Bocconi

Bachelor of Science in Economics, Management and Computer Science

Generative Modeling in AI and Stochastic Processes

Advisor:
Prof. OMIROS PAPASPILIOPOULOS

Bachelor of Science thesis by:
FRANCESCO VACCA
student ID no 3140929

Academic Year 2023-2024

Bocconi
Università Commerciale
Luigi Bocconi

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to all the people who have supported me throughout this degree journey and during the completion of this thesis.

First of all, I would like to express my deepest gratitude to my supervisor, Om, for his invaluable guidance in both the writing of this thesis and my future endeavors. His encouragement and the strong relationship we have built have been immensely meaningful. I would also like to extend my thanks to all the BEMACS professors who have taught and inspired me throughout my studies.

Un ringraziamento speciale va soprattutto ai miei nonni, a mia madre, a mio padre, a mia sorella Giulia, e a tutta la mia famiglia. Senza il vostro sostegno e amore incondizionato non sarei qui. Grazie per aver sempre creduto in me e per avermi sostenuto nei miei obiettivi e sogni. Grazie per avermi dato accesso a tutte le opportunità che ho avuto e che avrò; prometto di sfruttarle al meglio.

Un Grazie di cuore a Camilla, la cui presenza e il cui amore è stata la mia ancora in questi anni a Milano.

Una menzione poi per tutti i compagni di Università che mi hanno supportato e motivato lungo questo percorso. Un ringraziamento in particolare ai ragazzi di BSML, Cap, Moro, Tancre, Jack, e Ale. Grazie per la nostra amicizia, per le serate passate in biblio, per i pranzi al parchetto, e per tutti i bei ricordi che mi porterò dietro. Grazie soprattutto per aver mantenuto alta l'asticella, creando un sano livello di "peer pressure" e competizione, e per esservi sempre aiutati e spronati a vicenda a dare il massimo. Nonostante le nostre strade ora si separeranno, spero con tutto il cuore di mantenere questo bellissimo rapporto.

Un ringraziamento infine a tutti i miei amici non universitari, che sono stati parte fondamentale della mia vita in questi tre anni tra Milano, Roma, e Sydney. Grazie per tutte le esperienze insieme, per i viaggi, le serate in compagnia, e per le discussioni, con cui ho avuto modo di apprezzare punti di vista e prospettive di vita diversi da quella che è la mia bolla universitaria.

Contents

1 Introduction **1**

1.1 Purpose 1

1.2 Methods and Sources 1

2 Deep Generative Models **2**

2.1 Flow-Based Models 3

2.1.1 RealNVP 5

2.2 Latent Variable Models 6

2.2.1 Probabilistic Principal Component Analysis (pPCA) 7

2.2.2 Variational Auto Encoders 8

2.2.3 Deep Diffusion Generative Models 10

3 Stochastic Differential Equations **13**

3.1 Background 13

3.1.1 Ordinary Differential Equations 13

3.1.2 Stochastic Processes 14

3.2 Theory of Stochastic Differential Equations 15

3.2.1 Reverse Time Stochastic Differential Equations 17

3.2.2 Backward Stochastic Differential Equations 17

4 SDEs in Generative Modeling **19**

4.1 Score-Based Generative Modeling through SDEs 19

4.2 Deep Generative Modeling with Backward Stochastic Differential Equations 23

5 Implementation and Evaluation of DGMs **27**

5.1 Dataset 27

5.2 Models Implementation 27

5.2.1 Classes 28

5.2.2 Training 29
5.3 Results and Evaluation 29

6 Conclusions **32**

1 Introduction

1.1 Purpose

Generative modeling represents a fundamental area in artificial intelligence, focused on understanding and replicating the distribution of data. Unlike discriminative models, which aim to categorize or predict outcomes based on input data, generative models' goal is to generate new data instances that resemble the training data.

Stochastic processes play a central role in this approach, and very recent developments explore the connections of generative modeling with stochastic differential equations (SDEs). The purpose of this thesis is to review the foundations of generative modeling, of the stochastic processes used for this purpose, and to show new ideas on how SDEs can be used to build generative models. This models will be finally implemented, tested, and compared.

1.2 Methods and Sources

The first chapter proposes an introductory overview to deep generative models (DGMs), describing techniques such as Flow-Based Models [15], and various Latent Variable Models [16]. The second chapter provides the necessary background and theoretical framework on stochastic differential equations [7], including reverse time SDEs and backward SDEs.

These first two sections will then serve as building blocks for the third chapter, where the concepts are unified, with the aim to explore how stochastic differential equations can be employed in the context of deep generative models. In particular, we'll see how SDEs can be utilized to transform data distributions into a simpler prior distribution by gradually injecting noise [11], and then how backward stochastic differential equations can be employed to develop new generative models [17]. In the last chapter, the aforementioned models are implemented in Python and tested on the MNSIT dataset, comparing their performances in terms of qualitative and quantitative metrics.

2 Deep Generative Models

This chapter approaches the field of Deep Generative Modeling, providing an overview of the theory behind some of the most important generative models.

In general, a generative model is a probabilistic framework used to generate data. Starting from some sample data, the model learns a latent representation of them (i.e., the underlying probability distribution), and then, by sampling from this learnt distribution, it generates new data which should resemble the training set.

More formally, let's consider a dataset of samples $x_1, \ldots, x_n$, drawn from a true data distribution $p(x)$. In Figure 1, these samples are depicted as black dots within a blue region representing $p(x)$. The model proposes a new distribution $\hat{p}_\theta(x)$ (shown in green), constructed by initially drawing samples from a standard Gaussian distribution (shown in red). These Gaussian samples are transformed via a generative model – typically a neural network – into the space of the data samples. The parameters (or weights) θ of this neural network dictate the characteristics of the generated data, controlling how the generated distribution $\hat{p}_\theta(x)$ approximates the true distribution $p(x)$. The goal in training these models is to adjust θ so that $\hat{p}_\theta(x)$ is similar to $p(x)$. This adjustment process is carried out by minimizing a loss function which quantifies the discrepancy between the generated and true distributions, such as the Kullback-Leibler (KL) divergence. Throughout the training process, θ is iteratively adjusted to reduce this discrepancy, transforming $\hat{p}_\theta(x)$ from an initially arbitrary distribution into one that increasingly resembles the true data distribution $p(x)$, finally allowing to generate new data similar to the original ones.

We can identify four main categories of deep generative models:

- Flow-based models
- Latent variable models
- Autoregressive generative models (ARM)
- Energy-based models

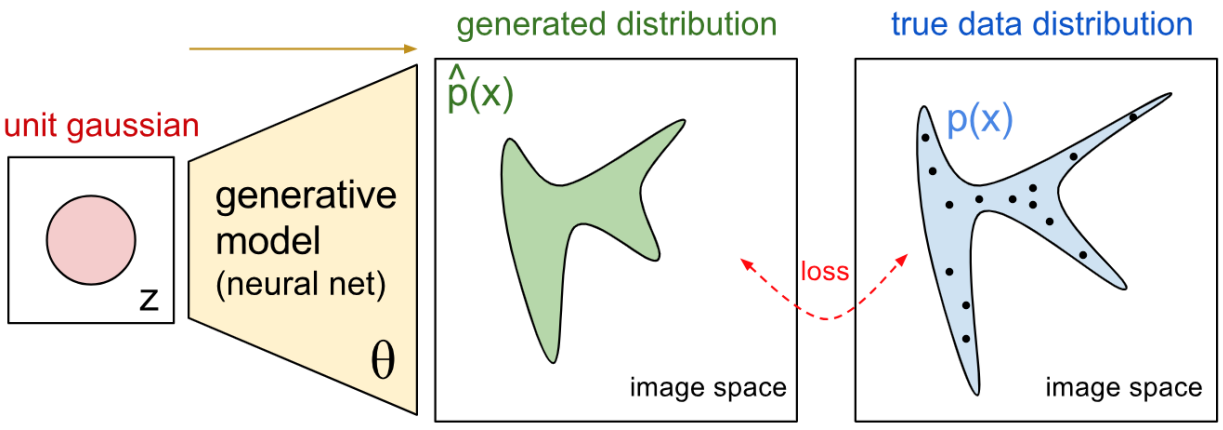

Figure 1: Generative model framework. The picture is taken from the OpenAI blog [4].

As for the purposes of this thesis, we will focus on the first two categories.

2.1 Flow-Based Models

Flow-based models, or flows, are a class of deep generative models that enable exact latent-variable inference and log-likelihood evaluation. They use a series of invertible transformations to generate complex multi-dimensional data distributions.

The core concept of flow-based models relies on the change of variables formula in probability theory, which allows the transformation of densities under invertible mappings. The change of variables formula in the multidimensional case is:

$$p(\mathbf{x}) = p(\mathbf{z} = f^{-1}(\mathbf{x}))|\mathbf{J}_f(\mathbf{x})|^{-1}, \tag{2.1}$$

where $p(\mathbf{x})$ is the density of the data, f is an invertible function (a bijection), and $\mathbf{J}_f(\mathbf{x})$ is the Jacobian matrix of the transformation f at point $\mathbf{x}$. The Jacobian matrix $\mathbf{J}_f(\mathbf{x})$ is the matrix of all first-order partial derivatives of f and $|\mathbf{J}_f(\mathbf{x})|$ represents the determinant.

Let's now consider a series of invertible functions, $f_k : \mathbb{R}^D \to \mathbb{R}^D$. Starting with a known

distribution $\pi(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0|0, \mathbf{I})$, we can apply these transformations, getting:

$$p(\mathbf{x}) = \pi\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})\right) \prod_{i=1}^{K} |\mathbf{J}_{f_i}(\mathbf{z}_{i-1})|^{-1}. \tag{2.2}$$

Figure 2 shows an example of how a simple (unimodal) distribution, like Gaussian, is turned into a complex (multimodal) one, employing a sequence of f_i. In general, we can obtain nearly any complex distribution and transform it back into a simple one.

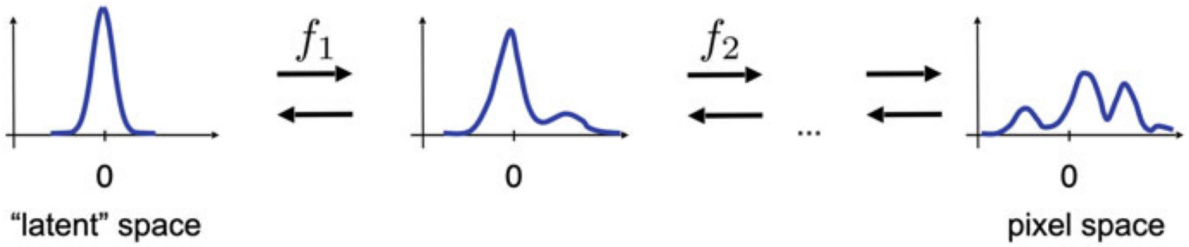

Figure 2: Visual example of transforming a unimodal distribution into a multimodal distribution (image taken from [15]).

Taking the logarithm of (2.2), with $\pi(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0|0, \mathbf{I})$, we get

$$\ln p(\mathbf{x}) = \ln \mathcal{N}\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})|0, \mathbf{I}\right) - \sum_{i=1}^{K} \ln |\mathbf{J}_{f_i}(\mathbf{z}_{i-1})|^{-1}, \tag{2.3}$$

where the first part of the equation is the *Mean Square Error* loss function, and the second part, which ensures that the distribution is normalized, acts as a *regularizer* by penalizing large changes of volume for the invertible functions.

Now, in order to model the invertible transformations, it's possible to use neural networks, due to their flexibility and simplicity in training. However, not every neural network can be employed, but we need neural networks that are invertible and in which the Jacobian-determinant is simple to compute. A *normalizing flow* or *flow-based model* is the model built using neural networks with such properties.

Among the principal flow-based models, we have RealNVP, ResNet Flows, and DenseNet Flows. For the scope of this thesis, we will limit to describe the first one.

2.1.1 RealNVP

One of the most significant classes of flow-based models is RealNVP, *Real-valued Non-Volume Preserving* flows, which are also a foundation for numerous other models. We will now briefly describe the main components and the learning objective of these models.

One of the building blocks of RealNVP is the so called *coupling layer*. The idea is to divide the input to the layer in two parts $\mathbf{x} = [\mathbf{x}_a, \mathbf{x}_b]$, and then to apply the following transformation:

$$\mathbf{z}_a = \mathbf{x}_a \tag{2.4}$$

$$\mathbf{z}_b = \exp s(\mathbf{x}_a) \odot \mathbf{x}_b + t(\mathbf{x}_a), \tag{2.5}$$

where s and t are the scaling and transition arbitrary neural networks, and $\odot$ is the symmetric tensor product operation.

With this transformation, we have that the Jacobian matrix $\mathbf{J}$ is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{I}_{d\times d} & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{z}_b}{\partial \mathbf{x}_a} & \operatorname{diag}(\exp(s(\mathbf{x}_a))) \end{bmatrix}, \tag{2.6}$$

whose determinant, given that it is a lower triangular matrix, is easily tractable, and is equivalent to:

$$\det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp(s(\mathbf{x}_a)_j) = \exp\left(\sum_{j=1}^{D-d} s(\mathbf{x}_a)_j\right). \tag{2.7}$$

Even if coupling layers are invertible, and the logarithm of the Jacobian-determinant is easily tractable, we are only processing part of the input, so we have to combine them with *permutation layers*, which being volume-preserving can be applied after each coupling layer, changing the order of variables.

Another important concept is that of *dequantization*. Since flow-based models assume that data are random variables, we often have a problem with many discrete objects. This can be overcome by adding (uniform) noise to the data, thus obtaining a continuous space.

As for the learning objective, it corresponds to the log-likelihood function. Applying the coupling and permutations layers as outlined above and plugging the logarithm of the Jacobian-determinant for the coupling layers in Eq. (2.3), we obtain

$$\ln p(\mathbf{x}) = \ln \mathcal{N}\left(\mathbf{z}_0 = f^{-1}(\mathbf{x})|0, \mathbf{I}\right) - \sum_{i=1}^{K} \left(\sum_{j=1}^{D-d} s_k \left(\mathbf{x}_a^k\right)_j \right), \tag{2.8}$$

where s_k is the scale network in the k-th coupling layer, and $\mathbf{x}_a^k$ is the input to the respective layer. The exponential in the determinant is then removed through the logarithm. The first part of the objective can be interpreted as a function of $-MSE(0, f^{-1}(\mathbf{x}))$, while for the second part, we have a sum over transformations, considering only the outputs of s_k for each coupling layer.

2.2 Latent Variable Models

A widely used approach in generative modeling is that based on *latent variables* [16]. These models incorporate hidden factors that represent underlying structures in high-dimensional data. Typically, data $\mathbf{x} \in \mathcal{X}^D$, where $\mathcal{X}$ is the data space, is generated through a process involving latent variables $\mathbf{z} \in \mathcal{Z}^M$, with $\mathcal{Z}^M$ being a latent space with lower dimensionality. To summarize the generative process, we start by sampling the latent variable $\mathbf{z}$, to then generate new data by sampling $\mathbf{x}$ from $p(\mathbf{x}|\mathbf{z})$. The factorization of the joint distribution is $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$, which expresses the aforementioned generative process. Since during training we only know the sample data $\mathbf{x}$, we need to *marginalize out* $\mathbf{z}$. Therefore, the marginal likelihood is

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}. \tag{2.9}$$

The difficulty now is to calculate this integral. The next sections present different approaches, both in the case of tractable integral, with the *probabilistic Principal Component Analysis* model, and non tractable, in which case *variational inference* techniques are employed, such as *Variational Auto Encoders* and *Deep Diffusion Generative Models*.

2.2.1 Probabilistic Principal Component Analysis (pPCA)

The *pPCA* [14] is a probabilistic model for which the PCA framework serves as a maximum likelihood estimate of its parameters. It assumes that

- $\mathbf{z} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^D$ are continuous random variables.
- $\mathbf{z} \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$.
- $\mathbf{z}$ and $\mathbf{x}$ are linearly dependent with Gaussian additive noise:

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \mathbf{b} + \epsilon. \tag{2.10}$$

where $\epsilon \sim \mathcal{N}(\epsilon|0, \sigma^2\mathbf{I})$.
Then the conditional distribution of $\mathbf{x}$ given $\mathbf{z}$ is also Gaussian:

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|\mathbf{W}\mathbf{z} + \mathbf{b}, \sigma^2\mathbf{I}), \tag{2.11}$$

The likelihood of the observed data integrates out the latent variables:

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z}. \tag{2.12}$$

The integration results in a marginal likelihood that is also Gaussian, given by:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}). \tag{2.13}$$

Moreover, pPCA facilitates the analytical computation of the true posterior distribution over the latent variables $\mathbf{z}$, which is Gaussian as well:

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{M}^{-1}\mathbf{W}^T(\mathbf{x} - \mathbf{b}), \sigma^{-2}\mathbf{M}), \tag{2.14}$$

where $\mathbf{M} = \mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I}$. By maximizing the log-likelihood, the parameters of the model can be estimated, allowing for the extraction of significant features from the data.

pPCA is a simple and effective latent variable model. However, if the dependencies are non-linear or distributions other than Gaussian are employed, it would be impossible to compute the integral precisely, leading to the necessity for approximation.

2.2.2 Variational Auto Encoders

When the integral in Eq. (2.9) is non-tractable, so that it can't be calculated analytically, pPCA can't be employed. A simple approach to calculate it would be by Monte Carlo approximation. However, it suffers from the *curse of dimensionality*, as we have an exponential growth of samples with respect to the latent space's dimensions.

A better way to approximate the posterior distribution of the latent variables is the *variational approach*, which allows to approximate complex posterior distributions by framing the problem as an optimization task.

Let's consider a class of variational distributions with parameters ϕ, $\{q_\phi(\mathbf{z})\}_\phi$, such as Gaussians with $\phi = \{\mu, \sigma^2\}$, and assume that all $\mathbf{z} \in \mathcal{Z}^M$ are assigned with a positive probability. Approximating the logarithm of the marginal distribution, and considering $q_\phi(\mathbf{z}|\mathbf{x})$ instead of $q_\phi(\mathbf{z})$ for each $\mathbf{x}$, i.e. an amortized variational posterior, which is useful as a single model is trained, which returns the parameters of a distribution for the given input, we get

$$\ln p(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x}|\mathbf{z})] - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln q_\phi(\mathbf{z}|\mathbf{x}) - \ln p(\mathbf{z})]. \tag{2.15}$$

We then get a *Variational Auto-Encoder*, which is a model very similar to an auto-encoder, with a *stochastic encoder* $q_\phi(\mathbf{z}|\mathbf{x})$, and a *stochastic decoder* $p(\mathbf{x}|\mathbf{z})$.

The Eq. (2.15) is the *Evidence Lower BOund (ELBO)*, which is the log-likelihood function's lower bound. In the ELBO, $\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x}|\mathbf{z})]$, represents the *reconstruction error* of encoding $\mathbf{x}$ in $\mathbf{z}$ and then deconding in the opposite way, while $\mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln q_\phi(\mathbf{z}|\mathbf{x}) - \ln p(\mathbf{z})]$, could be seen as a *regularizer*, which in the basic case corresponds to the Kullback-Leibler (KL) divergence.

In the VAE framework, almost any distribution can be chosen for the latent variables, and

the decision depends on how we want to represent the latent factors in the data. Later, neural networks can be used to set the parameters of both the encoders and the decoders. Usually, for ease, $\mathbf{z} \in \mathbb{R}^M$ is a vector of continuous r.v.s. After that, Gaussians can be used for the variational posterior and for the prior:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}|\mu_\phi(\mathbf{x}), \text{diag}[\sigma^2_\phi(\mathbf{x})]\right) \tag{2.16}$$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I}), \tag{2.17}$$

where $\mu_\phi(\mathbf{x})$ and $\sigma^2_\phi(\mathbf{x})$ are given by a neural network, like in the decoder's case.
An important component of VAEs is the *Reparameterization Trick*. This solves the problem of backpropagation, which cannot be used as it requires deterministic layers to apply gradient descent properly. It works by expressing the random variable as a deterministic variable transformed by a random noise ϵ:

$$\mathbf{z}_{\phi,n} = \mu_\phi(\mathbf{x}_n) + \sigma_\phi(\mathbf{x}_n) \odot \epsilon. \tag{2.18}$$

It is employed in the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ to reduce the gradient's variance, as now the randomness comes from $p(\epsilon)$, and so the gradient is computed w.r.t. a deterministic function.
Therefore, we start with the distributions (2.16), (2.17), and a categorical distribution $p_\theta(\mathbf{x}|\mathbf{z})$, with $x_d \in \mathcal{X} = \{1, 2, ..., L-1\}$, and we have the two networks:

- *Encoder:*

$$\mathbf{x} \in \mathcal{X}^D \to \text{Linear}(D, 256) \to \text{Leaky ReLU} \to$$
$$\text{Linear}(256, 2 \cdot M) \to \text{split} \to \mu \in \mathbb{R}^M, \log \sigma^2 \in \mathbb{R}^M.$$

- *Decoder:*

$$\mathbf{x} \in \mathcal{X}^D \to \text{Linear}(M) \to \text{Leaky ReLU} \to$$
$$\text{Linear}(256, D \cdot L) \to \text{reshape} \to \text{softmax} \to \theta \in [0,1]^{D \times L}.$$

where, since the distribution for x is categorical, the decoder network returns probabilities through the softmax function.

Finally, for a given dataset $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$, the aim during training is to minimize the (negative) ELBO:

$$-\text{ELBO}(\mathcal{D}; \theta, \phi) =$$
$$\sum_{n=1}^{N} \left[-\ln \text{Categorical}(\mathbf{x}_n | \boldsymbol{\theta}(z_{\phi,n})) + \ln \mathcal{N}(z_{\phi,n} | \mu_\phi(\mathbf{x}_n), \sigma_\phi^2(\mathbf{x}_n)) + \ln \mathcal{N}(z_{\phi,n} | 0, \mathbf{I}) \right].$$

The learning procedure can be then summarized as:

1. Apply the encoder network to $\mathbf{x}_n$, obtaining $\mu_\phi(\mathbf{x}_n)$ and $\ln \sigma_\phi^2(\mathbf{x}_n)$.
2. Apply the reparameterization trick, obtaining $\mathbf{z}_{\phi,n} = \mu_\phi(\mathbf{x}_n) + \sigma_\phi(\mathbf{x}_n) \odot \mathbf{e}$, where $\mathbf{e} \sim \mathcal{N}(0, \mathbf{I})$.
3. Apply the decoder network to $\mathbf{z}_{\phi,n}$ to get the probabilities $\boldsymbol{\theta}(\mathbf{z}_{\phi,n})$.
4. Compute the ELBO using $\mathbf{x}_n$, $\mathbf{z}_{\phi,n}$, $\mu_\phi(\mathbf{x}_n)$, and $\ln \sigma_\phi^2(\mathbf{x}_n)$.

2.2.3 Deep Diffusion Generative Models

Another important type of latent variable models is *hierarchical* latent variable models, which are used to capture dependencies among latent variables at different levels of abstraction, enabling the representation of complex hierarchical structures in data. A hierarchical latent variable model class that has become extremely popular recently is that of *Deep Diffusion Generative Models* (DDGMs).

DDGMs are hierarchical VAEs in which a reversed diffusion (DNN) parameters the top-down path, and a diffusion process, such as a Gaussian diffusion, defines the variational posterior (bottom-up path).

The original main idea [9] is to learn a reverse diffusion process to rebuild the structure in the data after having demolished it with a forward diffusion process.

We consider a set of latent variables $\mathbf{z}_{1:T} = [\mathbf{z}_1, \ldots, \mathbf{z}_T]$. The marginal likelihood is defined as:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}_{1:T}) d\mathbf{z}_{1:T}. \tag{2.19}$$

The joint distribution is represented as a Gaussian transitioning first-order Markov chain.:

$$p_\theta(\mathbf{x}, \mathbf{z}_{1:T}) = p_\theta(\mathbf{x}|\mathbf{z}_1) \left(\prod_{i=1}^{T-1} p_\theta(\mathbf{z}_i|\mathbf{z}_{i+1}) \right) p_\theta(\mathbf{z}_T), \tag{2.20}$$

where $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{z}_i \in \mathbb{R}^D$ for $i = 1, \ldots, T$.

We can now bring in a class of variational posteriors as specified below:

$$Q_\phi(\mathbf{z}_{1:T}|\mathbf{x}) = q_\phi(\mathbf{z}_1|\mathbf{x}) \left(\prod_{i=2}^{T} q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1}) \right), \tag{2.21}$$

where we use Q_ϕ to indicate the product of a sequence of variational posteriors $q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1})$.

Now, these distributions are formulated as this Gaussian diffusion process:

$$q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1}) = \mathcal{N}(\mathbf{z}_i|\sqrt{1-\beta_i}\mathbf{z}_{i-1}, \beta_i \mathbf{I}), \tag{2.22}$$

where $\mathbf{z}_0 = \mathbf{x}$.

A step of the diffusion, $q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1})$, works by taking the earlier generated object $\mathbf{z}_{i-1}$, scaling it by $\sqrt{1-\beta_i}$, and adding then noise with variance β_i.

Through the reparameterization trick, we can rewrite it more explicitly:

$$\mathbf{z}_i = \sqrt{1-\beta_i}\mathbf{z}_{i-1} + \sqrt{\beta_i} \odot \epsilon, \tag{2.23}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$.

The learning objective is again the ELBO, which can be rewritten in terms of KL divergences:

$$\begin{aligned}
\mathcal{L}(\mathbf{x}; \theta, \phi) &= \mathbb{E}_{Q_\phi(\mathbf{z}_{1:T}|\mathbf{x})}[\ln p_\theta(\mathbf{x}|\mathbf{Z}_1)] + \\
&- \sum_{i=2}^{T-1} \mathbb{E}_{Q_\phi(\mathbf{z}_{-i}|\mathbf{x})}[KL[Q_\phi(\mathbf{z}_i|\mathbf{z}_{i-1})||p_\theta(\mathbf{z}_i|\mathbf{z}_{i+1})]] + \\
&- \mathbb{E}_{Q_\phi(\mathbf{z}_{-T}|\mathbf{x})}[KL[Q_\phi(\mathbf{z}_T|\mathbf{z}_{T-1})||p_\theta(\mathbf{z}_T)]] + \\
&- \mathbb{E}_{Q_\phi(\mathbf{z}_{-1}|\mathbf{x})}[KL[Q_\phi(\mathbf{z}_1|\mathbf{x})||p_\theta(\mathbf{z}_1|\mathbf{z}_2)]].
\end{aligned}$$

At present, DDGMs are highly popular deep generative models, and thanks to advances in deep learning, a diffusion-based model which achieved state-of-the-art results in image synthesis was developed [3]. Further researches found that DDGMs are closely related to score-based generative models [13], giving a precise connection between DDGMs and stochastic differential equations [11], which we'll explore in Chapter 4.

3 Stochastic Differential Equations

Stochastic differential equations (SDEs) are a powerful tool for modeling uncertainty. While ordinary differential equations (ODEs) describe deterministic systems, SDEs produce a different solution trajectory each time they are solved, making them useful for modeling phenomena influenced by random forces.

This chapter provides an introduction to SDEs, starting with a review of ODEs and basic stochastic processes. Afterwards, the concept of stochastic differential equation in terms of Brownian motion is presented, to then focus on two types of SDEs, reverse time and backward SDEs, which will be fundamental for the ideas in the next chapter, where they are employed in novel deep generative models.

3.1 Background

3.1.1 Ordinary Differential Equations

An ordinary differential equation [6] is a mathematical equation that describes the relationship between a function and its derivatives. The general form for an ODE is:

$$F(t, x(t), \frac{dx(t)}{dt}, \frac{d^2x(t)}{dt^2}, ..., \frac{d^nx(t)}{dt^n}) = 0, \tag{3.1}$$

where t is the independent variable (usually time), $x(t)$ represents the dependent variable we're solving for, as a function of t, $d^ix(t)/dt^i$ represents the $i-th$ derivative of $x(t)$ with respect to t. F is a function that relates t, $x(t)$, and its derivatives, while n is the order of the ODE, referring to the highest derivative of $x(t)$ in the equation.

Differential equations of order n are often transformed into vector differential equations of order one:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t), t) + \mathbf{G}(\mathbf{x}(t), t)\mathbf{w}(t), \tag{3.2}$$

where $\mathbf{x}(t) \in \mathbb{R}^D$ is usually referred to as the state of the system, $\mathbf{G}$ and $\mathbf{f}$ are arbitrary functions, and $\mathbf{w}(t) \in \mathbb{R}^D$ is the forcing function or input to the system. This representation is named the state-space form of the differential equation, and it appears to be useful in the domain of stochastic differential equations.

A very useful class of differential equations are linear equations of the form

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(t)\mathbf{x}(t) + \mathbf{G}(t)\mathbf{w}(t), \tag{3.3}$$

which often appears in applications, and, unlike general nonlinear differential equations, we can actually solve these equations. Most solution methods for differential equations are beyond the scope of this thesis, but can involve analytical techniques for exact solutions, numerical methods for approximate solutions, and transformations such as the Laplace and Fourier transforms to simplify equations.

3.1.2 Stochastic Processes

A stochastic process is a collection of random variables indexed by time or space, representing the evolution of a system subject to randomness. Unlike ODEs, which model deterministic systems, stochastic processes can be used to model randomness.

Definition 3.1 (Stochastic Process) *Consider an arbitrary set T. A stochastic process is a class $X = \{X(t, \omega) : t \in T\}$ of random variables depending on t.*

A *probability space* associated with a stochastic process is a triple $(\Omega, \mathcal{F}, p)$, where Ω is the sample space, $\mathcal{F}$ is a sigma-algebra on Ω, representing all possible events, and p is a probability measure that gives probabilities to the elements in $\mathcal{F}$. Each random variable $X(t, \omega)$ in the family is a function from Ω to $\mathbb{R}$, for each fixed $t \in T$.

The *expected value* of a stochastic process at time t is the integral of $X(t, \omega)$ w.r.t. the probability measure p, denoted as $\mathbb{E}[X(t)]$, assuming the integral exists.

The simplest stochastic process is the random walk, which models the path of a particle that moves in random directions at fixed intervals.

Definition 3.2 (Random Walk) *A random walk is a stochastic process $\{S_n\}_{n=0}^{\infty}$, where each S_n represents the position of a point at step n, and is defined recursively by $S_n = S_{n-1} + X_n$. Here, S_0 is the starting position, often set to zero, and $\{X_n\}$ is a sequence of i.i.d. random variables representing the steps of the walk. The state space of the random walk is typically $\mathbb{Z}$, the set of all integers, and the steps X_n can take positive or negative integer values, or zero.*

Another fundamental concept in stochastic processes is the one of Markov chains, which are discrete-time processes having the property that the process's future state is determined only by its present state and not by the sequence of events that came before it.

Definition 3.3 (Markov Chain) *A Markov chain is a stochastic process $\{X_n\}_{n\geq 0}$ taking on a finite or countable number of states, distinguished by the Markov property. This property stipulates that the future state of the process only depends on the current state, and is independent from the sequence of precedent events. Mathematically, for any times s, t with $0 \leq s < t$ and any states i, j, the Markov property can be expressed as:*

$$p(X_t = j | X_s = i, X_{s-1}, X_{s-2}, \ldots, X_0) = p(X_t = j | X_s = i).$$

3.2 Theory of Stochastic Differential Equations

At the very core, a stochastic differential equation is an ODE of the form Eq. (3.2), where $\mathbf{w}(t)$ is a stochastic process. Then we have the general form

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}, t)dt + \mathbf{G}(\mathbf{x}, t)d\mathbf{w}(t) \tag{3.4}$$

where $\mathbf{w}(t)$ is a d-dimensional Brownian motion, such that $\mathbf{w}(t)$ and $\mathbf{w}(t')$ are independent for all $t \neq t'$. The function $\mathbf{f}(\mathbf{x}, t) \in \mathbb{R}^d$ is the *drift function*, and $\mathbf{G}(\mathbf{x}, t) \in \mathbb{R}^{d\times d}$ is the

dispersion matrix, defining how $\mathbf{w}(t)$ enters the system.

The concept of Brownian motion [8], also called Wiener process, is a generalization of the random walk process, with each increment being independent. Therefore, the magnitude and direction of the process's changes are random and independent with respect to the earlier ones.

Definition 3.4 (Brownian motion) *A Brownian motion* $\mathbf{w}(t) \in \mathbb{R}^d$ *is a continuous stochastic process with the following properties:*

1. *Any* $\Delta\mathbf{w}_k = \mathbf{w}(t_{k+1}) - \mathbf{w}(t_k)$ *is a Gaussian r.v. with* $\mu = 0$ *and covariance* $\mathbf{G}\Delta t_k$*, in which* $\mathbf{G}$ *is the diffusion matrix of the Brownian motion and* $\Delta t_k = t_{k+1} - t_k$*.*
2. *Not overlapping increments' time spans imply independent increments.*
3. $\mathbf{w}(0) = \mathbf{0}$*.*

$w(t)$ indicates a one-dimensional Brownian motion. If we are in the scalar case, the diffusion matrix is often called the diffusion coefficient and denoted as g.

Euler-Maruyama method to solve SDE A simple numerical technique used to approximate solutions of SDEs is the *Euler-Maruyama method* [5]. Given a general SDE as in Eq. (3.4), the Euler-Maruyama method discretizes time into small steps of size Δt. Starting from an initial state $\mathbf{x}(0)$, it iteratively computes the state at each time step using the update rule:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{f}(\mathbf{x}, t)\Delta t + \mathbf{G}(\mathbf{x}, t)\Delta\mathbf{w}(t), \tag{3.5}$$

where the usual notation is used and $\Delta\mathbf{w}(t)$ follows a Normal distribution with $\mu = 0$ and $\sigma^2 = \Delta t$. This method is computationally simple and easy to implement, but, being a first-order method, may exhibit strong numerical errors, especially for nonlinear SDEs, where more sophisticated methods should be used for accurate and reliable simulations.

3.2.1 Reverse Time Stochastic Differential Equations

Reverse time stochastic differential equations [1] are SDEs that model stochastic processes by reversing the direction of time. The mathematical formulation typically involves transforming a forward-time SDE like in Eq. (3.4) to a reverse-time SDE by reversing the time variable, $t \to T - t$, which affects both the drift and diffusion terms and requires a redefinition of the stochastic process dynamics. Hence, given the forward-time SDE

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}, t)dt + \mathbf{G}(\mathbf{x}, t)d\mathbf{w}(t), \tag{3.6}$$

the matching reverse-time SDE is

$$d\mathbf{x}(t) = \{\mathbf{f}(\mathbf{x}, t) - \nabla \cdot \left[\mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^T\right] - \mathbf{G}(\mathbf{x}, t)\mathbf{G}(\mathbf{x}, t)^T \cdot \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\}dt + \mathbf{G}(\mathbf{x}, t)d\overline{\mathbf{w}}(t), \tag{3.7}$$

where $d\overline{\mathbf{w}}(t)$ is the reverse-time Brownian motion and p_T is the pdf of $\overline{\mathbf{x}}(T)$ defined by the forward-time SDE.

This approach is particularly useful when the final state of a system is known, and the objective is to infer the historical states that led to this final state.

3.2.2 Backward Stochastic Differential Equations

Backward SDEs (BSDEs) [2] are SDEs in which we have a terminal condition, and the evolution is backward in time. While forward SDEs explain a process's evolution forward in time, they express the progression of a stochastic process backward in time. Pairing a BSDE with a forward SDE, we obtain a forward-backward stochastic differential equation (FBSDE), given by

$$\begin{aligned} \mathbf{x}(t) &= \xi + \int_0^t \mathbf{f}(s, \mathbf{x}(s))\, ds + \int_0^t \mathbf{G}(s, \mathbf{x}(s))\, d\mathbf{w}(s) \\ \mathbf{y}(t) &= \eta + \int_t^T \mathbf{b}(s, \mathbf{x}, \mathbf{y}, \mathbf{z})\, ds - \int_t^T \mathbf{z}(s)\, d\mathbf{w}(s), \end{aligned} \tag{3.8}$$

where $t \in [0, T]$, $\mathbf{x}(t) \in \mathbb{R}^{d_x}$ is the forward process, $\mathbf{y}(t) \in \mathbb{R}^{d_y}$ is the backward process, and $\mathbf{z}(t) \in \mathbb{R}^{d_y \times d_w}$ is the control process. The dynamics of $\mathbf{x}(t)$ are outlined by the drift function $\mathbf{f}(t, x)$ and diffusion function $\mathbf{G}(t, x)$. The relationship between $\mathbf{x}(t)$ and $\mathbf{y}(t)$ are specified by the generator function $\mathbf{b}(t, \mathbf{x}, \mathbf{y}, \mathbf{z})$. $\mathbf{y}(t)$.

$\mathbf{x}(0) = \xi$ is a given *initial condition*, and $\mathbf{y}(T) = \eta$ is the *terminal condition*. In some cases, such as the Markovian FBSDE, we have that $\mathbf{y}(T)$ is a function of $\mathbf{x}(T)$. Here the terminal condition $\mathbf{y}(T) = \phi\mathbf{x}(T)$ implies a dependence on the terminal value of the forward process. For the non-Markovian FBSDE, the terminal condition depends on the entire process of $\mathbf{x}$, denoted $\mathbf{y}(T) = \phi(\mathbf{x}(0 \leq t \leq T))$. It is assumed that the initial input of the forward process ξ is normally distributed, while the terminal value of the backward process η follows the target data distribution.

BSDEs have numerous applications in financial mathematics and risk management, for example they are used to model and price derivatives and future liabilities or payoffs that depend on the path of an underlying stochastic process.

4 SDEs in Generative Modeling

In recent years, various research directions that explore the use of stochastic differential equations in deep generative modeling have emerged. This chapter aims to put together the concepts explored in the previous sections, presenting two recent developments in this field: first, a cohesive framework that generalizes and improves prior work on score-based generative models using SDEs [11], and then, an innovative deep generative model, called *BSDE-Gen* [17], which blends the versatility of BSDEs with the strength of deep neural networks.

4.1 Score-Based Generative Modeling through SDEs

The idea to perturbate data with multiple noise scales is key to the success of various methods related to denoising diffusion probabilistic models [3][9].

Song et al. (2020) [11] propose to further generalize this idea to an infinite number of noise scales, such that, as noise intensifies, perturbed data distributions change according to an SDE. Score-based generative models are thus improved by employing SDEs, which transform data into a simple noise distribution. Estimating, through score matching, and thus knowing the score of marginal distributions at each time step, this SDE can be reversed to generate a new data sample. The process is visually summarized in Figure 3. The next paragraphs present more in detail the main concepts and the structure of the model in [11].

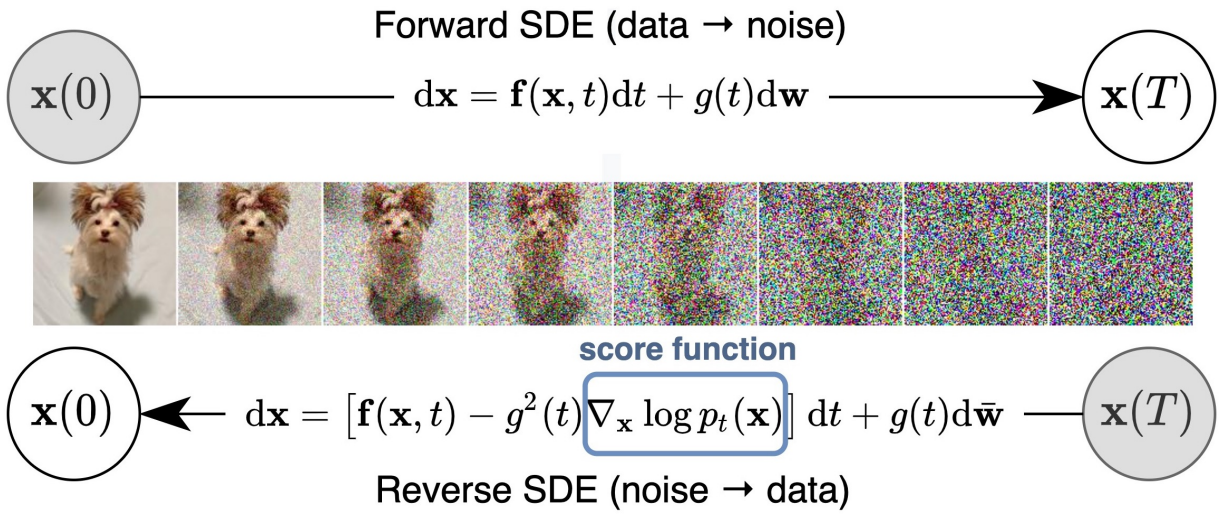

Figure 3: Model's overview (from [11]).

Background With the term *score-based generative models* we refer to two model classes:

- *Score matching with Langevin dynamics* (SMLD) [10] estimates the score (i.e., the gradient of the log probability density with respect to data) at each noise scale. During generation, Langevin dynamics is used to sample from a series of decreasing noise scales.
- *Denoising diffusion probabilistic modeling* (DDPM) [3][9] trains a series of probabilistic models to reverse each step of the noise corruption, utilizing information of the functional form of the reverse distributions to render training tractable.

Forward SDE The goal is to build a diffusion process $\{\mathbf{x}(t)\}_{t=0}^{T}$, parameterized by a continuous time variable $t \in [0, T]$, where $\mathbf{x}(0) \sim p_0$ and $\mathbf{x}(T) \sim p_T$. The initial distribution p_0 corresponds to a dataset of i.i.d. samples, and p_T represents the prior distribution. The diffusion process is described by the SDE:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\, dt + g(t)\, d\mathbf{w}, \tag{4.1}$$

where $d\mathbf{x}$ denotes the infinitesimal change in $\mathbf{x}$, $\mathbf{f}(\mathbf{x}, t)$ is a drift coefficient, $g(t)$ is a diffusion coefficient, and $d\mathbf{w}$ is an increment of a Wiener process. We denote by $p_t(\mathbf{x})$ the pdf of $\mathbf{x}$, and

use $p_{st}(\mathbf{x}(t)|\mathbf{x}(s))$ to indicate the transition kernel from $\mathbf{x}(s)$ to $\mathbf{x}(t)$, where $0 \leq s < t \leq T$. The distribution p_T is usually a prior distribution without any structure, thus uninformative with respect to p_0, for instance a Gaussian with fixed parameters.

Reverse SDE According to [1], the reverse of a diffusion process is itself a diffusion process, that runs backward in time and is defined by:

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\, dt + g(t)d\overline{\mathbf{w}}, \tag{4.2}$$

where $\overline{\mathbf{w}}$ is a standard Wiener process with time flowing backward from T to 0, and dt is an infinitesimal negative timestep. Starting from samples of $\mathbf{x}(T) \sim p_T$, after having estimated the score $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ of each marginal distribution for every t, we can determine the reverse diffusion process from Eq. (4.2) and simulate it in order to sample $\mathbf{x}(0) \sim p_0$.

Scores Estimation To estimate a distribution's score, it's possible to train a score-based model on samples with score matching. Hence, to estimate $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$, we can train a score-based model $\mathbf{s}_\theta(\mathbf{x}, t)$, generalizing the ELBOs in SMLD and DDPM, by finding:

$$\boldsymbol{\theta^*} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_t \left\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \left[\| \boldsymbol{s_\theta}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}} \log p_t(\mathbf{x}(t) \mid \mathbf{x}(0)) \|_2^2 \right] \right\}, \tag{4.3}$$

where $\lambda : [0, T] \to \mathbb{R}_{\geq 0}$ gives the weights, $t \sim U(0, T)$, $\mathbf{x}(0) \sim p_0(\mathbf{x})$ and $\mathbf{x}(t) \sim p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))$. With enough data and sufficient model size, score matching ensures that $\boldsymbol{s_{\theta^*}}(\mathbf{x}, t) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ for almost all $\mathbf{x}$ and t, with the solution $\mathbf{s}_{\theta^*}$ being optimal.
The parameter λ is usually picked such that $\lambda \propto 1/\mathbb{E}\left[\| \nabla_{\mathbf{x}} \log p_t(\mathbf{x}(t) \mid \mathbf{x}(0)) \|_2^2 \right]$, similarly to SMLD and DDPM. In order to efficiently solve Eq. (4.3), it is typically needed to know the transition kernel $p_{0t}(\mathbf{x}(t)|\mathbf{x}(0))$, which can be either a Gaussian distribution, in the case of affine drift coefficient $\mathbf{f}$, or obtained by solving Kolmogorov's forward equation, in the case of more general SDEs.

Samples Generation Afterwards, we can use the trained $\boldsymbol{s}_{\boldsymbol{\theta}^*}$ to construct the reverse-time SDE, and then apply a solver method such as the Euler-Maruyama to generate samples from p_0.

We have *controllable generation* if there are additional information or conditions under which we want to generate data, denoted by $\mathbf{y}$. If $p_t(\mathbf{y}|\mathbf{x}(t))$ is known, it is also possible to produce samples from $p_0(\mathbf{x}(0)|\mathbf{y})$. Given a forward SDE as in Eq. (4.1), it's possible to sample from $p_t(\mathbf{x}(t)|\mathbf{y})$. We start from $p_T(\mathbf{x}(T)|\mathbf{y})$ and solve the conditional reverse-time SDE:

$$d\mathbf{x} = \{\mathbf{f}(\mathbf{x}, t) - g(t)^2 \left[\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \log p_t(\mathbf{y}|\mathbf{x})\right]\}dt + g(t)d\overline{\mathbf{w}}. \tag{4.4}$$

Three useful applications of this type of generation are class-conditional generation, with $\mathbf{y}$ representing class labels, image imputation, and image colorization.

Contributions This framework's contributions include: improving the understanding of existing approaches, introducing novel sampling algorithms, enabling accurate likelihood computation, latent code manipulation, uniquely identifiable encoding, and bringing additional conditional generation capabilities to the score-based generative models class.

In particular, some theoretical and practical contributions are:

- *Flexible sampling and likelihood computation*, as different SDE solvers can be used in the reverse-time SDE integration in order to sample.
- *Controllable generation*, as the generation process can be modulated by conditioning on data unavailable during training, since the conditional reverse-time SDE can be estimated from unconditional scores in an efficient way.
- *Unified framework*, amalgamating methods of SMLD and DDPM as discretizations of two different SDEs, and providing a unified method to investigate and adjust SDEs to improve score-based generative models.

4.2 Deep Generative Modeling with Backward Stochastic Differential Equations

BSDE-Gen [17] is a novel deep generative model that exploits the ability of BSDEs to describe the evolution of a stochastic process backward in time and deep neural networks' capability to model data in high-dimensional spaces. Starting with a casual initial input following a standard normal distribution, BSDE-Gen moves toward a final value representing the target distribution. After training, the optimized parameters can be utilized to generate new samples through a forward scheme. The objective to minimize during training is an MMD loss function obtained from the BSDE-based dynamics. The result is a model able to produce new data that are similar to the initial dataset.

BSDE-based Generative Models Let's consider a forward-backward stochastic differential equation as in Eq. (3.8), where $\mathbf{x}(t)$ is the forward process, $\mathbf{y}(t)$ is the backward process, $\mathbf{z}(t)$ is the control process, $\mathbf{f}(t)$ is the drift function, $\mathbf{G}(t)$ is the diffusion function, and $\mathbf{b}(t, \mathbf{x}, \mathbf{y}, \mathbf{z})$ is the generator function, specifying the relation between the forward and backward processes.

To solve the FBSDE, we need to find the $\mathcal{F}(t)$-adapted stochastic process $(\mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t))$ for all $t \in [0, T]$ in a suitable space that satisfies the aforementioned equation, given $\mathbf{f}, \mathbf{G}, \mathbf{b}$, the Brownian motion $\mathbf{w}(t)$, and the initial and terminal conditions ξ, η.

Using the *Euler Maruyama method* in Eq. (3.5) for the two processes, we obtain

$$\begin{aligned} \mathbf{x}(t + \Delta t) &\approx \mathbf{x}(t) + \mathbf{f}(t, \mathbf{x}(t))\Delta t + \mathbf{G}(t, \mathbf{x}(t))\Delta \mathbf{w}(t) \\ \mathbf{y}(t + \Delta t) &\approx \mathbf{y}(t) - \mathbf{b}(t, \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t))\Delta t + \mathbf{z}(t)\Delta \mathbf{w}(t), \end{aligned} \tag{4.5}$$

where $\Delta \mathbf{w}(t) = \mathbf{w}(t + \Delta t) - \mathbf{w}(t)$.

Model Architecture Consider as known the drift function $\mathbf{f}(t)$, the diffusion function $\mathbf{G}(t)$, and the generator function $\mathbf{b}(t, \mathbf{x}, \mathbf{y}, \mathbf{z})$.

Then, to use the scheme in Eq. (4.5), we need to learn the initial value of the backward process $\mathbf{y}(0)$ and the control process $\mathbf{z}(t)$, which can be both approximated using the variable t and the respective forward process $\mathbf{x}(t)$. Hence, it's possible to use two deep neural networks, $\mathcal{N}^{\theta_{\mathbf{y}(0)}}(\mathbf{x}(0))$ and $\mathcal{N}^{\theta_{\mathbf{z}(t)}}(t, \mathbf{x}(t))$, such that

$$\mathbf{y}(0) \approx \mathcal{N}^{\theta_{\mathbf{y}(0)}}(\mathbf{x}(0)),\ \mathbf{z}(t) \approx \mathcal{N}^{\theta_{\mathbf{z}(t)}}(t, \mathbf{x}(t)), \tag{4.6}$$

with the following structure:

$$\mathcal{N}^{\theta}(\mathbf{x}) := \phi \circ \mathcal{L}_H \circ \sigma_{H-1} \circ \mathcal{L}_{H-1} \circ \cdots \circ \sigma_1 \circ \mathcal{L}_1(\mathbf{x}), \tag{4.7}$$

where H is the number of layers of the neural network, $\mathcal{L}_i(\mathbf{x})$ is a linear transformation, σ is the activation function, and ϕ is the function that maps to the state space. Dropout is used for regularization.

The complete structure is summarized in Figure 4. The BSDE-Gen model starts with the random initialization of an input $\mathbf{x}(0)$ and a series of Brownian motions $\mathbf{w}(t)$. We then obtain $\mathbf{x}(t)$ using the Euler Maruyama scheme, while $\mathbf{y}(0)$ and $\mathbf{z}(t)$ are obtained with the DNNs in Eq. (4.6), whose parameters $\theta = \{\theta_{\mathbf{y}}, \theta_{\mathbf{z}}\}$ are trained with the MMD Loss function (more details in the next paragraph). Combining these elements, $\mathbf{y}(t)$ is obtained using again Eq. (4.5). Finally, at time T, the generated sample $\mathbf{y}(T)$ is obtained.

Training The loss function used to train the deep neural networks in Eq. (4.6) is the MMD Loss. This is a statistical test that checks if two probability distributions are equivalent, computing their distance as the distance between mean embeddings of features through the reproduction of kernel Hilbert space (RKHS) $\mathcal{H}$.

Given two probabilities of random elements $\mathbb{P}$ and $\mathbb{Q}$ on $\mathcal{X}$, the MMD is:

$$\mathrm{MMD}^2(\mathbb{P}, \mathbb{Q}) = \|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|^2_{\mathcal{H}}, \tag{4.8}$$

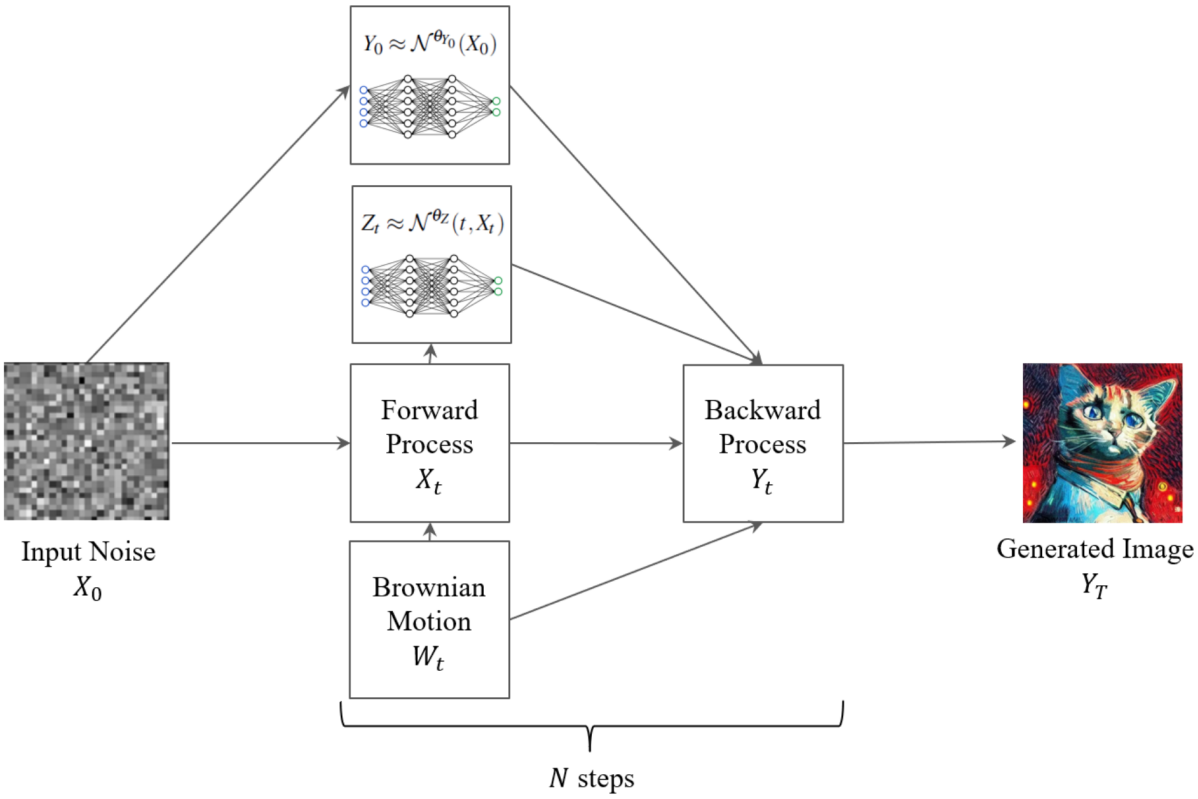

Figure 4: Model Architecture of BSDE-Gen (from [17]).

where $\mu_{\mathbb{P}}$ and $\mu_{\mathbb{P}}$ are the are the mean embeddings of $\mathbb{P}$ and $\mathbb{Q}$ in a RKHS $\mathcal{H}$. Under acceptable conditions, $\text{MMD}^2(\mathbb{P}, \mathbb{Q}) = 0 \iff \mathbb{P} = \mathbb{Q}$.

In the BSDE-Gen model, the MMD Loss is

$$L(\theta) = \hat{\text{MMD}}^2(\mathbf{y}(T), \eta), \tag{4.9}$$

where $\mathbf{y}(T)$ is the final value computed by the model and η represents the target data. After optimizing the parameters $\hat{\theta}$, it's possible to generate new samples using the Euler forward discrete scheme 4.5.

The BSDE-Gen models can be either trained with a decoder-only architecture or a encoder-decoder architecture. The first involves training the generative model to turn a noise vector into a realistic image, much like classic GANs. In contrast, the encoder-decoder method adds noise to the image, comparable to diffusion models (e.g. [3],[9],[10]).

Results and Limitations The BSDE-Gen model demonstrates a possible way for producing high-dimensional complex data, with potential applications in many fields, including computer vision and healthcare (e.g., drug discovery), by giving an instrument to model complex systems with uncertain dynamics and partial information.

However, there are some limitations. Firstly, the model is computationally complex, which can be a problem with large datasets. Also, BSDE-Gen necessitates the setting of some hyperparameters, namely the forward process and the generator function $\mathbf{b}$ of the backward process. Future research could focus on finding more efficient algorithms to decrease computational complexity, and on developing other encoder architectures to optimize the mapping between the BSDE-Gen model input and target data.

5 Implementation and Evaluation of DGMs

This chapter presents an implementation of the two models described in the previous section, i.e., a score-based generative model which uses SDEs [11], which we will refer to as *ScoreNet*, and a generative model which employs backward SDEs [17], called *BSDE-Gen*.
We will implement the models in Python using PyTorch, train them on the same dataset of images, and generate new samples. Finally, the results will be compared and evaluated, both through visual inspection and through different evaluation metrics, such as Inception Score and Reconstruction Error.

5.1 Dataset

The dataset employed to test the implementation of the two deep generative models is the MNIST dataset, a benchmark in the field of machine learning and computer vision. The dataset consists of a total of 70,000 28x28 pixels images, each depicting a handwritten digit from 0 to 9. The choice is driven by the simplicity of the dataset, which allows to train these models in an effecient way, yet allowing for a meaningful evaluation.

Figure 5: Examples from the MNIST dataset.

5.2 Models Implementation

The generative models implementations are inspired by the code published on Github by the authors of the respective papers [12] [18]. The code was revisited and adapted for the purposes of this analysis. In both cases, the framework used to implement the model is Pytorch.

5.2.1 Classes

ScoreNet The *ScoreNet* class, inheriting from *nn.Module*, implements a time-dependent score-based model with a U-Net architecture. This model employs Gaussian Fourier projections for encoding time steps and dense layers to reshape outputs into feature maps. The *GaussianFourierProjection* class generates random features for time encoding, while the *Dense* class reshapes outputs. The *ScoreNet* includes encoding and decoding paths. The encoding path reduces input resolution through convolutional layers, and the decoding path restores it using transposed convolutional layers.

ScoreNet initializes these layers along with group normalization and dense layers for time embeddings. The forward method processes input x and time t, embedding t and incorporating it into both encoding and decoding layers. The *Swish* activation function ensures non-linearity. During the *forward* pass, the encoding layers apply convolutions, normalization, and activations, integrating time embeddings. The decoding layers restore resolution, using skip connections from the encoding path, refining the output with transposed convolutions and normalization. The output is normalized dividing by the standard deviation of the perturbation kernel, ensuring consistency with input perturbation. This design allows *ScoreNet* to model temporal dynamics effectively, using the U-Net structure for precise reconstructions and temporal embeddings for time-dependent behavior.

BSDE-Gen For this model, a Python class *FBSDEGen*, inheriting from *nn.Module*, is defined. The class implements the Forward-Backward Stochastic Differential Equations (FBSDE) generator, defining its architecture with fully connected layers. The class constructor initializes the network layers and essential parameters such as the drift function b, the diffusion function *sigma*, and the generator *f*. The *forward* method implements the forward pass, where it generates the sample paths for the state variable x and the value function y over discrete time steps, using the network to predict the values at each step. The class utilizes a dropout layer to prevent overfitting and a GELU activation function to introduce

non-linearity.

5.2.2 Training

In both cases, the training process is handled by a *train* function, which iterates over the training data, adjusting the model parameters using the respective optimizer and loss function. As for *ScoreNet*, the loss function defined measures the difference between the model's predicted scores and the true noise added to the data. On the other hand, the loss function used for training *BSDE-Gen* is an implementation of the *MMD Loss* (Maximum Mean Discrepancy) described in Section 4.2.

The two models were trained on the MNIST dataset, using a batch size of 512.

	ScoreNet	**BSDE-Gen**
# Parameters	1,115,425	12,948,560
Time per epoch (sec)	~ 9	~ 350
Training epochs	31	31
Reconstruction Error	0.16	0.61
Inception Score	1.62	1.84

Table 1: Training characteristics of the two models

From the comparison in Table 1, it can be evicted how there are significant differences among the models, especially regarding computational efficiency and complexity. Indeed, BSDE-Gen has more than 10 times the parameters of ScoreNet, and this complexity results in a significantly longer training time.

5.3 Results and Evaluation

After having trained the models, it's possible to generate samples and evaluate their performances. We are going to compare the generated data under four main points of view: visual inspection, inception score (IS), reconstruction error (RE), and training running time.

Visual Inspection The first and most naive, yet effective, method to compare the models performances, is to visually inspect the generated images after n iterations.

Figure 6: Samples generated from *ScoreNet* model.

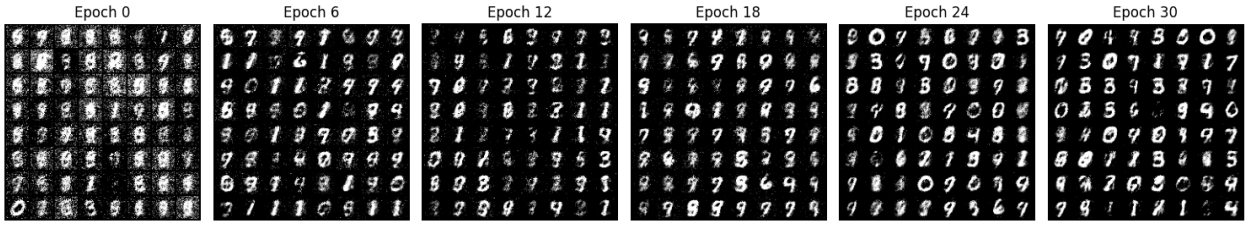

Figure 7: Samples generated from *BSDE-Gen* model.

From Figure 6 and 7, we can notice how, at the last iteration, both models generate images closely resembling the original data in Figure 5. However, we can see that the less parameterized *ScoreNet* model initially generates almost random images (see epoch 1 and 6 in Figure 6), to then significantly improve over training epochs. On the other hand, the more complex *BSDE-Gen* model generates images resembling digits from the start, and the improvement over epochs is less significant.

Inception Score and Reconstruction Error The previous observation is confirmed by the graphs in Figure 10, where the reconstruction error of *ScoreNet* is initially much higher compared to *BSDE-Gen*, but eventually the former model improves significantly, surpassing the latter in terms of Reconstruction Error (in red) and achieving a slightly lower Inception Score (in blue), at the last training epoch (see Table 1).

The Inception Score is a measure that evaluates the quality and diversity of images generated by a model. It uses a pre-trained Inception v3 classifier to predict the class probabilities for each generated image. Mathematically, IS is calculated using the Kullback-Leibler divergence between the conditional distribution of labels and the marginal distribution. A higher Inception Score indicates better image quality and diversity, with the score being higher when

generated images are both realistic and varied.

Reconstruction error, instead, measures the difference between original data and its reconstruction by a model, quantifying how accurately the model can recreate the input data from a compressed representation. Lower reconstruction errors indicate better model performance, reflecting its ability to capture and reproduce essential features of the data. Note that in Figure 10 the logarithm of the reconstruction error is plotted for visualization purposes.

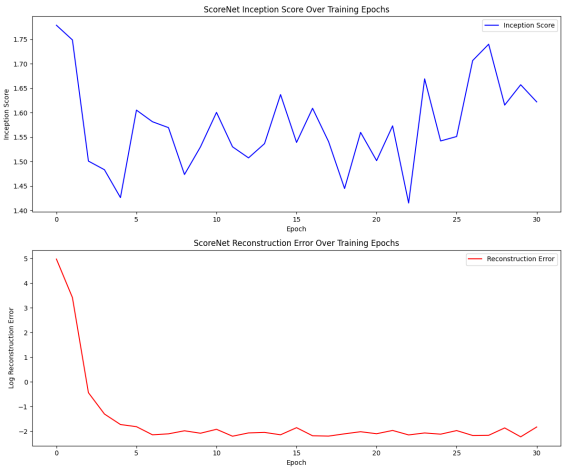

Figure 8: ScoreNet IS (in blue) and RE (in red)

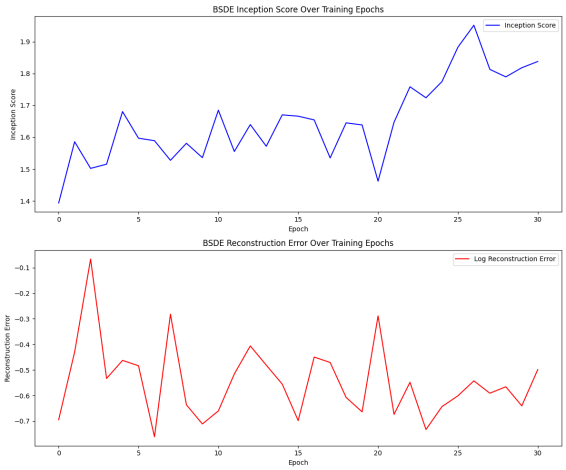

Figure 9: BSDE-Gen IS (in blue) and RE (in red)

Figure 10: Comparison of BSDE-Gen and ScoreNet Training Results

Running Time Finally, from a running time point of view, it must be noted that, using the same hardware, the *ScoreNet* total training time was 1/25 of the *BSDE-Gen* running time, showing how the first model proves to be the most efficient.

6 Conclusions

This thesis explored the integration of stochastic differential equations (SDEs) with deep generative models (DGMs), showing their potential in the field of generative modeling. The journey from theoretical concepts to practical implementations demonstrates the capabilities of SDEs to be employed in generative algorithms, possibly enhancing them.

We started with a foundational review of generative modeling, laying the foundation for the incorporation of stochastic processes. SDEs were thus introduced as a tool to model the randomness and uncertainty present in many real-world data generation processes.

This groundwork was fundamental for understanding the recent development of advanced models like the score-based and backward stochastic differential equation models, which represent the core focus of this investigation.

By implementing these models, we have showed the power of SDEs to enable precise control over the generative process. The score-based model, which employs a continuous spectrum of noise levels to refine data generation, and the BSDE model, which leverages backward time dynamics, both demonstrate how SDEs can lead to generated samples with high quality and similarity to the original data.

Offering a powerful tool for modeling complex systems with uncertain dynamics and incomplete information, the application of these models is potentially significant in many fields, such as computer vision and healthcare.

In conclusion, several promising directions can be explored by future research. In particular, enhancing the efficiency and scalability of SDE-based generative models can make them more practical for large-scale applications.

Moreover, as pointed out in [11], a meaningful research direction could be the identification of methods to combine the stable learning of these models with the efficient sampling of implicit models like GANs.

References

[1] Brian D.O. Anderson. "Reverse-time diffusion equation models." In: *Stochastic Process. Appl.*, (1982), pp. 313–326.

[2] Jean-Michel Bismut. "Conjugate convex functions in optimal stochastic control." In: *Journal of Mathematical Analysis and Applications* , (1973), pp: 384-404.

[3] Jonathan Ho, Ajay Jain, Pieter Abbeel. "Denoising diffusion probabilistic models". In: *arXiv:2006.11239*, (2020).

[4] Andrej Karpathy, Pieter Abbeel, Greg Brockman, Peter Chen, Vicki Cheung, Yan Duan, Ian Goodfellow, Durk Kingma, Jonathan Ho, Rein Houthooft, Tim Salimans, John Schulman, Ilya Sutskever, Wojciech Zaremba. "Generative models" In: *OpenAI Research*, (2016) Link: `https://openai.com/research/generative-models` (visited on 03/23/2024)

[5] Peter E. Kloeden , Eckhard Platen. "Numerical Solution of Stochastic Differential Equations", pp. 340-344. Springer, Berlin (1992). ISBN: 978-3-642-08107-1. DOI: 10.1007/978-3-662-12616-5.

[6] Simo Särkkä and Arno Solin. "Some Background on Ordinary Differential Equations" In: *Applied Stochastic Differential Equations*, pp.4–22. Cambridge University Press (2019). ISBN: 9781108186735. DOI: 10.1017/9781108186735. URL: `https://www.cambridge.org/core/services/aop-cambridge-core/content/view/D335EF751E2737063309276740AABA25/9781316510087c2_4-22.pdf/some_background_on_ordinary_differential_equations.pdf` (visited on 03/29/2024)

[7] Simo Särkkä and Arno Solin. "Pragmatic Introduction to Stochastic Differential Equations" In: *Applied Stochastic Differential Equations*, pp.23–41. Cambridge University Press (2019). ISBN: 9781108186735. DOI: 10.1017/9781108186735. URL:

https://www.cambridge.org/core/services/aop-cambridge-core/content/view/3C7E2790CCC56A4B61132706EB01F296/9781316510087c3_23-41.pdf/pragmatic_introduction_to_stochastic_differential_equations.pdf (visited on 03/30/2024)

[8] Simo Särkkä and Arno Solin. "Itô Calculus and Stochastic Differential Equations" In: *Applied Stochastic Differential Equations*, pp.42–58. Cambridge University Press (2019). ISBN: 9781108186735. DOI: 10.1017/9781108186735. URL: https://www.cambridge.org/core/services/aop-cambridge-core/content/view/C6BDDB0A7AE521BDE01A4EBFEAC91BCF/9781316510087c4_42-58.pdf/ito_calculus_and_stochastic_differential_equations.pdf (visited on 03/30/2024)

[9] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, Surya Ganguli. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *International Conference on Machine Learning*, PMLR (2015), pp. 2256–2265.

[10] Yang Song, Stefano Ermon. "Generative modeling by estimating gradients of the data distribution". In: *Advances in Neural Information Processing Systems*, (2019), pp. 11895–11907.

[11] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, Ben Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*, (2021). DOI:.

[12] Yang Song. "score_sde: Score-Based Generative Modeling through Stochastic Differential Equations". In: *GitHub*, (2021). URL: https://github.com/yang-song/score_sde (visited on 23/05/2024).

[13] Yang Song, Diederik P. Kingma. "How to train your energy-based models". In: *arXiv preprint arXiv:2101.03288*, (2021).

[14] Michael E. Tipping, Christopher M. Bishop. "Probabilistic principal component analysis". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, (1999), pp. 611–622.

[15] Jakub M. Tomczak. "Flow-Based Models" In: *Deep Generative Modeling*, pp.27-56. Springer (2021). ISBN: 978-3-030-93158-2. DOI: 10.1007/978-3-030-93158-2_3.

[16] Jakub M. Tomczak. "Latent Variable Models" In: *Deep Generative Modeling*, pp.57-127. Springer (2021). ISBN: 978-3-030-93158-2. DOI: 10.1007/978-3-030-93158-2_4.

[17] Xingcheng Xu. "Deep Generative Modeling with Backward Stochastic Differential Equations". In: *arXiv:2304.04049*, (2023).

[18] Xingcheng Xu. "BSDE-Gen: Backward Stochastic Differential Equations for Generative Models", In: *GitHub*, URL: `https://github.com/xingchengxu/BSDE-Gen` (visited on 24/05/2024).