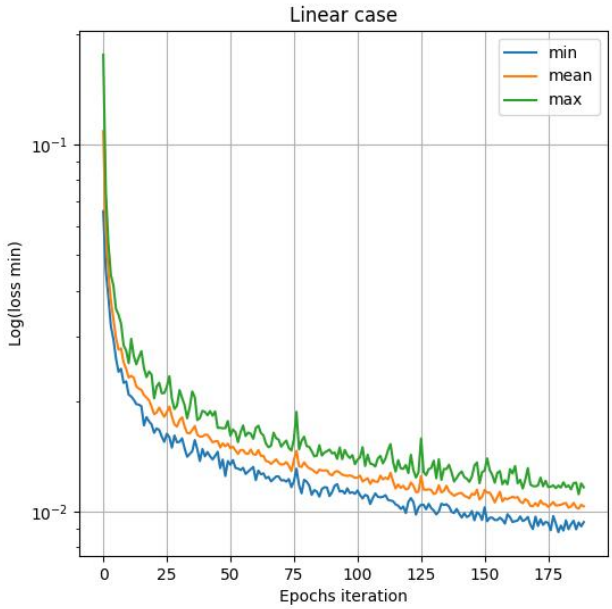


Figure 6.8: Loss function in the linear schedule up to 190 epoch steps, plotting the mean, min, and max value

ularly for the linear and wave schedules, where the improvements are most pronounced. On the other hand, using a *npy* is not necessary to resample because it is already in that size and has zero noise.
Similarly, the experiment involved initiating the process from five distinct random noise patterns. The results of this experiment are highly motivating as they successfully demonstrate the intended objective: the creation of entirely new data from noise.

In summary, we create two datasets, one in array format and another with higher resolution. To maintain the quality of the data and reduce imperfections, we convert the higher-resolution images into a smaller format while preserving the key features and minimizing the impact of noise around the figures.

7 Conclusions

The central objective of this thesis was to embark on a journey of image construction, using a seemingly rudimentary training set comprised of monochromatic triangles. This choice was driven by considerations of computational efficiency. With the successful implementation of our training code, we achieved an astounding feat – the generation of entirely novel triangle samples, each distinct from the original training dataset. This triumph underscores the remarkable capability of constructing fresh and innovative content through the utilization of a specific category of images.

One crucial revelation throughout this undertaking was the significance of selecting the appropriate training set. This choice, it turns out, can profoundly impact the quality of the results achieved. Remarkably, such improvements in result quality can be achieved with similar computational resources. We have demonstrated the marked differences in outcomes that arise from altering the training dataset. While our demonstrations were based on simple 32-bit images due to computational constraints, the potential for extension to higher resolutions with three-channel imagery for more intricate forms and colors is readily apparent. With slight adaptations, our methodology can be readily applied to these more complex scenarios.

Due to the scope of this work, an unexplored proposition is the application of denoising diffusion models to a Poisson distribution, in contrast to our exclusive focus on Gaussian distributions. The choice of Poisson holds particular allure, given its frequent occurrence in natural phenomena. This avenue represents a promising direction for future exploration in the realm of generative modeling.

It's worth noting that while this work may not entirely convey the countless hours invested in the coding and computational aspects of this research, it underscores the immense dedication and effort that underpin the realization of these results.

Bibliography

[Blo04] BLOOM, Dr. Jeremy Orloff J.: *Introduction To Probability And Statistics*. https://ocw.mit.edu/courses/18-05-introduction-to-probability-and-statistics-spring-2014/pages/syllabus/. Version: 2004. – MIT OpenCourseWare

[den22] *Diffusion Models | Paper Explanation | Math Explained*. YouTube video. https://www.youtube.com/watch?v=HoKDTa5jHvg. Version: 2022

[Gra90] GRAY, Robert M.: *Entropy and Information Theory*. https://ee.stanford.edu/~gray/it.pdf. Version: 1990. – Accessed on 2023-10-11

[GS18] GRIMMETT, Geoffrey ; STIRZAKER, David: *Probability and Random Processes*. Singapore : Springer, 2018. http://dx.doi.org/10.1007/978-981-13-3801-4. http://dx.doi.org/10.1007/978-981-13-3801-4. – ISBN 978–981–13–3801–4

[HJA20] HO, Jonathan ; JAIN, Ajay ; ABBEEL, Pieter: *Denoising Diffusion Probabilistic Models*. 2020

[Ken16] KENNEDY, Tom: *Monte Carlo Methods - a special topics course*. April 2016

[KW22] KINGMA, Diederik P. ; WELLING, Max: *Auto-Encoding Variational Bayes*. 2022

[PP] PETERSEN, Kaare B. ; PEDERSEN, Michael S.: *The Matrix Cookbook*. http://matrixcookbook.com, . – Version: November 15, 2012

[Rao11] RAO, Anup: *CSE599s: Extremal Combinatorics October 3, 2011 - Lecture 2: Jensen's Inequality and Probabilistic Magic*. https://homes.cs.washington.edu/~anuprao/pubs/CSE599sExtremal/lecture2.pdf. Version: 2011. – Accessed on [Insert Access Date]

[RC04] ROBERT, Christian P. ; CASELLA, George: *Monte Carlo Statistical Methods*. New York, Ny Springer New York, 2004

[RFB15] RONNEBERGER, Olaf ; FISCHER, Philipp ; BROX, Thomas: *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015

[Rin12] RINCÓN, Luis: *Introducción a los procesos estocásticos*. Primera edición. Prensas de Ciencias, 2012. – 328 S. – ISBN 9786070230448

[SDWMG15] SOHL-DICKSTEIN, Jascha ; WEISS, Eric A. ; MAHESWARANATHAN, Niru ; GANGULI, Surya: *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015

[Shi01] SHIRYAEV, Albert: *Probability-1*. Berlin : Springer, 2001

[WGBS23] WILDBERGER, Jonas ; GUO, Siyuan ; BHATTACHARYYA, Arnab ; SCHOLKOPF, Bernhard: On the Interventional Kullback-Leibler Divergence. In: *Proceedings of Machine Learning Research* Bd. 213. Tubingen, Germany, 2023, S. 1–22

[ZLC+21] ZHANG, Yufeng ; LIU, Wanwei ; CHEN, Zhenbang ; LI, Kenli ; WANG, Ji: On the Properties of Kullback-Leibler Divergence Between Gaussians. In: *CoRR* abs/2102.05485 (2021). https://arxiv.org/abs/2102.05485