Technical University of Applied Sciences Würzburg-Schweinfurt (THWS)
Faculty of Computer Science and Business Information Systems

Master Thesis

Loss Functions in Diffusion Models: A Comparative Study

Submitted to the Faculty of Computer Science and Business Information Systems at the Technical University of Applied Sciences Würzburg-Schweinfurt for the completion of a degree program in Master of Artificial Intelligence

Dibyanshu Kumar

Submitted on: May 15, 2025

Primary Supervisor: Prof. Dr. Magda Gregorová
Second Supervisor: Prof. Dr. Andreas Lehrmann

Abstract (en)

Diffusion models have established themselves as highly effective generative frameworks, inspiring significant research into their underlying mechanisms. An important aspect of these models lies in the choice of loss functions, which directly influences their training and performance. Over recent years, multiple formulations have been proposed [20, 56, 27, 49], each with distinct characteristics and theoretical foundations. This thesis provides a comprehensive exploration of these loss functions, systematically analyzing their theoretical relationships and unifying them under the framework of the variational lower bound objective. We complement this analysis with empirical studies that examine the conditions under which different objectives yield varying performance and provide insights into the factors driving these discrepancies. Additionally, we assess the impact of loss function selection on the model's ability to achieve specific objectives, such as producing high-quality samples or precisely estimating data likelihoods. By presenting a unified perspective, this study advances the understanding of loss functions in diffusion models, contributing to more efficient and goal-oriented model designs in future research.

Abstract (de)

Diffusionsmodelle haben sich als hochwirksame generative Frameworks etabliert und zu bedeutender Forschung zu ihren zugrunde liegenden Mechanismen geführt. Ein wichtiger Aspekt dieser Modelle liegt in der Wahl der Verlustfunktionen, die ihr Training und ihre Leistung direkt beeinflussen. In den letzten Jahren wurden verschiedene Formulierungen vorgeschlagen [20, 56, 27, 49], jede mit unterschiedlichen Merkmalen und theoretischen Grundlagen. Diese Arbeit bietet eine umfassende Untersuchung dieser Verlustfunktionen, analysiert systematisch ihre theoretischen Zusammenhänge und vereinheitlicht sie im Rahmen des Ziels der variationellen Untergrenze. Wir ergänzen diese Analyse durch empirische Studien, die die Bedingungen untersuchen, unter denen unterschiedliche Ziele zu unterschiedlichen Leistungen führen, und Einblicke in die Faktoren geben, die diese Diskrepanzen verursachen. Darüber hinaus bewerten wir den Einfluss der Wahl der Verlustfunktion auf die Fähigkeit des Modells, bestimmte Ziele zu erreichen, wie z. B. die Erstellung qualitativ hochwertiger Stichproben oder die präzise Schätzung von Datenwahrscheinlichkeiten. Durch die Darstellung einer einheitlichen Perspektive fördert diese Studie das Verständnis von Verlustfunktionen in Diffusionsmodellen und trägt zu effizienteren und zielorientierteren Modelldesigns in der zukünftigen Forschung bei.

Acknowledgment

I sincerely thank everyone who supported and guided me throughout the completion of this thesis. First and foremost, I extend my gratitude to my supervisor, Prof. Dr. Magda Gregorová, for providing me with the opportunity to explore my interest in generative modeling and diffusion models. Her guidance and advice have been very useful in shaping both my research and my scientific writing.

I would also like to thank Philip Väth for his suggestions and advice. Additionally, I appreciate CAIRO for providing the computational resources that enabled me to conduct my experiments seamlessly.

On a personal note, I am thankful to my family who support me in my studies, even from afar. Their belief in me sustained my motivation during challenging times. I am grateful to my friends, especially Bhumika, for her continued moral support which have been a source of encouragement and comfort.

Contents

Notations

a, ϵ, x, y	scalars
$\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{x}$, $\boldsymbol{y}$	vectors
$\mathbf{a}$, $\boldsymbol{\epsilon}$, $\mathbf{x}$, $\mathbf{y}$	random vectors
$\boldsymbol{A}$, $\boldsymbol{B}$	matrices
$a_{i\cdot}$, $a_{\cdot j}$, a_{ij}	i-th row, j-th column, and (i,j)-th element of matrix $\boldsymbol{A}$
$\boldsymbol{I}$	identity matrix
$\boldsymbol{\Sigma}$	covariance matrix
$\mathcal{N}(\mathbf{0}, \boldsymbol{I})$	multivariate standard normal distribution
$p_{\boldsymbol{\theta}}(\mathbf{x})$	probability distribution of $\mathbf{x}$ parameterized by $\boldsymbol{\theta}$
Z	normalizing constant
$E_{\boldsymbol{\theta}}$	energy function parameterized by $\boldsymbol{\theta}$
D	dimension of the data
d	dimension of the latent variable in AEs
$\mathbb{R}$, $\mathbb{R}^{\mathrm{D}}$	real numbers and D-dimensional real vectors
G, D	generator and discriminator
D_{KL}	KL divergence
$\det(\boldsymbol{A})$	determinant of the matrix $\boldsymbol{A}$
$\boldsymbol{A}^\top$	transpose of matrix $\boldsymbol{A}$
$\mathrm{Tr}(\boldsymbol{A})$	trace of the square matrix $\boldsymbol{A}$
$\lvert\det(\boldsymbol{A})\rvert$	absolute value of the determinant of matrix $\boldsymbol{A}$
t	timestep
T	total number of timesteps
$\frac{\partial f}{\partial x}$	partial derivative of f with respect to x
$\odot$	element-wise multiplication
$\sum(\cdot)$	summation of elements
$\lVert\mathbf{a}\rVert$	ℓ_2-norm of vector $\mathbf{a}$
$\mathbb{E}[\cdot]$	expectation of a random variable
$\nabla_{\mathbf{x}}$	gradient with respect to data $\mathbf{x}$
$\nabla_{\boldsymbol{\theta}}$	gradient with respect to parameters $\boldsymbol{\theta}$
$\mathbf{u}$	random projection direction (random unit vector)
$\mathrm{U}\{1, \mathrm{T}\}$	discrete uniform distribution between 1 and T
$\mathcal{U}[0, 1]$	continuos uniform distribution over [0,1]
L	NELBO loss formulation
$\mathcal{L}$	weighted loss formulation
$\tilde{\mathcal{L}}$	rescaled loss formulation

Acronyms

GMM	Gaussian Mixture Models
HMM	Hidden Markov Models
DNN	Deep Neural Network
AE	Autoencoder
VAE	Variational Autoencoder
VQVAE	Vector Quantized Variational Autoencoder
HVAE	Hierarchical Variational Autoencoders
MHVAE	Markovian Hierarchical Variational Autoencoders
GAN	Generative Adversarial Network
DCGAN	Deep Convolutional Generative Adversarial Network
SNGAN	Spectral Normalization Generative Adversarial Network
WGAN-GP	Wasserstein Generative Adversarial Network-Gradient Penalty
DDPM	Denoising Diffusion Probabilistic Models
DDPM	Improved Denoising Diffusion Probabilistic Models
ADM	Ablated Diffusion Models
VDM	Variational Diffusion Model
SNR	Signal-to-Noise Ratio
ELBO	Evidence Lower Bound
NELBO	Negative Evidence Lower Bound
EBM	Energy Based Model
MCMC	Markov Chain Monte Carlo
CNN	Convolutional Neural Network
KLD	Kullback-Leibler Divergence
FID	Frechet Inception Distance
VP	Variance-Preserving
VE	Variance-Exploding
DSM	Denoising Score Matching
SSM	Sliced Score Matching
ReLU	Rectified Linear Unit
SiLU	Sigmoid Linear Unit
SDE	Stochastic Differential Equation

1 Introduction

In this section we begin by providing an overview of the context and background of diffusion models, followed by the motivation behind this study and the specific research focus areas.

1.1 Context

There is a very famous quote by the great physicist Richard Feynman that states *"What I cannot create, I do not understand"* [13]. While he referred to understanding physical systems or mathematical proofs, this analogy applies to generative models in the sense that if a model understands a data distribution, it can produce new samples from it. In more technical terms, if a model approximates the true data distribution using techniques like adversarial training or probabilistic modeling, it can generate new outputs, which is one of the main task in generative modeling.

The roots of generative modeling lie in statistical methods, such as histogram-based density estimation. These methods estimate data densities by dividing the data space into discrete bins, and samples can be drawn using techniques like the inverse cumulative distribution function. However, because of their simplicity, histogram-based methods struggle to generalize in sparsely populated regions and become impractical in high-dimensional spaces due to the curse of dimensionality.

Other classical methods for modeling the data distribution are Gaussian mixture models (GMMs) [7], which approximate the data distribution by representing it as a weighted sum of Gaussian components and therefore flexibly model multimodal distributions. Hidden Markov model (HMM) [43] is another technique used in modeling sequential data, for example speech and time series, by assuming that the data is generated from a series of latent states and follows a Markov process. Though these models are powerful in specific contexts, they rely on strong statistical assumptions about the underlying data and are computationally very expensive to scale.

In the last decade, there have been significant advancements in generative modeling because of the increase in computational power and the ability to leverage the strengths

of deep neural networks (DNNs). DNN, with their capacity to learn representations of data, replaced handcrafted features and statistical approximations with end-to-end learning. This shift improved the performance of models and made it possible for generative modeling to address more complex and high-dimensional datasets.

One of the very earliest approaches in generative modeling is the autoencoders (AEs) [18]. They compress data into a lower dimensional latent space using an encoder and then reconstruct it with a decoder. While AEs are very good at capturing the representation of data in a low-dimensional latent space, they struggle to generate new samples. Following that, Kingma et al.[29] introduced Variational Autoencoders (VAEs) that enhanced this framework by using probabilistic principles, smooth interpolations, meaningful latent representations, and efficient sampling in the latent space. Generative Adversarial Networks (GANs) [14] soon followed and transformed the field of generative modeling. By framing the problem as an adversarial game between a generator and a discriminator, GANs achieved significant breakthroughs in generating high-quality images. Subsequent GAN variants further refined the architecture, producing remarkably realistic outputs and setting new benchmarks for generative tasks across various domains.

In parallel, autoregressive models also emerged as another powerful class of generative models. Approaches like PixelCNN [64], Gated PixelCNN [40], and later transformers [65], modeled data by predicting each element sequentially, conditioned on its predecessors. For example, in the case of images, they generate pixels one at a time, with each pixel depending on those previously generated. These models are highly effective mostly in sequence generation tasks such as text generation where capturing conditional dependencies is important. However, their primary limitation lies in their strict sequential dependence, which may not accurately represent all types of data.

Amidst the rapid evolution of generative modeling, diffusion models [20, 27, 56] have gained prominence as a powerful alternative to GANs, VAEs, and autoregressive models. The principles of diffusion models are rooted in stochastic processes that iteratively refine random noise into structured data. Inspired by concepts from non-equilibrium thermodynamics [52], diffusion models learn to reverse a noise injection process to generate new samples from a learned data distribution. This approach is conceptually very robust and overcomes several challenges faced by earlier generative models, such as mode collapse in GANs and blurry reconstructions in VAEs.

In the last few years diffusion models have become very popular due to their versatility and ability to generate high-quality outputs. Extensive research has focused on enhancing these models and extending their applications beyond image generation into various domains. However, even with these advancements, some questions remain about their underlying mechanisms, leaving room for further exploration of their fundamental principles.

One such area of ambiguity lies in the design and implementation of loss functions. Various approaches have been proposed to model the reverse process, each using different target objectives and loss formulations. Although researchers have drawn theoretical links between these loss functions, there is no systematic study that demonstrate their equivalence. Additionally, there is a lack of experimental comparisons that clarify their differing behaviors or provide a clear justification for selecting one loss function over another based on specific objectives.

This thesis focuses on the role of loss functions in diffusion models. By systematically exploring and comparing different loss formulations, we aim to elucidate their effects on training stability, convergence, and sample quality. The study begins with a mathematical analysis of these losses, unifying their theoretical origins under a single framework, followed by empirical evaluations to assess their practical implications. Through this research we try to bridge the existing knowledge gaps and provide a clearer understanding of how the choice of loss function effects the performance of diffusion models. Moreover, we aim to establish a criteria for selecting the most appropriate objective based on specific goals, such as likelihood estimation or high-quality sample generation.

1.2 Motivation and research area

In this research we do a comparative study of various loss formulations in diffusion models, aiming to address critical gaps in understanding their theoretical and empirical differences. Diffusion models have achieved remarkable success in generative modeling due to their ability to generate high-quality data samples across several domains. The performance of these models stems primarily from the design and refinement of training objectives, which have evolved significantly over time.

There have been numerous notable contributions in diffusion models. Starting with score-based models, as introduced by Song et al. [56], where the reverse diffusion process is modeled by optimizing a denoising score-matching objective. Ho et al. [20] proposed Denoising Diffusion Probabilistic Models (DDPM), which redefined the objective by predicting noise ($\boldsymbol{\epsilon}$) during training, achieving better results in image generation. Subsequently, Variational Diffusion Models (VDM) by Kingma et al. [27] introduced a training objective in terms of Signal-to-Noise Ratio (SNR) which gave good results for likelihood estimation. More recent developments, such as Progressive Distillation [49], have further refined loss functions by jointly modeling the data representation ($\mathbf{x}$) and noise ($\boldsymbol{\epsilon}$), significantly improving efficiency by reducing the number of sampling steps.

Existing research has explored the theoretical equivalence of various training objectives used in diffusion models. For instance, [58] made connections between score matching and diffusion-based generative frameworks by using stochastic differential equations

to model the forward process, thereby aligning it with continuous distributions that evolve over time. Similarly, [27] used the evidence lower bound (ELBO) objective for diffusion, inspired by Variational Autoencoders [28]. More recently, [26] demonstrated that diffusion model objectives are fundamentally equivalent and closely related to the ELBO framework. However, while these works highlight the theoretical equivalence of the loss functions, they lack a structured analysis of their formulations under a single framework. Moreover, whether the theoretical equivalences among objectives translate into consistent performance during the training of deep neural networks remains largely unexplored.

In this study, we conduct a comprehensive comparison of training objectives, formulated for four different target predictions of the diffusion models: data $\mathbf{x}$, noise $\boldsymbol{\epsilon}$, rate of change in the data distribution $\mathbf{v}$, and score $\mathbf{s}$. We derive the negative ELBO loss in terms of these targets and establish mathematical relationships with the most commonly used diffusion loss functions. These relationships help us to design experiments that evaluate whether the theoretical equivalence between these objectives holds in practice when used for training over the same datasets. Our experiments highlight the differences and similarities in the theoretical foundations and practical behavior of these loss functions, particularly in terms of loss convergence during training and the quality of generated samples. We explore the loss behavior across different diffusion timesteps, providing insights into the mechanisms that drive their performance and functionality. Additionally, we compare the outcomes of these training objectives in terms of data density estimation and sample quality, summarizing with a comprehensive understanding of their roles in optimizing diffusion models.

By providing a structured comparison of loss formulations, this thesis aims to deepen the understanding of their practical implications and guide the development of more effective and efficient diffusion models. The findings are expected to contribute to the broader class of diffusion models by offering insights into the connections between theoretical design and empirical performance.

2 Background

Our work is centered around diffusion models, which fall under the broader class of generative modeling. In this chapter, we look into the basic concepts of generative modeling. We explore the major types of generative models, their practical applications, and their inherent limitations. Furthermore, we discuss the motivation behind diffusion-based approaches and analyze the family of models to which they are closely related.

2.1 Generative modeling

Generative Modeling is a branch of machine learning that is used to create models which can generate new data samples resembling a given dataset. Unlike discriminative models, which aim to classify or predict labels based on input data, the basic idea in generative models is to estimate the probability distribution of the data and then sample from that in order to produce new, realistic samples. This ability makes them powerful tools for tasks such as image synthesis [14, 28, 63, 20], text generation [45, 8, 30], music generation [62, 41, 68], and more.

In statistics and machine learning, we often assume that the data points $\mathbf{x} \in \mathbb{R}^{\mathrm{D}}$ in a dataset are independent and identically distributed samples drawn from an underlying data distribution $p(\mathbf{x})$, where D is the dimensionality of the data. However, we typically lack the exact analytical form of $p(\mathbf{x})$. To estimate this distribution, we construct a model that represents a parameterized probability distribution, referred to as the model distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$, where $\boldsymbol{\theta}$ represent the model parameters. By tuning these parameters, our goal is to ensure that $p_{\boldsymbol{\theta}}(\mathbf{x})$ closely approximates $p(\mathbf{x})$. When $p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p(\mathbf{x})$, the model can be used to generate new data points by sampling from $p_{\boldsymbol{\theta}}(\mathbf{x})$. Moreover, we can use it to compute the likelihood for any given data point.

A simple way to model the data distribution $p(\mathbf{x})$ is to approximate it using a Gaussian distribution. While a Gaussian distribution is inherently simple and may not accurately capture the complexity of $p(\mathbf{x})$, it serves as a good starting point for modeling. We can define a simple Gaussian model as $p_{\boldsymbol{\mu}}(\mathbf{x}) = \frac{1}{(2\pi)^{\mathrm{D}/2}} \exp\left(-\frac{1}{2}\|\mathbf{x} - \boldsymbol{\mu}\|^2\right)$, where $\boldsymbol{\mu} \in \mathbb{R}^{\mathrm{D}}$ is the tunable mean parameter, and the covariance is fixed to the identity matrix. This model can be represented as a computational graph with two layers: the first layer

represents the input data point $\mathbf{x}$, and the second layer consists of a single unit that computes the probability density function of the Gaussian distribution as illustrated in fig. 2.1.

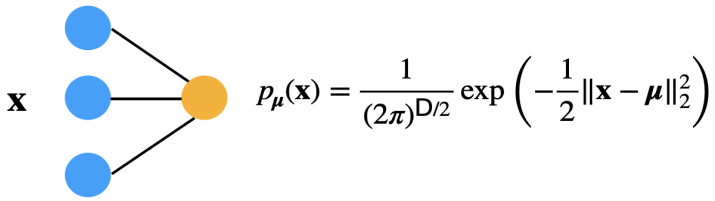


Figure 2.1: Computation graph for modeling $p(\mathbf{x})$ as a simple Gaussian distribution. The first layer represents the input $\mathbf{x} \sim p(\mathbf{x})$, and second layer with single unit that computes the probability density of the Gaussian distribution, where mean $\boldsymbol{\mu}$ is the tunable parameter.

As discussed, Gaussian models are often too simple to capture the complexity of $p(\mathbf{x})$. To address this, we can use DNN, which allows us to model more complex and flexible probability distributions. However, the raw output from our DNN, denoted as $f_{\boldsymbol{\theta}}(\mathbf{x})$, where $\boldsymbol{\theta}$ represent the network parameters, is not a valid probability density. To transform this output into a proper probability density, we first apply the exponential function to ensure it is positive for all $\mathbf{x}$ and then we normalize it by dividing by a constant $Z_{\boldsymbol{\theta}} = \int \exp(f_{\boldsymbol{\theta}}(\mathbf{x}))\, d\mathbf{x}$, to ensure the result integrates to one, as illustrated in fig. 2.2. The final output is given by,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(f_{\boldsymbol{\theta}}(\mathbf{x}))/Z_{\boldsymbol{\theta}} \tag{2.1}$$

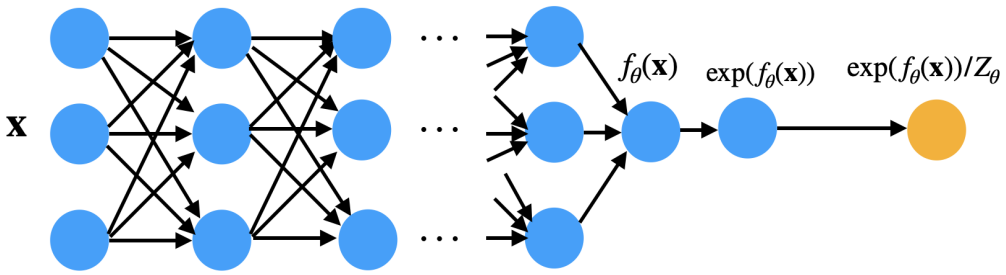


Figure 2.2: Computation graph for modeling $p(\mathbf{x})$ as a DNN with n hidden layers. The raw output $f_{\boldsymbol{\theta}}(\mathbf{x})$ is passed through an exponential function and then normalized in order to get a valid probability density.

2.2 Intractability of the normalizing constant

A significant challenge in modeling the data using DNN is that the normalizing constant $Z_{\boldsymbol{\theta}}$ involves integrating a highly complex function $\exp(f_{\boldsymbol{\theta}}(\mathbf{x}))$. Due to the non linearity of $f_{\boldsymbol{\theta}}(\mathbf{x})$ and the high dimensionality of $\mathbf{x}$, this integral is generally intractable, making direct computation impractical. Researchers have come up with several strategies for tackling the intractability of $Z_{\boldsymbol{\theta}}$ and can be broadly classified into three major categories as discussed below.

2.2.1 Approximating the normalizing constant

One of the approach is to use energy based models (EBM) [1] where we define an energy function $E_{\boldsymbol{\theta}}(\mathbf{x})$ as a neural network parameterized by $\boldsymbol{\theta}$. The energy function maps each data point $\mathbf{x}$ to a scalar value, where a lower value of $E_{\boldsymbol{\theta}}(\mathbf{x})$ corresponds to a higher likelihood of the sample, while higher value indicates a lower likelihood. The model distribution is given as,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(-E_{\boldsymbol{\theta}}(\mathbf{x}))/Z_{\boldsymbol{\theta}} \tag{2.2}$$

A key advantage of EBMs is their flexibility, we can design $E_{\boldsymbol{\theta}}$ without any constraints that enables the model to capture complex data distributions. However the issue of intractability of $Z_{\boldsymbol{\theta}}$ still persists because of which we cannot train EBMs by typically maximizing the likelihood estimation and hence alternative methods are used.

A widely used method is contrastive divergence, introduced in [19]. This approach approximates the gradient of the negative log-likelihood by comparing the energy of samples from $p(\mathbf{x})$ with the energy of samples drawn from $p_{\boldsymbol{\theta}}(\mathbf{x})$. The gradient is expressed as,

$$\nabla_\theta \mathcal{L}_{\text{MLE}}(\theta) = \mathbb{E}_{p(\mathbf{x})}\left[\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})\right] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\mathbf{x})}\left[\nabla_{\boldsymbol{\theta}} E_{\boldsymbol{\theta}}(\mathbf{x})\right] \tag{2.3}$$

This gradient updates the energy function to assign lower energy to real data points and higher energy to samples generated by the model that deviate from the true distribution. Sampling from $p_{\boldsymbol{\theta}}(\mathbf{x})$ is done using Markov Chain Monte Carlo (MCMC), such as langevin dynamics (discussed in section 3.2.4). Starting from a sample drawn from $p(\mathbf{x})$, a small number of MCMC steps are run to produce a sample that approximates $p_{\boldsymbol{\theta}}(\mathbf{x})$, allowing the model to estimate the gradient without directly computing $Z_{\boldsymbol{\theta}}$. This gradient is then used to update the energy function, i.e. assign lower energy to real data and higher energy to unlikely samples. While this approach is computationally efficient, the reliance on a limited number of MCMC steps introduces approximation errors, which prevents the model from fully converging to the true data distribution.

2.2.2 Restricting the neural network models

Another strategy is designing neural network architectures in such a way that the normalizing constant is tractable by construction. For example autoregressive models like PixelCNN [64], which factorize the joint distribution into a product of conditionals and therefore simplifies the likelihood calculation as given in equation (2.4).

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \prod_{i=1}^{\mathrm{D}} p_{\boldsymbol{\theta}}(\mathbf{x}_i \mid \mathbf{x}_{1:i-1}) \tag{2.4}$$

Here, each conditional probability is modeled by a neural network. Since all of them are normalized individually (i.e. integrates to 1), the full joint probability is also normalized. This factorization simplifies likelihood calculation but can introduce sequential dependencies that affect efficiency.

Normalizing flows [10] also comes in this category. They maintain tractability by defining a deterministic, invertible transformation between a known distribution $p(\mathbf{z})$ and the data distribution $p(\mathbf{x})$ denoted as $\mathbf{x} = f_{\boldsymbol{\theta}}(\mathbf{z})$, where $\mathbf{z} \sim p(\mathbf{z})$. Here, $f_{\boldsymbol{\theta}}(\mathbf{z})$ is an invertible function with computable Jacobian determinant. Using the change of variables formula $p_{\boldsymbol{\theta}}(\mathbf{x})$ is computed as,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = p(f_{\boldsymbol{\theta}}^{-1}(\mathbf{x})) \cdot \left| \det \left(\frac{\partial f_{\boldsymbol{\theta}}^{-1}(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \tag{2.5}$$

While normalizing flows ensures exact density computation, they impose architectural constraints, requiring transformations that are bijective and therefore limiting the use of arbitrary deep networks. It creates a tradeoff between expressiveness and efficiency as more expressive transformations can be computationally expensive or harder to invert, whereas simpler ones may restrict model flexibility.

Variational Autoencoder (VAE) [29] is another way in which we model $p(\mathbf{x})$ indirectly through latent variable $\mathbf{z}$, therefore optimizing the tractable lower bound to the data log likelihood also known as evidence lower bound (ELBO) that is given by,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) \right] - D_{\mathrm{KL}} \left(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z}) \right) \tag{2.6}$$

where $q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x})$ is the approximate posterior, $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ is our generative model and $p(\mathbf{z})$ is the known prior of the latent variable. Since diffusion models are closely related to VAEs we discuss it in detail in section 2.4.2.

2.2.3 Modeling the generation process

A third approach avoids modeling the density function explicitly and instead focuses on the generation process. Generative Adversarial Networks (GANs) [14] train two

neural networks, a generator G and a discriminator D in an adversarial setup. The generator takes random noise $\mathbf{z} \sim p(\mathbf{z})$, generally from a gaussian distribution, and produces synthetic samples which are close to the true data distribution $p(\mathbf{x})$, while the discriminator evaluates if samples are real or fake. The training objective of GAN is expressed as a minimax game and is given as,

$$\min_G \max_D V(G, D) = \mathbb{E}_{p(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{p(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))] \tag{2.7}$$

where $p(\mathbf{z})$ is the prior distribution of the generator's input noise and $G(\mathbf{z})$ generates synthetic data samples. The discriminator is trained to maximize the probability of correctly classifying real and fake samples, while the generator is trained to minimize this objective, by fooling the discriminator.

In previous research [3, 25] it is shown that GANs are able to produce high-quality outputs, like realistic images, with variants such as DCGANs [44] and Conditional GANs [34] enhancing their capabilities. However, training GANs is challenging due to instability and mode collapse, where the generator fails to produce diverse samples. While GANs are good at generating high quality samples, they do not explicitly define a density function and cannot provide meaningful likelihood estimates for data points. This makes them less suitable for tasks requiring explicit probability modeling.

2.3 Framework for a good generative model

A good framework for generative modeling is one which balances between flexibility, tractability and practical applications as illustrated in fig 2.3. Flexibility ensures that the model can capture the complex, high-dimensional distributions often found in real-world data, such as natural images, text, or audio signals. Deep neural networks are particularly well suited for this purpose due to their ability to learn hierarchical representations and model intricate, multi-modal distributions.

At the same time, the framework must remain computationally tractable to ensure that both training and inference are feasible, even when dealing with high dimensional datasets. Many generative models suffer from the curse of dimensionality, where the computational cost grows exponentially with the number of dimensions. A well designed framework mitigates this issue through efficient parameterization, approximate inference techniques, or better optimization strategies.

Another important aspect is the ability to estimate the probability density of any given sample. Density estimation is fundamental not only for generating new data but also for evaluating the model's confidence in its predictions. A model that provides meaningful likelihoods can be used for a variety of downstream applications like model comparison, outlier detection, and data compression.

Finally, a generative framework must excel at producing high quality, diverse samples that closely mirror the true data distribution. This aspect is particularly important for applications like image synthesis, text generation, and data augmentation, where the goal is to generate realistic and novel outputs.

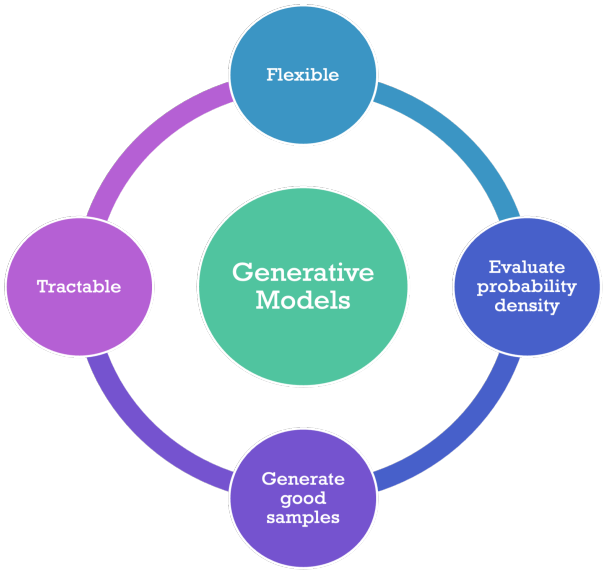


Figure 2.3: Framework for an efficient generative model

Diffusion models stand out as a promising approach that fulfills these key criteria for a good generative model. They utilize the flexibility of deep neural networks with no architectural restrictions, allowing the use of complex structures like U-Nets or transformers for parameterizing the denoising process. Their computational tractability stems from maximizing a variational lower bound on the likelihood, similar to VAEs, but they often outperform VAEs by using a fixed forward process and a longer, more gradual denoising process, which reduces approximation errors and captures finer data details. They can evaluate probability densities through the learned reverse process, while also excelling at generating high-quality, diverse samples by iteratively refining noise into outputs that closely match the true data distribution.

While we explore diffusion models in detail (chapter 3), we first focus on the family of latent space models, which form the foundational framework underlying many modern generative approaches, including diffusion models. They provide essential concepts such as latent space representations and iterative refinement, which are crucial in understanding the mechanics of diffusion-based methods.

2.4 Latent space models for generative modeling

Learning compact and meaningful representations of $p(\mathbf{x})$ is crucial for many modern machine learning techniques, particularly for generative modeling. In this section we explore a family of encoder-decoder architecture that use latent space to capture the underlying structure of complex datasets. Beginning with AEs, which focus on deterministic data reconstruction through a bottleneck representation, we progress to VAEs, which introduce a probabilistic framework for generation. Building on this we discuss about Hierarchical Variational Autoencoders (HVAEs), which generalize VAEs by introducing multi-scale latent hierarchies, making them closely related to diffusion models.

2.4.1 Autoencoders

An AE is a neural network architecture designed to learn a compressed representation of input data and reconstruct it as accurately as possible. It consists of two primary functions: (i) an encoder network $f_{\boldsymbol{\theta}}(\mathbf{x})$, which maps the input $\mathbf{x}$ to a lower dimensional latent space $\mathbf{z}$, where $\mathbf{z} \in \mathbb{R}^{\mathrm{d}}$, and (ii) a decoder network $g_{\boldsymbol{\phi}}(\mathbf{z})$, which reconstructs the input from this latent representation given as $\tilde{\mathbf{x}} = g_{\boldsymbol{\phi}}(f_{\boldsymbol{\theta}}(\mathbf{x}))$. The block diagram shown in fig. 2.4 is a high level representation of an AE.

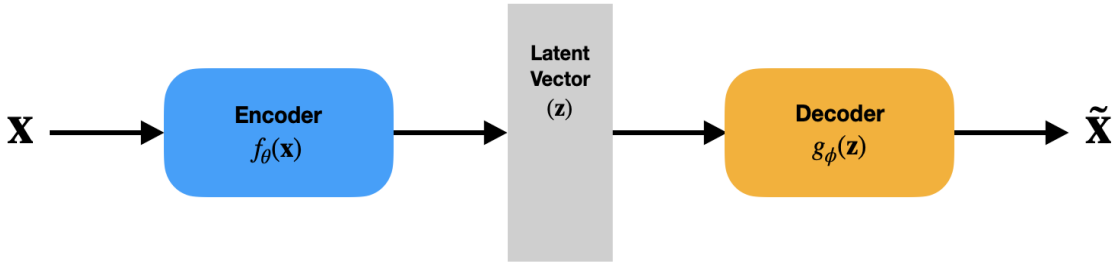


Figure 2.4: High level representation of autoencoders, where the encoder network $f_{\boldsymbol{\theta}}(\mathbf{x})$ takes in input $\mathbf{x}$ and convert it to a low dimensional latent vector $\mathbf{z}$ which is then reconstructed by the decoder network $g_{\boldsymbol{\phi}}(\mathbf{z})$ giving output $\tilde{\mathbf{x}}$.

The AE is trained by minimizing the reconstruction loss that measures the discrepancy between the input and its reconstruction. A common choice is the ℓ_2 loss, between the input $\mathbf{x}$ and the reconstructed output $g_{\boldsymbol{\phi}}(f_{\boldsymbol{\theta}}(\mathbf{x}))$. It is defined as the expected loss over the data distribution $p(\mathbf{x})$ and is given as,

$$\mathcal{L}_{recon} = \mathbb{E}_{p(\mathbf{x})} \left\| \mathbf{x} - g_{\boldsymbol{\phi}}(f_{\boldsymbol{\theta}}(\mathbf{x})) \right\|^2 \tag{2.8}$$

The overall optimization problem for training the AE is to find the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ that minimize this empirical reconstruction loss,

$$\boldsymbol{\theta}, \boldsymbol{\phi} = \arg\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - g_{\boldsymbol{\phi}}(f_{\boldsymbol{\theta}}(\mathbf{x}_i))\|^2 \tag{2.9}$$

AEs are good at tasks like dimensionality reduction, denoising and feature extraction, often capturing meaningful patterns in the data. However, their deterministic framework which ensures that the same input always produces the same latent representation and reconstruction, leads to a discontinuous latent space without any probabilistic structure. This makes them unsuitable for generating new samples as random sampling from the latent space produces incoherent outputs. In the next subsection, we explore how VAE address the shortcomings of AE by introducing a probabilistic framework to the latent space.

2.4.2 Variational autoencoders

VAEs [29] extend the framework of AEs by introducing a probabilistic approach to the latent space. Each input in the data space is mapped to a probability distribution in the latent space, rather than a fixed point, allowing the model to produce meaningful outputs even when sampling randomly from this latent distribution.

VAEs assume a generative process where the latent variable $\mathbf{z}$ is sampled from a fixed prior distribution $p(\mathbf{z})$, that is generally chosen to be a standard Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and the data is generated from the conditional distribution $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ that is a parameterized neural network. The generative process can be defined as,

$$\begin{aligned} p_{\boldsymbol{\theta}}(\mathbf{x}) &= \int_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})\, d\mathbf{z} \\ &= \int_{\mathbf{z}} p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p(\mathbf{z})\, d\mathbf{z} \end{aligned} \tag{2.10}$$

Since $\mathbf{z}$ is continuous and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is a highly complex function, therefore the integral in equation (2.10) is intractable to compute. This intractability of $p_{\boldsymbol{\theta}}(\mathbf{x})$ directly affects our ability to compute the true posterior distribution $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ in VAEs, which we need for inference. By Bayes theorem, this posterior can be expressed as,

$$p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) p(\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \tag{2.11}$$

As $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ cannot be computed directly due to the intractability of $p_{\boldsymbol{\theta}}(\mathbf{x})$, we approximate it with a tractable distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, which is a neural network parameterized by $\boldsymbol{\phi}$ and is known as the inference model. It can be interpreted as the encoder of an AE model that models the distribution of the latent vector $\mathbf{z}$ given $\mathbf{x}$, whereas $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ is like the decoder that is used for reconstruction of the data.

In order to make the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ close to the true posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ we minimize the Kullback-Leibler divergence (KLD) between them, which by definition is given as,

$$\begin{aligned} D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right] &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) - \log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z}) + \log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})}\right] - \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] + \log p_{\boldsymbol{\theta}}(\mathbf{x}) \end{aligned} \tag{2.12}$$

The first term in the equation (2.12) can be written as the KL divergence between $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ that measures, the deviation of the learned approximate posterior from the fixed prior. By substituting this KL term and rearranging the equation, we can express the relationship in terms of the log-likelihood as follows,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right] + D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right] \tag{2.13}$$

While we are able to derive an expression for the log likelihood, we still don't have an analytical form for $D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right]$, however we know that it is greater than or equal to 0, by definition. Therefore, equation (2.13) can be expressed as,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right] \tag{2.14}$$

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathrm{ELBO} = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right] \tag{2.15}$$

Since we cannot maximize the log likelihood we maximize the lower bound to the log likelihood, which is called the Evidence Lower Bound (ELBO). By maximizing the ELBO we indirectly minimize the KL divergence between $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$.

The ELBO consists of two terms,

- Reconstruction term: The first term is the expected log likelihood of the observed data $\mathbf{x}$ given the latent variables $\mathbf{z}$, where the expectation is taken over the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Maximizing this term trains the decoder $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ to

accurately reconstruct inputs from their latent representation, and thus minimizes the reconstruction loss. The specific form of this loss depends on the choice of the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. For a Gaussian distribution, the loss is equivalent to the mean squared error, while for a Bernoulli distribution, it becomes the binary cross-entropy, both ensuring generated outputs match original inputs.

- Regularization term: The second term, defined as $D_{\mathrm{KL}}\left[q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right]$, measures the KL divergence between the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ and the prior distribution $p(\mathbf{z})$. Minimizing this divergence, contributes to maximizing the ELBO. By doing so, the model ensures that the latent space is regularized by constraining the encoder's output distribution to remain close to the given prior.

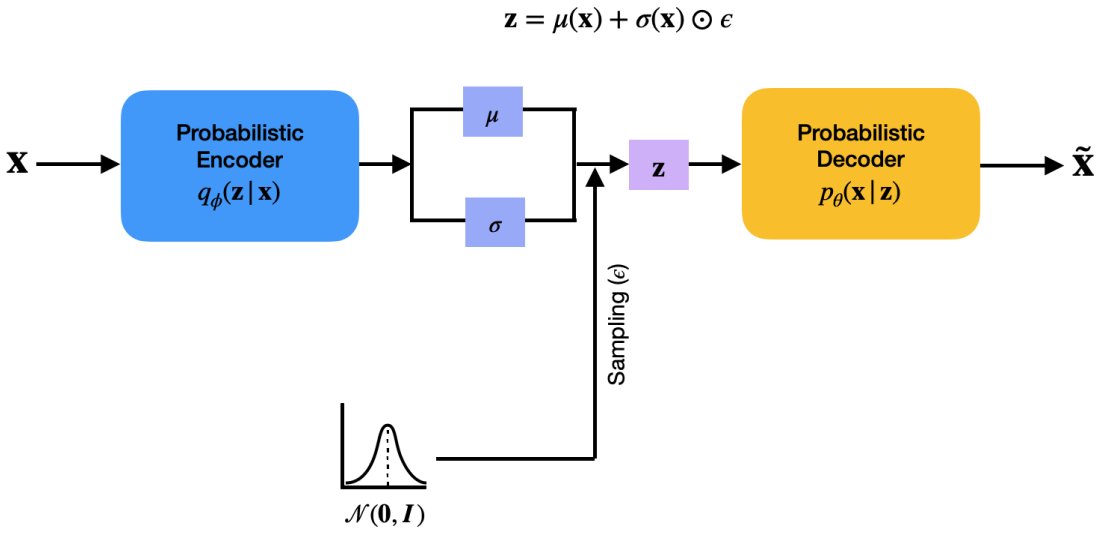


Figure 2.5: High level representation of VAE. The probabilistic encoder maps input $\mathbf{x}$ to a latent distribution, parameterized by mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$. A latent variable $\mathbf{z}$ is sampled using the reparameterization trick, then passed through the probabilistic decoder to reconstruct $\tilde{\mathbf{x}}$.

To make the ELBO tractable for optimization, we use reparameterization trick that addresses the challenge of passing gradients through the sampling block. Typically, we model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ as mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$. As direct sampling of $\mathbf{z}$ from $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is non-differentiable, we reparameterize $\mathbf{z}$ as $\mathbf{z} = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $\odot$ denotes element-wise multiplication. Here we shift the stochasticity to $\boldsymbol{\epsilon}$ and take it out of the main network allowing the gradients to flow through the encoder network during backpropagation. A high level representation of VAE is illustrated in fig. 2.5.

By simultaneously optimizing $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, the VAE develops a generative model capable of accurately reconstructing inputs and generating new samples from the prior. While

this makes the VAE effective for generative modeling, it does have limitations. The regularization term often causes the approximate posterior $q_\phi(\mathbf{z}|\mathbf{x})$ to closely align with the simple prior $p(\mathbf{z})$, such as a standard Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. This alignment results in very smooth and simple latent representations that does not capture complex or multi-modal data distributions, leading to generated samples that lack diversity and sharpness. To address this limitation, HVAE introduce a multi-level latent structure, that enhances the model's capacity to represent complex distributions, which we explore in the next section.

2.4.3 Hierarchical variational autoencoders

HVAEs [53] can be seen as an extension of VAEs where we increase the complexity of the latent space by using a multi level hierarchy structure of latent variables. The key idea is to stack multiple layers of latent variables, where each layer models different levels of abstraction or granularity. In a general HVAE, the generative process allows arbitrary dependencies between latent variables, where each latent $\mathbf{z}_t$ may depend on all higher-level latents $\mathbf{z}_{t+1}, \ldots, \mathbf{z}_\mathrm{T}$. However, a special case is the Markovian HVAE (MHVAE) [33], which imposes the Markov property on the hierarchy and is illustrated in fig. 2.6 .

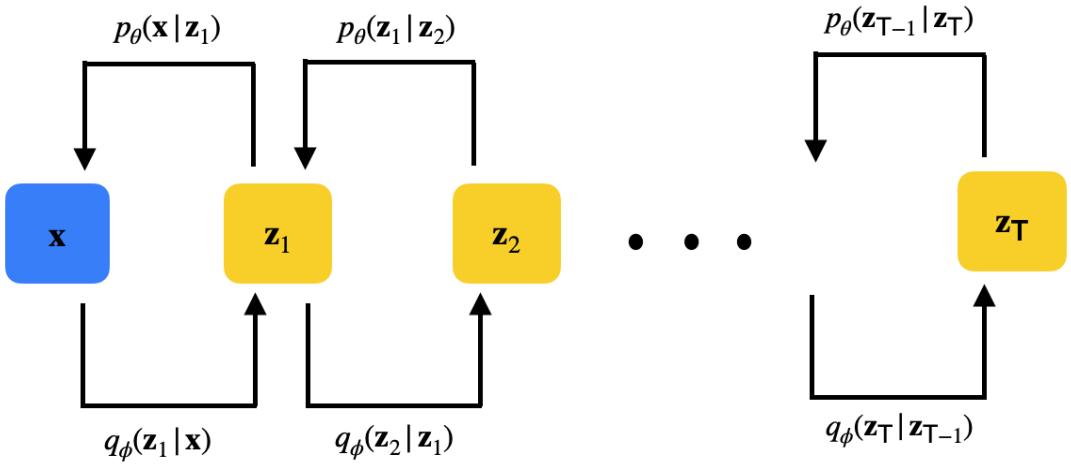


Figure 2.6: Illustration of a MHVAE with T hierarchical latent variables. It is a special case of HVAE where the generative process follows the Markov property, i.e., each latent variable $\mathbf{z}_t$ is conditioned solely on the variable $\mathbf{z}_{t+1}$.

Consider a hierarchy with T levels of latent variables, denoted as $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_\mathrm{T}$, where $\mathbf{z}_\mathrm{T}$ is at the highest level and $\mathbf{z}_1$ at the lowest, closest to the observed data $\mathbf{x}$. In MHVAE the generative process follows the Markov property which imposes a structured dependency across these levels. Specifically, each latent variable $\mathbf{z}_t$ is conditionally dependent only

on the immediate higher level latent $\mathbf{z}_{t+1}$, rather than the entire set of higher latents. This is expressed as:

$$p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \mathbf{z}_{t+1}, \ldots, \mathbf{z}_{\mathrm{T}}) = p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \mathbf{z}_{t+1}) \tag{2.16}$$

where $\boldsymbol{\theta}$ parameterizes the generative model. This simplification reduces computational complexity and enforces a chain-like flow of information from the top of the hierarchy.

Using the Markovian structure defined in Equation (2.16), we can specify the joint distribution over the observed data $\mathbf{x}$ and the latent variables $\mathbf{z}_{1:\mathrm{T}}$, as well as the approximate posterior used for inference in the MHVAE. The generative joint distribution as given in equation (2.17), starts with the prior $\mathbf{z}_{\mathrm{T}}$ (generally a standard normal, $p(\mathbf{z}_{\mathrm{T}}) = \mathcal{N}(\mathbf{z}_{\mathrm{T}}|\mathbf{0}, \boldsymbol{I})$), and proceeds through the hierarchy to generate $\mathbf{x}$,

$$p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:\mathrm{T}}) = p(\mathbf{z}_{\mathrm{T}}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1) \prod_{t=2}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}|\mathbf{z}_t) \tag{2.17}$$

Here, $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1)$ is the likelihood of the data given the latent at lowest level, and each $p_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}|\mathbf{z}_t)$ is a conditional distribution, often parameterized as a Gaussian with mean and variance determined by $\mathbf{z}_t$.

The approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})$ is defined as,

$$q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z}_1|\mathbf{x}) \prod_{t=2}^{\mathrm{T}} q_{\boldsymbol{\phi}}(\mathbf{z}_t|\mathbf{z}_{t-1}) \tag{2.18}$$

where $\boldsymbol{\phi}$ denotes the parameters of the inference network.

Similar to VAEs, we can also derive the ELBO for MHVAE by defining the generative process as follows,

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \log \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:\mathrm{T}})\, d\mathbf{z}_{1:\mathrm{T}} \\
&= \log \int \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:\mathrm{T}}) q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})}{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})}\, d\mathbf{z}_{1:\mathrm{T}} \\
&= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})} \left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:\mathrm{T}})}{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})} \right] \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}_{1:\mathrm{T}})}{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})} \right] \qquad \text{(Using Jensen's Inequality)}
\end{aligned} \tag{2.19}$$

Substituting the values from equations (2.17) and (2.18) we obtain,

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathrm{ELBO} = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}_{1:\mathrm{T}}|\mathbf{x})} \left[\log \frac{p(\mathbf{z}_{\mathrm{T}}) p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}_1) \prod_{t=2}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}|\mathbf{z}_t)}{q_{\boldsymbol{\phi}}(\mathbf{z}_1|\mathbf{x}) \prod_{t=2}^{\mathrm{T}} q_{\boldsymbol{\phi}}(\mathbf{z}_t|\mathbf{z}_{t-1})} \right] \tag{2.20}$$

The ELBO in equation (2.20) provides a foundation for training the MHVAE. This objective function shares fundamental mathematical similarities with the ELBO used in diffusion models, though their architectural implementations differ. Specifically, while both frameworks have a hierarchical latent structure, in diffusion models we have:

- A fixed, non-learned forward process that gradually transforms the data distribution into an isotropic Gaussian, and
- a latent space with the same dimensionality as the input data.

In the next chapter, we expand on this topic by examining the forward and reverse processes of diffusion models in depth, while also refining the ELBO formulation into simpler terms.

3 Diffusion models

The fundamental goal in generative modeling is to learn the distribution of the given data. As discussed in section 2.3, designing a good generative model requires carefully balancing between: the flexibility to model complex distributions, computational tractability, likelihood estimation, and the ability to generate good quality samples. While traditional approaches have made progress, they have significant drawbacks that limit their efficiency. For instance, GANs produce high quality samples but are prone to mode collapse, require extremely stable training dynamics, and cannot explicitly evaluate probability densities. VAEs, on the other hand, provide a probabilistic framework but generate blurry samples due to their reliance on a simplistic latent space. Normalizing flows ensure exact likelihood estimation but impose invertibility constraints on the model architecture. These challenges point to the need for more robust approaches to generative modeling.

Diffusion models [20], have recently emerged as a powerful alternative, addressing these limitations while maintaining strong theoretical foundations. Unlike GANs, diffusion models avoid the need for adversarial training and explicitly evaluate probability densities. Dhariwal et al. [9], demonstrated that diffusion models can outperform GANs in image synthesis. In contrast to VAEs, they do not use a traditional low dimensional latent space. Instead, the latent space is the same dimension as the data space and is a series of progressively noisier versions of the data rather than a compressed encoding. Additionally, diffusion models impose no strict architectural restrictions, providing greater flexibility in model design. These characteristics make diffusion models a good fit in the framework of generative modeling tasks. Their effectiveness is further demonstrated in table 3.1, which compares quality of generated samples from different models based on FID score (discussed in section 4.3.2). The results show that diffusion models consistently achieve better performance in generating high-quality samples.

Diffusion models have shown good performance across a wide range of tasks, proving useful for both density estimation [27, 55] and generative tasks. They have been instrumental in high quality text-to-image synthesis, as demonstrated by Stable Diffusion [46] and GLIDE [37], which generate photorealistic images from textual descriptions. They also excel in image-to-image translation tasks, including super resolution [48], image inpainting [32], and deblurring applications [67]. Beyond image domain, diffusion models are also used in text-to-speech generation [4], video generation [21], 3D shape synthesis [42] and molecular design [22]. This wide range success across different modalities

Table 3.1: Comparison of Generative Models by FID Scores (Lower is Better)

Model	CIFAR-10	ImageNet -64	ImageNet -128	CelebA -64
GANs				
DCGAN [51, 16]	35.6	–	–	12.5
SNGAN [51]	11.8	34.4	33.2	–
WGAN-GP [51, 31]	15.0	35.8	79.5	30.0 ± 1.0
BigGAN-deep [3, 35]	9.59	–	7.4	–
VAEs				
Vanilla VAE [31]	155.7 ± 11.6	–	–	85.7 ± 3.8
VQ-VAE-2 [50]	51.9	–	–	22.89
NVAE [61]	25.1	–	–	18.3
Flows				
Glow [5]	46.90	–	–	–
Residual Flow [5]	46.37	–	–	–
Diffusion				
DDPM [20, 39]	3.17	–	–	1.27
IDDPM [9]	–	2.92	–	–
ADM [9, 39]	–	2.61	5.91	1.60

shows the ability of diffusion models to learn complex, high-dimensional distributions and generate from them while maintaining stable training dynamics.

In this chapter, we provide a detailed exploration of diffusion models, starting with the forward and reverse processes. We introduce key concepts such as the signal-to-noise ratio (SNR), negative evidence lower bound (NELBO), and the loss formulations for different target objectives including $\mathbf{x}$-prediction and $\boldsymbol{\epsilon}$-prediction [20, 27]. Next, we examine diffusion models from a score-based generative perspective, shifting the focus to the score function, or $\mathbf{s}$-prediction [56], which represents the gradient of the data's probability distribution, $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. To make score modeling tractable, we discuss techniques such as sliced score matching and denoising score matching. Lastly, we introduce progressive distillation in diffusion models where the distributional change rate is captured through $\mathbf{v}$-prediction [49], a novel target prediction that integrates both the data representation $\mathbf{x}$ and the noise component $\boldsymbol{\epsilon}$.

We show that for each of these four targets i.e data $\mathbf{x}$, noise $\boldsymbol{\epsilon}$, the rate of change in the data distribution $\mathbf{v}$, and the score function $\mathbf{s}$, we can derive an equivalent NELBO loss formulation and establish its mathematical connections with other widely used diffusion loss functions. This allows us to construct a unified theoretical framework that links these objectives, providing an understanding of their equivalence and distinctions.

3.1 Diffusion models as probabilistic denoising models

Diffusion models generate high-quality data by iteratively refining noisy samples into structured outputs. Given a sample $\mathbf{x} \sim q(\mathbf{x})$, the forward process incrementally perturbs the data by adding Gaussian noise across multiple timesteps T. This is defined as $q(\mathbf{z}_t \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}_t; \alpha_t \mathbf{x}, \sigma_t^2 \boldsymbol{I})$, where $\mathbf{z}_t$ represents the noisy version of $\mathbf{x}$ at time t, α_t scales the data $\mathbf{x}$, and σ_t governs the magnitude of the added noise. The objective for the model is to learn a reverse process which enables the generation of new samples starting from a pure Gaussian noise $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ by iteratively denoising it to reconstruct realistic samples. This probabilistic framework enables diffusion models to model complex data distributions, making them useful for generative applications. In the following sections, we delve into the mathematical details of the forward and reverse processes and formulate the training objectives that drive these models.

3.1.1 Forward diffusion process

In diffusion models, the forward process operates as a Markov process, gradually decreasing the information from the data $\mathbf{x}$ by introducing noise over a series of timesteps as shown in fig. 3.1. This generates intermediate latent variables, denoted as $\mathbf{z}_t$, with $t \in [0, 1]$ indicating a specific timestep. The addition of noise is governed by a predefined schedule which determines the amount of noise to be added and signal to be removed at each timestep, regulated by parameters α_t and σ_t.

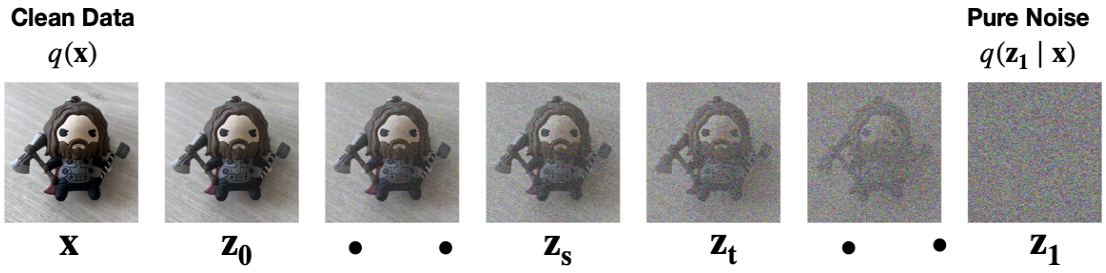


Figure 3.1: Illustration of the forward diffusion process, where clean data $\mathbf{x}$ is progressively corrupted into pure noise $\mathbf{z}_1$ through a sequence of intermediate latent variables $\mathbf{z}_t$ at different timesteps $t \in [0, 1]$. The latent variable $\mathbf{z}_s$ precedes $\mathbf{z}_t$, where $s < t$.

The evolution of latent variables and the Markov transition distribution in the forward process are defined as follows:

$$\begin{aligned} q(\mathbf{z}_t \mid \mathbf{x}) &= \mathcal{N}(\mathbf{z}_t; \alpha_t \mathbf{x}, \sigma_t^2 \boldsymbol{I}) \\ q(\mathbf{z}_t \mid \mathbf{z}_s) &= \mathcal{N}(\mathbf{z}_t; \alpha_{t|s} \mathbf{z}_s, \sigma_{t|s}^2 \boldsymbol{I}) \end{aligned} \tag{3.1}$$

where $0 \leq s \leq t \leq 1$, $\alpha_{t|s} = \frac{\alpha_t}{\alpha_s}$ and $\sigma_{t|s}^2 = \sigma_t^2 - \alpha_{t|s}^2 \sigma_s^2$

The scheduling parameters α_t and σ_t are both strictly positive and smooth functions of time, with α_t decreasing and σ_t increasing monotonically. These parameters control the trade-off between the retained signal and the added noise at each timestep.

A key quantity that characterizes the relative balance between signal and noise in the forward process is the Signal-to-Noise Ratio, $\mathrm{SNR}(t)$. Since α_t controls the signal strength and σ_t determines the level of noise, their ratio provides a useful measure of how much information remains in $\mathbf{z}_t$. The $\mathrm{SNR}(t)$ is defined as,

$$\mathrm{SNR}(t) = \frac{\alpha_t^2}{\sigma_t^2} \tag{3.2}$$

This ratio decreases over time as the forward process progresses. This means that $\mathrm{SNR}(s) > \mathrm{SNR}(t)$, for $s < t$. At $t = 0$, the data has the most signal and is least noisy, while at $t = 1$, the signal is completely lost, leaving only pure noise. As a result the conditional distribution of $\mathbf{z}_1$ given the original data $\mathbf{x}$ reduces to a isotropic gaussian, i.e. $q(\mathbf{z}_1 \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}_1; \mathbf{0}, \boldsymbol{I})$.

The rate at which SNR decays depends on the choice of the noise schedule, making it a crucial factor in diffusion model performance. In particular, different scheduling strategies correspond to different SNR profiles, which impact the efficiency of training and inference. Fig. 3.2 illustrates the behavior of log(SNR) with respect to time across different noise schedules in diffusion models.

The authors in DDPM [20] used a linear noise schedule over 1000 discrete timesteps. Nichol and Dhariwal [38] later proposed a cosine schedule, which provided a smoother transition between low and high noise levels, resulting in improved performance. A more advanced approach was introduced in Variational Diffusion Models (VDM) [27], where the forward noise schedule was learned. This method demonstrated that increasing the number of timesteps and transitioning to continuous-time modeling could further minimize the loss, leading to better likelihood estimation. The noise schedules used in the above mentioned works was variance-preserving (VP) i.e. $\alpha_t^2 = 1 - \sigma_t^2$, ensuring that the variance of the data remained constant throughout the forward process. Alternatively, in the case of variance-exploding (VE) schedules [58, 57], $\alpha_t^2 = 1$, meaning that the noise variance increases exponentially over time. It was demonstrated by Kingma et al.

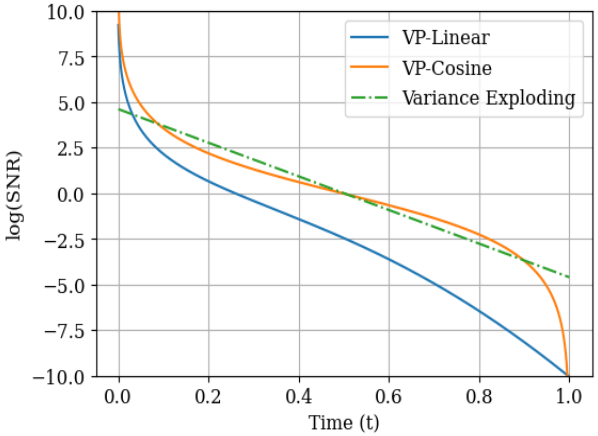


Figure 3.2: Log(SNR) vs. time graph for different noise schedules used in the forward process. The vp-cosine schedule provides a smoother noise decay compared to vp-linear and variance-exploding schedule.

[27] that the variance preserving and variance exploding formulations can be considered equivalent in continuous time.

In this study, we use a variance-preserving cosine schedule. Specifically, we define $\alpha_t = \cos(0.5\pi t)$, ensuring that the variance of the noise evolves smoothly over time. Under this schedule, the noisy image $\mathbf{z}_t$ at timestep t, given the original data $\mathbf{x}$, is formulated as:

$$\mathbf{z}_t = \cos(0.5\pi t)\,\mathbf{x} + \sin(0.5\pi t)\,\boldsymbol{\epsilon} \tag{3.3}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ represents standard Gaussian noise.

3.1.2 Reverse generative process

The reverse diffusion process $q(\mathbf{z}_s \mid \mathbf{z}_t)$ is also a Markov chain with Gaussian transition probabilities and is responsible for gradually reconstructing the original data $\mathbf{x}$ from its noisy version $\mathbf{z}_t$. However, as the exact reverse process is mathematically intractable, it is approximated using a learned distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t)$. This approximation forms the basis of a hierarchical generative model, where a sequence of latent variables $\mathbf{z}_t$ is progressively sampled in reverse, starting from the most noisy state at t = 1 and moving backward to t = 0. Over the course of T steps, the noise is gradually reduced, ultimately recovering a clean data sample from the original distribution.

In the discrete-time setting, the total number of steps T is finite and time is divided into evenly spaced intervals of width $\frac{1}{\mathrm{T}}$, with $s(i) = \frac{i-1}{\mathrm{T}}$ and $t(i) = \frac{i}{\mathrm{T}}$.

The overall reverse process is defined as,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}_1) p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_0) \prod_{i=1}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)}) d\mathbf{z} \tag{3.4}$$

Introducing the posterior $q(\mathbf{z}_{0:1} \mid \mathbf{x})$ to further simplify,

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \log \int q(\mathbf{z}_{0:1} \mid \mathbf{x}) \frac{p(\mathbf{z}_1) p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_0) \prod_{i=1}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)})}{q(\mathbf{z}_{0:1} \mid \mathbf{x})} \, d\mathbf{z} \\
&= \log \mathbb{E}_{q(\mathbf{z}_{0:1} \mid \mathbf{x})} \left[\frac{p(\mathbf{z}_1) p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_0) \prod_{i=1}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)})}{q(\mathbf{z}_{0:1} \mid \mathbf{x})} \right]
\end{aligned}$$

Using Jensen's inequality:

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &\geq \mathbb{E}_{q(\mathbf{z}_{0:1} \mid \mathbf{x})} \left[\log \frac{p(\mathbf{z}_1) p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_0) \prod_{i=1}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)})}{q(\mathbf{z}_{0:1} \mid \mathbf{x})} \right] \\
&\geq \mathbb{E}_{q(\mathbf{z}_{0:1} \mid \mathbf{x})} \left[\log \frac{p(\mathbf{z}_1) p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_0) \prod_{i=1}^{\mathrm{T}} p_{\boldsymbol{\theta}}(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)})}{q(\mathbf{z}_1 \mid \mathbf{x}) \prod_{i=1}^{\mathrm{T}} q(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)}, \mathbf{x})} \right]
\end{aligned} \tag{3.5}$$

By applying logarithmic properties and the definition of KL divergence, Equation (3.5) can be simplified into,

$$\begin{aligned}
-\log p_{\boldsymbol{\theta}}(\mathbf{x}) \leq \mathrm{NELBO}(\mathbf{x}) = \underbrace{D_{\mathrm{KL}}\left[q(\mathbf{z}_1 \mid \mathbf{x}) \,\|\, p(\mathbf{z}_1)\right]}_{\text{Prior Loss}} + \\
\underbrace{\mathbb{E}_{q(\mathbf{z}_0 \mid \mathbf{x})}\left[-\log p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}_0)\right]}_{\text{Reconstruction Loss}} + \\
\underbrace{\sum_{i=1}^{\mathrm{T}} \mathbb{E}_{q(\mathbf{z}_{t(i)} \mid \mathbf{x})} D_{\mathrm{KL}}\left[q(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)}, \mathbf{x}) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{z}_{s(i)} \mid \mathbf{z}_{t(i)})\right]}_{\text{Diffusion Loss } (L_{\mathrm{T}}(\mathbf{x}))}
\end{aligned} \tag{3.6}$$

To approximate the true data distribution we aim to minimize the negative log likelihood. However, since this is intractable, we instead optimize the tractable negative Evidence Lower Bound (NELBO) as shown in equation (3.6). The NELBO can be decomposed into three main components: the prior loss, the reconstruction loss, and the diffusion loss.

With a sufficiently low SNR at $t = 1$, the forward process transforms the data distribution into a pure Gaussian as discussed in section 3.1.1. This allows us to approximate

the posterior $q(\mathbf{z}_1 \mid \mathbf{x})$ as a standard normal distribution and therefore we can set the prior $p(\mathbf{z}_1)$ that matches this form and is given as $p(\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_1; \mathbf{0}, \boldsymbol{I})$. Since the prior is fixed and already matches the posterior $q(\mathbf{z}_1 \mid \mathbf{x})$, there is no additional information to be learned from the prior loss in equation (3.6). As a result, it can be dropped from the equation. Moreover, given the assumptions of the forward process, the latent variable $\mathbf{z}_0$ remains nearly identical to the original data $\mathbf{x}$, as only a small amount of noise is added at the start of the diffusion process. Therefore the reconstruction loss term becomes negligible and can also be omitted from the NELBO. The remaining term is the diffusion loss $L_{\mathrm{T}}(\mathbf{x})$. This plays a crucial role in guiding the learning process and depends on the total number of timesteps T, which determines the depth of the generative model.

For further simplifying the diffusion loss $L_{\mathrm{T}}(\mathbf{x})$ we need the analytical form of the posterior $q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x})$ and unlike the reverse transition $q(\mathbf{z}_s \mid \mathbf{z}_t)$, this posterior distribution can be approximated using Bayesian inference as discussed in [27]. Due to the Markov property of the forward process, we can factorize the joint distribution of the latent variables as $q(\mathbf{z}_s, \mathbf{z}_t|\mathbf{x}) = q(\mathbf{z}_s|\mathbf{x})q(\mathbf{z}_t|\mathbf{z}_s)$. Therefore, $q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x})$ can be interpreted as a Bayesian posterior, derived from a prior $q(\mathbf{z}_s|\mathbf{x})$ and updated with the likelihood $q(\mathbf{z}_t|\mathbf{z}_s)$.

$$q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x}) \propto q(\mathbf{z}_s|\mathbf{x})q(\mathbf{z}_t|\mathbf{z}_s) \tag{3.7}$$

To compute this posterior analytically, we use a standard result from Bayesian inference for Gaussian distributions. Consider a scenario, where the prior $m(x)$ is a Gaussian distribution with mean μ_A and variance σ_A^2, i.e., $m(x) = \mathcal{N}(\mu_A, \sigma_A^2)$. and the linear Gaussian likelihood is given as $m(y|x) = \mathcal{N}(ax, \sigma_B^2)$ then the general solution for the posterior $m(x|y)$ is,

$$m(x|y) = \mathcal{N}(\tilde{\mu}, \tilde{\sigma}^2) \tag{3.8}$$

where, $\tilde{\sigma}^{-2} = \sigma_A^{-2} + a^2\sigma_B^{-2}$ and $\tilde{\mu} = \tilde{\sigma}^{-2}\left(\sigma_A^{-2}\mu_A + a\sigma_B^{-2}y\right)$

From equation (3.1) we get $q(\mathbf{z}_s|\mathbf{x})$ and $q(\mathbf{z}_t|\mathbf{z}_s)$. By substituting them in the general solution for the posterior i.e. equation (3.8), we get,

$$q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x}) = \mathcal{N}\left(\boldsymbol{\mu}_Q(\mathbf{z}_t, \mathbf{x}; s, t), \sigma_Q^2(s, t)\boldsymbol{I}\right) \tag{3.9}$$

where,

$$\sigma_Q^2(s, t) = \frac{\sigma_{t|s}^2\sigma_s^2}{\sigma_t^2} \tag{3.10}$$

$$\boldsymbol{\mu}_Q(\mathbf{z}_t, \mathbf{x}; s, t) = \alpha_{t|s}\frac{\sigma_s^2}{\sigma_t^2}\mathbf{z}_t + \alpha_s\frac{\sigma_{t|s}^2}{\sigma_t^2}\mathbf{x} \tag{3.11}$$

We can now define the conditional model distribution as,

$$p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t) = q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x} = \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)) \tag{3.12}$$

This is similar to $q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x})$ but we replace $\mathbf{x}$ with $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)$ which is the prediction of the original data $\mathbf{x}$ by the denoising model given the noisy data $\mathbf{z}_t$ at time t.

3.1.3 Loss formulation

In the previous section, we introduced the NELBO objective, as defined in equation (3.6). This objective serves as the foundational loss function for training diffusion models. After simplifying the NELBO by omitting the prior and reconstruction terms, it reduces to the diffusion loss $L_{\mathrm{T}}(\mathbf{x})$, which governs the denoising process. When designing the denoising model, there are multiple choices for the target prediction beyond directly estimating the data $\mathbf{x}$. For example, a common approach involves predicting the noise $\boldsymbol{\epsilon}$ added during the forward process [20, 54, 38]. Another method focuses on estimating the rate of change in the data distribution over time, known as $\mathbf{v}$-prediction [49]. Additionally, some approaches aim to predict the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, which represents the gradient of the log-probability density of the data [56, 58].

For each of these target predictions, we can formulate an equivalent NELBO loss (L) from equation (3.6). This unified loss, ensures that the model's predictions align with the data generation process, even though different parameterizations are used. Beyond the standard NELBO loss, alternative loss formulations have been proposed in the literature, designed for specific goals, such as enhancing the perceptual quality of generated samples or improving computational efficiency during training and inference. We refer to these alternative losses as weighted loss functions ($\mathcal{L}$) as they can all be expressed as a weighted function of the NELBO as shown in equation (3.13). Here, the weight $w(t)$ is an appropriate chosen weighting function.

$$\mathcal{L} = w(t)L \tag{3.13}$$

In the following sections, we provide a detailed exploration of these target predictions and their corresponding loss formulations. We begin with the two most widely used approaches: $\mathbf{x}$-prediction, where the model directly estimates the original data, and $\boldsymbol{\epsilon}$-prediction, where the model predicts the noise component. Next, we discuss score-based modeling, demonstrating how the NELBO can be reformulated in terms of the score function, leading to $\mathbf{s}$-prediction. Finally, we examine $\mathbf{v}$-prediction, also referred to as angular parameterization, which focuses on capturing the rate of change of the data distribution.

Through this systematic review, we unify these different formulations under the broader framework of the NELBO objective. Specifically, we derive the NELBO in terms of these alternative targets and show that all the different objectives, whether predicting

the original data $\mathbf{x}$, noise $\boldsymbol{\epsilon}$, rate of change in the data distribution $\mathbf{v}$, or score $\mathbf{s}$ can be expressed as weighted functions of the NELBO. For consistency throughout this study, we refer to different target objectives as $\mathbf{x}$-space, $\boldsymbol{\epsilon}$-space, $\mathbf{v}$-space, and $\mathbf{s}$-space.

3.1.4 Loss formulation in x-space

The loss formulation in $\mathbf{x}$-space has been explored in several studies [20, 27, 54]. In this work, we follow the derivation method presented by Kingma et al. [27]. To express the loss in $\mathbf{x}$-space, we revisit equation (3.12), where we define our conditional model distribution $p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t)$ similar to the posterior $q(\mathbf{z}_s|\mathbf{z}_t, \mathbf{x})$ and replace $\mathbf{x}$ with the estimation of our denoising model $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)$. This was expressed as:

$$p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t) = q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x} = \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t))$$

From the above equation and equation (3.9) we can define $p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t)$ as,

$$p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t) = \mathcal{N}\left(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\mathbf{z}_t, \mathbf{x}; s, t), \sigma_Q^2(s,t)\boldsymbol{I}\right) \tag{3.14}$$

Since the distributions $q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{z}_s \mid \mathbf{z}_t)$ are Gaussian with same variances, the KL divergence between these distributions has a closed form solution that simplifies to,

$$D_{\mathrm{KL}}[q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x}) \parallel p(\mathbf{z}_s \mid \mathbf{z}_t)] = \frac{1}{2\sigma_Q^2(s,t)} \|\boldsymbol{\mu}_Q - \boldsymbol{\mu}_{\boldsymbol{\theta}}\|^2 \tag{3.15}$$

Substituting the expressions for $\boldsymbol{\mu}_Q$ and $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ from equation (3.11),

$$\begin{aligned}
D_{\mathrm{KL}}[q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x}) \parallel p(\mathbf{z}_s \mid \mathbf{z}_t)] &= \frac{\sigma_t^2}{2\sigma_{t|s}^2\sigma_s^2} \frac{\alpha_s^2 \sigma_{t|s}^4}{\sigma_t^4} \|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|^2 \\
&= \frac{1}{2\sigma_s^2} \frac{\alpha_s^2 \sigma_{t|s}^2}{\sigma_t^2} \|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|_2^2 \\
&= \frac{1}{2\sigma_s^2} \frac{\alpha_s^2(\sigma_t^2 - \alpha_{t|s}^2\sigma_s^2)}{\sigma_t^2} \|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|_2^2 \\
&= \frac{1}{2} \frac{\alpha_s^2\sigma_t^2/\sigma_s^2 - \alpha_t^2}{\sigma_t^2} \|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|_2^2 \\
&= \frac{1}{2}\left(\frac{\alpha_s^2}{\sigma_s^2} - \frac{\alpha_t^2}{\sigma_t^2}\right) \|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|_2^2
\end{aligned} \tag{3.16}$$

We can write equation (3.16) in terms of $\mathrm{SNR}(t)$ using equation (3.2),

$$D_{\mathrm{KL}}[q(\mathbf{z}_s \mid \mathbf{z}_t, \mathbf{x}) \parallel p(\mathbf{z}_s \mid \mathbf{z}_t)] = \frac{1}{2}(\mathrm{SNR}(s) - \mathrm{SNR}(t))||\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{x}_t; t)||_2^2 \tag{3.17}$$

Next, we substitute the above equation into the diffusion loss $L_{\mathrm{T}}(\mathbf{x})$. For simplicity, we adopt a shorter notation by representing $s(i)$ and $t(i)$ as s and t respectively.

$$L_{\mathrm{T}}(\mathbf{x}) = \frac{1}{2}\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\sum_{i=1}^{\mathrm{T}}(\mathrm{SNR}(s) - \mathrm{SNR}(t))\left\|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\right\|_2^2\right] \tag{3.18}$$

The loss formulation in equation (3.18) applies to the discrete setting, where the number of diffusion steps T is finite, as discussed in section 3.1.2. However, in the continuous-time formulation, the number of steps approaches infinity, i.e., $\mathrm{T} \to \infty$, and the timestep t is treated as a continuous variable. This leads to what is known as the continuous-time diffusion process.

In this scenario, we introduce a very small time interval τ, such that the total number of steps can be expressed as $\mathrm{T} = \frac{1}{\tau}$. Consequently, we define the timesteps as $s = \tau(i-1)$ and $t = \tau i$ which allows us to transition from the discrete formulation to the continuous setting as follows:

$$L(\mathbf{x}) = \frac{1}{2}\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}),i\sim\mathrm{U}\{1,\mathrm{T}\}}\left[\frac{\mathrm{SNR}(t-\tau) - \mathrm{SNR}(t)}{\tau}\left\|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\right\|_2^2\right] \tag{3.19}$$

As $\mathrm{T} \to \infty$, $\tau \to 0$, which simplifies the equation (3.19) to,

$$L(\mathbf{x}) = -\frac{1}{2}\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}),t\sim\mathcal{U}[0,1]}\left[\mathrm{SNR}'(t)\left\|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}; t)\right\|_2^2\right] \tag{3.20}$$

To distinguish between the notations used for sampling in discrete and continuous settings, we use $\mathrm{U}\{1, \mathrm{T}\}$ that represents sampling from a discrete uniform distribution over the integers between 1 and T. On the other hand, in the continuous-time framework, $\mathcal{U}[0, 1]$ denotes sampling from a continuous uniform distribution over the interval $[0, 1]$, showing the continuous nature of the diffusion process. Additionally, we can remove the factor $\frac{1}{2}$ from the equation (3.20), as it is a constant scaling factor that does not affect the optimization process in terms of finding the optimal parameters $\boldsymbol{\theta}$. This gives,

$$L(\mathbf{x}) = -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\int_0^1 \mathrm{SNR}'(t)\left\|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\right\|_2^2 dt \tag{3.21}$$

Equation (3.21) represents the NELBO loss formulation in $\mathbf{x}$-space. In this formulation, $\mathrm{SNR}'(t)$ denotes the time derivative of $\mathrm{SNR}(t)$. Specifically, under the cosine scheduling of the forward process, we define the scaling factors as $\alpha_t = \cos(\pi/2)t$ and $\sigma_t = \sin(\pi/2)t$.

Given these definitions, we can derive the expression for $\mathrm{SNR}'(t)$ as,

$$\begin{aligned}\mathrm{SNR}'(t) &= \frac{d}{dt}\left(\frac{\cos^2\left(\frac{\pi}{2}t\right)}{\sin^2\left(\frac{\pi}{2}t\right)}\right) \\ &= \frac{-\pi\cos(\pi/2)t}{\sin^3(\pi/2)t} = \frac{-\pi\alpha_t}{\sigma_t^3}\end{aligned} \tag{3.22}$$

We define the weighted loss function in the $\mathbf{x}$-space as follows,

$$\mathcal{L}(\mathbf{x}) = -\mathbb{E}_{t\sim\mathcal{U}(0,1),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[w_{\mathbf{x}}(t)\,\mathrm{SNR}'(t)\,\|\mathbf{x}-\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)\|_2^2\right] \tag{3.23}$$

By choosing the weight function as $w_{\mathbf{x}}(t) = -\frac{1}{\mathrm{SNR}'(t)}$, the above expression simplifies to an expectation of the mean squared error (MSE) between the original data $\mathbf{x}$ and its prediction $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)$,

$$\mathcal{L}(\mathbf{x}) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\|\mathbf{x}-\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)\|_2^2\right] = w_{\mathbf{x}}(t)L(\mathbf{x}) \tag{3.24}$$

3.1.5 Loss formulation in ϵ-space

The $\boldsymbol{\epsilon}$-space loss formulation is among the most widely used objectives in diffusion models, initially proposed in the DDPM framework [20]. Rather than training the model to directly reconstruct the original data sample $\mathbf{x}$, this approach shifts the focus to predicting the noise term $\boldsymbol{\epsilon}$ that is added at each timestep during the forward diffusion process.

This formulation offers a number of advantages. First, it aligns naturally with the stochastic structure of the diffusion process, where noise is incrementally injected into the data. By modeling this noise directly, the learning task becomes simpler, as the model is tasked with estimating a well defined quantity. Second, predicting $\boldsymbol{\epsilon}$ allows for a stable and consistent training objective across different noise levels, making it easier for the network to generalize over the entire range of timesteps. As a result, this objective has become a standard in diffusion-based generative modeling and is the foundation for many subsequent advances in the field.

To derive the NELBO formulation in the $\boldsymbol{\epsilon}$-space, we begin by further simplifying equation (3.21). This is achieved by expressing the original data $\mathbf{x}$ in terms of the noise variable $\boldsymbol{\epsilon}$, using the forward diffusion equation, $\mathbf{z}_t = \alpha_t\mathbf{x} + \sigma_t\boldsymbol{\epsilon}$. By rearranging this expression, we can substitute $\mathbf{x}$ in the loss and reformulate the objective directly in terms of the noise, which forms the basis of the $\boldsymbol{\epsilon}$-prediction loss. The derivation is as follows,

$$
\begin{aligned}
L(\boldsymbol{\epsilon}) &= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t)\, \|(\mathbf{z}_t - \sigma_t\boldsymbol{\epsilon})/\alpha_t - (\mathbf{z}_t - \sigma_t\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t))/\alpha_t\|_2^2\, dt \\
&= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t)\, \|(\mathbf{z}_t - \sigma_t\boldsymbol{\epsilon})/\alpha_t - (\mathbf{z}_t - \sigma_t\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t))/\alpha_t\|_2^2\, dt \\
&= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t)\frac{\sigma_t^2}{\alpha_t^2}\, \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)\|_2^2\, dt \\
&= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \frac{\mathrm{SNR}'(t)}{\mathrm{SNR}(t)}\, \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)\|_2^2\, dt
\end{aligned} \tag{3.25}
$$

Hence, the NELBO loss in epsilon space is given by,

$$
L(\boldsymbol{\epsilon}) = -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}),t\sim\mathcal{U}[0,1]} \left[\frac{\mathrm{SNR}'(t)}{\mathrm{SNR}(t)}\, \|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)\|_2^2\right] \tag{3.26}
$$

The loss proposed in the original DDPM framework is different from the exact NELBO formulation derived in Equation (3.26). Instead of modeling the scaled noise introduced in the forward process, DDPM employs a weighted $\boldsymbol{\epsilon}$-prediction loss. In this approach, the model is trained to predict the standard Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ directly, rather than the rescaled version that was actually added to the data $\mathbf{x}$ at each timestep.

The weighted $\boldsymbol{\epsilon}$ loss is given as follows and can be seen as a weighted function of (3.26) with weight $w_{\boldsymbol{\epsilon}}(t) = -\frac{\mathrm{SNR}(t)}{\mathrm{SNR}'(t)}$:

$$
\mathcal{L}(\boldsymbol{\epsilon}) = \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}),t\sim\mathcal{U}[0,1]}\left[\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t;t)\|_2^2\right] = w_{\boldsymbol{\epsilon}}(t)L(\boldsymbol{\epsilon}) \tag{3.27}
$$

In this section, we examined both the forward and reverse diffusion processes and derived the corresponding NELBO objectives, along with their weighted loss formulations, in the $\mathbf{x}$-space and $\boldsymbol{\epsilon}$-space. These two perspectives form the foundation for many existing diffusion based generative models.

However, there exists another significant class of diffusion models, namely score-based diffusion models, which take a different approach by directly estimating the gradient of the data log-density, also known as the score function. In the next section (3.2), we shift our focus to this family of models and formulate the NELBO and weighted loss in the $\mathbf{s}$-space, showing that score modeling is also equivalent to other loss formulations through appropriate transformations.

3.2 Diffusion models as score based models

While earlier formulations of diffusion models focused on predicting either the original data $\mathbf{x}$ or the noise $\boldsymbol{\epsilon}$ introduced during the forward process, score-based diffusion models adopt a fundamentally different approach. Instead of reconstructing the data or the noise, these models aim to directly estimate the score function of the data distribution. For a given data distribution $p(\mathbf{x})$, the score function is defined as the gradient of the log-probability with respect to the data, i.e., $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. The score represents a vector field that gives the direction in which the data distribution increases most rapidly. This is also referred to as the Stein score function, and it should not be confused with the Fisher score function, which involves taking the gradient of the log-likelihood with respect to the model parameters $\boldsymbol{\theta}$.

The motivation for using the score function is that it is often easier to handle computationally and analytically compared to the probability density function itself. Since the score function uses the gradient of the log-density, it bypasses the need to compute the intractable normalizing constant (section 2.1) as shown below.

From equation (2.1) we have,

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(f_{\boldsymbol{\theta}}(\mathbf{x}))/Z_{\boldsymbol{\theta}}$$

where, $Z_{\boldsymbol{\theta}}$ is the intractable normalizing constant. By taking the logarithm of both sides and then computing the gradient with respect to $\mathbf{x}$, we obtain:

$$\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x}) - \nabla_{\mathbf{x}} \log Z_{\boldsymbol{\theta}} \tag{3.28}$$

Since $Z_{\boldsymbol{\theta}}$ is a normalizing constant that does not depend on $\mathbf{x}$, its gradient with respect to $\mathbf{x}$ is zero. Therefore, the expression simplifies to:

$$\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x}) = \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x}) \tag{3.29}$$

In this formulation, $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$ represents the score model, which is a neural network parameterized by $\boldsymbol{\theta}$. The goal is to train this model to accurately approximate the true score function of the data distribution, $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. It means learning a vector field $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$ that is optimized to match the vector field of ground truth score as closely as possible. This concept is illustrated in fig. 3.3.

Since both the true score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ and the model score $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$ are vector fields defined over the same data space, we can directly compute the difference between them at each data point $\mathbf{x}$. This allows us to define a scalar valued objective that quantifies how closely the model's predictions align with the true score function. Specifically, we can measure the difference between these two vector fields using the squared Euclidean

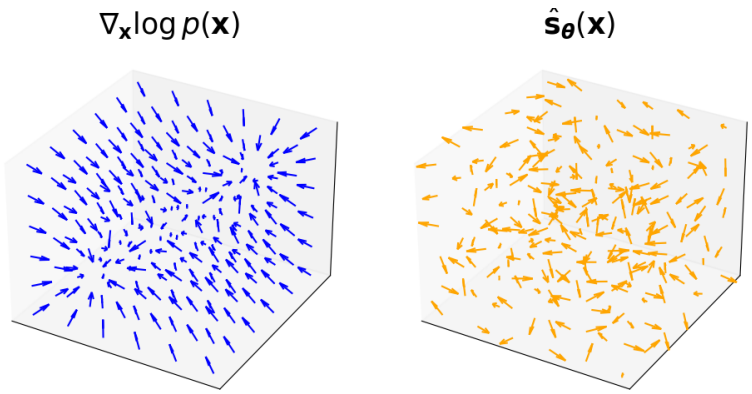


Figure 3.3: Illustration of score-based learning through vector fields. Left: The ground truth score function of the data distribution, $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, representing the true gradient of the log-density. Right: The score model $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$, a neural network which we need to train to approximate the true score function. The training objective is to match this model generated vector field to the ground truth field as closely as possible.

distance between their corresponding vectors at each point, and then average that over the data distribution. This is expressed through the Fisher divergence [23], which is used as an objective function for training score-based models. The Fisher divergence measures the expected squared difference between the true data score and the model's score and is given as,

$$L_{\mathcal{F}}(\mathbf{x}) = \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2\right] \tag{3.30}$$

However, the Fisher divergence cannot be computed directly, since it requires access to the true score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, which is generally intractable for most real world data distributions. To overcome this challenge, we turn to score matching methods, which provide a way to estimate the model parameters without explicitly needing the ground truth score. These methods are discussed in detail in the following subsections.

3.2.1 Score matching

Score matching [23][36] uses integration by parts to transform the Fisher divergence into an equivalent, tractable objective. This reformulated expression is equal to the original Fisher divergence up to an additive constant. Since such constants do not affect the optimization, minimizing the score matching objective yields the same optimal parameters as minimizing the Fisher divergence. The derivation is provided below,

The Fisher divergence as expressed in equation (3.30) is given as,

$$L_{\mathcal{F}}(\mathbf{x}) = \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}\left[\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2\right]$$

Let's first consider the one-dimensional case of Fisher divergence, i.e. $x \in \mathbb{R}$,

$$\begin{aligned} L_{\mathcal{F}}(x) &= \frac{1}{2}\int p(x)\left(\nabla_x \log p(x) - \nabla_x \log p_\theta(x)\right)^2 dx \\ &= \frac{1}{2}\int p(x)(\nabla_x \log p(x))^2 dx + \frac{1}{2}\int p(x)(\nabla_x \log p_\theta(x))^2 dx \\ &\quad - \int p(x)\nabla_x \log p(x)\nabla_x \log p_\theta(x) dx \end{aligned} \tag{3.31}$$

In equation (3.31), the first term can be treated as a constant because it does not depend on any model parameters. The second term remains tractable, as it does not require knowledge of the ground truth score. However, the third term depends on both the model parameters and the ground truth score, and therefore requires further simplification.

Simplifying just the third term of equation (3.31):

$$\begin{aligned} &= -\int p(x)\nabla_x \log p(x)\nabla_x \log p_\theta(x) dx \\ &= -\int p(x)\frac{1}{p(x)}\nabla_x p(x)\nabla_x \log p_\theta(x) dx \\ &= -\int \nabla_x \log p_\theta(x)\nabla_x p(x) dx \end{aligned} \tag{3.32}$$

Using integration by parts ($\int u\,dv = uv - \int v\,du$)

where, $u = \nabla_x \log p_\theta(x)$ and $dv = \nabla_x p(x)$. From this we get, $du = \nabla_x^2 \log p_\theta(x) dx$ and $v = p(x) + C$.

Since we are dealing with indefinite integrals, we can safely ignore the integral constant C as it cancels out in the integration by parts process. Thus, the integral in equation (3.32) simplifies to,

$$-p(x)\nabla_x \log p_\theta(x)|_{x=-\infty}^{x=\infty} + \int p(x)\nabla_x^2 \log p_\theta(x) dx = \int p(x)\nabla_x^2 \log p_\theta(x) dx \tag{3.33}$$

Here the first term is set to 0, as for any general probability distribution $p(x) = 0$, for $x = \pm\infty$.

Finally, substituting the above result in equation (3.31):

$$\begin{aligned} L_{\mathcal{F}}(x) &= \text{const} + \frac{1}{2}\mathbb{E}_{p(x)}\left[(\nabla_x \log p_\theta(x))^2\right] + \mathbb{E}_{p(x)}\left[\nabla_x^2 \log p_\theta(x)\right] \\ &= \text{const} + \mathbb{E}_{p(x)}\left[\frac{1}{2}\|\nabla_x \log p_\theta(x)\|_2^2 + \nabla_x^2 \log p_\theta(x)\right] \end{aligned} \tag{3.34}$$

Equation (3.34) expresses the Fisher divergence objective for one-dimensional data $x \in \mathbb{R}$. For multivariate data $\mathbf{x} \in \mathbb{R}^{\mathrm{D}}$, the Fisher divergence takes the following form:

$$L_{\mathcal{F}}(\mathbf{x}) = \mathbb{E}_{p(\mathbf{x})}\left[\frac{1}{2}\left\|\nabla_{\mathbf{x}} \log p_{\boldsymbol{\theta}}(\mathbf{x})\right\|_2^2 + \mathrm{Tr}\left(\nabla_{\mathbf{x}}^2 \log p_{\boldsymbol{\theta}}(\mathbf{x})\right)\right] + \text{const} \tag{3.35}$$

where, $\nabla_{\mathbf{x}}^2 \log p_{\boldsymbol{\theta}}(\mathbf{x}) \in \mathbb{R}^{\mathrm{D}\times\mathrm{D}}$ is the Hessian matrix of the log-density with respect to $\mathbf{x}$ and Tr(.) denotes the trace i.e. the sum of diagonal elements of a matrix. Since the constant term does not influence optimization, it can be omitted. Rewriting equation (3.35) in terms of $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$,

$$L_{\mathcal{F}}(\mathbf{x}) = \mathbb{E}_{p(\mathbf{x})}\left[\frac{1}{2}\left\|\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})\right\|_2^2 + \mathrm{Tr}\left(\nabla_{\mathbf{x}}\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})\right)\right] \tag{3.36}$$

The Jacobian $\nabla_{\mathbf{x}}\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$ is given as,

$$\nabla_{\mathbf{x}}\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}1}(\mathbf{x})}{\partial \mathbf{x}_1} & \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}1}(\mathbf{x})}{\partial \mathbf{x}_2} & \cdots & \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}1}(\mathbf{x})}{\partial \mathbf{x}_{\mathrm{D}}} \\ \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}2}(\mathbf{x})}{\partial \mathbf{x}_1} & \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}2}(\mathbf{x})}{\partial \mathbf{x}_2} & \cdots & \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}2}(\mathbf{x})}{\partial \mathbf{x}_{\mathrm{D}}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}\mathrm{D}}(\mathbf{x})}{\partial \mathbf{x}_1} & \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}\mathrm{D}}(\mathbf{x})}{\partial \mathbf{x}_2} & \cdots & \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}\mathrm{D}}(\mathbf{x})}{\partial \mathbf{x}_{\mathrm{D}}} \end{bmatrix} \tag{3.37}$$

Its trace is the sum of the diagonal entries:

$$\mathrm{Tr}(\nabla_{\mathbf{x}}\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})) = \sum_{i=1}^{\mathrm{D}} \frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}i}(\mathbf{x})}{\partial \mathbf{x}_i} \tag{3.38}$$

The objective in equation (3.36) is known as score matching objective. A key advantage of this formulation is that it eliminates the dependence on the ground truth score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, which is typically intractable. Moreover, the expectation in score matching can be efficiently approximated using the empirical mean computed over the training dataset.

However, the score matching objective is not scalable when using deep neural networks (DNNs) to model high-dimensional data. The objective consists of two terms: the squared Euclidean norm of the score model output $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$, and the trace of its Jacobian with respect to $\mathbf{x}$. The first term is straightforward to compute, requiring only a forward pass followed by a simple ℓ_2 norm calculation. In contrast, the second term is considerably more complex to evaluate. To compute each diagonal entry of the Jacobian (equation (3.37)), we must perform a forward pass to obtain the corresponding component of the score function output, followed by a separate backward pass; this process must be repeated for every dimension. As a result, the overall complexity is $\mathcal{O}(\mathrm{D})$

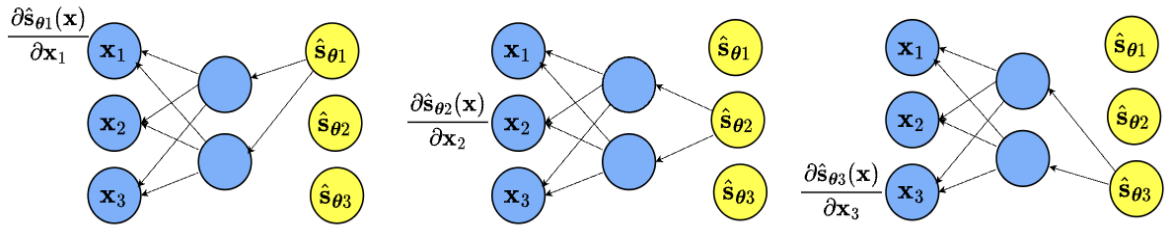


Figure 3.4: This image illustrates the process of obtaining each diagonal element $\frac{\partial \hat{\mathbf{s}}_{\boldsymbol{\theta}_i}(\mathbf{x})}{\partial \mathbf{x}_i}$ through separate backward passes for the corresponding score function outputs. Since this computation must be repeated for every dimension, the overall complexity is $\mathcal{O}(\mathrm{D})$ times that of a single backward pass. This shows the inefficiency of this approach for high-dimensional data.

times that of standard backpropagation, making the computation highly inefficient and non-scalable. A pictorial representation of this is shown in fig. 3.4.

To address the computational challenges of score matching, several approaches are proposed in the literature. Notably, sliced score matching and denoising score matching offer practical alternatives for computing the Jacobian trace. These methods keeps the core objective of learning the score function while avoiding the expensive per dimension backpropagation required by vanilla score matching. We explore these approaches in more detail, highlighting their formulations, and advantages in high-dimensional settings.

3.2.2 Sliced score matching

Sliced Score Matching [56] is a scalable alternative to traditional score matching, designed to overcome the computational challenges posed by high-dimensional data. The idea here is to simplify the problem by breaking it down into multiple one-dimensional components as learning patterns in one dimension is significantly easier than doing so in high dimensions. Sliced score matching does this by projecting both the ground truth score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ and the estimated score function $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$ onto randomly sampled one-dimensional directions. For a random unit vector $\mathbf{u} \in \mathbb{R}^{\mathrm{D}}$, the projections of both the true and estimated score function onto the direction $\mathbf{u}$ is given by $\mathbf{u}^\top \nabla_{\mathbf{x}} \log p(\mathbf{x})$ and $\mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$ respectively. This reduces the high-dimensional score matching problem into a series of one-dimensional subproblems that are computationally much cheaper to solve.

A justification behind this approach lies in the geometric intuition that two vector fields are close in high-dimensional space if their projections onto many random directions are also close. By averaging over multiple such random projections, we can obtain a good approximation of the score matching objective. This technique not only maintains

the statistical fidelity of the original objective but also makes it far more tractable for modern deep learning applications. This idea is defined through a new objective, called the sliced Fisher divergence, and is given as,

$$L_{\mathcal{SF}}(\mathbf{x}) = \frac{1}{2}\mathbb{E}_{p(\mathbf{u})}\mathbb{E}_{p(\mathbf{x})}\left[(\mathbf{u}^\top \nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x}))^2\right] \tag{3.39}$$

Here, $\mathbf{u}$ denotes a random projection direction, sampled from a distribution $p(\mathbf{u})$ which is generally a uniform distribution over the unit sphere. The objective measures the expected squared error between the projections of the true data score and the model score along direction $\mathbf{u}$, averaged over both the data distribution and the distribution of random projections. Similar to standard score matching, this formulation also involves the intractable ground truth score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. However, we can again use the technique of integration by parts to eliminate this term, resulting in the following equivalent form,

$$L_{\mathcal{SF}}(\mathbf{x}) = \mathbb{E}_{p(\mathbf{u})}\mathbb{E}_{p(\mathbf{x})}\left[\mathbf{u}^\top \nabla_{\mathbf{x}} \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})\mathbf{u} + \frac{1}{2}\left(\mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})\right)^2\right] \tag{3.40}$$

The scalability issue now vanishes as mathematically, $\mathbf{u}^\top \nabla_{\mathbf{x}} \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})\mathbf{u} = \mathbf{u}^\top \nabla_{\mathbf{x}}(\mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x}))$, which corresponds to computing the derivative of the score function output along $\mathbf{u}$. This derivative requires only a single backward pass, as compared to computing the trace of the Jacobian. This is illustrated in fig. 3.5. Consequently, the computational complexity is reduced to that of standard gradient computation, making sliced score matching scalable to high-dimensional data.

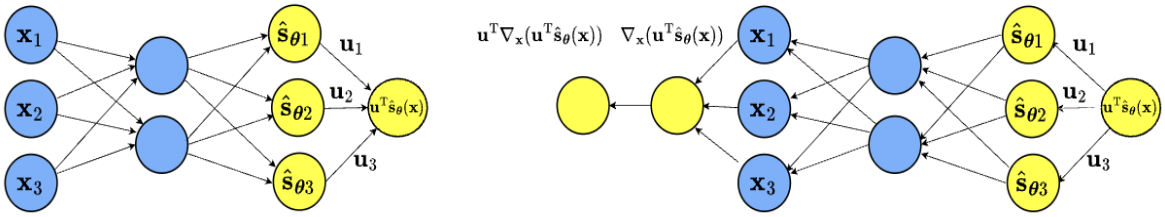


Figure 3.5: Left: Forward pass through the network to compute the projected score $\mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$. Right: Backward pass to compute the gradient $\nabla_{\mathbf{x}}(\mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x}))$, followed by the directional projection $\mathbf{u}^\top \nabla_{\mathbf{x}}(\mathbf{u}^\top \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x}))$. This approach requires only a single backward pass, avoiding the need to compute the full Jacobian or its trace.

3.2.3 Denosing score matching

An alternative to the computationally expensive vanilla score matching is Denoising Score Matching (DSM) [66]. DSM provides a workaround for calculating ground truth

score $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, by adding Gaussian noise to the data, which makes the score easier to compute and estimate. The key idea is to perturb the data $\mathbf{x}$ by adding noise $\boldsymbol{\epsilon} \in \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$, to get a noisy version $\mathbf{z}$ and therefore learn the score of the perturbed distribution $\nabla_{\mathbf{x}} \log q(\mathbf{z}|\mathbf{x})$ which can be mathematically derived, instead of the original score that is not tractable.

The perturbed distribution $q(\mathbf{z}|\mathbf{x})$ is typically Gaussian, given by:

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mathbf{x}, \sigma^2 \boldsymbol{I}) \tag{3.41}$$

and its log-probability is:

$$\log q(\mathbf{z}|\mathbf{x}) = -\frac{1}{2\sigma^2} \|\mathbf{z} - \mathbf{x}\|^2 - \frac{\mathrm{D}}{2} \log(2\pi\sigma^2) \tag{3.42}$$

From equation (3.42) we can easily get the score of perturbed data distribution, given as,

$$\nabla_{\mathbf{z}} \log q(\mathbf{z}|\mathbf{x}) = -\frac{(\mathbf{z} - \mathbf{x})}{\sigma^2} \tag{3.43}$$

This formulation is analytically tractable because the score $\nabla_{\mathbf{z}} \log q(\mathbf{z}|\mathbf{x})$ has a closed-form expression that depends only on the difference between the noisy input and the original data, scaled by the noise variance. The DSM objective trains the model $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z})$, to approximate this score by minimizing the loss function given as:

$$L_{\mathcal{DSM}}(\mathbf{x}) = \mathbb{E}_{p(\mathbf{x}), q(\mathbf{z}|\mathbf{x})} \left[\|\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}) - \nabla_{\mathbf{z}} \log q(\mathbf{z}|\mathbf{x})\|^2 \right] \tag{3.44}$$

3.2.4 Sample generation using Langevin dynamics

Having established that we can use principled statistical techniques, such as score matching, to train a model and estimate the score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ from given data $\mathbf{x}$, the next challenge is to use this score to construct a generative model. Specifically, we need a method to generate new data points from the vector field defined by the estimated score function.

Suppose we start with a collection of random points drawn from a standard Gaussian distribution. Now, the question is how can we transform these points into valid samples from the target distribution $p(\mathbf{x})$ using the score function. One approach is to move each point along the direction indicated by the score, which points toward regions of higher probability density. The problem in this approach is that without additional regularization, all data points would converge to the modes of the distribution, collapsing into small set of high density locations and would fail to capture the full diversity of the target distribution.

Input: Score function $\nabla_{\mathbf{x}} \log p(\mathbf{x})$, step size α, number of steps T
Output: Sample $\mathbf{x}^{\mathrm{T}} \sim p(\mathbf{x})$
Initialize $\mathbf{x}^0 \sim \pi(\mathbf{x})$ `// Prior distribution, e.g., Gaussian`
for $t = 1$ **to** T **do**
  Sample $w^t \sim \mathcal{N}(0, \boldsymbol{I})$
  Update $\mathbf{x}^t = \mathbf{x}^{t-1} + \alpha \nabla_{\mathbf{x}} \log p(\mathbf{x}^{t-1}) + \sqrt{2\alpha}\, w^t$
end
return $\mathbf{x}^{\mathrm{T}}$

Algorithm 1: Langevin Dynamics with Score Function

To address this, we introduce randomness into the process by combining the score function with Gaussian noise. This addition of random noise prevents data points from simply piling up at the modes and instead allows them to explore the distribution more broadly. If we keep the sampling procedure long enough to reach convergence, and set the step size to be very small then we can generate good quality diverse samples. This method is known as Langevin dynamics, which is a stochastic method widely used for sampling.

3.2.5 Score matching at multiple noise levels

While Langevin dynamics provides a good framework for generating samples using the estimated score function. In practice, there still exists some challenges to generate samples using score estimation starting from a randomly generated sample. In principle, the score $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{x})$, learned via score matching, captures local information near high density regions of the data distribution but provides poor estimates in low density areas. Therefore, if we generate samples by starting from random noise in low density regions and evolve them using the score, the model struggles to guide samples toward the true data manifold. As a result, Langevin dynamics often fails to generate diverse, high-quality samples.

Researchers found that this issue can be resolved through multi-scale score modeling framework. Rather than training a score network to estimate the score of the data distribution directly, the idea is to corrupt the data with Gaussian noise at various levels and train the network to recover the score of these noisy distributions. We define a sequence of noise scales as $\{\sigma_1, \sigma_2, \ldots, \sigma_K\}$ where $\sigma_1 < \sigma_2 < \cdots < \sigma_K$, ranges from low to high noise levels. For each noise scale σ_i, the data $\mathbf{x}$ is corrupted to produce a noisy version $\mathbf{z}_i = \mathbf{x} + \boldsymbol{\epsilon}_i$, where $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \sigma_i^2 \boldsymbol{I})$. The perturbed distribution is then given by, $q_{\sigma_i}(\mathbf{z}_i|\mathbf{x}) = \mathcal{N}(\mathbf{z}_i; \mathbf{x}, \sigma_i^2 \boldsymbol{I})$. The score network, parameterized by $\boldsymbol{\theta}$ and denoted as $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_i; \sigma_i)$, is trained to approximate the score of this noisy distribution, $\nabla_{\mathbf{z}} \log q_{\sigma_i}(\mathbf{z}_i|\mathbf{x})$ which can be derived analytically as shown in equation (3.43). The training objective for

this multi-scale denoising score matching framework is to minimize the expected squared error across all noise levels, given by:

$$L_{\mathcal{DSM}}(\mathbf{x}) = \sum_{i=1}^{K} \mathbb{E}_{p(\mathbf{x})q(\mathbf{z}|\mathbf{x})} \left[\|\nabla_{\mathbf{z}} \log q_{\sigma_i}(\mathbf{z}_i|\mathbf{x}) - \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_i; \sigma_i)\|_2^2 \right] \tag{3.45}$$

where $\mathbf{z}_i$ is the noisy data at scale σ_i. This loss guides the score network to accurately predict the direction from $\mathbf{z}_i$ back to the clean data $\mathbf{x}$ at each noise level.

Training the model across multiple noise levels makes it more robust, allowing it to capture the structure of the entire data manifold, including regions with low data densities. This framework, forms the backbone of modern score-based generative models, achieving state-of-the-art performance in generating diverse, high-quality samples. Building on this foundation, we will next explore how the principles of score matching connect to denoising diffusion models and will formulate the equivalent NELBO and weighted loss in $\mathbf{s}$-space.

3.2.6 Loss formulation in s-space

The concept of multiscale score estimation naturally transitions into the formulation of diffusion based generative models. However, unlike score-based models, which add noise at discrete levels, diffusion based models incorporate noise progressively over several time steps, providing a smooth and dynamic noising process.

As discussed in section 3.1, diffusion models operate over time interval $t \in [0, 1]$. In this framework, the data is progressively corrupted by adding noise according to the equation:

$$\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon}$$

where α_t and σ_t are the scheduling parameters and noise $\boldsymbol{\epsilon}$ is sampled from a standard normal distribution.

In $\mathbf{s}$-space the goal is to learn a time dependent score model $\hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t, t)$ that approximates the true score of the data distribution at time t.

$$\nabla_{\mathbf{z}_t} \log q(\mathbf{z}_t \mid \mathbf{x}) \approx \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t) \tag{3.46}$$

The NELBO loss formulation in $\mathbf{x}$-space as derived in section 3.1.4 is expressed as,

$$L(\mathbf{x}) = -\mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t) \, \|\mathbf{x} - \hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|_2^2 \, dt$$

To express the equivalent NELBO loss in $\mathbf{s}$-space, we use Tweedie's formula [11], which provides a connection between the score function and the denoising model. It states

that, given samples from a distribution, we can estimate its true mean by the maximum likelihood estimate of samples which is the empirical mean and a correctional term that uses the score of the estimate. For a random Gaussian variable $\mathbf{z} \sim \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$, Tweedie's formula is expressed as:

$$\mathbb{E}[\boldsymbol{\mu}_z \mid \mathbf{z}] = \mathbf{z} + \boldsymbol{\Sigma}_z \nabla_{\mathbf{z}} \log p(\mathbf{z}). \tag{3.47}$$

In our case, we have $q(\mathbf{z}_t \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}_t; \alpha_t \mathbf{x}, \sigma_t^2 \boldsymbol{I})$, and we use Tweedie's formula to get the true posterior mean of $q(\mathbf{z}_t \mid \mathbf{x})$,

$$\alpha_t \mathbf{x} = \mathbf{z}_t + \sigma_t^2 \nabla \log q(\mathbf{z}_t|\mathbf{x}) \tag{3.48}$$

For notational simplicity we use just ∇ for $\nabla_{\mathbf{z}_t}$. From equation (3.48) we can formulate $\mathbf{x}$ in terms of score $\nabla \log q(\mathbf{z}_t|\mathbf{x})$, and can derive loss in $\mathbf{s}$-space,

$$\mathbf{x} = \frac{\mathbf{z}_t + \sigma_t^2 \nabla \log q(\mathbf{z}_t|\mathbf{x})}{\alpha_t} \tag{3.49}$$

Substituting this in equation (3.21),

$$\begin{aligned} L(\mathbf{s}) &= -\mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t) \left\| \frac{\mathbf{z}_t}{\alpha_t} + \frac{\sigma_t^2}{\alpha_t} \nabla \log q(\mathbf{z}_t|\mathbf{x}) - \frac{\mathbf{z}_t}{\alpha_t} - \frac{\sigma_t^2}{\alpha_t} \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t) \right\|_2^2 dt \\ &= -\mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), t \sim \mathcal{U}[0,1]} \left[\mathrm{SNR}'(t) \left\| \frac{\mathbf{z}_t}{\alpha_t} + \frac{\sigma_t^2}{\alpha_t} \nabla \log q(\mathbf{z}_t|\mathbf{x}) - \frac{\mathbf{z}_t}{\alpha_t} - \frac{\sigma_t^2}{\alpha_t} \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t) \right\|_2^2 \right] \\ &= -\mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t) \frac{\sigma_t^4}{\alpha_t^2} \left\| \nabla \log q(\mathbf{z}_t|\mathbf{x}) - \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t) \right\|_2^2 dt \end{aligned} \tag{3.50}$$

Hence, the NELBO loss in $\mathbf{s}$-space is given as,

$$L(\mathbf{s}) = -\mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), t \sim \mathcal{U}[0,1]} \left[\mathrm{SNR}'(t) \frac{\sigma_t^4}{\alpha_t^2} \left\| \nabla \log q(\mathbf{z}_t|\mathbf{x}) - \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t) \right\|_2^2 \right] \tag{3.51}$$

In $\mathbf{s}$-space the weighted loss can be formulated as the weighted function of NELBO with weight $w_{\mathbf{s}}(t) = -\frac{\alpha_t^2 \mathrm{SNR}'(t)}{\sigma_t^4}$,

$$\mathcal{L}(\mathbf{s}) = \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), t \sim \mathcal{U}[0,1]} \left[\left\| \nabla \log q(\mathbf{z}_t \mid \mathbf{x}) - \hat{\mathbf{s}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t) \right\|_2^2 \right] = w_{\mathbf{s}}(t) L(\mathbf{s}) \tag{3.52}$$

Moreover as discussed in section 3.2.3, for Gaussian noise perturbation, the ground truth score simplifies to,

$$\nabla \log q(\mathbf{z}_t \mid \mathbf{x}) = -\frac{(\mathbf{z}_t - \alpha_t \mathbf{x})}{\sigma_t^2} \tag{3.53}$$

3.3 Angular parameterization in diffusion models

A common choice for the loss formulation in standard diffusion models is the $\boldsymbol{\epsilon}$-space loss. However, this approach faces challenges during sampling, particularly when the SNR(t) approaches zero. In such cases, the scaling factor α_t becomes very small, leading to inaccuracies in reconstructing the data from the predicted noise $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t)$. Therefore the reconstructions using $\hat{\mathbf{x}}_{\boldsymbol{\theta}}(\mathbf{z}_t) = \frac{1}{\alpha_t}\left(\mathbf{z}_t - \sigma_t \hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\mathbf{z}_t)\right)$ becomes unreliable. To address this instability we generally use large number of sampling steps and clip the reconstructed data to a valid range at each step. However, it becomes a significant problem for model distillation, where the goal is to reduce the number of sampling steps for improved efficiency. One solution to mitigate this issue is to directly predict $\mathbf{x}$ in the reverse process instead of $\boldsymbol{\epsilon}$.

Salimans and Ho [49] propopsed a novel approach to address this challenge by modeling the reverse process in diffusion models through the estimation of the rate of change in the data distribution, also known as velocity. This method, detailed in their work on progressive distillation, proved especially powerful for model distillation, and performed better than the $\mathbf{x}$-space loss. By adopting this velocity framework, they achieved faster sampling, producing high-quality outputs with significantly fewer steps.

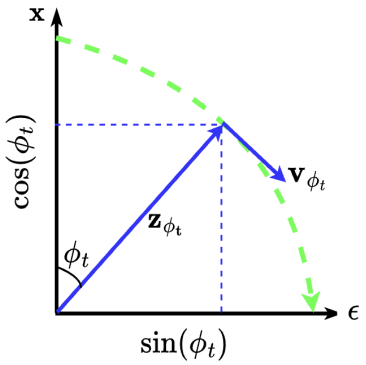


Figure 3.6: Illustration of angular parameterization in diffusion models: The noisy data $\mathbf{z}_{\phi_t}$ lies on a circular trajectory between clean data $\mathbf{x}$ and $\boldsymbol{\epsilon}$, with the angle ϕ_t defining the mixture. The velocity $\mathbf{v}_{\phi_t}$ indicates the rate of change in data along this path.

Angular parameterization interprets the noisy data $\mathbf{z}_{\phi_t}$ on a circular trajectory in a two dimensional space, parameterized by angle ϕ_t. The data evolution can be visualized as motion along this trajectory as illustrated in fig. 3.6. This can be mathematically formulated as the relationship between the clean data $\mathbf{x}$ and noise $\boldsymbol{\epsilon}$, given in equation (3.54), where $\phi_t = \arctan\left(\frac{\sigma_t}{\alpha_t}\right)$.

$$\mathbf{z}_{\phi_t} = \cos(\phi_t)\mathbf{x} + \sin(\phi_t)\boldsymbol{\epsilon} \tag{3.54}$$

The angular parameterization weights the contributions of data and noise, with $\cos(\phi_t)$ and $\sin(\phi_t)$ respectively. The instantaneous rate of change of $\mathbf{z}_{\phi_t}$ with respect to ϕ_t, denoted by $\mathbf{v}_{\phi_t}$ is given by,

$$\mathbf{v}_{\phi_t} = \frac{d\mathbf{z}_{\phi_t}}{d\phi_t} = \cos(\phi_t)\boldsymbol{\epsilon} - \sin(\phi_t)\mathbf{x} \tag{3.55}$$

3.3.1 Loss formulation in v-space

We will now formulate the NELBO loss in $\mathbf{v}$-space. As $\phi_t = \arctan\left(\frac{\sigma_t}{\alpha_t}\right)$, we can express,

$$\cos(\phi_t) = \frac{\alpha_t}{\sqrt{\alpha_t^2 + \sigma_t^2}} \tag{3.56}$$

$$\sin(\phi_t) = \frac{\sigma_t}{\sqrt{\alpha_t^2 + \sigma_t^2}} \tag{3.57}$$

From equation (3.55), we have,

$$\mathbf{v}_{\phi_t} = \frac{d\mathbf{z}_{\phi_t}}{d\phi_t} = \frac{d\cos(\phi_t)\mathbf{x}}{d\phi_t} + \frac{d\sin(\phi_t)\boldsymbol{\epsilon}}{d\phi_t} = \cos(\phi_t)\boldsymbol{\epsilon} - \sin(\phi_t)\mathbf{x}$$

By rearranging the terms in above equation we get,

$$\begin{aligned}
\sin(\phi_t)\mathbf{x} &= \cos(\phi_t)\boldsymbol{\epsilon} - \mathbf{v}_{\phi_t} \\
\sin(\phi_t)\mathbf{x} &= \frac{\cos(\phi_t)(\mathbf{z}_{\phi_t} - \cos(\phi_t)\mathbf{x})}{\sin(\phi_t)} - \mathbf{v}_{\phi_t} \\
\mathbf{x}(\sin^2(\phi_t) + \cos^2(\phi_t)) &= \cos(\phi_t)\mathbf{z}_{\phi_t} - \sin(\phi_t)\mathbf{v}_{\phi_t} \\
\mathbf{x} &= \cos(\phi_t)\mathbf{z}_{\phi_t} - \sin(\phi_t)\mathbf{v}_{\phi_t}
\end{aligned} \tag{3.58}$$

Substiuting the value of x from equation (3.58) in equation (3.21),

$$\begin{aligned}
L(\mathbf{v}) &= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t) \Big\| (\cos(\phi_t)\mathbf{z}_{\phi_t} - \sin(\phi_t)\mathbf{v}) \\
&\quad - (\cos(\phi_t)\mathbf{z}_{\phi_t} - \sin(\phi_t)\hat{\mathbf{v}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)) \\
&= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \mathrm{SNR}'(t) \sin^2\phi_t \left\|\mathbf{v} - \hat{\mathbf{v}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\right\|_2^2 dt
\end{aligned} \tag{3.59}$$

By substituting the value for $\sin^2\phi_t$ from equation (3.57), we get,

$$\begin{aligned}
L(\mathbf{v}) &= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \int_0^1 \frac{\sigma_t^2}{\alpha_t^2 + \sigma_t^2} \mathrm{SNR}'(t) \left\|\mathbf{v} - \hat{\mathbf{v}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\right\|_2^2 dt \\
L(\mathbf{v}) &= -\mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}),t\sim\mathcal{U}[0,1]} \left[\frac{\sigma_t^2}{\alpha_t^2 + \sigma_t^2} \mathrm{SNR}'(t) \left\|\mathbf{v} - \hat{\mathbf{v}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\right\|_2^2 \right]
\end{aligned} \tag{3.60}$$

Equation (3.60) represents the NELBO formulation in $\mathbf{v}$-space. The weighted $\mathbf{v}$ loss can be formulated as the weighted function of NELBO with weight $w_{\mathbf{v}}(t) = -\frac{(\alpha_t^2+\sigma_t^2)}{\sigma_t^2 \mathrm{SNR}'(t)}$,

$$\mathcal{L}(\mathbf{v}) = \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}),t\sim\mathcal{U}[0,1]}\left[\|\mathbf{v} - \hat{\mathbf{v}}_{\boldsymbol{\theta}}(\mathbf{z}_t; t)\|_2^2\right] = w_{\mathbf{v}}(t)L(\mathbf{v}) \tag{3.61}$$

3.4 Equivalence of loss functions

In the previous sections we derived the NELBO and weighted loss formulations in $\mathbf{x}$, $\boldsymbol{\epsilon}$, $\mathbf{s}$ and $\mathbf{v}$ space. Each of the NELBO loss originates from the same foundational equation, as presented in equation (3.21). This shared origin implies that, these formulations are theoretically equivalent, as they all aim to optimize the same underlying objective of modeling the data distribution through the diffusion process. However, the practical implementation of these losses changes when we transition to their weighted forms. To obtain the weighted loss from the NELBO loss, we introduce specific weighting factors different for every target objective. These weights essentially removes the SNR scalings associated with the ℓ_2 difference between the target and the prediction. As a consequence of this weighting process, the resulting weighted loss functions are no longer equivalent.

To establish a unified relationship between the weighted loss formulations, we rescale the losses in the $\boldsymbol{\epsilon}$, $\mathbf{v}$, and $\mathbf{s}$ space to make them equivalent to $\mathcal{L}(\mathbf{x})$. We denote these as the rescaled losses $\widetilde{\mathcal{L}}$, which provide consistency across all target spaces and can be efficiently obtained using the weights derived in the previous sections.

$$\widetilde{\mathcal{L}}(\boldsymbol{\epsilon}) = \frac{\sigma_t^2}{\alpha_t^2}\mathcal{L}(\boldsymbol{\epsilon}) = \mathcal{L}(\mathbf{x}) \tag{3.62}$$

$$\widetilde{\mathcal{L}}(\mathbf{v}) = \frac{\sigma_t^2}{\alpha_t^2+\sigma_t^2}\mathcal{L}(\mathbf{v}) = \mathcal{L}(\mathbf{x}) \tag{3.63}$$

$$\widetilde{\mathcal{L}}(\mathbf{s}) = \frac{\sigma_t^4}{\alpha_t^2}\mathcal{L}(\mathbf{s}) = \mathcal{L}(\mathbf{x}) \tag{3.64}$$

Table 3.2 provides a summary of the various loss formulations across different targets for the denoising model.

Although researchers have suggested that certain training objectives outperform others in diffusion models, the underlying reasons for these differences remain unclear and are often based on empirical observations rather than solid theoretical foundations. For example, while some works prefer more complex weights for loss functions [6, 24, 15], others [20, 38] have found that simpler objectives, such as the ℓ_2 loss between target and prediction can perform just as well or even better in practice. This inconsistency raises important questions about the fundamental role of loss formulations in training diffusion

Table 3.2: Overview of all the loss formulations across different scenarios. While the NELBO and the rescaled loss are equivalent and comparable, the weighted losses are not equivalent and are expected to exhibit different empirical performance.

Target	**NELBO loss** (L)	**Weighted loss** ($\mathcal{L}$)	**Rescaled loss** ($\widetilde{\mathcal{L}}$)
$\mathbf{x}$	$-\mathbb{E}\big[\mathrm{SNR}'(t)\,\|\mathbf{x}-\hat{\mathbf{x}}_{\boldsymbol{\theta}}\|_2^2\big]$	$\mathbb{E}\big[\|\mathbf{x}-\hat{\mathbf{x}}_{\boldsymbol{\theta}}\|_2^2\big]$	$\mathbb{E}\big[\|\mathbf{x}-\hat{\mathbf{x}}_{\boldsymbol{\theta}}\|_2^2\big]$
$\boldsymbol{\epsilon}$	$-\mathbb{E}\left[\frac{\mathrm{SNR}'(t)}{\mathrm{SNR}(t)}\|\boldsymbol{\epsilon}-\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\|_2^2\right]$	$\mathbb{E}\big[\|\boldsymbol{\epsilon}-\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\|_2^2\big]$	$\mathbb{E}\left[\frac{\sigma_t^2}{\alpha_t^2}\|\boldsymbol{\epsilon}-\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}\|_2^2\right]$
$\mathbf{v}$	$-\mathbb{E}\left[\frac{\sigma_t^2}{\alpha_t^2+\sigma_t^2}\,\mathrm{SNR}'(t)\,\|\mathbf{v}-\hat{\mathbf{v}}_{\boldsymbol{\theta}}\|_2^2\right]$	$\mathbb{E}\big[\|\mathbf{v}-\hat{\mathbf{v}}_{\boldsymbol{\theta}}\|_2^2\big]$	$\mathbb{E}\left[\frac{\sigma_t^2}{\alpha_t^2+\sigma_t^2}\|\mathbf{v}-\hat{\mathbf{v}}_{\boldsymbol{\theta}}\|_2^2\right]$
$\mathbf{s}$	$-\mathbb{E}\left[\frac{\sigma_t^4}{\alpha_t^2}\mathrm{SNR}'(t)\,\|\mathbf{s}-\hat{\mathbf{s}}_{\boldsymbol{\theta}}\|_2^2\right]$	$\mathbb{E}\big[\|\mathbf{s}-\hat{\mathbf{s}}_{\boldsymbol{\theta}}\|_2^2\big]$	$\mathbb{E}\left[\frac{\sigma_t^4}{\alpha_t^2}\|\mathbf{s}-\hat{\mathbf{s}}_{\boldsymbol{\theta}}\|_2^2\right]$

models, and whether the observed performance variations stem from the loss functions themselves or from other factors, such as model architecture, training dynamics, or noise schedules.

In our theoretical analysis, we derived the NELBO loss for various denoising models. We demonstrated that different formulations of the learned model such as predicting the original data $\mathbf{x}$, the noise $\boldsymbol{\epsilon}$, the rate of change of the data distribution $\mathbf{v}$, and the score function $\mathbf{s}$, can be mapped to one another, with their corresponding NELBO objectives being mathematically interchangeable. Additionally, we formulated the relationships between the weighted loss formulations.

In principle, the mathematical equivalence between different target objectives we established, should hold when training the various denoising models with equivalent loss formulations, provided the models are trained under similar conditions (dataset, model architecture etc.). In the next section, we outline the experiments conducted to validate this hypothesis and provide a detailed analysis of the results obtained.

4 Experiments

In this chapter, we start by providing an overview of the experimental setup used to conduct our experiments. This includes a description of the datasets used, the model architectures chosen as well as the evaluation metrics used to assess performance. Following the setup, we present the results obtained from these experiments, highlighting notable observations and an in-depth analysis of the findings.

4.1 Datasets

For the experiments we work on two types of datasets: 2-dimensional synthetic datasets and high-dimension image dataset. 2D datasets, are particularly useful for their simplicity and interpretability. These datasets allow for straightforward visualization of the whole data manifold and provide an intuitive understanding of how the model operates. By working with 2D data, we can directly observe the effects of various noise levels, diffusion steps, and loss formulations on the generated samples. This clarity makes 2D datasets an excellent starting point for testing hypotheses and debugging models. To complement the insights gained from 2D datasets, we also conduct experiments on a high dimensional image dataset. Although we evaluate our approach on an image dataset, our primary goal is to analyze how different loss functions influence model behavior rather than conduct large scale image generation experiments. These findings establish a framework for future work to more thoroughly investigate the role of loss formulations in image based tasks.

4.1.1 2D datasets

To ensure generalizability, we use four types of 2D synthetic datasets: Gaussian clusters, intersecting rings, Swiss roll, and parallel waves as shown in fig. 4.1. Each dataset has a distinct shape and structure, helping us evaluate model performance across a variety of data patterns. These datasets are generated programmatically and contain 50K samples each. The diversity of these datasets enables us to thoroughly assess the model's ability to handle different geometric complexities and data distributions.

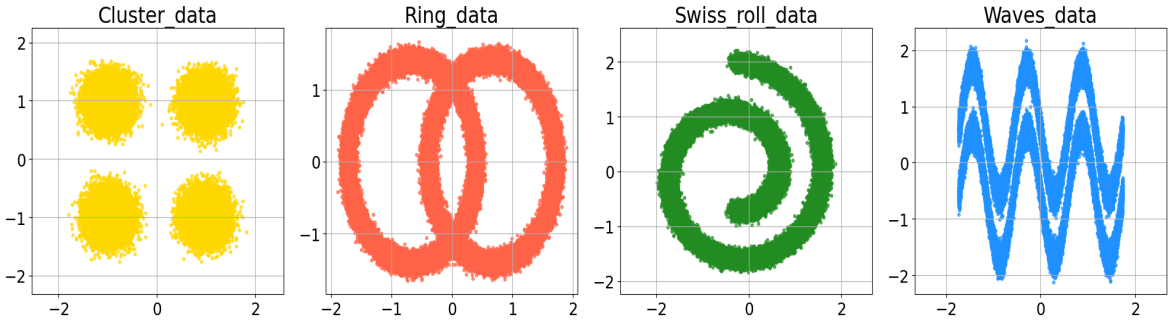


Figure 4.1: Scatter plots of the 2D synthetic datasets, each containing 50K samples, used in the experiments. From left to right: (a) Gaussian clusters, with four separate groups; (b) Intersecting rings, showing two overlapping circular bands; (c) Swiss roll, exhibiting a spiral shape; and (d) Parallel waves, consisting of two adjacent sinusoidal curves. Each dataset presents unique geometric challenges for evaluating model performance.

The Gaussian clusters dataset comprises four distinct groups of points drawn from normal distributions centered at the corners of a square, testing the model's ability to identify well separated, localized groups. The intersecting rings dataset features two overlapping circular bands, with points sampled at different angles and radius to create thickness, challenging the models to capture non-linear, overlapping structures. The swiss roll dataset is a classical example in generative modeling. It forms a spiral like pattern in 2D, with points sampled along a continuous curve and noise added for variability, evaluating the model's ability to learn smooth, curved manifolds. Lastly, the waves dataset consists of two parallel sinusoidal curves with controlled noise, assessing the model's sensitivity to closely spaced, non-linear patterns. All these datasets are standardized to ensure consistent scale.

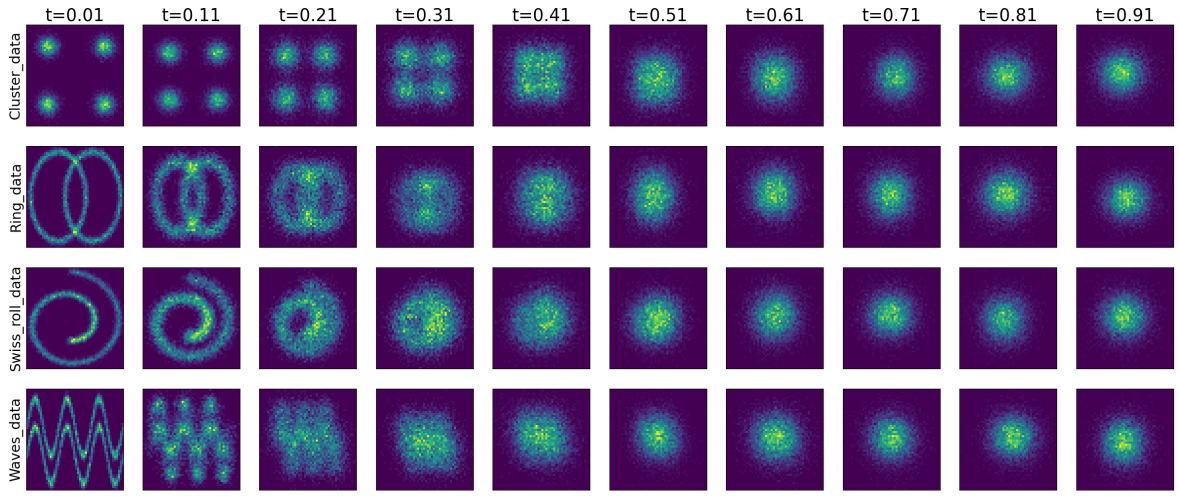


Figure 4.2: Illustration of the forward diffusion process on 2D synthetic datasets, showing the effect of adding cosine-scheduled Gaussian noise over time $t \in [0, 1]$.

Fig. 4.2 shows the effect of the forward noising process with a cosine schedule applied to all four datasets at different time steps. As noise is progressively added, the datasets transform, with their original structures becoming more blurred and distorted, ultimately converging to an isotropic Gaussian distribution.

4.1.2 Image dataset

For the high dimensional image dataset, we use the CIFAR-10 dataset, which is a widely adopted benchmark in computer vision tasks. CIFAR-10 consists of 60,000 color images, each of size 32×32 pixels, divided into 10 distinct classes, including objects like airplanes, cars, birds, and cats, among others. The dataset is split into 50,000 training images and 10,000 test images. Despite its relatively small image size, CIFAR-10 dataset poses significant challenges for models due to the diversity of object types, variations in their appearance, and complex background patterns. Its balanced class distribution and well-defined structure make it suitable for evaluating generative and classification models. This dataset is widely recognized as a standard benchmark, providing a consistent framework to test and compare model performance. Fig. 4.3 shows 100 randomly selected samples from the CIFAR-10 dataset.

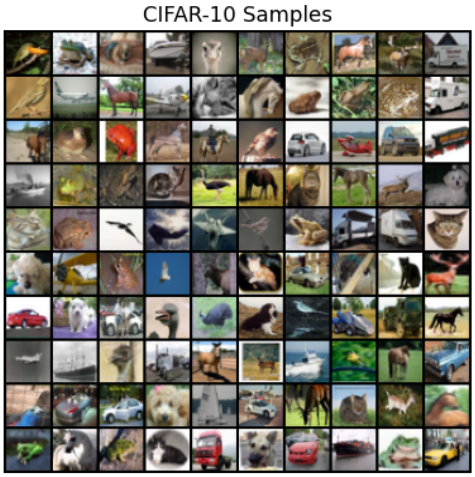


Figure 4.3: 100 random samples from the CIFAR-10 dataset with each image representing one of the 10 classes showing the diversity of object classes, appearances, and backgrounds.

4.2 Model architecture

For modeling the 2D dataset, we chose a simple neural network architecture consisting of 7 fully connected layers, each followed by a ReLU activation function as shown in fig 4.4. This architecture is chosen for its simplicity and efficiency, working well across all four datasets used in our experiments. The decision to use fully connected layers is driven by their ability to model the data efficiently while keeping the model interpretable and computationally manageable. The ReLU activation is selected due to its effectiveness in preventing vanishing gradients and adding non-linearity to the model.

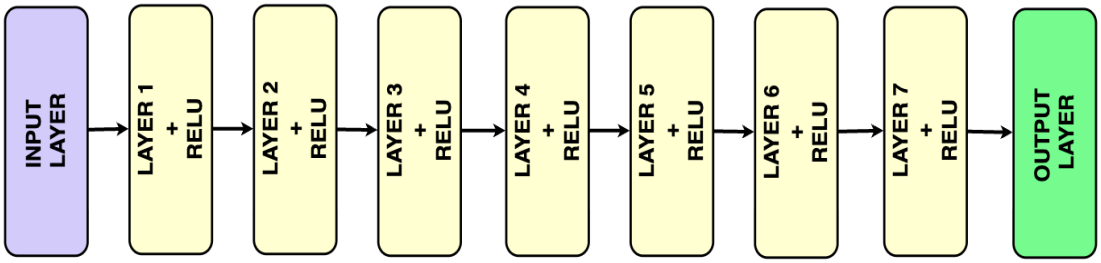


Figure 4.4: Neural network architecture with 7 fully connected layers and ReLU activation for 2D dataset modeling

In case of high-dimensional image dataset, we used a more complex UNet-based architecture inspired by [47] [12], which is a classic deep learning framework for image segmentation tasks. The UNet architecture is known for its success in handling high-dimensional data due to its encoder-decoder structure. The encoder consists of a series of downsampling blocks, each of which typically reduces the spatial dimensions of the input by half while increasing the number of feature channels. The decoder, on the other hand, is composed of upsampling blocks that progressively restore the spatial dimensions and reduce the feature channels to match the input shape. The upsampling blocks also combine the outputs from corresponding downsampling blocks at the same resolution using residual connections. In addition to these downsampling and upsampling components, the architecture includes mid blocks which takes in the input from the downsampling block, work at the same spatial resolution and pass the output to the upsampling blocks. Most diffusion model architectures follow a similar structural framework and differ mostly in the specific operations performed within these blocks. Moreover, the number of blocks can be adjusted based on the specific use case requirements. In this study, we used four blocks each for upsampling and downsampling, along with three mid blocks. A high-level diagram of this framework is shown in fig. 4.5. The following section provides a more detailed explanation of the UNet architecture used in this study.

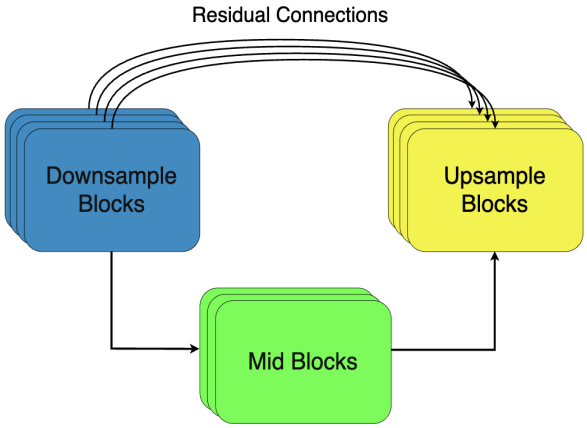


Figure 4.5: A high level illustration of UNet framework

4.2.1 UNet architecture

Before detailing the downsampling, upsampling, and mid-blocks, we first introduce their core components: time embedding block, resnet block, and the attention block as illustrated in fig. 4.6.

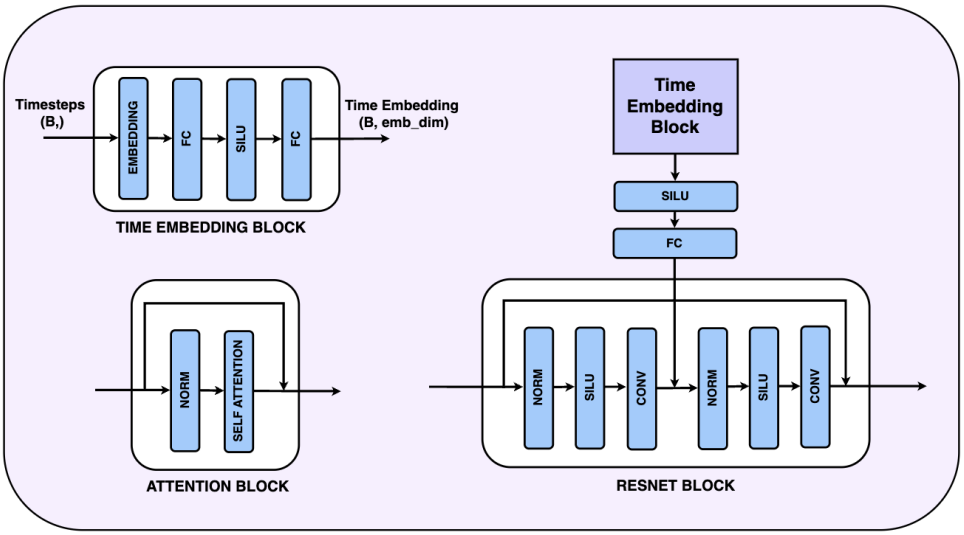


Figure 4.6: Illustration of time embedding, resnet and attention block

The time embedding block takes in a 1D tensor of batch size as input and generates embeddings for each timestep. It begins with an embedding layer that converts the timesteps into vector representations using sinusoidal encoding [65]. This is followed by a linear layer, an activation layer using the sigmoid linear unit (SiLU) activation function, and another linear layer.

The resnet block begins with a groupnorm layer, followed by an activation function and a convolutional layer. The output of this sequence is passed through another groupnorm, activation, and convolutional layer. A residual connection is added by summing the input of the first groupnorm layer with the output of the second convolutional layer. Additionally, timestep information from the time embedding block is added into the resnet block. The time embedding is first passed through an activation function and a linear layer, which reshapes it to match the number of channels after the convolution. The resulting time embedding is then added to the output of the first convolutional layer in the resnet block.

Finally, we have the attention block, which is composed of a groupnorm layer followed by a self-attention layer. A residual connection is applied by adding the input of the groupnorm layer directly to the output of the self-attention layer.

With the time embedding, resnet, and attention blocks defined, designing the downsampling, upsampling, and mid-blocks becomes clear as illustrated in fig. 4.7. The downsampling block consists of a resnet block followed by an attention block, that forms a resnet layer. Multiple resnet layers can be stacked in a block. In this study, we use a fixed number of 6 resnet layers. The mid blocks always maintains the same spatial resolution. It has one resnet block, followed by layers of resent and attention block. The upsampling block is similar to downsampling block except it starts with an upsampling layer which increases the spatial dimension and then concatenates the output from the corresponding downsampling block across the channel dimension. After that its the same layers of resnet and attention block.

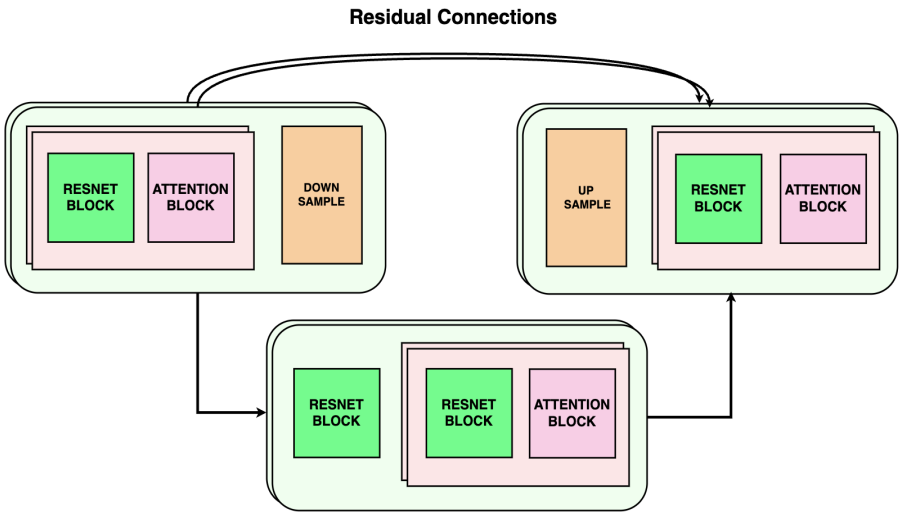


Figure 4.7: A detailed UNet architecture used in this study, with downsampling blocks which reduces the spatial dimensions, mid blocks, working at a same resolution and upsampling blocks which increase the spatial resolution and have residual connections from the corresponding down sample block.

4.3 Evaluation metrics for sample quality

To evaluate the performance of generative models in terms of sample quality, selecting appropriate metrics is important to measure how well the generated samples capture the characteristics of the target data distribution. For 2D datasets, we use moment-based metrics [2] which provide a robust way to assess statistical alignment between generated and real data, focusing on global distributional properties. For image datasets, the Fréchet Inception Distance (FID) [17] is widely used. It measures the similarity between feature distributions of generated and real images in a perceptually meaningful embedding space. In this section, we provide a detailed discussion of these metrics and show their relevance in our use case.

4.3.1 Moment based metrics

Moment based metrics [2] are useful for evaluating generative models on low dimensional datasets, where the data distribution can be characterized by statistical moments. These metrics are used to measure the difference between the real and generated data samples. In this study, we focus on the first and second moments i.e. mean and covariance, which represent the central tendency and spread of the data, respectively. Specifically, we measure the mean distance and covariance distance between the two distributions. These metrics are computationally efficient and easy to interpret which makes them a good choice for evaluating generative models in 2D settings.

Mean distance

The mean distance metric evaluates how closely the means of the real and generated data distributions align. Given a real dataset $\boldsymbol{x} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ and a generated data $\boldsymbol{y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_m\}$, where $\boldsymbol{x}_i, \boldsymbol{y}_j \in \mathbb{R}^{\mathrm{D}}$, the mean vectors $\boldsymbol{\mu}_{\boldsymbol{x}}, \boldsymbol{\mu}_{\boldsymbol{y}} \in \mathbb{R}^{\mathrm{D}}$ are defined such that their k-th components are:

$$\boldsymbol{\mu}_{\boldsymbol{x},k} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_{i,k}, \quad \boldsymbol{\mu}_{\boldsymbol{y},k} = \frac{1}{m}\sum_{j=1}^{m} \boldsymbol{y}_{j,k} \tag{4.1}$$

The mean distance is then defined as the Euclidean norm of the difference between these mean vectors and is given as,

$$\text{Mean Distance} = \|\boldsymbol{\mu}_{\boldsymbol{x}} - \boldsymbol{\mu}_{\boldsymbol{y}}\|_2 = \sqrt{\sum_{k=1}^{\mathrm{D}} (\boldsymbol{\mu}_{\boldsymbol{x},k} - \boldsymbol{\mu}_{\boldsymbol{y},k})^2} \tag{4.2}$$

where $\boldsymbol{\mu}_{\boldsymbol{x},k}$ and $\boldsymbol{\mu}_{\boldsymbol{y},k}$ denote the k-th components of $\boldsymbol{\mu}_{\boldsymbol{x}}$ and $\boldsymbol{\mu}_{\boldsymbol{y}}$, respectively.

Covariance distance

The distance between the covariance matrices measures the similarity between the spread and orientation of the real and generated data distributions. For real data $\boldsymbol{x}$ and generated data $\boldsymbol{y}$ as defined in previous section, the covariance matrices $\boldsymbol{\Sigma_x}$ and $\boldsymbol{\Sigma_y}$ are computed using the mean vectors $\boldsymbol{\mu_x}$ and $\boldsymbol{\mu_y}$ (equation (4.1)) as,

$$\boldsymbol{\Sigma_x} = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \boldsymbol{\mu_x})(\boldsymbol{x}_i - \boldsymbol{\mu_x})^\top \tag{4.3}$$

$$\boldsymbol{\Sigma_y} = \frac{1}{m-1}\sum_{j=1}^{m}(\boldsymbol{y}_j - \boldsymbol{\mu_y})(\boldsymbol{y}_j - \boldsymbol{\mu_y})^\top \tag{4.4}$$

To find the distance between the covariance matrices of the real and generated distribution, we use the Frobenius norm defined as,

$$\text{Covariance Distance} = \|\boldsymbol{\Sigma_x} - \boldsymbol{\Sigma_y}\|_F = \sqrt{\sum_{i=1}^{\mathrm{D}}\sum_{j=1}^{\mathrm{D}}(\boldsymbol{\Sigma_{x}}_{,ij} - \boldsymbol{\Sigma_{y}}_{,ij})^2} \tag{4.5}$$

where $\boldsymbol{\Sigma_{x}}_{,ij}$ and $\boldsymbol{\Sigma_{y}}_{,ij}$ denote the (i,j)-th elements of $\boldsymbol{\Sigma_x}$ and $\boldsymbol{\Sigma_y}$, respectively.

4.3.2 Fréchet inception distance

Fréchet Inception Distance (FID) [17], is a widely used metric for evaluating the quality of generative models, particularly for image datasets. It measures the similarity between the feature distributions of real and generated data, using the embeddings extracted from a pretrained inception network [59]. The FID score measures both the mean and covariance differences between these distributions in the embedding space. Let the feature embeddings of the real and generated datasets be denoted as $\boldsymbol{z_x}$ and $\boldsymbol{z_y}$, respectively. Assuming $\boldsymbol{z_x} \sim \mathcal{N}(\boldsymbol{\mu_x}, \boldsymbol{\Sigma_x})$ and $\boldsymbol{z_y} \sim \mathcal{N}(\boldsymbol{\mu_y}, \boldsymbol{\Sigma_y})$, where $\boldsymbol{\mu_x}$, $\boldsymbol{\mu_y}$ are the mean vectors, and $\boldsymbol{\Sigma_x}$, $\boldsymbol{\Sigma_y}$ are the covariance matrices of the real and generated data feature embeddings. The FID score is defined as:

$$\text{FID}(\mathbf{z}_{\boldsymbol{x}}, \mathbf{z}_{\boldsymbol{y}}) = \|\boldsymbol{\mu_x} - \boldsymbol{\mu_y}\|_2^2 + \text{Tr}\left(\boldsymbol{\Sigma_x} + \boldsymbol{\Sigma_y} - 2\left(\boldsymbol{\Sigma_x}\boldsymbol{\Sigma_y}\right)^{\frac{1}{2}}\right) \tag{4.6}$$

where $\|\boldsymbol{\mu_x} - \boldsymbol{\mu_y}\|_2^2$ is the squared Euclidean distance between the mean vectors, $\text{Tr}(\cdot)$ denotes the trace of a matrix, summing its diagonal elements and $(\boldsymbol{\Sigma_x}\boldsymbol{\Sigma_y})^{\frac{1}{2}}$ represents the matrix square root of the product of the covariance matrices.

Lower FID values mean that the generated data is close to the real data in the feature space, which indicates that results are of high-quality. This metric is especially useful for evaluating image quality because it uses meaningful features extracted by a pretrained inception network. In our work, we compute FID to evaluate generative performance on image datasets, providing a clear and widely accepted way to assess generative models.

4.4 Experimental results and discussion

In this section, we present the experiments conducted along with the results and key insights derived from the findings. Our analysis evaluates the performance of diffusion models trained with various loss formulations from three primary perspectives: (*i*) Loss convergence over epochs, which reflects the efficiency and stability of the training process; (*ii*) Quality of generated samples, that shows the model's ability to generate realistic and high quality samples; and (*iii*) Samples generated from varying sampling steps, giving insights into how different loss formulations affect the reverse diffusion process over time. The loss over epochs curves reported in this analysis are averaged over three independent experimental runs.

4.4.1 Loss convergence over epochs

We begin by training the denoising model using the weighted loss formulations $\mathcal{L}$ for different datasets as shown in fig. 4.8 (left). As discussed in Section 3.4, the weighted loss formulations are inherently not equivalent and cannot be directly compared in their original forms. To have meaningful comparison, we rescale the weighted test loss according to the definitions provided in equations (3.62), (3.63), and (3.64). This rescaling transforms the losses into a common scale, giving the rescaled loss, $\tilde{\mathcal{L}}$, which is mathematically equivalent to $\mathcal{L}(\mathbf{x})$.

The rescaled loss are shown in fig. 4.8 (right). After rescaling, the loss curves for different formulations closely align, demonstrating the equivalence of these losses in their rescaled forms. Since $\mathcal{L}(\mathbf{x}) = \tilde{\mathcal{L}}(\mathbf{x})$, the $\mathbf{x}$-space losses remain unchanged between the weighted and rescaled plots. The rescaling of $\mathcal{L}(\boldsymbol{\epsilon})$, $\mathcal{L}(\mathbf{v})$ and $\mathcal{L}(\mathbf{s})$ to $\tilde{\mathcal{L}}(\boldsymbol{\epsilon})$, $\tilde{\mathcal{L}}(\mathbf{v})$ and $\tilde{\mathcal{L}}(\mathbf{s})$ respectively, aligns these curves close to $\tilde{\mathcal{L}}(\mathbf{x})$.

Next, we train the model using the NELBO loss formulations L for various target predictions. Based on their theoretical equivalence, as outlined in section 3.4, we expect that they behave similarly in the experiments too. To evaluate this, we run experiments using the NELBO loss formulation across different datasets, as shown in fig. 4.9. The loss curves for predictions in the $\mathbf{v}$ and $\boldsymbol{\epsilon}$ space are closely aligned and so is the loss

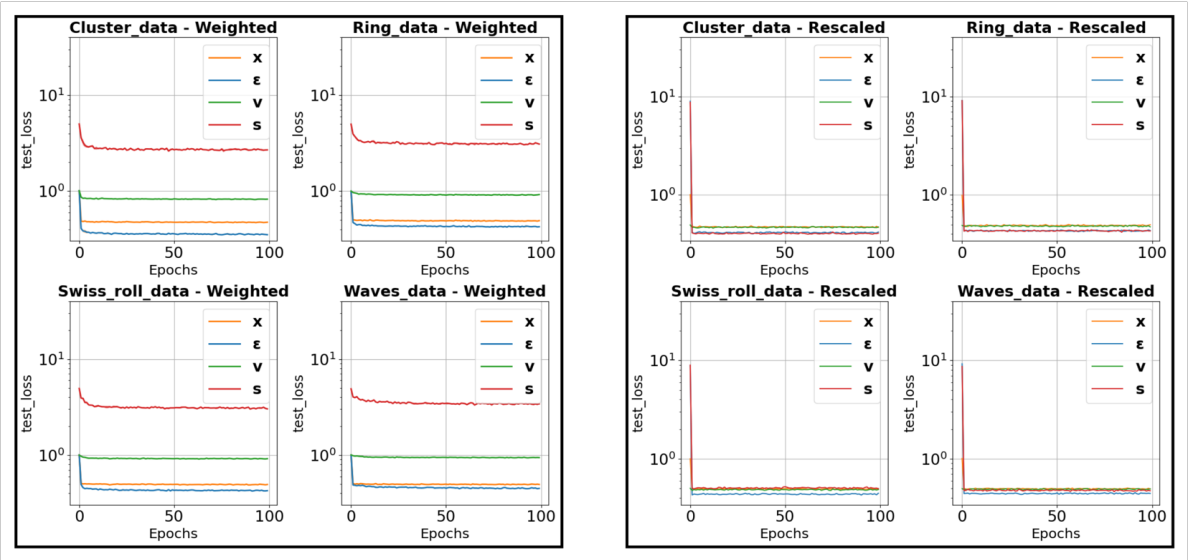


Figure 4.8: The weighted test loss $\mathcal{L}$ (left), is not directly comparable across different target predictions. However, the rescaled test loss $\tilde{\mathcal{L}}$ (right), is comparable and demonstrates the mathematical equivalence discussed in section 3.4.

curves for predictions in the $\mathbf{x}$ and $\mathbf{s}$ space. However, there is a significant gap between these two groups across all tested scenarios, which indicates differences in their training dynamics despite their theoretical equivalence.

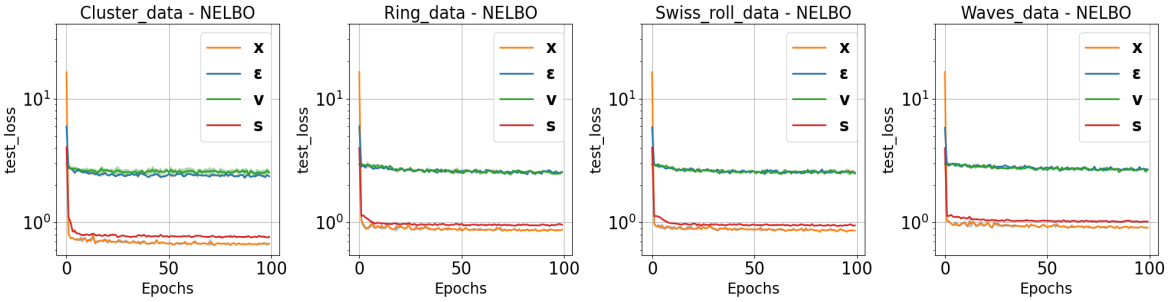


Figure 4.9: NELBO test loss over 100 epochs for different datasets. The plot illustrates the significant difference between the loss curves for predictions in the $\mathbf{v}$ and $\boldsymbol{\epsilon}$ space compared to those in the $\mathbf{x}$ and $\mathbf{s}$ space. The discrepancy between the two groups can be seen across all datasets.

We attribute these variations to differences in SNR scaling of the targets in the NELBO formulation and the variability of target objectives across different timesteps. The SNR scaling is inversely proportional to the weighting function, and is expressed as $\frac{1}{w(t)}$. These scaling factors determine the contribution of each timestep to the overall loss, thereby influencing the model's learning dynamics during training. As illustrated in fig. 4.10, the scaling factors for the $\boldsymbol{\epsilon}$ and $\mathbf{v}$ space are significantly larger at early timesteps, where the added noise is minimal. In the $\mathbf{x}$ space, the scaling is also high initially but

decreases more gradually over time. In contrast, the **s** space has higher scaling at later timesteps.

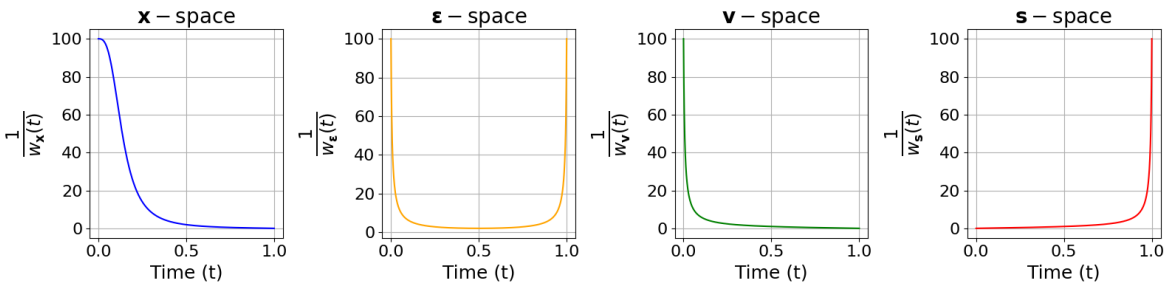


Figure 4.10: SNR scalings ($\frac{1}{w(t)}$) with respect to timesteps for different NELBO formulations. The plot compares the scaling behavior of $\mathbf{x}$, $\boldsymbol{\epsilon}$, $\mathbf{v}$ and $\mathbf{s}$ space showing their varying contributions to the loss at different stages of the diffusion process.

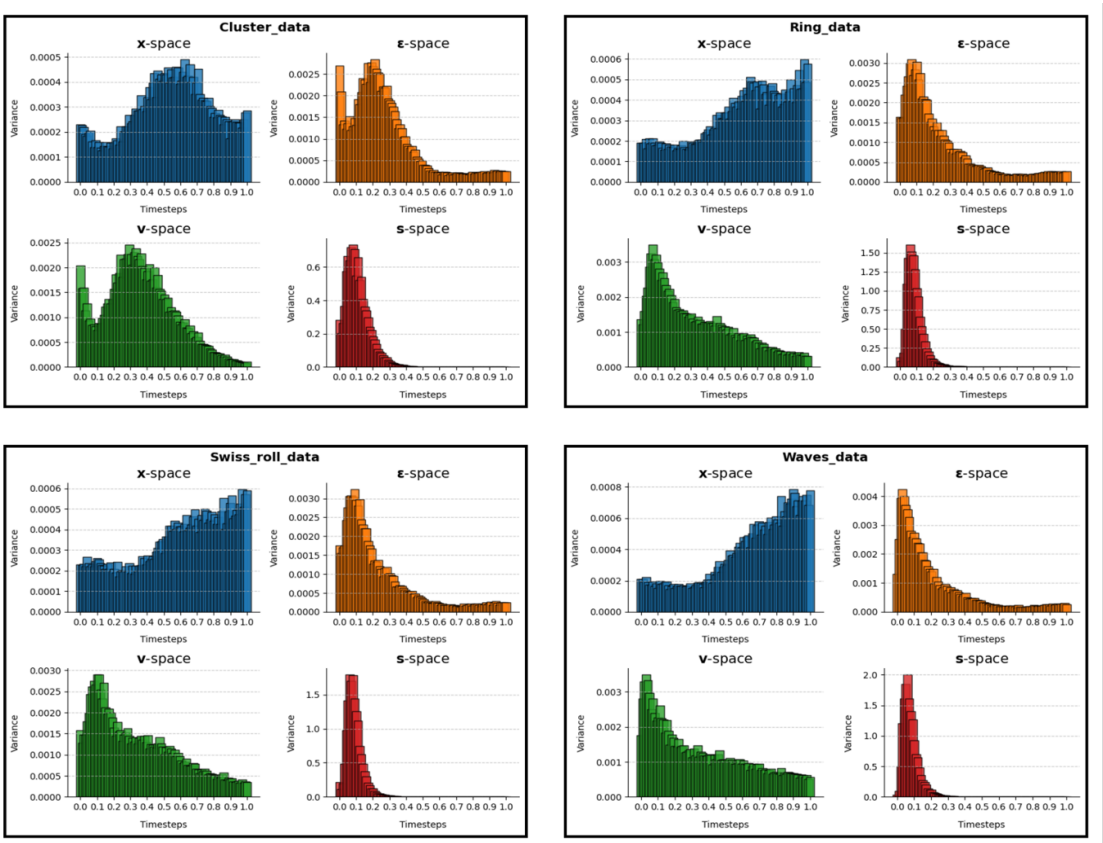


Figure 4.11: Variance of weighted loss across timesteps for different target objectives and datasets. The plots illustrate the variance of the weighted loss in the $\mathbf{x}$, $\boldsymbol{\epsilon}$, $\mathbf{v}$ and $\mathbf{s}$ space across time for all the datasets. The $\mathbf{x}$-space show low variance at the start while the $\boldsymbol{\epsilon}$, $\mathbf{v}$ and $\mathbf{s}$ space have higher variance at early timesteps.

Fig. 4.11 illustrates the variance in the objective when the model is trained with weighted loss (i.e., without scaling factors). In the $\mathbf{x}$-space, variance is lower at early timesteps. In contrast, the $\boldsymbol{\epsilon}$, $\mathbf{v}$, and $\mathbf{s}$ space show higher variance at early timesteps, with the $\mathbf{s}$-space having the highest variance. This pattern can be seen across all four datasets.

The NELBO loss is nothing but the SNR scalings applied over these weighted loss formulations. The scaling factor has limited impact in the $\mathbf{x}$-space due to its gradual decline over time and the low variance in loss at early timesteps. However, in the $\boldsymbol{\epsilon}$ and $\mathbf{v}$-space, high scaling factors at early timesteps, coupled with high loss variance, degrade model performance, resulting in poor likelihood outcomes. In the $\mathbf{s}$-space, lower scaling at early timesteps results in a better likelihood. These findings suggest that excessively large scaling factors at timesteps, where the variability of target objective is high, negatively impacts the model's overall likelihood performance.

Because of different scaling factors across target objectives the training dynamics under NELBO loss formulation are different and hence the performance in terms of likelihood also varies. The weighted loss formulations $\mathcal{L}$ on the other hand discards any such scalings and thus results in a more stable learning process compared to NELBO loss formulations.

4.4.2 Generated samples

The quality of generated samples shows a different pattern compared to the behavior observed in loss convergence. It suggests that better likelihood estimation does not necessarily mean good quality of generated samples. This phenomenon is also discussed by Theis et al. [60], where they emphasize on the limitations of likelihood based metrics in evaluating generative models. They show that optimizing for likelihood tends to prioritize capturing the entire data distribution, often at the expense of perceptual quality and sample diversity, making it an unreliable measure for evaluating the perceptual quality and diversity of generated samples.

To examine the differences in the quality of generated samples across various loss formulations, we use moment based metrics for comparison. Specifically, we calculate the mean distance (Euclidean distance between the means of real and generated datasets) and the covariance distance (Frobenius norm of the difference between their covariance matrices), as described in section 4.3.1. These metrics provide a quantitative assessment of the alignment between the real and generated data distributions. The results of this analysis are presented in table 4.1.

We observe that although the $\mathbf{x}$ and $\mathbf{s}$ space achieve lower NELBO values, the $\boldsymbol{\epsilon}$ and $\mathbf{v}$ space consistently produce higher-quality samples across all datasets. This is also evident from fig. 4.12, which shows 2000 samples generated over 512 sampling steps.

Table 4.1: Performance of NELBO and weighted loss formulations on 2D datasets. The table compares NELBO loss (L) and weighted loss $\mathcal{L}$ across cluster, ring, swiss roll, and waves datasets, reporting NELBO, loss values, mean distance (Euclidean), and covariance distance (Frobenius norm) to assess model convergence and sample quality.

Data	**Loss Form**	**NELBO Loss (L)**			**Weighted loss ($\mathcal{L}$)**		
		NELBO↓	**Mean dist.↓**	**Covar dist.↓**	**Loss↓**	**Mean dist.↓**	**Covar dist.↓**
Cluster data	$\mathbf{x}$	0.6777	0.1754	0.5746	0.4754	0.4300	0.2715
	$\boldsymbol{\epsilon}$	2.3636	0.0364	0.0706	0.3522	0.0634	0.1430
	$\mathbf{v}$	2.5396	0.0307	0.0409	0.8264	0.0389	0.0363
	$\mathbf{s}$	0.7657	0.2279	0.1498	2.6934	0.2688	0.1633
Ring data	$\mathbf{x}$	0.8785	0.2744	0.3807	0.4932	0.2807	0.3974
	$\boldsymbol{\epsilon}$	2.5700	0.0914	0.1107	0.4266	0.0983	0.0366
	$\mathbf{v}$	2.5452	0.0459	0.0718	0.9227	0.0453	0.0088
	$\mathbf{s}$	0.9577	0.2254	0.1563	3.0981	0.2220	0.1637
Swiss data	$\mathbf{x}$	0.8640	0.1133	0.2645	0.4934	0.5256	0.6875
	$\boldsymbol{\epsilon}$	2.5324	0.0689	0.0941	0.4261	0.0857	0.0824
	$\mathbf{v}$	2.4861	0.0418	0.0598	0.9171	0.0427	0.0269
	$\mathbf{s}$	0.9493	0.1266	0.1972	3.0274	0.0893	0.1227
Waves data	$\mathbf{x}$	0.9104	0.1593	0.5559	0.4939	0.1869	0.6911
	$\boldsymbol{\epsilon}$	2.6805	0.0405	0.0757	0.4500	0.0748	0.0778
	$\mathbf{v}$	2.6873	0.0447	0.0738	0.9411	0.0131	0.0271
	$\mathbf{s}$	1.0210	0.0353	0.1369	3.4165	0.0178	0.1676

The samples from the $\boldsymbol{\epsilon}$ and $\mathbf{v}$ space align more closely with the true data manifold compared to those from the $\mathbf{x}$ and $\mathbf{s}$ space, regardless of whether the weighted or NELBO loss is used. Additionally, the sample quality between the weighted and NELBO loss formulations is comparable in most cases. These results indicate that while the scaling of NELBO affects model convergence, its influence on sample quality is minimal.

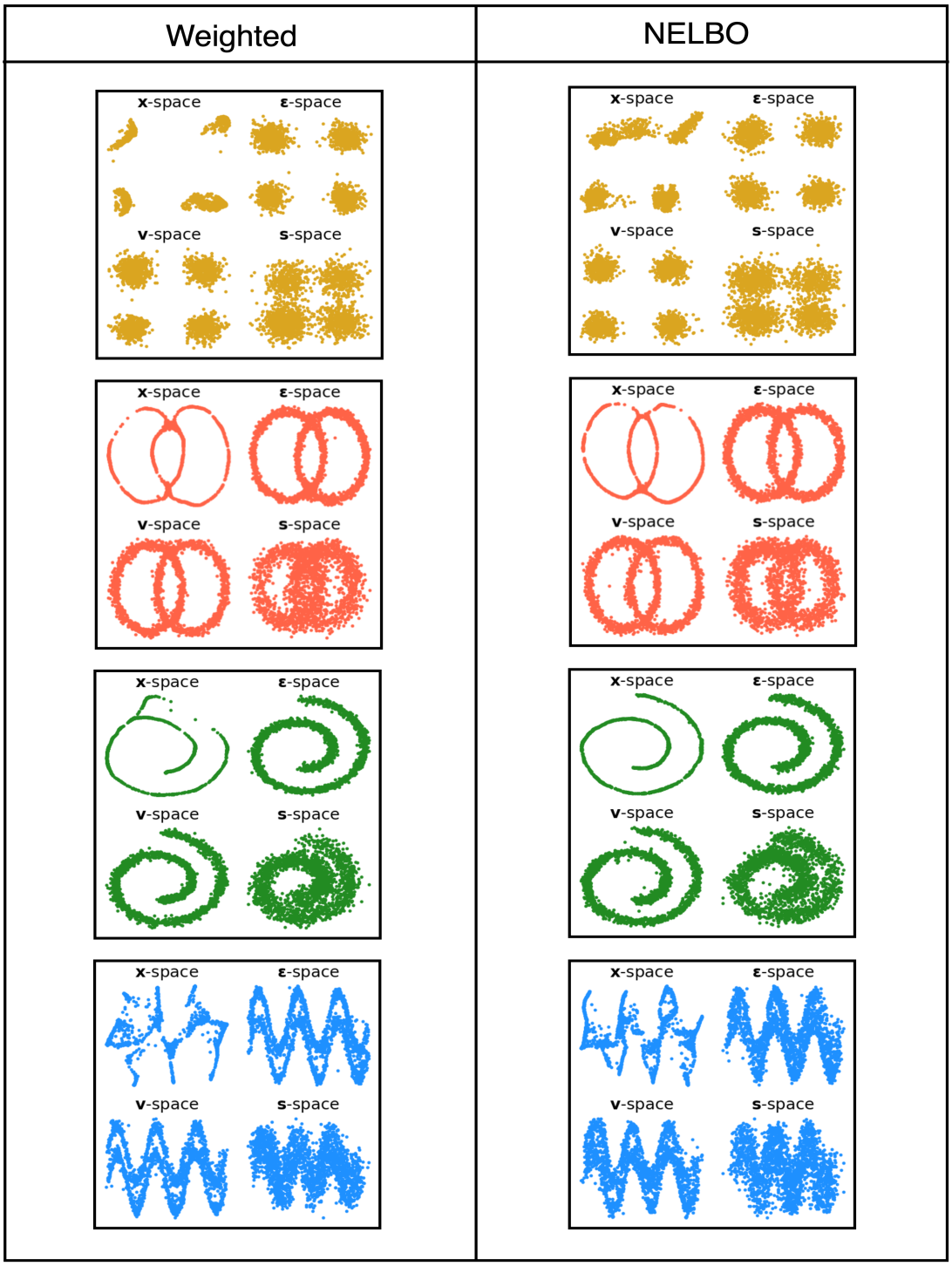


Figure 4.12: 2K generated samples after 512 sampling steps from model trained using weighted loss formulation (left) and NELBO formulation (right) for cluster data, ring data, swiss roll data and waves data.

4.4.3 Samples generated from varying sampling steps

Fig. 4.13 shows the generation of samples of ring data using different numbers of sampling steps in the reverse process. The results are similar for other datasets, and is detailed in the appendix. For both weighted and NELBO loss formulations, the following variations in sample quality can be seen as the number of sampling steps changes.

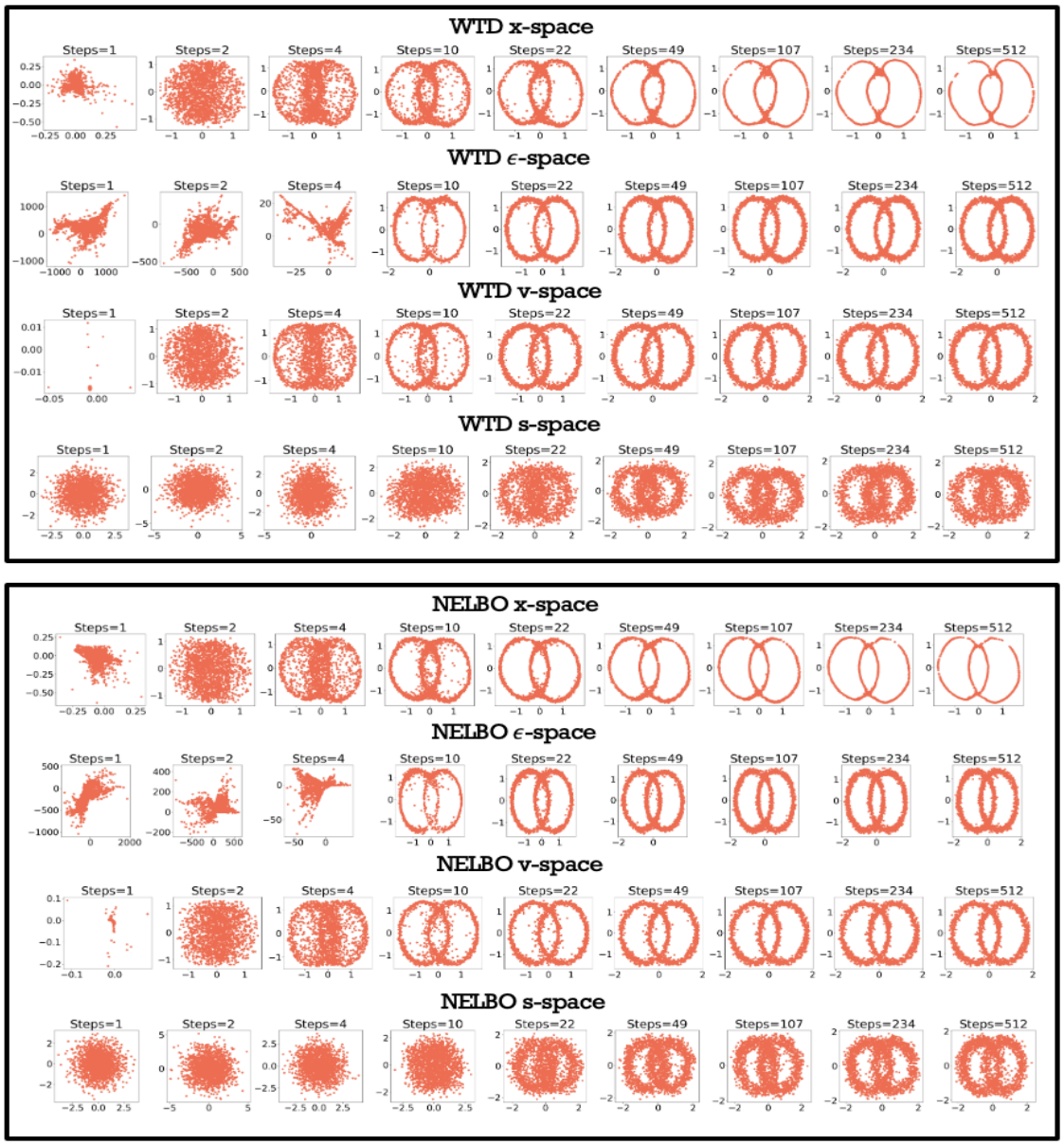


Figure 4.13: Generated samples for ring data for different number of sampling steps from model trained on weighted loss (top) and NELBO loss (bottom). The figure highlights the differences in sample quality across the target objectives when using different numbers of sampling steps.

For the **x**-space, sample quality declines with more sampling steps but outperforms other objectives with fewer steps by effectively capturing data structure and scale. In contrast, the $\boldsymbol{\epsilon}$-space produces poorer samples with fewer steps, and the sample quality gradually increases. The **v**-space, captures the data structure well even with fewer sampling steps and the sample quality continues to improve with more steps. The quality of samples generated in the **s**-space is not good, however, it improves with the number of steps.

While the variation in the quality of samples generated with different sampling steps under various target objectives is not fully established, a plausible explanation may lie in how the loss function behaves for different target predictions over timesteps. To understand this better, we visualize the weighted train loss across timesteps for all target predictions on the ring dataset, as shown in fig 4.14. Similar trends are observed for other datasets, with corresponding graphs provided in the appendix.

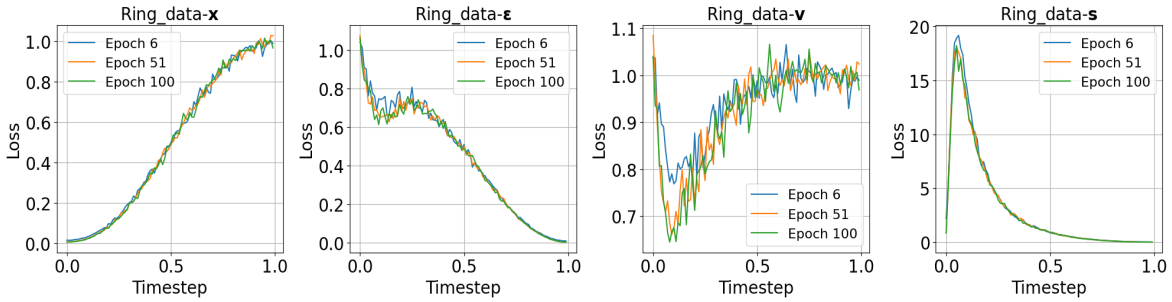


Figure 4.14: This figure illustrates the behavior of weighted train loss $\mathcal{L}$ with respect to timesteps for ring data at 3 different epochs. In **x**-space the loss increases while for $\boldsymbol{\epsilon}$-space it decreases with timesteps. The **v**-space tries to balance the loss and in the **s**-space, the loss is more at the start due to the sensitivity of score matching to noise variance.

In the **x**-space, the model predicts the original data point at each timestep during the reverse diffusion. As noise increases in the forward process (fig. 4.2), the SNR drops significantly, making prediction harder and resulting in higher losses at later timesteps. In contrast, for $\boldsymbol{\epsilon}$-space, the task is to predict the noise that was added to the data at each timestep. As more noise is introduced, predicting it becomes progressively easier. The **v**-space formulation as shown in section 3.3.1 interpolates between data **x** and noise $\boldsymbol{\epsilon}$, weighted by time dependent functions, requiring the model to find a balance between the two. In **s**-space the loss is significantly higher in the starting timesteps due to the sensitivity of score matching to noise variance (σ_t), evident from equation (3.53). At early timesteps, σ_t^2 becomes negligible and the score function becomes very large in magnitude as $\mathbf{s} \propto \frac{1}{\sigma^2}$, leading to a significant rise in the loss.

4.4.4 Results on image dataset

The outcomes of various loss formulations applied to the image dataset are summarized in table 4.2. We excluded the results for score-based metrics, as their precise computation for continuous time diffusion models in high dimensional image spaces requires modeling reverse and forward Stochastic Differential Equations (SDEs), which is beyond the scope of this study and left for future research.

Table 4.2: Performance comparison of NELBO and weighted loss formulations for CIFAR10 dataset, evaluating likelihood estimation (NELBO) and sample quality (FID) across different target objectives $\mathbf{x}$, $\boldsymbol{\epsilon}$ and $\mathbf{v}$.

Data	**Loss Form**	**NELBO Loss (L)**		**Weighted Loss ($\mathcal{L}$)**	
		NELBO↓	**FID↓**	**Loss↓**	**FID↓**
CIFAR10	$\mathbf{x}$	0.09	13.54	0.05	16.96
	$\boldsymbol{\epsilon}$	0.84	24.26	0.06	13.75
	$\mathbf{v}$	0.86	18.18	0.12	18.24

To evaluate the models, we used the NELBO to measure how well the model approximates the data likelihood, and the FID score (section 4.3) to assess the quality of 50,000 samples generated by each model, which were trained with different objectives. The sample generation process involved 500 reverse diffusion steps.

First we evaluate the models based on probability density estimation. Our findings show that the equivalent NELBO formulations performed differently for high-dimensional image dataset as well. Specifically, the NELBO in $\mathbf{x}$-space achieved the best performance in terms of probability density estimation. In $\boldsymbol{\epsilon}$ and $\mathbf{v}$-space NELBO loss did not performed so well and there is a significant difference as compared to $\mathbf{x}$-space. The NELBO in $\boldsymbol{\epsilon}$ and $\mathbf{v}$-space is nearly identical which is consistent with the results observed in 2D datasets as discussed in section 4.4.1.

We now evaluate the performance based on the quality of generated samples, as measured by the FID score. The NELBO formulation in $\mathbf{x}$-space and the weighted loss in $\boldsymbol{\epsilon}$-space gave the best results, which is different as compared to the 2D datasets, where NELBO in $\mathbf{x}$-space showed relatively weaker performance. Notably, while the weighted $\boldsymbol{\epsilon}$ loss achieved a good FID score, the NELBO $\boldsymbol{\epsilon}$ loss performed the worst among all configurations. This means that, despite NELBO in $\boldsymbol{\epsilon}$-space minimizes the negative log likelihood, the perceptual quality of samples generated using the weighted $\boldsymbol{\epsilon}$ loss is better and again indicates that a more accurate likelihood estimation does not necessarily correspond to better sample quality.

In **v**-space, both the NELBO and weighted loss formulations gave nearly identical FID results. While it was expected that weighted loss in **v**-space will generate better samples, it was not seen in the experiments. The reason for this could be the simplistic weighting $w(t)$ which we used here, whereas in the original study [49], the researchers utilized a more complex weighting strategy, which likely better optimized sample quality.

We show the samples generated using loss formulations in $\mathbf{x}$, $\boldsymbol{\epsilon}$, and $\mathbf{v}$ spaces in fig. 4.15, 4.16, and 4.17, respectively. By comparing the generated samples, we observe noticeable differences in quality for different loss formulations. The samples generated using the weighted $\boldsymbol{\epsilon}$ loss appear closer to the original data. These images are slightly brighter, making objects like cars, birds, and airplanes more distinct and easier to identify.

On the other hand, the images generated with the NELBO loss in $\mathbf{x}$ space are also good but tend to look a little blurry compared to weighted $\boldsymbol{\epsilon}$. This makes it harder to recognize different objects clearly. Finally, the images produced using the loss formulation in **v**-space show a lower quality. Some images appear hazy, with unclear details, which affects their overall appearance and makes object identification more difficult.

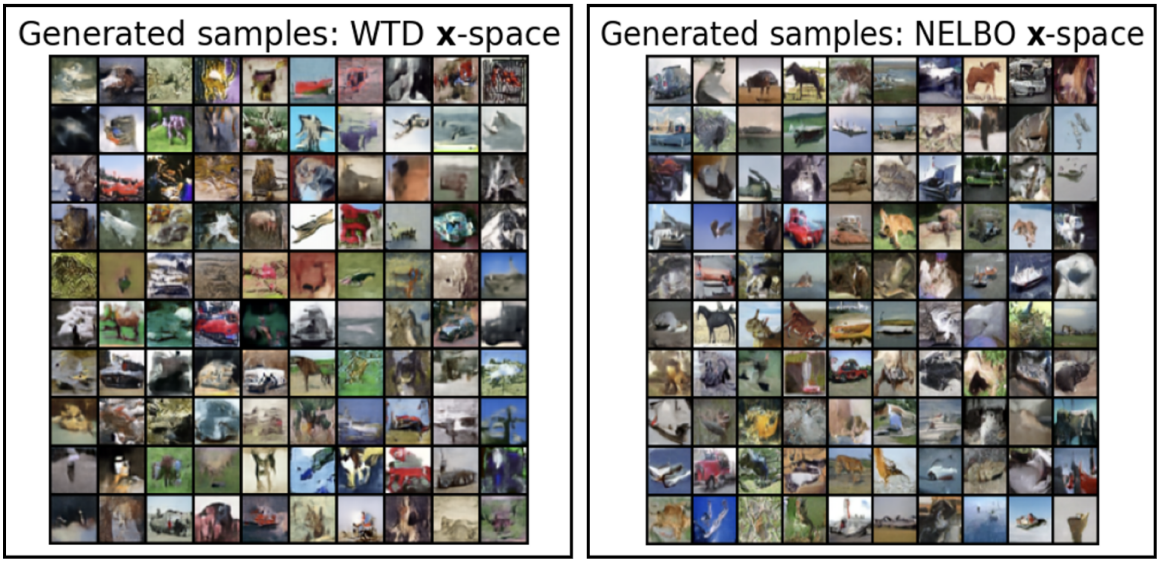


Figure 4.15: Images generated using different loss formulations in $\mathbf{x}$-space. Left: Weighted $\mathbf{x}$ loss (FID: 16.96), Right: NELBO $\mathbf{x}$ loss (FID: 13.54). Although the NELBO $\mathbf{x}$ loss gave a good FID score compared to other objectives, the images generated are a little blurry. In some of the images we can identify the objects, however for most of them its very difficult.

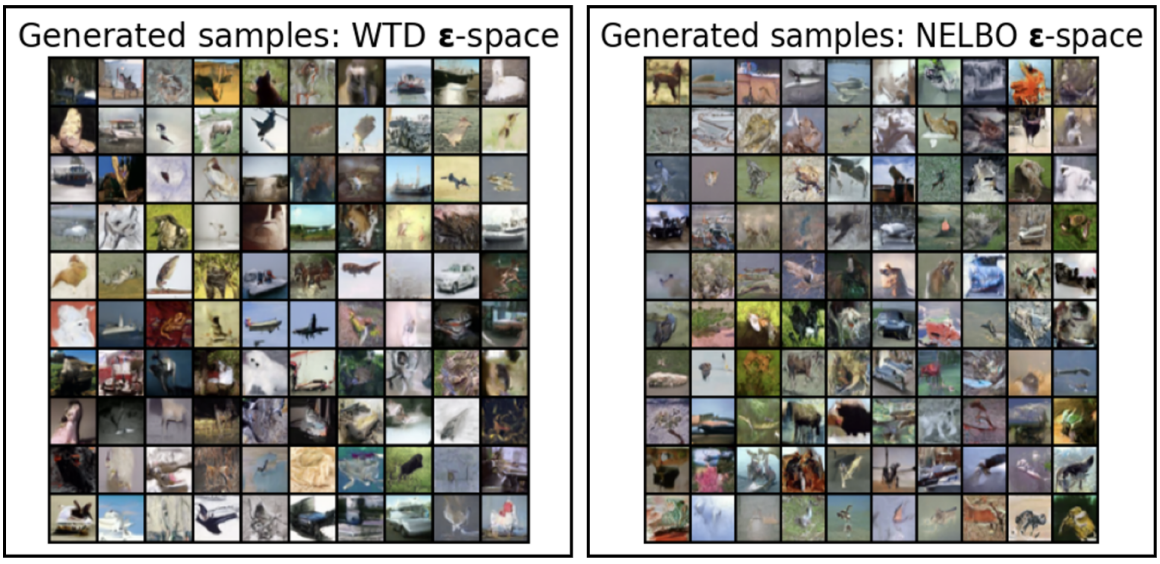


Figure 4.16: Images generated using different loss formulations in $\boldsymbol{\epsilon}$-space. Left: Weighted $\boldsymbol{\epsilon}$ loss (FID: 13.75), Right: NELBO $\boldsymbol{\epsilon}$ loss (FID: 24.26). The images generated using the weighted $\boldsymbol{\epsilon}$ loss show the best visual quality compared to other loss formulations. In contrast, the samples produced with the NELBO $\boldsymbol{\epsilon}$ loss are of lower quality and are easily distinguishable from those generated by the weighted $\boldsymbol{\epsilon}$ loss.

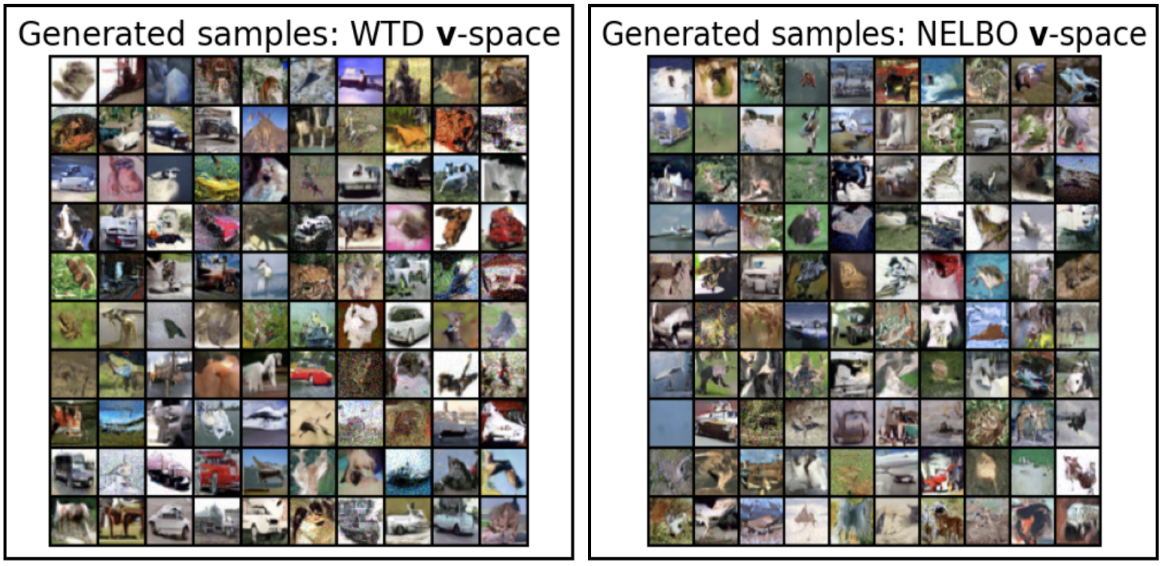


Figure 4.17: Images generated using different loss formulations in $\mathbf{v}$-space. Left: Weighted $\mathbf{v}$ loss (FID: 18.24), Right: NELBO $\mathbf{v}$ loss (FID: 18.18). The images generated using both the NELBO and weighted formulations in $\mathbf{v}$-space show poor visual quality, with many appearing hazy and making object recognition very difficult.

5 Conclusions

We conclude this thesis by summarizing the key findings and insights gained from the research. Additionally, we explore potential directions for future work. On a personal note, I finally reflect on the learnings and takeaways I gained as an individual through this journey.

5.1 Summary and key insights

Our study began by exploring why diffusion models are among the most effective generative models and how their structure naturally fits within the framework of generative modeling. We studied the foundational aspects of diffusion models, including the forward and reverse processes, and discussed key concepts such as the SNR and the NELBO. These concepts helped us understand how diffusion objectives can be framed mathematically and optimized during training. We also delved into different loss formulations for predicting key targets such as the original data $\mathbf{x}$, the noise $\boldsymbol{\epsilon}$, and the rate of change in the data distribution $\mathbf{v}$, which integrates both data and noise.

To extend this understanding, we examined diffusion models from a score-based generative modeling perspective. This approach focuses on predicting the score function $\mathbf{s}$. By avoiding the need to compute complex normalizing constants, score-based modeling simplifies generative tasks while maintaining good performance. We explored practical techniques such as sliced score matching and denoising score matching which make this approach tractable.

We systematically analyzed the four main target space i.e $\mathbf{x}$, $\boldsymbol{\epsilon}$, $\mathbf{v}$, and $\mathbf{s}$. For each one of them, we derived a corresponding NELBO and weighted loss formulation and demonstrated their mathematical equivalence. This allowed us to create a unified mathematical framework that connects these loss functions, providing new insights into their similarities and differences.

While the theoretical equivalence of these loss formulations is significant, practical training introduces complexities that may lead to performance differences. To anazlyze this, we conducted a series of experiments designed to evaluate whether these theoretical rela-

tionships translates into similar empirical performance. Moreover we evaluated different objectives based on the quality of generated samples. Our findings are summarized below:

- **Divergence in NELBO loss behavior:** The NELBO loss L showed different behavior depending on the target objective. This difference is due to the distinct scaling factors applied to targets, leading to variations in their empirical performance.

- **Stability of weighted loss formulation:** Unlike NELBO loss, equivalence of the rescaled loss $\tilde{\mathcal{L}}$ is confirmed through empirical results. Rescaled loss are essentially the scaled version of the weighted loss $\mathcal{L}$, used to make them equivalent to each other. Hence, the weighted loss formulation proved to be more stable compared to the NELBO loss. This stability also points the advantages of weighted formulations in training diffusion models.

- **Influence of sampling steps on sample quality:** The quality of generated samples varied across target objectives when we generated samples using different sampling steps. This variation is likely due to the differing behaviors of the loss functions over timesteps and the specific influence of target predictions. This observation point out the importance of considering sampling strategies during training.

- **Likelihood estimation vs. sample quality:** We also observe that a better likelihood estimation does not necessarily means good quality of generated samples which aligns with the previous studies. This shows a trade-off between optimizing for likelihood estimation and generating good quality samples from the model.

- **Optimal objectives for specific tasks:** For likelihood estimation, the NELBO loss in $\mathbf{x}$-space gave the best results. For quality of generated samples, NELBO $\mathbf{x}$ loss and weighted $\boldsymbol{\epsilon}$ loss performed better than other objectives in case of image generation. Notably, the samples generated using the weighted $\boldsymbol{\epsilon}$ were visually more realistic.

In conclusion, this study offers a theoretical foundations for diffusion models, a unified mathematical framework that clarifies the construction of loss functions, and experimental insights on the comparison of their empirical performance. It offers a comprehensive resource for researchers working with diffusion models or those seeking to delve deeper into the field of generative modeling. By exploring the working dynamics, recent advancements, and different interpretations of these models, our work stands as a valuable reference for those who want to understand or extend the capabilities of generative modeling.

5.2 Future works

While this study provides significant insights into the theoretical and empirical aspects of diffusion models, there are several directions for future research to build upon these findings. Below are some potential directions for extending and enhancing the work presented:

- **Exploration of complex weighted loss functions** This study primarily focused on simple weighted loss functions (ℓ_2). Future work could explore the impact of more complex weightings $w(t)$ in order to understand how they influence the training dynamics and model performance.

- **Evaluation with alternative noise schedules** The experiments conducted in this study utilized a cosine noise schedule for the forward process. Further research could examine the effects of other variants of noise schedules on the reverse diffusion process.

- **Detailed experiments in high-dimensional image data** Although this work was primarily focused on 2D synthetic datasets, the insights gained can serve as a foundation for extending multiple experiments on high-dimensional image datasets.

- **Score matching in high-dimensional spaces** Score matching was not explored for high-dimensional image datasets in this study as it needed to model reverse and forward SDE. Research can be done on comparative studies of score-matching techniques for continuous-time diffusion models in these settings.

5.3 Personal takeaways

As I conclude this thesis, I take a moment to reflect on my personal journey through this research by highlighting the lessons learned, the challenges faced, and the growth I have experienced along the way.

This journey began with an exploration of various generative modeling techniques. While I had some prior knowledge in this field, going deeper into the underlying theoretical frameworks broadened my understanding and my skills. This thesis required me to focus on the finer mathematical details, which proved essential in developing the concepts. The process of deriving the formulations, analyzing them, and connecting them to practical outcomes was challenging and exciting at the same time. It pushed me to expand my thinking and sharpen my problem-solving skills. During this period, I read several research papers, trying to understand different approaches and methodologies.

This process not only increased my knowledge but also helped me improve my scientific writing skills, and helped me to effectively communicate my ideas in a clear and structured manner.

While this thesis marks the end of this particular project, it also marks the beginning of a new path in my academic journey. The insights and skills gained here will help in my research in generative modeling, and I look forward to continuing to explore and contribute to this field in the future.

Bibliography

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. "A learning algorithm for boltzmann machines". In: *Cognitive Science* (1985).

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale GAN training for high fidelity natural image synthesis". In: *arXiv:1809.11096* (2018).

[4] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J Weiss, Mohammad Norouzi, Najim Dehak, and William Chan. "Wavegrad 2: Iterative refinement for text-to-speech synthesis". In: *arXiv:2106.09660* (2021).

[5] Ricky T. Q. Chen, Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. "Residual Flows for Invertible Generative Modeling". In: (2020).

[6] Jooyoung Choi, Jungbeom Lee, Chaehun Shin, Sungwon Kim, Hyunwoo Kim, and Sungroh Yoon. "Perception prioritized training of diffusion models". In: *Proceedings of the IEEE/CVF Conference on CVPR*. 2022.

[7] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society: series B (methodological)* (1977).

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *North American chapter of the association for computational linguistics*. 2019.

[9] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis". In: *Advances in neural information processing systems* (2021).

[10] Laurent Dinh, David Krueger, and Yoshua Bengio. "Nice: Non-linear independent components estimation". In: *arXiv:1410.8516* (2014).

[11] Bradley Efron. "Tweedie's formula and selection bias". In: *Journal of the American Statistical Association* (2011).

[12] ExplainingAI. *Denoising Diffusion Probabilistic Models Code — DDPM Pytorch Implementation*. YouTube video. 2023.

[13] James Gleick. *Genius: The Life and Science of Richard Feynman*. Pantheon Books, 1992.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014.

[15] Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. "Efficient diffusion training via min-snr weighting strategy". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2023.

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018.

[17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". In: *Advances in neural information processing systems* (2017).

[18] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* (2006).

[19] Geoffrey E. Hinton. "Training products of experts by minimizing contrastive divergence". In: (2002).

[20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* (2020).

[21] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. "Video diffusion models". In: *Advances in Neural Information Processing Systems* (2022).

[22] Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. "Equivariant Diffusion for Molecule Generation in 3D". In: *arXiv:2203.17003* (2022).

[23] Aapo Hyvärinen and Peter Dayan. "Estimation of non-normalized statistical models by score matching." In: *Journal of Machine Learning Research* (2005).

[24] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. "Elucidating the design space of diffusion-based generative models". In: *Advances in neural information processing systems* (2022).

[25] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

[26] Diederik Kingma and Ruiqi Gao. "Understanding diffusion objectives as the elbo with simple data augmentation". In: *Advances in Neural Information Processing Systems* (2024).

[27] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. "Variational diffusion models". In: *Advances in neural information processing systems* (2021).

[28] Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: *Foundations and Trends® in Machine Learning* (2019).

[29] Diederik P Kingma, Max Welling, et al. *Auto-encoding variational bayes*. 2013.

[30] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv:1910.13461* (2019).

[31] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. *Are GANs Created Equal? A Large-Scale Study*. 2018.

[32] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. "Repaint: Inpainting using denoising diffusion probabilistic models". In: *Proceedings of the IEEE/CVF*. 2022.

[33] Calvin Luo. "Understanding diffusion models: A unified perspective". In: *arXiv:2208.11970* (2022).

[34] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv:1411.1784* (2014).

[35] MIT-HAN-LAB. *DiffAugment for BigGAN (CIFAR)*. Github. 2020.

[36] MITCBMM. *Diffusion and Score-Based Generative Models*. YouTube video. 2023.

[37] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. "Glide: Towards photorealistic image generation and editing with text-guided diffusion models". In: *arXiv:2112.10741* (2021).

[38] Alexander Quinn Nichol and Prafulla Dhariwal. "Improved denoising diffusion probabilistic models". In: *International conference on machine learning*. 2021.

[39] Mang Ning, Enver Sangineto, Angelo Porrello, Simone Calderara, and Rita Cucchiara. "Input Perturbation Reduces Exposure Bias in Diffusion Models". In: (2023).

[40] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. "Conditional Image Generation with PixelCNN Decoders". In: (2016).

[41] Abhilash Pal, Saurav Saha, and R Anita. "Musenet: Music generation using abstractive and generative methods". In: *International Journal of Innovative Technology and Exploring Engineering* (2020).

[42] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. "Dreamfusion: Text-to-3d using 2d diffusion". In: *arXiv:2209.14988* (2022).

[43] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* (1989).

[44] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv:1511.06434* (2015).

[45] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. "Improving language understanding by generative pre-training". In: (2018).

[46] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. "High-resolution image synthesis with latent diffusion models". In: *Proceedings of the IEEE/CVF*. 2022.

[47] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *Medical image computing and computer-assisted intervention–MICCAI 2015*. Springer. 2015.

[48] Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. "Palette: Image-to-image diffusion models". In: *ACM SIGGRAPH 2022 conference proceedings*. 2022.

[49] Tim Salimans and Jonathan Ho. "Progressive distillation for fast sampling of diffusion models". In: *arXiv:2202.00512* (2022).

[50] Jiwan Seo and Joonhyuk Kang. *Rate-Adaptive Quantization: A Multi-Rate Codebook Adaptation for Vector Quantization-based Generative Models*. 2024.

[51] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. "How good is my GAN?" In: (2018).

[52] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International conference on machine learning*. 2015.

[53] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. "Ladder variational autoencoders". In: *Advances in neural information processing systems* (2016).

[54] Jiaming Song, Chenlin Meng, and Stefano Ermon. "Denoising diffusion implicit models". In: *arXiv:2010.02502* (2020).

[55] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. "Maximum likelihood training of score-based diffusion models". In: *Advances in neural information processing systems* (2021).

[56] Yang Song and Stefano Ermon. "Generative modeling by estimating gradients of the data distribution". In: *Advances in neural information processing systems* (2019).

[57] Yang Song and Stefano Ermon. "Improved techniques for training score-based generative models". In: *Advances in neural information processing systems* (2020).

[58] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. "Score-based generative modeling through stochastic differential equations". In: *arXiv:2011.13456* (2020).

[59] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[60] Lucas Theis, Aäron van den Oord, and Matthias Bethge. "A note on the evaluation of generative models". In: *arXiv:1511.01844* (2015).

[61] Arash Vahdat and Jan Kautz. *NVAE: A Deep Hierarchical Variational Autoencoder*. 2021.

[62] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu, et al. "Wavenet: A generative model for raw audio". In: *arXiv:1609.03499* (2016).

[63] Aaron Van Den Oord, Oriol Vinyals, et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* (2017).

[64] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *International conference on machine learning*. 2016.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* (2017).

[66] Pascal Vincent. "A connection between score matching and denoising autoencoders". In: *Neural computation* (2011).

[67] Jay Whang, Mauricio Delbracio, Hossein Talebi, Chitwan Saharia, Alexandros G Dimakis, and Peyman Milanfar. "Deblurring via stochastic refinement". In: *Proceedings of the IEEE/CVF*. 2022.

[68] Botao Yu, Peiling Lu, Rui Wang, Wei Hu, Xu Tan, Wei Ye, Shikun Zhang, Tao Qin, and Tie-Yan Liu. "Museformer: Transformer with fine-and coarse-grained attention for music generation". In: *Advances in neural information processing systems* (2022).

List of Figures

Appendix

Weighted loss vs timestep

Below we show how the weighted train loss behave with respect to time steps for cluster data, swiss roll data and waves data

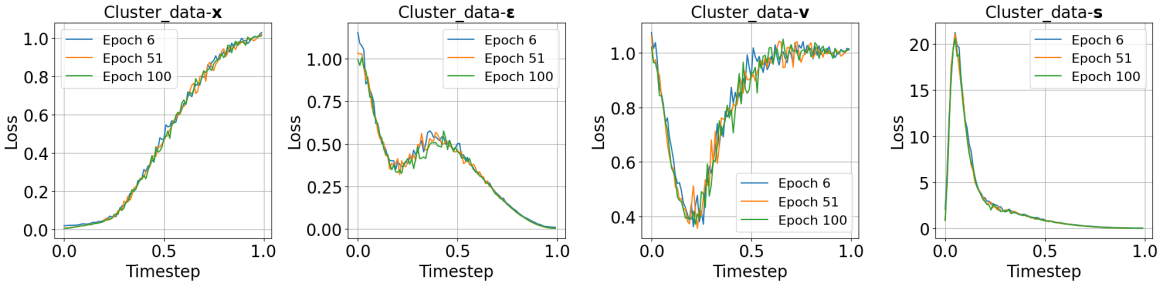


Figure 1: Weighted loss vs timestep for cluster data

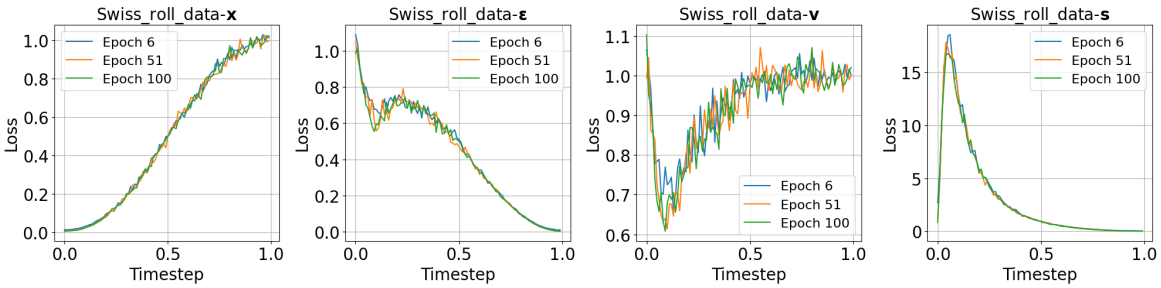


Figure 2: Weighted loss vs timestep for swiss roll data

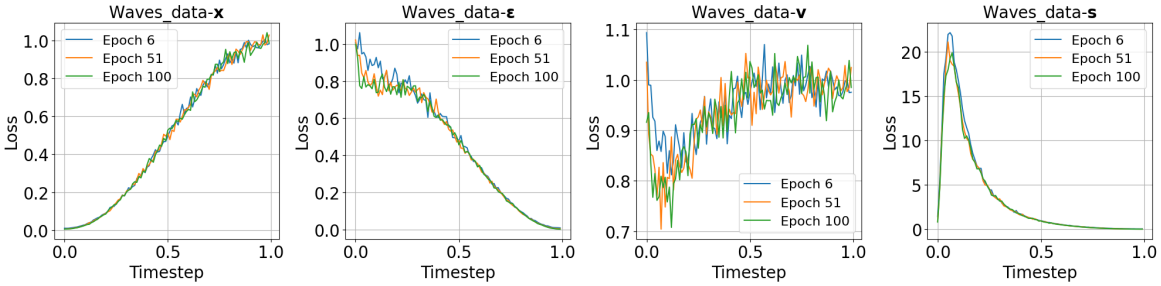


Figure 3: Weighted loss vs timestep for waves data

Generated samples from varying sample steps

Below are the generated samples from different number of sampling steps for cluster data, swiss roll data and waves data in the NELBO and weighted loss formulations.

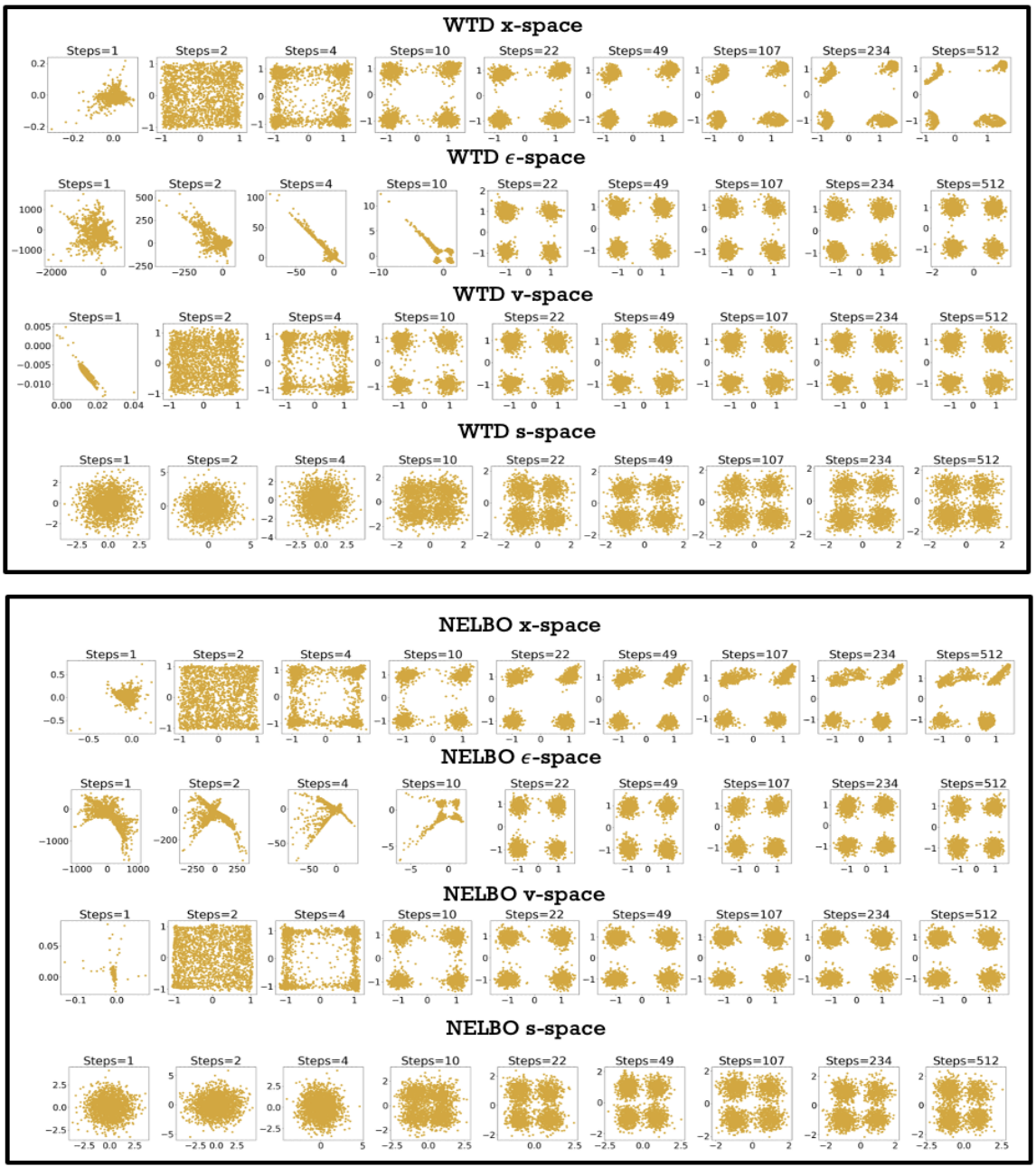


Figure 4: Generated samples for cluster data for different number of sampling steps from model trained on weighted loss (top) and NELBO loss (bottom).

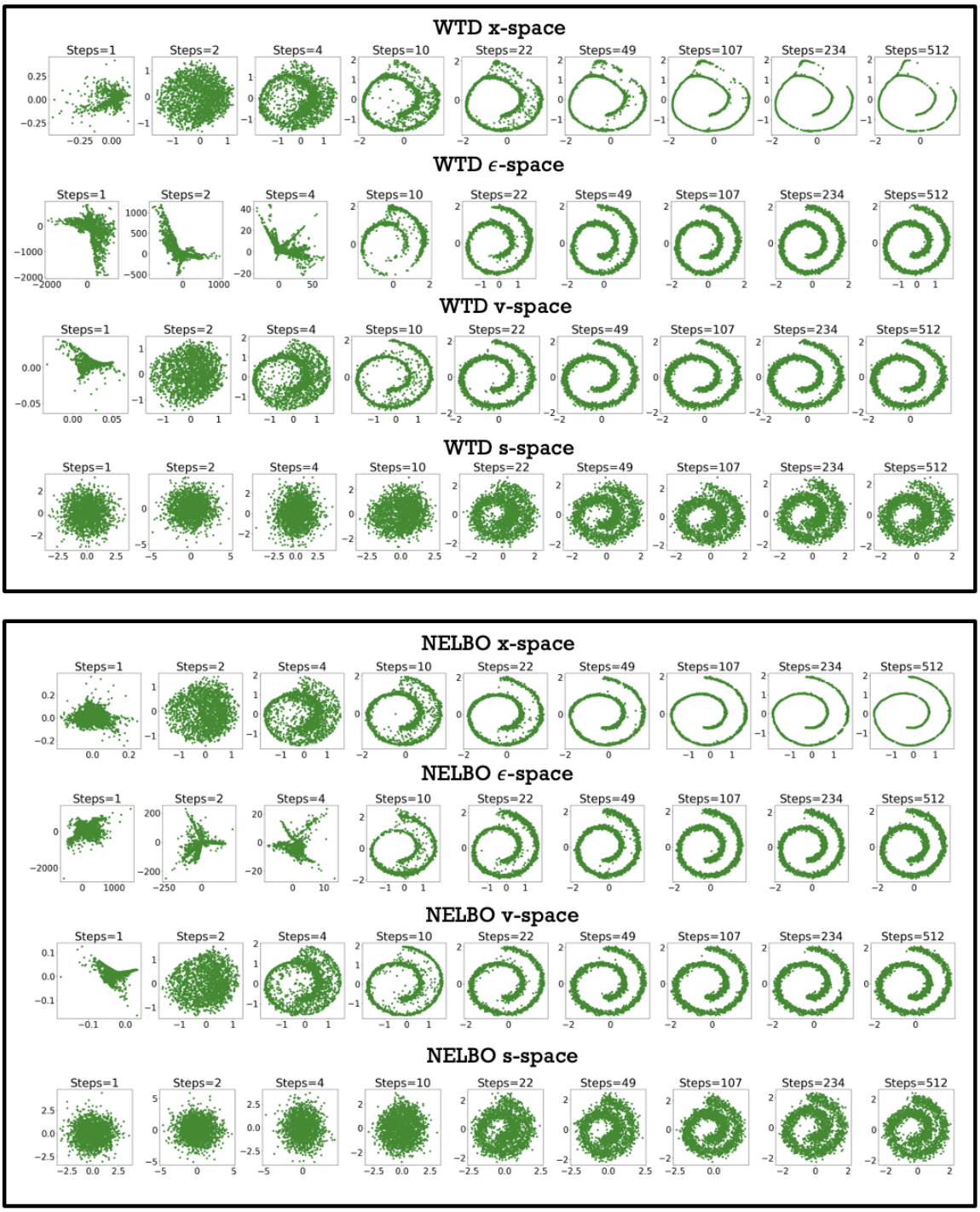


Figure 5: Generated samples for swiss data for different number of sampling steps from model trained on weighted loss (top) and NELBO loss (bottom).

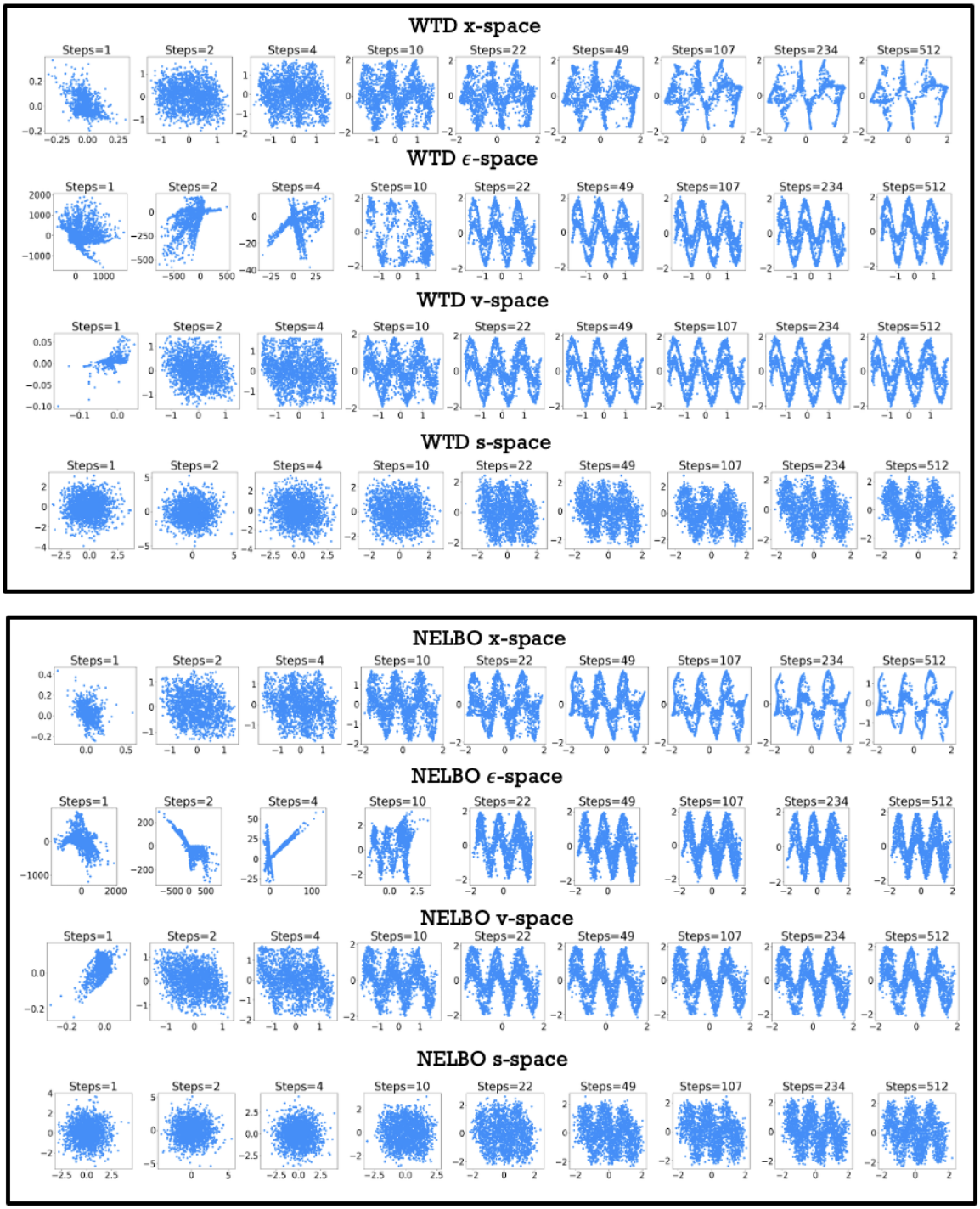
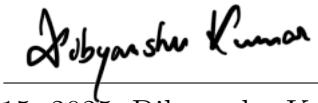


Figure 6: Generated samples for waves data for different number of sampling steps from model trained on weighted loss (top) and NELBO loss (bottom).

Declaration on oath

I hereby certify that I have written my master thesis independently and have not yet submitted it for examination purposes elsewhere. All sources and aids used are listed, literal and meaningful quotations have been marked as such.

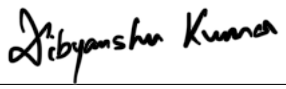


May 15, 2025, Dibyanshu Kumar

Consent to plagiarism check

I hereby agree that my submitted work may be sent to PlagAware (https://my.plagaware.com/) in digital form for the purpose of checking for plagiarism and that it may be temporarily (max. 5 years) stored in the database maintained by PlagScan as well as personal data which are part of this work may be stored there.

Consent is voluntary. Without this consent, the plagiarism check cannot be prevented by removing all personal data and protecting the copyright requirements. Consent to the storage and use of personal data may be revoked at any time by notifying the faculty.



May 15, 2025, Dibyanshu Kumar