Master thesis on Mathematical Sciences

Utrecht University

Stock Price Simulation under Jump-Diffusion Dynamics: A WGAN-Based Framework with Anomaly Detection

Renren Gan

Supervisor: Prof. dr. ir. C.W. Oosterlee

Dr. ir. L.A. Grzelak

N.T. Mücke

January 2023

Universiteit Utrecht

Acknowledgement

I would like to express my sincere gratitude to my supervisors for their support and helps throughout the research project. First, I thank Prof. C.W. Oosterlee for his weekly instruction, pointing out the right direction of the research. Next, I thank N.T. Mücke for his practical help. I also thank Dr. L.A. Grzelak for his reviews.

I would also like to thank my family and my cat for their support and accompany.

Abstract

Jump-diffusion path simulation is a popular topic in the finance area. In numerical path simulation, it is usually split into a diffusion part and a jump part. We propose a GAN-based framework that gives a general pattern for the jump-diffusion path simulation. The framework consists of two parts: 1) the diffusion learning part for the simulation of the diffusion part; 2) the jump detection part related to the jump simulation. The diffusion simulation is achieved by a conditional Wasserstein GAN with gradient penalty (called SDE-WGAN). The SDE-WGAN is an adapted model from a GAN-based SDEs simulation methodology, showing its advantages in stable training. The jump detection model is designed to detect the jump instances in a jump-diffusion path and estimate the jump parameters. The jump instances are detected by introducing a GAN-based anomaly detection method, as the jumps can be viewed as anomalies that are inconsistent with the non-jump data and rare to occur in the real market. The SDE-WGAN is well-combined since it can only generate non-jump states. The jumps are then recognized when the SDE-WGAN generated pattern significantly differs from the actual state. The maximum likelihood estimation is then applied to approximate the jump parameters based on the detected jump instances. We perform the proposed framework for simulating the Merton's model and obtain promising results. However, the framework may fail when the jump magnitude is small.

Contents

Chapter 1

Introduction

Financial market participants, such as investors, are interested in asset price simulation, which is a popular topic in the financial area. Simulation in this context, usually related to forecasting or prediction, refers to generating asset price paths in the future to show the possibility or the trend of future markets. In this thesis, we stand from the perspective of investors and focus on the basic asset class, i.e., stocks.

Return, risk and liquidity are three basic concepts in portfolio theory [1], illustrating the so-called magic triangle of investment. Investors would like the highest possible return with acceptable risk, while the risk is usually proportional to the return and inversely proportional to the asset liquidity. Regarding stocks, those listed on major exchanges are generally good at liquidity. However, the entire market may collapse during a crisis, for example, when the black swan events occur. Therefore, the simulation is helpful for investors to make crucial decisions.

In the financial area, stock prices are typically modeled as stochastic processes, in particular, Markov stochastic processes: The future stock price is a random variable indexed by time; Moreover, financial researchers assume that for the future stock price, the current price is only relevant, while not related to the past prices, following a so-called Markov property. The dynamics or the behavior of the stock prices are usually characterized by stochastic differential equations (SDEs), where a Wiener process or a standard Brownian motion is involved. One of the most common stochastic models for the behavior of stock prices is the geometric Brownian motion (GBM) process.

The GBM model forms the basis of the classic Black-Scholes pricing model [2] proposed in 1973. However, it is still popular for researchers nowadays and is still used for real-market simulation [3; 4]. Even though the GBM model is basic and widely used, it is not realistic enough. In empirical studies, the log-returns of the historical stock prices display a distribution which has a high peak and asymmetric heavy tails [5], while the GBM model is not able to present this feature. Regarding the so-called leptokurtic feature, many alternative models have been proposed. Among them, the jump-diffusion model is appealing since it provides a good explanation, by definition, for the jump patterns exhibited by stocks [6].

The jump-diffusion model is derived from the GBM model. It has an additional term driven by a compound Poisson process aiming to model the stock price's discontinuous behavior (i.e., the jump). The mathematical formula for the corresponding dynamics is also called an SDE with jumps [7]. The jump-diffusion model is a simple extension of the GBM model, which is not only computationally friendly but also shows better performance to stocks than the GBM model [8]. Except that, empirical tests show that the jump-diffusion processes can reproduce the leptokurtic feature of the stock returns and the volatility smile in option prices [9; 10].

In practice, the above stochastic models are often simulated via the Monte Carlo approach. The Monte Carlo simulation is an essential tool in finance, especially in option pricing and risk management [11]. It is a random sampling method based on probability theory: By repeatedly sampling the random variables in the system, we can obtain a range or a distribution of the outputs; Based on the law of large numbers, the expectation of the outputs converges to the actual mean, and from the central limit theorem, the error information is provided [12].

When simulating the price paths following the stochastic models, solutions of the corresponding SDEs are approximated, as most SDEs do not have closed-form solutions. Investors often resort to numerical approximations and time-discretization methods are usually used to approximate the continuous-time dynamics. The Euler discretization scheme [13] perhaps is the simplest approximation, and other discrete-time schemes, such as the Milstein scheme, are proposed to improve the accuracy of the approximation [14]. The Monte-Carlo simulation then plays a role because of the involved random increments.

To improve the estimation accuracy for the stochastic integrals, [15] proposes a new direction for the approximation instead of applying higher-order Taylor expansions or other discretization methods. The authors of [15] use a neural network, especially a generative adversarial network (GAN), to learn the relation between the two adjacent prices. Such methodology is tested successfully on the GBM model. Compared to non-machine learning methods, the neural network approach gives a general pattern for the stochastic model simulation and it can theoretically be applied to simulate all kinds of stochastic processes with the same precision. Moreover, the machine learning-based method shows its advantages, especially when dealing with high-dimensional problems and facing the curse of dimensionality [16].

This thesis will use the jump-diffusion model to simulate stock paths. With the investment background, we focus on the log-price dynamics under $\mathbb{P}$-measure, that is, the real-world measure. Usually, the simulation can be divided into two aspects: 1) parameter estimation and 2) path simulation (see Figure 1.1). Since the jump-diffusion process is an addition of a diffusion component (i.e., the part derived from the GBM model) and a jump component (i.e., the part driven by the compound Poisson process), we simulate the paths by generating the two components, respectively. When the model parameters are estimated from the empirical market, we generate the diffusion component following a usual log-GBM model. As for the jump component, the time instances for jumps and the jump magnitudes are sampled respectively [17; 18].

Considering the benefits of the neural network approach, we propose a GAN-based Monte Carlo simulation framework (see Figure 1.2). The details of each component are

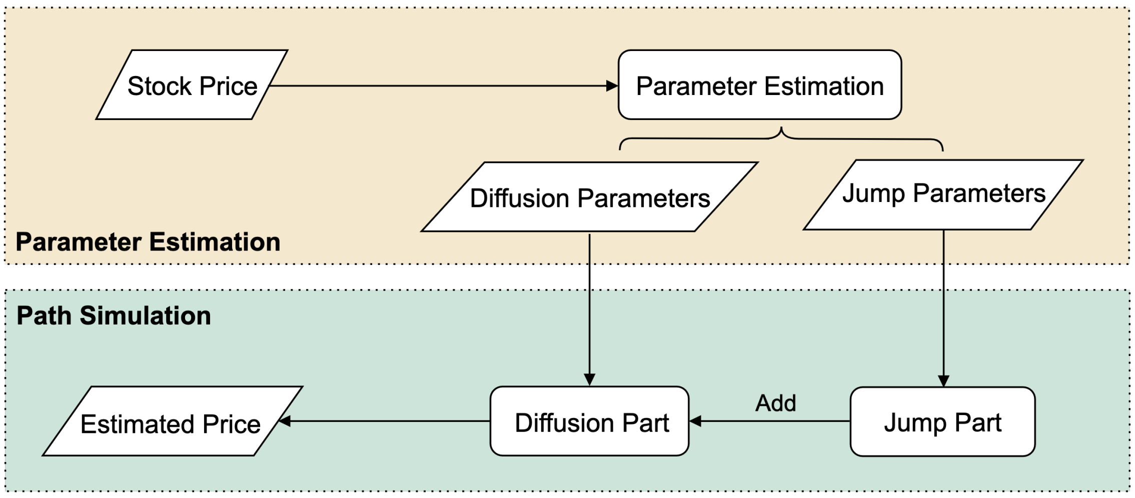


Figure 1.1: The simple framework for jump-diffusion model simulation.

demonstrated as follows:

- **Path simulation:**

  Following the general approach above, we separate the path simulation into diffusion and jump parts.

  - **The diffusion part:**

    An improved GAN architecture called SDE-WGAN will be used to simulate the diffusion component. It is essentially a condition GAN using the Wasserstein loss with an addition constrain so-called gradient penalty on the critic[1] loss. Contrary to the conditional GAN architecture in [15], SDE-WGAN is significantly stable.

  - **The jump part:**

    Instead of using Poisson random variables to sample the jump instances in the time horizon [17; 11], we generate independent Bernoulli random variables at each timestamp to determine the jump occurrences [18]. The jump magnitudes or the jump sizes are governed by some distribution, such as a normal distribution in Merton's model [19] and a double exponential distribution in Kou's model [9]. The jump magnitudes will influence the prices when the Bernoulli variable is equal to 1.

- **Parameter estimation:**

  Many statistic parameter estimation methods can be applied for estimating parameters of the jump-diffusion model. For example, [20; 8] use the maximum likelihood estimation (MLE) method, as the corresponding density function can be derived. Since the proposed SDE-WGAN can directly simulate the diffusion

[1]In Wasserstein GAN, the discriminator is often called a critic.

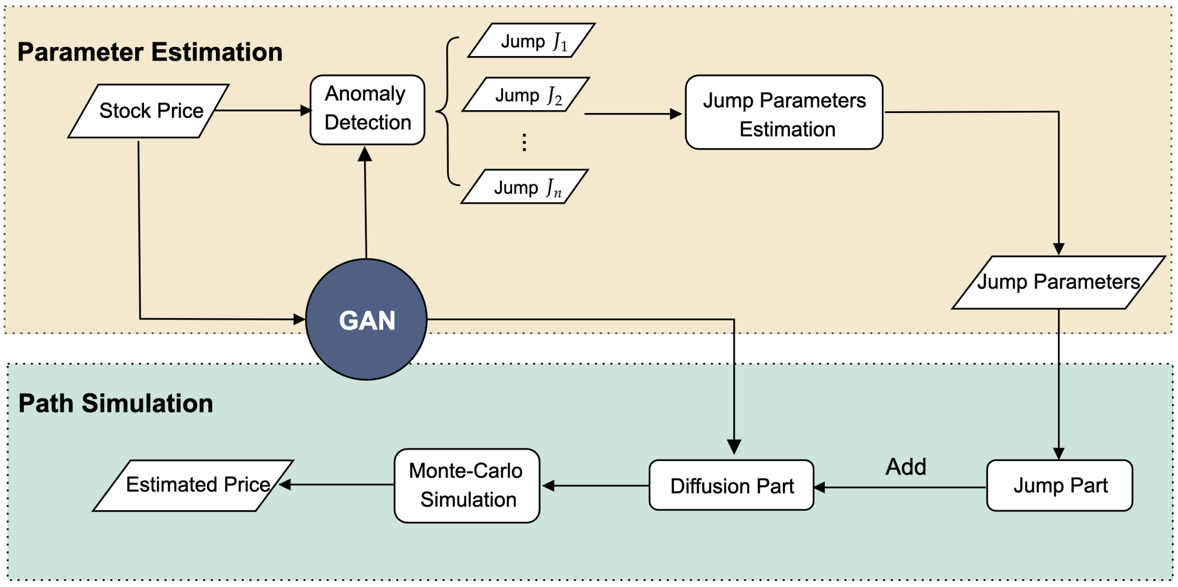


Figure 1.2: The proposed GAN-based framework for jump-diffusion model simulation.

part based on the adjacent prices, in this thesis, we only need to focus on the estimation for the jump parameters, that is, the jump intensity and the jump magnitude. Moreover, we introduce an anomaly detection technology when dealing with the jump part.

As crashes and large valleys are rare to happen [21] in the real market, jumps can therefore be viewed as anomalies. The prices not influenced by the jump part are the normal data. We can then apply anomaly detection methods to detect the jumps in the price path. With the benefit of the well-trained GAN in the diffusion simulation, the normal pattern is already learned. It is then natural to introduce a GAN-based anomaly detection method [22], where the GAN is actually the same. Based on the detected jump instances, we can then estimate the parameters of the jumps.

Research on jump-diffusion model simulation always retains its appeal, and our main contribution is proposing a general simulation framework. The GAN-based framework can work on the jump-diffusion process and easily adapt to other complex stochastic models. Moreover, the introduced anomaly detection method gives a new view on jump detection, which may be extended to the Hawkes jumps [23]. On the other hand, it is also another example of applying machine learning-based methods in finance.

1.1 Research Question

This thesis aims to examine the possibility of using GANs to simulate jump-diffusion processes, and a GAN-based framework is proposed: First, we improve the conditional GAN in [15] to simulate the diffusion part; Next, we introduce a GAN-based anomaly detection to recognize the jumps, the jump part is then simulated based on jump parameters estimated from the detected jumps.

In brief, the main research question of this work is "Can we use GANs and anomaly detection technology to simulate a jump-diffusion process?"

The detailed sub-questions are listed as follows:

- Given the necessary model parameters, can we use a GAN to simulate jump-diffusion paths?
- Given a path following the jump-diffusion dynamics, can we use the well-trained GAN to detect the jumps?
- How can we estimate the jump parameters as accurate as possible from the detected jumps?
- The Robustness and sensitivity of the proposed framework. For example, when the model parameters change, is the GAN still able to simulate jump-diffusion paths? What jump size can be detected?

1.2 Thesis Outline

The thesis is structured as follows: In Chapter 2, we describe some related backgrounds or preliminaries, including the stochastic processes to simulate, the numerical Monte Carlo simulation methods, the artificial neural networks and the anomaly detection methods. In Chapter 3, the generative adversarial networks are illustrated thoroughly. We present two GAN applications that are associated with the proposed framework. Chapter 3 ends with the GAN training problems. In Chapter 4, we discuss the improved GAN model-Wasserstein GAN and Wasserstein GAN with gradient penalty, which can effectively prevent the GANs failure modes because of the introduced Wasserstein loss. Chapter 5 explains the implementation of the proposed framework part by part, where the Merton's model is taken as an example. Experiments of the proposed framework on the artificial dataset are illustrated in Chapter 6, and we also examine the robustness of the framework. Chapter 7 discusses our choices and outlooks regarding the proposed framework. Finally, we make a brief conclusion.

Chapter 2

Preliminaries

This chapter presents the basic knowledge required in our proposed simulation framework (Figure 1.2). The contents are structured as follows. First, we discuss stochastic processes, which are the goal of the simulation. Next, the Monte Carlo path simulation is described. It is a simulation method we introduce to generate stochastic paths. After that, artificial neural networks are illustrated, which are the essential techniques in the framework. In the final part, anomaly detection is briefly explained, relating to our proposed creative idea.

2.1 Stochastic Processes in Finance

Stock prices are often modelled as a stochastic process, and their dynamics are usually described as stochastic differential equations (SDEs). A stochastic process is a collection of random variables $\{S(t), t \in T\}$ depending on time t, defined on a probability space $(\Omega, \mathcal{F}, P)$, where Ω is a sample space, $\mathcal{F}$ is a σ-field and P is a probability measure. It can also be viewed as a function of two variables $S(t) = S(t, \omega)$, with $t \in T$ and $\omega \in \Omega$: For a fixed $t \in T$, the function $S(t, \cdot)$ is measurable with respect to $\mathcal{F}(t)$; For some fixed $\omega \in \Omega$, the function $S(\cdot, \omega) : T \mapsto \mathbb{R}$ is called a trajectory or a path of the process [24]. $\{S(t), t \in T\}$ is said to be adapted to the filtration $\mathcal{F}(t)$, if $\sigma(S(t)) \subseteq \mathcal{F}(t)$.

In this section, which is heavily inspired by [25], two basic stochastic models are described in detail, related to two fundamental stochastic processes called Wiener process and Poisson process.

Definition 2.1.1 (Wiener process). *A Wiener process, $W(t)$, which is also called a standard Brownian motion, is a continuous-time stochastic process characterized by the following properties:*

- $W(0) = 0$,
- $W(t)$ *is almost surely continuous,*
- $W(t)$ *has independent Gaussian increments, i.e.* $\forall\, 0 < t_1 \leq t_2 \leq t_3 \leq t_4$, $W(t_2) - W(t_1) \perp\!\!\!\perp W(t_4) - W(t_3)$, *with distribution* $W(t) - W(s) \sim \mathcal{N}(0, t-s)$[1] *for* $0 \leq s < t$.

[1] $\mathcal{N}(\mu, \sigma^2)$ denotes a normal distribution with expectation μ and variance σ^2.

Definition 2.1.2 (Poisson process). *A stochastic process $\{X_{\mathcal{P}}(t), t \geq t_0 = 0\}$, with parameter $\lambda_p > 0$ is called a Poisson process, if:*

- $X_{\mathcal{P}}(0) = 0$;
- $\forall t_0 = 0 < t_1 < \cdots < t_n$, *the increments* $X_{\mathcal{P}}(t_1) - X_{\mathcal{P}}(t_0), \ldots, X_{\mathcal{P}}(t_n) - X_{\mathcal{P}}(t_{n-1})$ *are independent random variables;*
- *For* $s \geq 0$, $t > 0$ *and integers* $k \geq 0$, *the increments are integer-valued and have the Poisson distribution*

$$\mathbb{P}[X_{\mathcal{P}}(s+t) - X_{\mathcal{P}}(s) = k] = \frac{(\lambda_p t)^k e^{-\lambda_p t}}{k!}. \tag{2.1.1}$$

Proposition 2.1.3 (Poisson process). *The Poisson process in Definition 2.1.2 has the following properties:*

1) $\{X_{\mathcal{P}}(t), t \geq 0\}$ *is right-continuous and nondecreasing;*
2) $\mathbb{E}[X_{\mathcal{P}}(t)] = \lambda_p t$, $\mathbb{V}ar[X_{\mathcal{P}}(t)] = \lambda_p t$.

Proof. 1) is obvious from the definition. We then give the proof of 2).

For a time interval $\mathrm{d}t > 0$, the increment $\mathrm{d}X_{\mathcal{P}}(t) = X_{\mathcal{P}}(s + \mathrm{d}t) - X_{\mathcal{P}}(s)$ is a Poisson random variable with parameter $\lambda_p \mathrm{d}t$. The probability generating function of $\mathrm{d}X_{\mathcal{P}}(t)$ is

$$\begin{aligned} G(z) = \mathbb{E}[z^k] &= \sum_{k=0}^{\infty} z^k \mathbb{P}[\mathrm{d}X_{\mathcal{P}}(t)] \\ &= \sum_{k=0}^{\infty} z^k \frac{(\lambda_p \mathrm{d}t)^k e^{-\lambda_p \mathrm{d}t}}{k!} = e^{-\lambda_p (1-z)\mathrm{d}t}. \end{aligned}$$

Therefore,

$$\begin{aligned} \mathbb{E}[\mathrm{d}X_{\mathcal{P}}(t)] &= G^{'}(1^-) = \lambda_p \mathrm{d}t, \\ \mathbb{V}\mathrm{ar}[\mathrm{d}X_{\mathcal{P}}(t)] &= G^{''}(1^-) + G^{'}(1^-) - (G^{'}(1^-))^2 = \lambda_p \mathrm{d}t. \end{aligned}$$

With $X_{\mathcal{P}}(0) = 0$, $\mathbb{E}[X_{\mathcal{P}}(t)] = \mathbb{V}\mathrm{ar}[X_{\mathcal{P}}(t)] = \lambda_p t$. □

Example 2.1.4 (Paths of a Wiener process and a Poisson process). *We present some discrete paths that are generated by a Wiener process and a Poisson process respectively (see Figure 2.1).*

One reason for the two stochastic processes being appealing could be the Markov property, which is extremely important in modelling asset price movements. The Markov property states that the future behaviour of the process is independent of its past when the present state is given [26], that is, the process has no "memory".

Definition 2.1.5 (Markov property (simple version)). *A stochastic process $\{S(t), t \in T\}$ possesses the Markov property, if for any $s > 0$, the conditional distribution $S(t+s)$ given $\mathcal{F}(t) := \mathcal{F}_t$ is the same as the conditional distribution of $S(t+s)$ given $S(t)$, that is, $\forall y \in \mathbb{R}$,*

$$\mathbb{P}(S(t+s) \leq y | \mathcal{F}_t) = \mathbb{P}(S(t+s) \leq y | S(t)), \quad a.s. \tag{2.1.2}$$

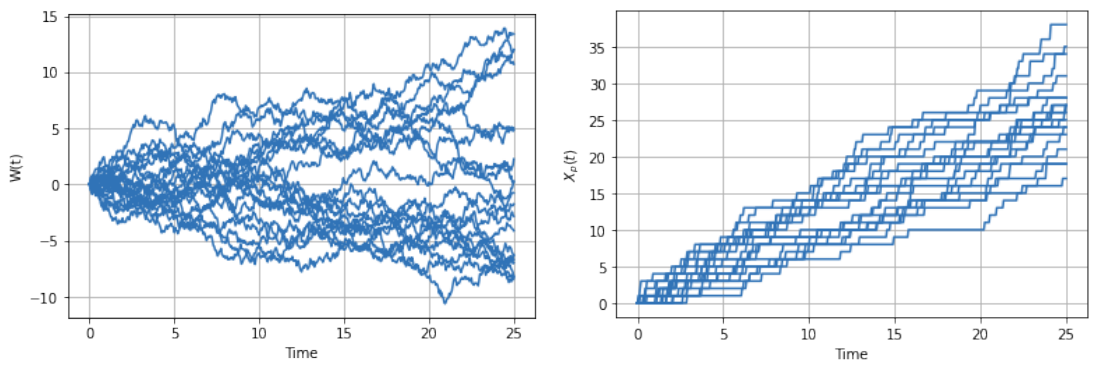


Figure 2.1: Discrete paths for a Wiener process (left) and a Poisson process (right), with $\Delta t = 0.05$ and $\lambda_p = 1$.

Theorem 2.1.6. *We state the Markov property of Wiener and Poisson processes:*

1. *Wiener process $W(t)$ has Markov property.*
2. *Poisson process $X_p(t)$ has Markov property.*

Proof. See Appendix A.2 □

2.1.1 Geometric Brownian Motion

The geometric Brownian motion (GBM) is perhaps the most common model for simulating stock movements in finance, and Figure 2.2 shows a realization of a GBM process. A geometric Brownian motion $S(t)$ is a continuous-time stochastic process which has the form:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t), \tag{2.1.3}$$

where μ, σ are constants, and $W(t)$ is a Wiener process adapted to the filtration $\mathcal{F}(t)$.

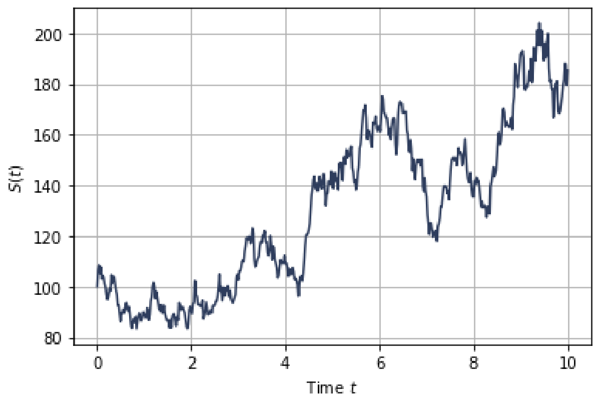


Figure 2.2: A realization of a GBM process, with $S_0 = 100, \mu = 0.05, \sigma = 0.2, \Delta t = 0.02$
.

The stock prices $\{S(t), t \geq 0\}$ are said to follow a geometric Brownian motion (GBM) process, if the following dynamics are satisfied:

$$\mathrm{d}S(t) = \mu S(t)\mathrm{d}t + \sigma S(t)\mathrm{d}W^{\mathbb{P}}(t), \text{ with } S(0) = S_0, \tag{2.1.4}$$

where the adapted Wiener process $W^{\mathbb{P}}(t)$ is under the real-world measure $\mathbb{P}$.

The corresponding integral formula is given by:

$$S(t) = S_0 + \int_0^t \mu S(u)\mathrm{d}u + \int_0^t \sigma S(u)\mathrm{d}W^{\mathbb{P}}(u) \tag{2.1.5}$$

Note that the equations (2.1.4) and (2.1.5) are of Itô's form and the integration here is handled by Itô's calculus. Itô's lemma is a cornerstone in stochastic processes that enables people dealing with the nowhere-differentiable Wiener process [25]. To present Itô's Lemma, we first give a relevant definition so-called Itô process.

Definition 2.1.7 (Itô process). *An Itô process $X(t)$ is defined to be an adapted stochastic process, whose corresponding SDE is given by:*

$$dX(t) = \bar{\mu}(t, X(t))dt + \bar{\sigma}(t, X(t))dW(t), \textit{with } X(0) = X_0, \tag{2.1.6}$$

where $W(t)$ is a Wiener process, and two general functions $\bar{\mu}(t,x)$ and $\bar{\sigma}(t,x)$ satisfy the following Lipschitz conditions:

$$|\bar{\mu}(t,x) - \bar{\mu}(t,y)|^2 + |\bar{\sigma}(t,x) - \bar{\sigma}(t,y)|^2 \le K_1|x-y|^2,$$
$$|\bar{\mu}(t,x)|^2 + |\bar{\sigma}(t,x)|^2 \le K_2(1+|x|^2),$$

for some constants K_1, $K_2 \in \mathbb{R}^+$ and $x, y \in \mathbb{R}$.

Lemma 2.1.8 (Itô's Lemma). *Suppose $X(t)$ is an Itô process defined by* (2.1.6). *Let $g(t,X)$ be a function of $X = X(t)$ and time t, with continuous partial derivatives, $\frac{\partial g}{\partial X}$, $\frac{\partial^2 g}{\partial X^2}$, $\frac{\partial g}{\partial t}$. A stochastic variable $Y(t) := g(t,X)$ then also follows an Itô process that is governed by the same Wiener process $W(t)$, i.e.*

$$dY(t) = (\frac{\partial g}{\partial t} + \bar{\mu}(t,X(t))\frac{\partial g}{\partial X} + \frac{1}{2}\bar{\sigma}^2(t,X(t))\frac{\partial^2 g}{\partial X^2})dt + \bar{\sigma}(t,X(t))\frac{\partial g}{\partial X}dW(t). \tag{2.1.7}$$

Proof. An intuitive proof is derived by applying the 2D Taylor series expansion, see Appendix A.2. □

By applying Itô's Lemma and using log-transformation, we can show the distribution of $S(t)$ in (2.1.4), which is stated as follows.

Remark 2.1.9 (Itô multiplication table). *The product rule of the terms dt and dW forms a so-called Itô multiplication table, shown in Table 2.1.*

With Itô's lemma, excellent results can be derived. In the following part, we show the distributions of $S(t)$ and $\log S(t)$ by Itô's lemma.

Table 2.1: Itô multiplication table for Wiener process.

	$\mathrm{d}t$	$\mathrm{d}W(t)$
$\mathrm{d}t$	0	0
$\mathrm{d}W(t)$	0	$\mathrm{d}t$

Proposition 2.1.10 (Lognormal distribution)**.** *The random variable $S := S(t)$ in* (2.1.4) *is from a lognomal distribution, i.e.* $\log S$ *is normally distributed* [2].

Proof. Let $X(t) = g(t, S) = \log S$, then $\frac{\partial g}{\partial S} = \frac{1}{S}$, $\frac{\partial^2 g}{\partial S^2} = -\frac{1}{S^2}$ and $\frac{\partial g}{\partial t} = 0$. By Itô's Lemma, we have

$$\begin{aligned} \mathrm{d}X(t) &= (\mu \cancel{S}\frac{1}{\cancel{S}} - \frac{1}{2}\sigma^2 \cancel{S^2}\frac{1}{\cancel{S^2}})\mathrm{d}t + \sigma \cancel{S}\frac{1}{\cancel{S}}\mathrm{d}W^{\mathbb{P}}(t) \\ &= (\mu - \frac{1}{2}\sigma^2)\mathrm{d}t + \sigma \mathrm{d}W^{\mathbb{P}}(t), \text{ with } X_0 = \log S_0. \end{aligned} \tag{2.1.8}$$

From Definition 2.1.1, the Wiener increment $\mathrm{d}W^{\mathbb{P}}(t)$ is normally distributed, that is, $\mathrm{d}W^{\mathbb{P}}(t) \sim \mathcal{N}(0, \mathrm{d}t)$. Therefore, $\mathrm{d}X(t)$ is normally distributed, with expectation $(\mu - \frac{1}{2}\sigma^2)\mathrm{d}t$ and variance $\sigma^2\mathrm{d}t$. □

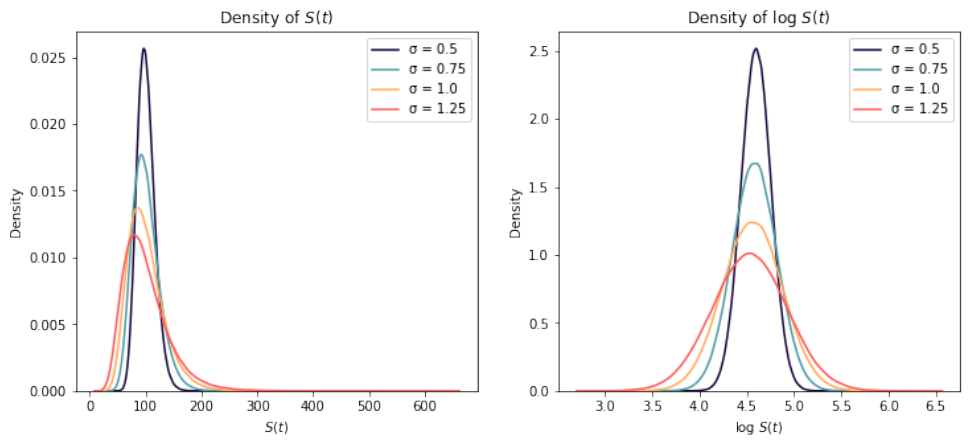


Figure 2.3: Density of $S(t)$ and $\log S(t)$ in (2.1.4) for varying σ, with $\mu = 0.05$.

Figure 2.3 illustrates the distributions of $S(t)$ and $\log S(t)$, showing Proposition 2.1.10 intuitively.

On the other hand, the integral formulation of (2.1.8) from time 0 to t ($t \in \mathbb{R}^+ \cup \{0\}$) is given by:

$$\int_0^t \mathrm{d}X(u) = \int_0^t (\mu - \frac{1}{2}\sigma^2)\mathrm{d}u + \int_0^t \sigma \mathrm{d}W^{\mathbb{P}}(u), \tag{2.1.9}$$

from which the solution is found to be:

$$X(t) = X_0 + (\mu - \frac{1}{2}\sigma^2)t + \sigma W^{\mathbb{P}}(t). \tag{2.1.10}$$

[2] Here, log denotes the natural logarithm ln.

We can therefore obtain the solution for $S(t)$ in (2.1.4), as

$$S(t) = S_0 \exp((\mu - \frac{1}{2}\sigma^2)t + \sigma W^{\mathbb{P}}(t)). \tag{2.1.11}$$

2.1.2 Is GBM Realistic Enough: Towards to Jump Processes

The GBM model is simple, widely applicable and remains attractive to the recent researchers [27; 3]. However, the model itself is not completely realistic, which makes the simulation inconsistent with the market behaviour. For example, the stock prices often show discontinuous behavior, while the GBM path is continuous. In order to build a more realistic model, researchers make extensions based on the GBM model. Among them, a so-called jump-diffusion process allows stock prices to jump by adding an extra discrete-state component [28]. [8] performs empirical tests showing that the jump-diffusion model fits stock data better than the GBM model.

There are several reasons for introducing jumps into stochastic processes. First of all, stock prices do jump in real life, which are caused by unpredictable events (e.g. the so-called black swan events). Second, the leptokurtic feature, i.e. a feature of a high peak and two heavy tails compared to the normal distribution, is evidently observed when illustrating the histogram of log returns $R(t) = \log \frac{S(t)}{S(t-1)}$. [9] shows that with the introduced jumps, the model is able to reproduce the leptokurtic feature. The rigorous description of the leptokurtic feature is given as follows:

Definition 2.1.11 (Leptokurtic Distribution)**.** *Let X be a random variable, the kurtosis of X is defined as $K = \mathbb{E}[(\frac{X-\mu}{\sigma})^4]$, where μ is the mean and σ is the standard deviation. Leptokurtic distributions are statistical distributions with $K > 3$.*

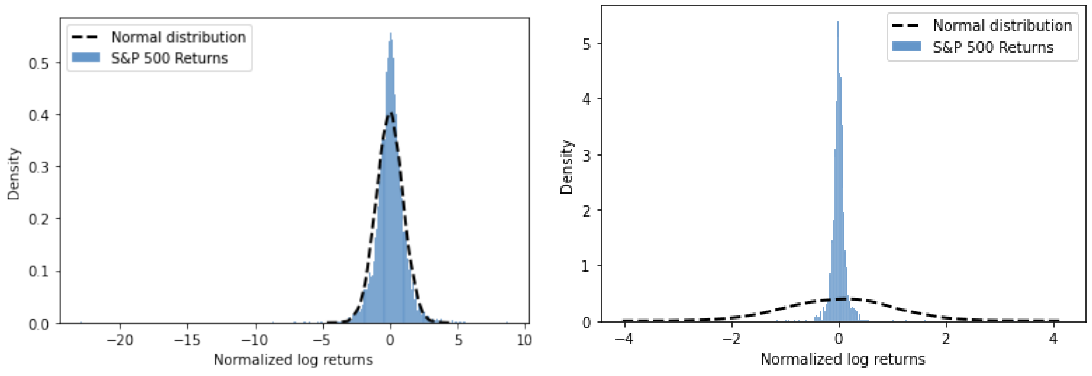


Figure 2.4: The histograms of the normalized log returns of S&P 500 index compared with the standard normal distribution $\mathcal{N}(0, 1)$ (Left: daily returns from Jan 2, 1980 to Dec 31, 2005; Right: 5-mimute returns from Nov 1, 2022 to Nov 30, 2022).

Example 2.1.12 (The leptokurtic feature of S&P 500 index data)**.** *Figure 2.4 displays the histograms of the normalized S&P 500 log returns for the long-term and intraday prices, with the sample kurtoses about 42.20 and 121.65 respectively.*

Jump-diffusion processes form a special class in the general Lévy processes, from which only finite jumps can occur in any finite time period. Except applying Poisson and Lévy processes to drive jumps, other jump processes such as Hawkes processes are attractive nowadays to better replica to the market jumps. Figure 2.5 illustrates two main families of jump processes: exponential Lévy process and Hawkes process. The Exponential Lévy process deals with independent jumps, while the Hawkes process can explain the clustering of jumps (or the contagious behaviour) [23; 29]. In this thesis, we simply focus on the basic jump-diffusion processes, particularly the Merton jump-diffusion model.

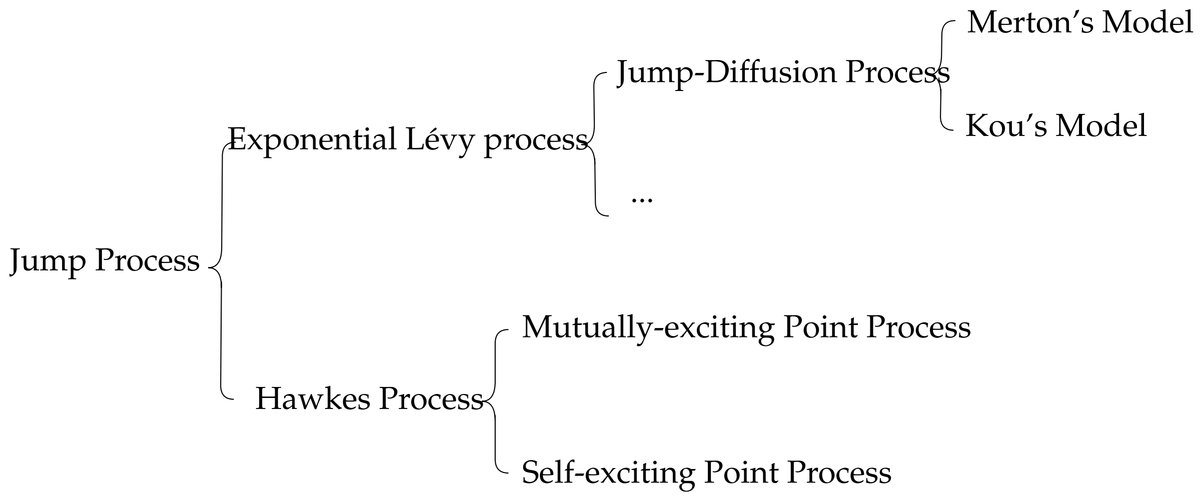


Figure 2.5: Family of jump processes.

2.1.3 Jump-Diffusion Process

Considering the reproduction of the stock price discontinuities, jump-diffusion processes are the simplest extensions based on the GBM process, from which Poisson processes are introduced for generating the jumps. In general, a jump diffusion process consists of two components: a geometric Brownian motion term called a diffusion part, and a compound Poisson process term called a jump part [30].

We first show the dynamics of jump-diffusion processes for the log-stock prices $X(t) = \log S(t)$, $t \geq 0$, which is given by:

$$\mathrm{d}X(t) = \mu \mathrm{d}t + \sigma \mathrm{d}W^{\mathbb{P}}(t) + J\mathrm{d}X_{\mathcal{P}}(t), \text{ with } X(0) = \log S(0), \tag{2.1.12}$$

where μ, σ are constants, meaning the drift and volatility respectively; $W^{\mathbb{P}}(t)$ is a Wiener process under $\mathbb{P}$-measure; J follows a distribution F_J, giving the jump magnitude; and $X_{\mathcal{P}}(t)$ is a Poisson process with parameter $\lambda_p > 0$. Note that $W^{\mathbb{P}}(t)$ and $X_{\mathcal{P}}(t)$ are usually considered independent.

The stochastic integral with respect to (2.1.12) is defined by:

$$\begin{aligned} X(t) - X(0) &= \int_0^t \mu \mathrm{d}u + \int_0^t \sigma \mathrm{d}W^{\mathbb{P}}(u) + \int_0^t J\mathrm{d}X_{\mathcal{P}}(u) \\ &:= \mu t + \sigma W^{\mathbb{P}}(t) + \sum_{k=1}^{X_{\mathcal{P}}(t)} J_k, \end{aligned} \tag{2.1.13}$$

where $\{J_k, k \geq 1\}$ is a sequence of independent and identically distributed random variables with distribution F_J. Here, the part in (2.1.13) resulting in jumps is so-called a compound Poisson process, given by $\left\{ \bar{X}(t) = \sum_{k=1}^{X_{\mathcal{P}}(t)} J_k, t \geq 0 \right\}$.

Similar to the derivation of the dynamics of log-GBM process, the dynamics of the original stock price $S(t) = e^{X(t)}$ can also be derived via Itô's lemma. The Itô's differential formula for the discrete-state Poisson process is illustrated as follows.

Corollary 2.1.13 (Itô's lemma for Poisson process). *Consider a stochastic process $X(t)$ which is right-continuous and has left-limit everywhere, defined as*

$$dX(t) = \bar{\mu}(t, X(t))dt + \bar{J}(t, X(t))dX_{\mathcal{P}}(t), \text{ with } X(0) \in \mathbb{R}, \tag{2.1.14}$$

where $\bar{\mu}, \bar{J} : [0, \infty] \times \mathbb{R} \to \mathbb{R}$ are deterministic, continuous functions and $X_{\mathcal{P}}(t)$ is a Poisson process starting at $t = 0$.

The Itô differential of a differentiable function $g : [0, \infty] \times \mathbb{R} \to \mathbb{R}$ is given by

$$dg(t, X(t)) = \left[\frac{\partial g}{\partial t} + \bar{\mu}(t, X(t)) \frac{\partial g}{\partial X} \right] dt + \left[g(t, X(t\_) + \bar{J}(t, X(t\_))) - g(t, X(t\_)) \right] dX_{\mathcal{P}}(t), \tag{2.1.15}$$

where $X(t\_) := \lim_{s \to t, s<t} X(s)$ denotes the left limit, and $\lim_{s \to t, s<t} \bar{J}(s, X(s)) = \bar{J}(t, X(t\_))$ because of the continuity of $\bar{J}$.

Proof. See Appendix A.2. $\square$

Furthermore, with the assumption of the independence of the Wiener process $W(t)$ and the Poisson process $X_{\mathcal{P}}(t)$, the dynamics of (2.1.14) with an additional Wiener term $\bar{\sigma}(t, X(t))\mathrm{d}W(t)$ is given by

$$\begin{aligned} \mathrm{d}g(t, X(t)) = & \left[\frac{\partial g}{\partial t} + \bar{\mu}(t, X(t)) \frac{\partial g}{\partial X} + \frac{1}{2} \bar{\sigma}^2 \frac{\partial^2 g}{\partial X^2} \right] \mathrm{d}t \\ & + \bar{\sigma} \frac{\partial g}{\partial X} \mathrm{d}W(t) + \left[g(t, X(t\_) + \bar{J}(t, X(t\_))) - g(t, X(t\_)) \right] \mathrm{d}X_{\mathcal{P}}(t) \end{aligned} \tag{2.1.16}$$

Applying Itô's lemma to the function $S(t) = e^{X(t)}$, with $X(t)$ in (2.1.12), we have

$$\mathrm{d}e^{X(t)} = \left[\mu e^{X(t)} + \frac{1}{2} \sigma^2 e^{X(t)} \right] \mathrm{d}t + \sigma e^{X(t)} \mathrm{d}W^{\mathbb{P}}(t) + \left[e^{X(t)+J} - e^{X(t)} \right] \mathrm{d}X_{\mathcal{P}}(t).$$

Therefore, the jump-diffusion dynamics of the stock price $S(t)$ under $\mathbb{P}$-measure is give by:

$$\frac{\mathrm{d}S(t)}{S(t)} = \left(\mu + \frac{1}{2} \sigma^2 \right) \mathrm{d}t + \sigma \mathrm{d}W^{\mathbb{P}}(t) + \left(e^J - 1 \right) \mathrm{d}X_{\mathcal{P}}(t). \tag{2.1.17}$$

The jump-diffusion processes given in (2.1.12) can be further subdivided into a number of models according to the distribution of the jump magnitude J. The Merton jump-diffusion (MJD) model [19] and the Kou jump-diffusion (KJD) model [9] are two popular choices. In the MJD model, J follows a normal distribution, while in the KJD model, J has an asymmetric double exponential distribution. In empirical practice, [8] points out the KJD model performs better than the MJD model for stocks.

Definition 2.1.14 (The Merton jump-diffusion model). *In the classical Merton's model, the dynamics for the log-stock prices $X(t)$ is given in* (2.1.12)*, where the jump magnitude variable J is normally distributed, that is, $J \sim \mathcal{N}(\mu_J, \sigma_J^2)$. Therefore, $dF_J(x) = f_J(x)dx$, where*

$$f_J(x) = \frac{1}{\sigma_J\sqrt{2\pi}}\, exp\left(-\frac{(x-\mu_J)^2}{2\sigma_J^2}\right). \tag{2.1.18}$$

Definition 2.1.15 (The Kou jump-diffusion model). *In Kou's model, the dynamics for the log-stock prices $X(t)$ is given in* (2.1.12)*, where the jump magnitude variable J has an asymmetric double exponential distribution with the density*

$$f_J(x) = p_1\alpha_1 e^{-\alpha_1 x}\mathbb{1}_{\{x\geq 0\}} + p_2\alpha_2 e^{\alpha_2 x}\mathbb{1}_{\{x\leq 0\}}, \alpha_1 > 1, \alpha_2 > 0, \tag{2.1.19}$$

where $p_1, p_2 \geq 0$ satisfying $p_1 + p_2 = 1$, denote the probabilities of upward and downward jumps. $\alpha_1 > 1$ is required to ensure $\mathbb{E}\left[e^J\right] < \infty$ and $\mathbb{E}[S(t)] < \infty$.

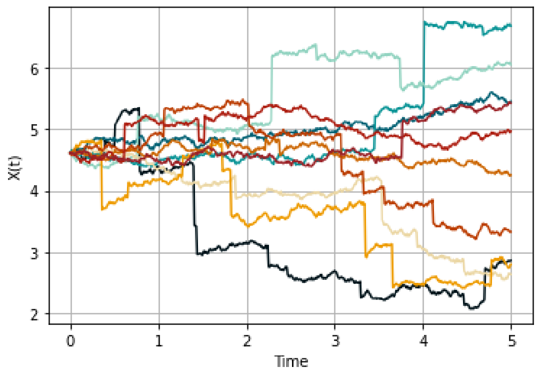
Example 2.1.16 (Paths of jump-diffusion processes). *Figure 2.6 illustrates 10 random paths of processes $X(t)$ in* (2.1.12) *and $S(t)$ in* (2.1.17) *from the MJD model and the KJD model respectively. Here, the parameters are set to $S(0) = 100$, $T = 5$, $\mu = 0.05$, $\sigma = 0.2$,, $\lambda_p = 1$, $\mu_J = 0$, $\sigma_J = 0.5$, $p_1 = p_2 = 0.5$, $\alpha_1 = \alpha_2 = 2$.*

2.2 Monte Carlo Path Simulation

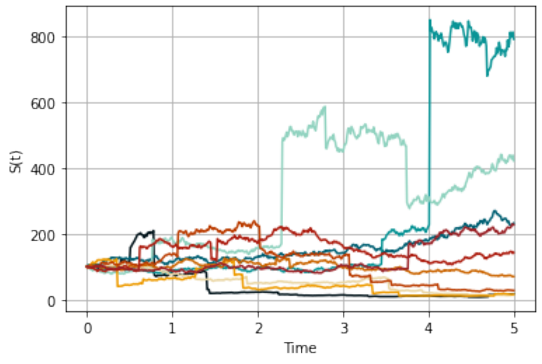
Monte Carlo simulation is a numerical sampling method that repeats the experiments many times to obtain a distribution of outcomes for analysis. Monte Carlo techniques are powerful and widely used in many areas, such as science, engineering, and finance [31]. In financial applications, Monte Carlo simulation is especially popularized in pricing and risk management. For the fundamental problem of solving stochastic differential equations, Monte Carlo methods also shows their advantages on path simulation.

The mathematics involved in Monte Carlo simulation are mainly the law of large numbers and the central limit theorem. The law of large numbers supports the key idea of Monte Carlo simulation, that is, for an arbitrary function $f : \Omega \mapsto \mathcal{F} \subseteq \mathbb{R}$, its expectation under a probability distribution $p(x)$ for the input $x \in \Omega$ can be approximated by

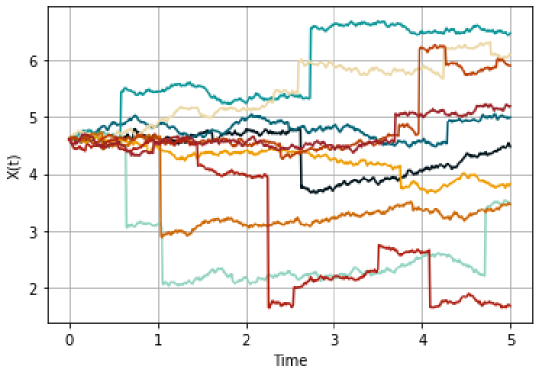
$$\begin{aligned} I = \mathbb{E}_{x\sim p(x)}[f(x)] &= \int_\Omega p(x)f(x)\mathrm{d}x \\ &\approx \frac{1}{N}\sum_{n=1}^{N} f(x_n) := \hat{I}, \quad x_n \overset{i.i.d}{\sim} p(x). \end{aligned} \tag{2.2.1}$$



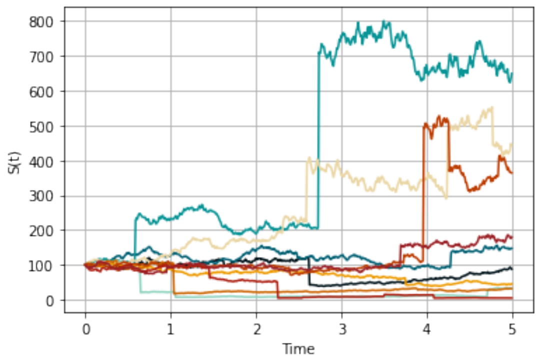
(a)



(b)



(c)



(d)

Figure 2.6: (a) and (b) present the paths of $X(t)$ and $S(t)$ from the MJD model; (c) and (d) present the paths of $X(t)$ and $S(t)$ from the KJD model.

The error of the estimator $\hat{I}$ is then structured via the central limit theorem:

$$\sqrt{N}(\hat{I} - I) \overset{d}{\mapsto} \mathcal{N}(0, \sigma^2),^{3} \tag{2.2.2}$$

where $\mathbb{V}\mathrm{ar}[f(x)] = \sigma^2 < \infty$.

The Monte Carlo technique is also popular on numerical integration, which shows its advantages regarding higher-dimensional integrals [32] and stochastic integration problems. With respect to solving SDEs, the Monte Carlo method is introduced to give numerical approximations of stochastic integrals, particularly in which a Wiener process $W(t)$ appears. The reader is referred to [25] for the details of Monte Carlo integration for $W(t)$-involved integrals.

Since the explicit solutions are usually not available for SDEs, researchers are interested in solving SDEs numerically. The numerical procedure is generally described as follows with an example. Consider a general one dimensional SDE whose Itô formula is defined by

$$\mathrm{d}S(t) = \bar{\mu}(t, S(t))\mathrm{d}t + \bar{\sigma}(t, S(t))\mathrm{d}W(t), t \geq 0, \tag{2.2.3}$$

[3] $\overset{d}{\mapsto}$ represents convergence in distribution.

with its solution in the time interval $[0, T]$ given by

$$S(T) = S(0) + \int_0^T \bar{\mu}(t, S(t))\mathrm{d}t + \int_0^T \bar{\sigma}(t, S(t))\mathrm{d}W(t). \tag{2.2.4}$$

We first partition the time interval, that is, define an equidistant grid in the temporal direction $0 = t_0 < t_1 < \cdots < t_m = T$, which is also named as time discretization. For each subinterval $[t_i, t_{i+1}] \subseteq [0, T]$, we obtain

$$s_{i+1} = s_i + \int_{t_i}^{t_{i+1}} \bar{\mu}(t, S(t))\mathrm{d}t + \int_{t_i}^{t_{i+1}} \bar{\sigma}(t, S(t))\mathrm{d}W(t), \tag{2.2.5}$$

where $s_i = S(t_i)$. The Monte Carlo method is then involved to numerically approximate the stochastic integrals in (2.2.5). At each time point t_i, a large number of s_{i+1} are generated, and a realization $\{s_0, s_1, \ldots, s_m\}$ is referred as a simulation path. The error convergence of the Monte Carlo path simulation above is defined as follows:

Definition 2.2.1 (Convergence). *Let s_m be an approximation for $S(T)$ and Δt be the time step size of the time discretization. s_m converges in a strong sense to $S(T)$, with order $\alpha > 0$, if*

$$\epsilon^s := \mathbb{E}\left[|s_m - S(T)|\right] = \mathcal{O}((\Delta t)^{\alpha}). \tag{2.2.6}$$

s_m converges in a weak sense to $S(T)$, with respect to a sufficiently smooth function $g(\cdot)$, with order $\beta > 0$, if

$$\epsilon^w := |\mathbb{E}\left[g(s_m)\right] - \mathbb{E}\left[g(S(T))\right]| = \mathcal{O}((\Delta t)^{\beta}). \tag{2.2.7}$$

Remark 2.2.2. *Note that the strong error ϵ^s explains the path-wise difference comparing to the exact solution $S(T)$, while the weak error ϵ^w describes the difference in the distributional sense.*

In the following part, we focus on the Monte Carlo path simulation with respect to the GBM and the jump-diffusion models and illustrate the simulation algorithm in detail.

2.2.1 Path Simulation of the GBM Model

The dynamics of a GBM process $\{S(t)\}_{t\geq 0}$ are given in (2.1.4), with the solution in (2.1.5). For the time discretization $0 = t_0 < t_1 < \cdots < t_m = T$, $m \geq 1$, with time step size $\Delta t = \frac{T}{m}$ and $t_i = i\Delta t$ for $\forall i = 0, 1, \ldots, m$, the dynamics at each time point t_i is given by

$$s_{i+1} = s_i + \int_{t_i}^{t_{i+1}} \mu S(t)\mathrm{d}t + \int_{t_i}^{t_{i+1}} \sigma S(t)\mathrm{d}W(t), \tag{2.2.8}$$

where $s_i = S(t_i)$.

Based on the stochastic Taylor expansions, the so-called Euler scheme and Milstein scheme are usually used for approximating the stochastic integrals. In the Euler scheme, the integrands uses the values at the left-side boundary of the integration interval as approximations; The Milstein scheme is based on the Euler scheme, with an additional component intuitively derived from the second order Taylor expansion (we refer to [14] for the detailed derivation).

Regarding (2.2.8), the Euler discretization scheme is given by:

$$\begin{aligned} s_{i+1} &\approx s_i + \int_{t_i}^{t_{i+1}} \mu s_i \mathrm{d}t + \int_{t_i}^{t_{i+1}} \sigma s_i \mathrm{d}W(t) \\ &\overset{\text{def}}{=} s_i + \mu s_i \Delta t + \sigma s_i \Delta t (W(t_{t+i}) - W(t_i)) \\ &\overset{\text{def}}{=} s_i + \mu s_i \Delta t + \sigma s_i \sqrt{\Delta t} Z, \end{aligned} \tag{2.2.9}$$

and the Milstein discretization scheme is defined as:

$$\begin{aligned} s_{i+1} &\approx s_i + \mu s_i \Delta t + \sigma s_i \Delta t (W(t_{t+i}) - W(t_i)) + \frac{1}{2}\sigma^2 s_i (W(t_{t+i}) - W(t_i))^2 - \Delta t) \\ &\overset{\text{def}}{=} s_i + \mu s_i \Delta t + \sigma s_i \sqrt{\Delta t} Z + \frac{1}{2}\sigma^2 s_i (\Delta t Z^2 - \Delta t), \end{aligned} \tag{2.2.10}$$

where $Z \sim \mathcal{N}(0,1)$ is a random variable since the Wiener increment $W(t_{t+i}) - W(t_i) := \Delta W(t_{i+1})$ is normally distributed with mean 0 and variance Δt.

Remark 2.2.3. (2.2.8) *has the exact solution given by*

$$S(t_{i+1}) = S(t_i)\, exp\left((\mu - \frac{1}{2}\sigma^2)\Delta t + \sigma(W(t_{i+1}) - W(t_i) \right). \tag{2.2.11}$$

The path simulation algorithm for the GBM process is then formed in Algorithm 1. With respect to the error convergence, the Euler scheme converges in a strong sense with order $\frac{1}{2}$ and in a weak sense with order 1; the Milstein scheme converges in both strong and weak senses with order 1 [33]. An example of both discrete-schemes is shown in the following part, together with the error convergence plots.

Example 2.2.4 (Path simulation of the GBM process)**.** *With parameters setting* $S(0) = 100$, $T = 1$, $\mu = 0.1$ *and* $\sigma = 0.3$, *Monte Carlo paths are generated (see Figure 2.7 (a) and (b)) by using the Euler scheme given in* (2.2.9) *and the Milstein scheme given in* (2.2.10) *respectively. The error convergence of the path simulation is also measured (see Figure 2.7 (c) and (d)), verifying the order of the strong and weak convergence regarding the two discrete-schemes.*

2.2.2 Path Simulation of the Jump-Diffusion Process

Our goal is to simulate a log-price jump-diffusion path $\{X(t_0), X(t_1), \ldots, X(t_m)\}$, with equal time partition $0 = t_0 < t_1 < \cdots < t_m$, time step length $\Delta t = \frac{T}{m}$ and the initial $X(t_0) = \log S(0)$. Comparing (2.1.8) and (2.1.12), the dynamics of a jump-diffusion process can be divided into a diffusion part and a jump part

$$\mathrm{d}X(t) = \underbrace{\mu \mathrm{d}t + \sigma \mathrm{d}W^{\mathbb{P}}(t)}_{\text{diffusion part}} + \underbrace{J\mathrm{d}X_{\mathcal{P}}(t)}_{\text{jump part}},$$

and so as the path simulation process. We therefore first focus on modeling the jump part, namely the Poisson process.

The key point of simulating the jump part is to determine the jump instances, that is,

Algorithm 1: Path simulation of GBM process, where $\eta = 0$ for the Euler scheme and $\eta = 1$ for the Milstein scheme.

```
Input  : Initial price $S(0)$, time interval $[0, T]$, drift parameter $\mu$, volatility
         parameter $\sigma$, the number of paths $n$ and the number of time steps $m$.
Output: $n$ simulation paths $\{s_{i,j}\}$of the GBM process with $m$ time steps, where
         $i = 0, 1, \dots, m$ and $j = 0, 1, \dots, n$.
$\Delta t \leftarrow \frac{T}{m}$ ;                /* Partition the time interval $[0, T]$ evenly:
 $0 = t_0 < t_1 < \cdots < t_m = T$, where $t_{i+1} - t_i = \frac{T}{m} := \Delta t$, $\forall i = 0, 1, \dots, m-1$.
 */
for $j = 1, \dots, n$ do
    $s_{j,0} \leftarrow S(0)$
    for $i = 0, 1, \dots, m-1$ do
        $z_{j,i} \leftarrow$ sample$(\mathcal{N}(0,1))$ ; /* $z_{j,i}$ is an i.i.d sample from the standard
         normal distribution.  */
        $s_{j,i+1} \leftarrow s_{j,i} + \mu s_{j,i}\Delta t + \sigma s_{j,i}\sqrt{\Delta t} z_{j,i} + \eta\left[\frac{1}{2}\sigma^2 s_{j,i}(\Delta t z_{j,i}^2 - \Delta t)\right]$ ; /* $\eta = 0$ for
         the Euler scheme and $\eta = 1$ for the Milstein scheme.  */
    end
end
```

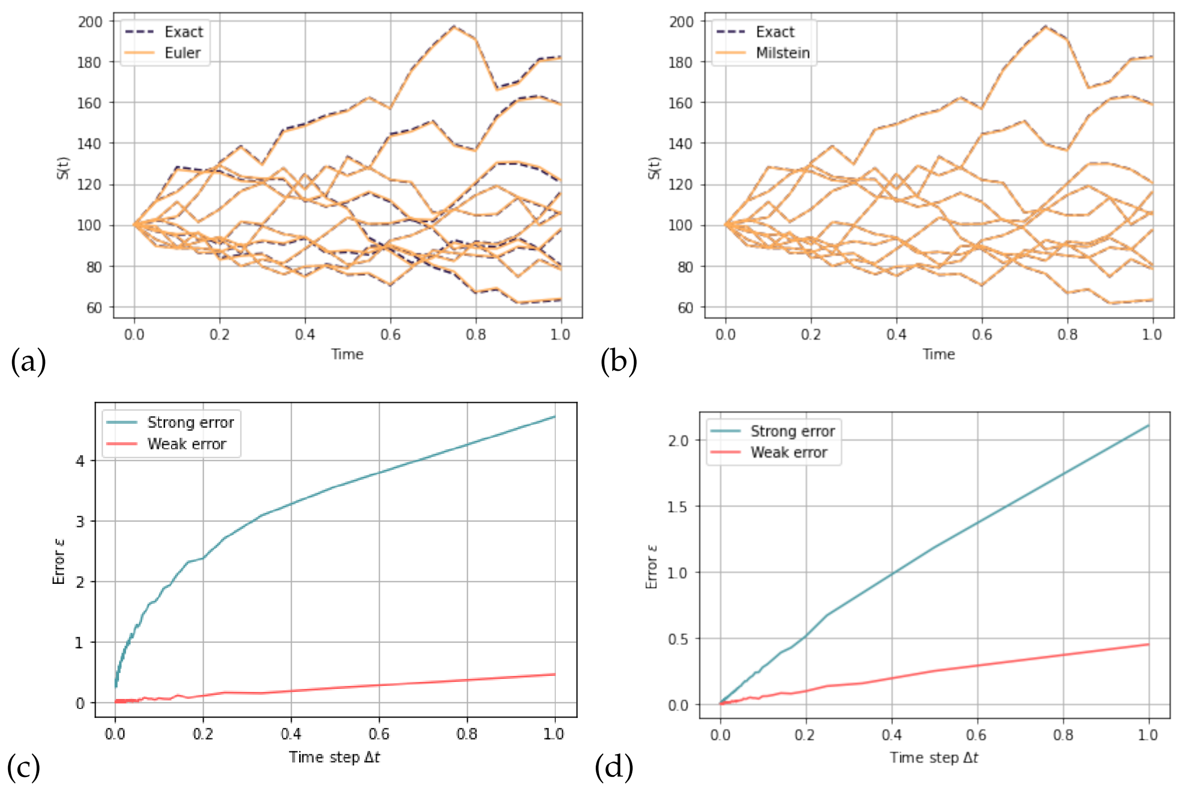


Figure 2.7: (a) and (b) illustrate the paths generated by the Euler scheme and the Milstein scheme respectively, compared with the exact solution given in (2.2.11); (c) and (d) present strong and weak convergence of both approximation schemes.

the timestamps $\{t_i\}$ when jumps occur. There are mainly three ways to calculate jump instances: 1) Since the Poisson increments follow Poisson distribution with parameter $\lambda_p \Delta t$, we can use Poisson random variables to model the number of jumps that arrive during every Δt; 2) As a Poisson distribution can be approximated by Bernoulli experiments (see Appendix A.1), at every time instance t_i, we can then introduce Bernoulli random variables to decide whether a jump occurs at t_i or not [18]; 3) Based on the result that the interarrival times of a Poisson process are exponentially distributed, we can then use exponential random variables to model the arrival time of a jump [34].

Note that following the time discretization method, it is natural to assume that a jump can only occur at some time t_i, that is, no jump happens during the period of two timestamps. Therefore, the third method that can return jump instances between two timestamps is improper. In programming, using Bernoulli random variables is easier to model the jump part than Poisson random variables, as at most one jump can occur at t_i. Figure 2.8 shows the error between two methods, and we can see that when the time step Δt is small, the two methods perform closely. On the other hand, the probability of k jumps occurring during a time period of length $\mathrm{d}t$ is $\mathbb{P}[\mathrm{d}X_{\mathcal{P}}(t) = k] = \frac{(\lambda_p \mathrm{d}t)^k e^{-\lambda_p \mathrm{d}t}}{k!}$.By Taylor series, the probability of exactly one jump occurring in a small $\mathrm{d}t$ is

$$\mathbb{P}[\mathrm{d}X_{\mathcal{P}}(t) = 1] = \frac{(\lambda_p \mathrm{d}t) e^{-\lambda_p \mathrm{d}t}}{1!} = \lambda_p \mathrm{d}t + o(\mathrm{d}t),$$

and

$$\mathbb{P}[\mathrm{d}X_{\mathcal{P}}(t) = k > 1] = \frac{(\lambda_p \mathrm{d}t) e^{-\lambda_p \mathrm{d}t}}{k!} = o(\mathrm{d}t),$$

which also give a theoretical support for using Bernoulli variables.

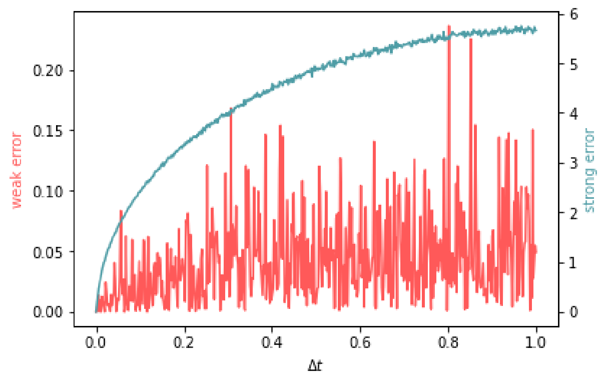


Figure 2.8: The strong error and weak error of simulating a 50-step Poisson process with $\lambda_p = 1$ by using Bernoulli random variables, compared to using Poisson random variables.

All in all, in this thesis, we use the second method, namely independent Bernoulli random variables to determine jump instances. Next, we return to the procedure of path simulation method for jump-diffusion models.

Similar to (2.2.9), the Euler scheme for a jump-diffusion process (2.1.13) is given by:[4]

$$\begin{aligned} x_{t+1} &= x_i + \int_{t_i}^{t_{i+1}} \mu \mathrm{d}t + \int_{t_i}^{t_{i+1}} \sigma \mathrm{d}W(t) + \int_{t_i}^{t_{i+1}} J \mathrm{d}X_{\mathcal{P}}(t) \\ &= x_i + \int_{t_i}^{t_{i+1}} \mu \mathrm{d}t + \int_{t_i}^{t_{i+1}} \sigma \mathrm{d}W(t) + \sum_{k=1}^{X_{\mathcal{P}}(t_{i+1})-X_{\mathcal{P}}(t_i)} J_k \\ &\overset{\text{def}}{=} x_i + \underbrace{\mu\Delta t + \sigma\sqrt{\Delta t}Z}_{\text{diffusion part}} + \underbrace{\sum_{k=1}^{X_{\mathcal{P}}(t_{i+1})-X_{\mathcal{P}}(t_i)} J_k}_{\text{jump part}} \end{aligned} \tag{2.2.12}$$

Regarding the compound Poisson process $\left\{ \sum_{k=1}^{X_{\mathcal{P}}(t_{i+1})-X_{\mathcal{P}}(t_i)} J_k \right\}$, we introduce a Bernoulli random variable $B(t_i)$ to determine whether a jump occurs at t_{i+1} or not, given by:

$$\begin{aligned} \mathbb{P}[\mathrm{B}(t_i) = 1] &= \lambda_p \Delta t, \\ \mathbb{P}[\mathrm{B}(t_i) = 0] &= 1 - \lambda_p \Delta t. \end{aligned} \tag{2.2.13}$$

If $B(t_i) = 0$, then $X_{\mathcal{P}}(t_i) = X_{\mathcal{P}}(t_{i+1})$, which means the log-price not jumping at t_{i+1} and (2.2.12) reduces to the Euler discretization of the GBM process. Otherwise $X_{\mathcal{P}}(t_{i+1}) = X_{\mathcal{P}}(t_i) = 1$, which indicates that the log-price at t_{i+1} increases additionally by J_k. The jump size J_k is also an independent random variable governed by the model setting.

Within the Monte Carlo simulation of the jump-diffusion process, the log-price $x_{j,i}$ for the j^{th} path at i^{th} time step is then given by:

$$x_{j,i+1} = x_{j,i} + \mu\Delta t + \sigma\sqrt{\Delta t}Z + J_{j,i}B(t_{j,i}). \tag{2.2.14}$$

We summarize the whole simulation procedure by Algorithm 2.

Remark 2.2.5. *The jump-diffusion process with dynamics for the log-prices $X(t) = \log S(t)$, $dX(t) = \mu dt + \sigma dW(t) + JdX_p$, has as exact solution in the time interval $[t_i, t_{i+1}]$,*

$$S(t_{i+1}) = S(t_i) \exp\left(\mu\Delta t + \sigma(W(t_{i+1}) - W(t_i)) + JX_p(t_{i+1} - X_p(t_i))\right)$$

Example 2.2.6 (Path simulation of the jump-diffusion process)**.** *Figure 2.9 shows ten simulated paths of the jump diffusion following Algorithm 2. Here, the jump sizes are governed under Merton's model and Kou's model respectively, and the parameters are set to $X(0) = \log S(0) = \log 100$, $T = 10$, $\mu = 0.05$, $\sigma = 0.2$, $\lambda_p = 1$, $\mu_J = 0$, $\sigma_J = 0.5$, $p_1 = p_2 = 0.5$, $\alpha_1 = \alpha_2 = 2$.*

Furthermore, we briefly analyze the error convergence of the Monte Carlo simulation regarding the jump-diffusion models, see Figure 2.10. [35] concludes that the weak convergence has the same order as the scheme used. Generally, the weak convergence

[4]If $X_{\mathcal{P}}(t_{i+1}) - X_{\mathcal{P}}(t_i) = 0$, then the summation term is equal to 0.

Algorithm 2: Path simulation of jump-diffusion process.

Input : Initial log-price $X(0) = \log S(0)$, time interval $[0, T]$, drift parameter μ, volatility parameter σ, jump intensity λ_p, jump magnitude $J \sim \mathbb{P}_J$, the number of paths n and the number of time steps m.

Output: n simulation paths $\{s_{i,j}\}$of jump-diffusion process with m time steps, where $i = 0, 1, \ldots, m$ and $j = 0, 1, \ldots, n$.

```
$\Delta t \leftarrow \frac{T}{m}$ ;   /* Partition the time interval $[0,T]$ evenly: $0 = t_0 < t_1 < \cdots < t_m = T$, where $t_{i+1} - t_i = \frac{T}{m} := \Delta t$, $\forall i = 0,1,\ldots,m-1$. */
for $j = 1, \ldots, n$ do
    $x_{j,0} \leftarrow X(0)$
    $s_{j,0} \leftarrow S(0)$
    for $i = 0, 1, \ldots, m-1$ do
        $z_{j,i} \leftarrow$ sample$(\mathcal{N}(0,1))$ ;   /* $z_{j,i}$ is an i.i.d sample from the standard normal distribution. */
        $B_{j,i} \leftarrow$ sample$(\mathcal{B}(1, \lambda_p \Delta t))$ ;   /* $B_{j,i}$ is an i.i.d sample from the Bernoulli distribution with parameter $\lambda_p \Delta t$. */
        $J_{j,i} \leftarrow$ sample$(\mathbb{P}_J)$ ;   /* $J_{j,i}$ is an i.i.d sample from the jump magnitude distribution $\mathbb{P}_J$, for example $\mathbb{P}_J$ is normally distributed in Merton's model. */
        $x_{j,i+1} \leftarrow x_{j,i} + \mu \Delta t + \sigma \sqrt{\Delta t} z_{j,i} + J_{j,i} B_{j,i}$
        $s_{j,i+1} \leftarrow \exp\left(x_{j,i}\right)$
    end
end
```

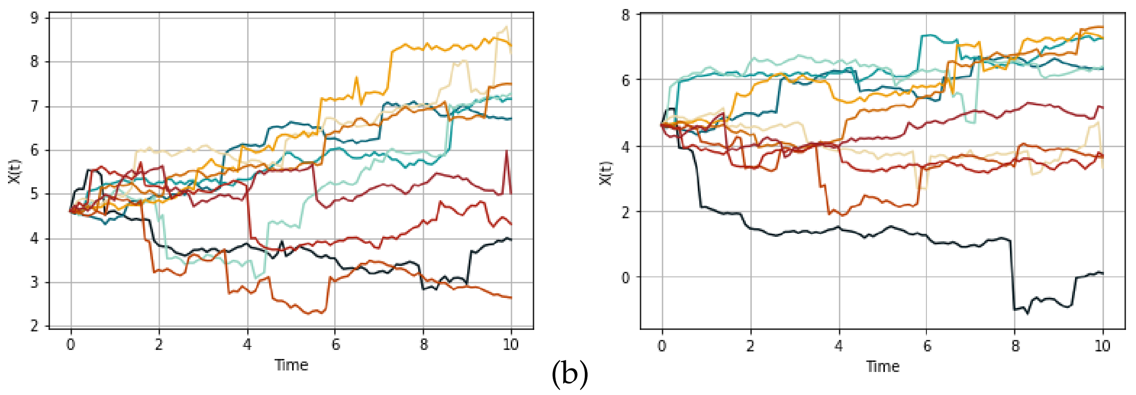


Figure 2.9: Ten simulated log-price paths of the jump-diffusion processes. Left: the dynamics follow the MJD model. Right: the dynamics follow the KJD model.

illustrated in Figure 2.10 is consistent with the result. Regarding the strong convergence, we find it almost uniform (compared to the slope of weak convergence) with respect to the time step size.

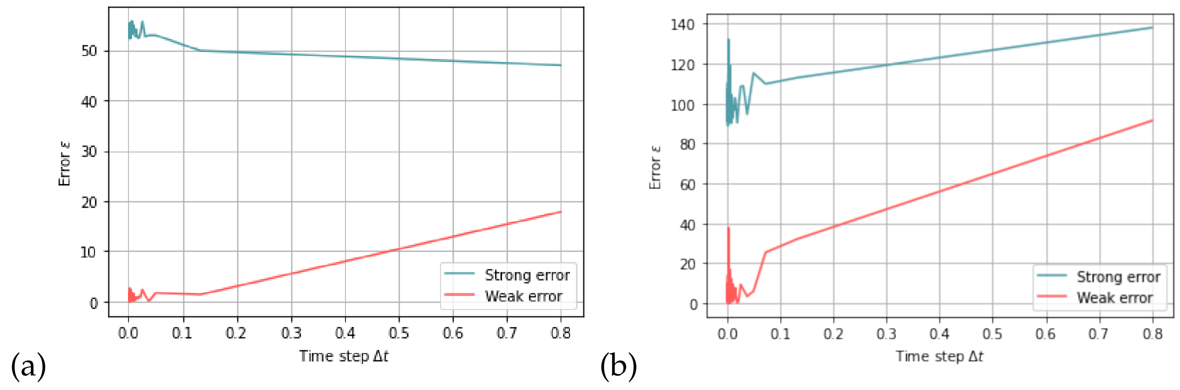


Figure 2.10: The weak and strong error convergence, compared to the exact solution where the jump instances are determined by Poisson random variables. Left: the convergence plot of the MJD model. Right: the convergence plot of the KJD model.

2.3 Artificial Neural Networks

The concept of artificial neural networks (ANNs) has been in the spotlight for decades and still leads innovation today. From a high level view, an artificial neural network is an input-output mapping: with provided training data, it learns the mapping from input to output data by seeing pairs of input and output data.

In this section, we describe the basics of artificial neural networks, in particular, a class of fully connected neural networks called multilayer Perceptron (MLP), which is a keystone in our proposed framework. The section is structured as follows: First, we illustrate the general architecture of a neural network; Then, the universal approximation theorem is stated without proof, which is a mathematical guarantee of using neural networks for approximation. After that, the so-called backpropagation is explained, which is the essence of neural network training. Various optimization methods are then introduced to descend the gradient calculated via backpropagation, such as Stochastic Gradient Descent (SGD), Root Mean Squared Propagation (RMSProp) and Adaptive Moment Estimation (Adam). Last but not least, we turn to activation functions, which also play a primary role in neural networks by adding non-linearity.

Structure of ANNs

As the name implies, artificial neural networks are inspired by neurons in biological brains, which receive input signals, and output reactions based on the inputs. In biology, usually many neurons work together to accomplish one action, and they transmit information via a structure called synapse. Neurons are then connected by synapses, forming a network of neurons.

Artificial neural networks have similar components, and Figure 2.11 displays a typical architecture of a fully connected feedforward ANN. Neurons in an artificial neural network are called nodes or units, which can process inputs and produce outputs. The connections are called edges, showing the direction of signal transmission. The ANN is typically divided into layers, where the nodes in one layer have the same level of connections. Usually, different layers execute different transformations of their inputs: The first layer, called the input layer, receives the initial signals; The last layer, called

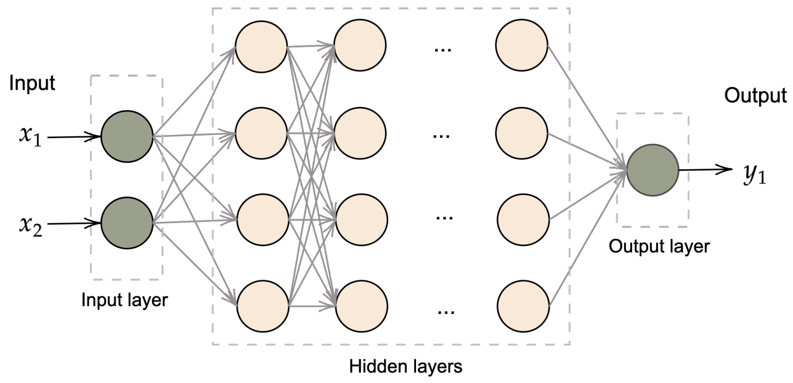


Figure 2.11: A general structure of a fully connected feedforward artificial neural network.

the output layer, produces the processed results; And the layers between the input and output layers, called the hidden layers, transform signals multiple times.

From the mathematical point of view, an artificial neural network is a set of weights, biases and activation functions, which are also the parameters of signal transformations. The mathematical formula of ANNs can be expressed as follows:

Suppose the input of an ANN layer is a real-valued vector $\mathbf{x} \in \mathbb{R}^m$ and the layer contains n nodes. Each edge between the input and a node conveys a linear transformation $f(\cdot)$ with weight $\mathbf{w} \in \mathbb{R}^{n \times m}$, formulated by

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}. \tag{2.3.1}$$

The node then produces output $\mathbf{y} \in \mathbb{R}^n$ by adding a bias $\mathbf{b} \in \mathbb{R}^n$ and further transforms the biased input via an activation function $h(\cdot)$, given by

$$\mathbf{y} = h\left[f(\mathbf{x}, \mathbf{w}) + \mathbf{b}\right] = h\left(\mathbf{w}^T \mathbf{x} + \mathbf{b}\right). \tag{2.3.2}$$

Hence, consider a fully connected feedforward ANN with m initial inputs $\mathbf{x} = \{x_1, \dots, x_m\}$, n outputs $\mathbf{y} = \{y_1, \dots, y_n\}$ and L hidden layers with k nodes in each hidden layer, we can describe the ANN as a mapping, given by:

$$\mathbf{y} = \Phi(\mathbf{x}|\Theta) = h^{(L+1)} \circ \alpha^{(L+1)}\left(\cdot|\theta^{(L+1)}\right) \circ h^{(L)} \circ \alpha^{(L)}\left(\cdot|\theta^{(L)}\right) \circ \cdots \circ h^{(1)} \circ \alpha^{(1)}\left(\mathbf{x}|\theta^{(1)}\right), \tag{2.3.3}$$

where $\Theta := \left(\theta^{(1)}, \dots, \theta^{(L+1)}\right) = \left(\mathbf{w}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{w}^{(L+1)}, \mathbf{b}^{(L+1)}\right)$ is a collection concatenating all weights and biases, here $\mathbf{w}^{(1)} \in \mathbb{R}^{m \times k}, \mathbf{b}^{(1)} \in \mathbb{R}^k, \mathbf{w}^{(L+1)} \in \mathbb{R}^{k \times n}, \mathbf{b}^{(L+1)} \in \mathbb{R}^n$, and $\mathbf{w}^{(i)} \in \mathbb{R}^{m \times m}, \mathbf{b}^{(i)} \in \mathbb{R}^m$ for $i = 2, \dots, L$; $h^{(i)}$ represents an activation function;

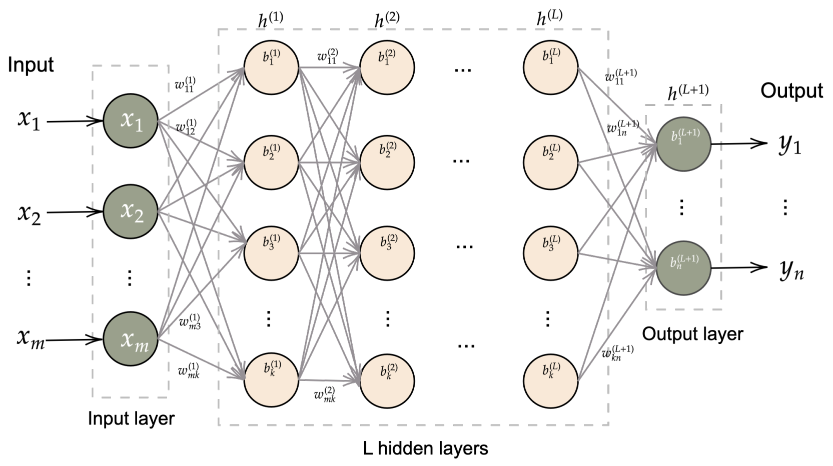


Figure 2.12: A detailed structure of a fully connected feedforward artificial neural network, where w_{ij}^l is the weight of an edge, b_i^l is the bias of a node and h^l is the activation function of a layer.

$\alpha^{(i)}$ denotes the operator for the i^{th} layer with expression

$$\alpha^{(i)}\left(\mathbf{z}|\theta^{(i)}\right)=\left(\mathbf{w}^{(i)}\right)^T\mathbf{z}+\mathbf{b}^{(i)}.$$

Figure 2.12 is illustrated for a vivid description of (2.3.3).

Activation Functions
The activation function is an important component in artificial neural network modeling, which makes non-linear approximations possible. However, the activation function is a hyperparameter in neural network setting, that is, there is no clear criterion for choosing activation functions.

Not all functions in real space can be introduced as activation functions. Because of backpropagation and gradient descent algorithms, activation functions are restricted to not shift the gradient towards zero. Continuity and differentiability (at least in sufficient ranges) are then necessary. In this section, we focus on four widely used activation functions: logistic or sigmoid function, hyperbolic tangent, ReLU and LeakyReLU.

The sigmoid function is given by

$$h_1(x)=\frac{1}{1+e^{-x}}. \qquad (2.3.4)$$

Since $h_1(x)\in(0,1)$ and everywhere differentiable, it is usually chosen as an output layer activation in binary classification models [36].

Hyperbolic tangent also presents 'S'-like shape as the sigmoid function, which has formula

$$h_2(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.3.5}$$

Hyperbolic tangent function works similar to the sigmoid, but $h_2(x) \in (-1, 1)$ and is centrosymmetric about zero.

Even though the sigmoid and hyperbolic tangent are commonly used in neural network architecture, it may cause unsuccessful training. As the saturating region of both functions are plain, vanishing gradients may occur which may lead to a neural network getting stuck at the training time.

ReLU [37], short for rectified linear unit, is one of the popular choices nowadays, especially for convolutional neural networks. ReLU is defined by

$$h_3(x) = \max(x, 0). \tag{2.3.6}$$

The range of ReLU is $[0, \infty)$. ReLU is differentiable everywhere, except at $x = 0$, and both the function itself and its derivative are monotonic. However, ReLU cannot appropriately map negative values, by which all negative inputs turn to zero. Because of this, nodes with negative inputs may learn nothing. LeakyReLU is then proposed in [38] to improve such problem.

LeakyReLU has formula

$$h_4(x) = \max(x, 0) + \zeta \min(x, 0), \tag{2.3.7}$$

where $\zeta \in \mathbb{R}$ is a hyperparameter and is typically set to 0.01. Unlike ReLU, LeakyReLU can still provide non-zero gradient in the region $x < 0$.

Universal Approximation Theorem
Although ANNs have achieved great success in practice, the development of theoretical results with vigorous proof is delayed. People are curious about the approximation ability of an ANN, namely, which kind of maps can be learned by an ANN.

As a theoretical guarantee for neural networks, the universal approximation theorem, similar to the classical Weierstrass approximation theorem [39], states that neural networks can approximate any continuous function defined on $[0, 1]$.

Theorem 2.3.1 (Universal approximation theorem [40]). *Suppose $\Phi(\cdot)$ is a multilayer feedforward neural network with one hidden layer, given by*

$$\Phi(x|\Theta) = h^{(2)} \circ \alpha^{(2)} \left(\cdot|\theta^{(2)}\right) \circ h^{(1)} \circ \alpha^{(1)} \left(x|\theta^{(L)}\right), \tag{2.3.8}$$

where $h^{(1)}$ is sigmoidal[5] *and $h^{(2)}$ is linear. Then $\forall f \in C[0, 1]$ and $\forall \epsilon > 0$ there exists a neural network formed by $\Phi(x|\Theta^*)$* (2.3.8) *such that*

$$\sup_{x \in [0,1]} |\Phi(x|\Theta^*) - f(x)| < \epsilon.$$

[5]An activation function $h : \mathbb{R} \mapsto \mathbb{R}$ is sigmoidal, if it is continuous and $\lim_{x \mapsto -\infty} h(x) = 0$, $\lim_{x \mapsto \infty} h(x) = 1$

In recent papers, the universal approximation theorem for neural networks has been extended. For example, it holds for feedforward neural networks with arbitrary depth but bounded width to approximate any Lebesgue integrable function [41].

Backpropagation and Gradient Descent
An artificial neural network provides a target output by weights and biases. The process of finding the best weights and biases is called training. The feedforward ANN is trained aiming to minimize the difference between the neural network output and the target output, i.e., the loss function, using gradient descent algorithms. The backpropagation is then applied to calculate the gradient of the loss function with respect to the weights and biases. For the detailed expression of the backpropagation, we refer to [42].

Gradient descent is a basic algorithm for finding the (local) minimization of the loss function. Consider a feedforward ANN of form (2.3.3). Our goal is to find the global optimal parameter set Θ^* to minimize the loss function L. Since no closed-form expression for Θ^* exists, Gradient Descent is then used to obtain a local optimal numerically. In general, the parameter set Θ is updated iteratively in the direction of the negative gradient. The i^{th} iteration can be expressed by

$$\Theta^{(i+1)} := \Theta^{(i)} - \lambda \nabla_{\Theta^{(i)}} L, \tag{2.3.9}$$

here $\lambda \in \mathbb{R}$ is a constant called learning rate, which is a hyperparameter that determines the step size along the descent direction.

Optimization Methods
In practice, Gradient Descent is usually not used in training neural networks. All training samples are involved in computing the loss function, which requires large memory. On the other hand, when the dataset is large, the algorithm is found hard to converge. In this part, three variants, Stochastic Gradient Descent (SGD), Root Mean Squared Propagation (RMSProp), and Adaptive Moment Estimation (Adam), are illustrated as alternatives to Gradient Descent.

Stochastic Gradient Descent [43] alters the gradient computation by introducing randomness. The training dataset X is randomly divided into several batches, and the loss function is calculated based on the samples in each subset. The parameter set is updated after every batch:

$$\Theta^{(i+1)} := \Theta^{(i)} - \lambda \frac{1}{|B|} \nabla_{\Theta^{(i)}} \sum_{\mathbf{x}_B \in B} L\left(\Phi\left(\mathbf{x}_B, \Theta^{(i)}\right), \mathbf{Y}\right), \tag{2.3.10}$$

where B represents a random batch with size $|B|$ and $\mathbf{Y}$ denotes the desired output. SGD algorithm usually requires less memory and converges faster on large datasets. However, as the parameters are updated more frequently, the convergence path of SGD is noisier than Gradient Descent.

Root Mean Squared Propagation (RMSProp) [44] is an extension to Gradient Descent associated with another extension, the so-called Adagrad[6] [46], where the learning rate

[6]In [44], RMSProp is an adaption of Rprop [45] for mini-batch learning.

is adapted based on the gradient. The update iteration is formulated as follows:

$$
\begin{aligned}
E\left[g^2\right]_i &= \beta E\left[g^2\right]_{i-1} + (1-\beta) g_i^2, \\
\Theta^{(i+1)} &:= \Theta^{(i)} - \frac{\lambda}{\sqrt{E\left[g^2\right]_i + \epsilon}} g_i,
\end{aligned}
\tag{2.3.11}
$$

where $g_i \triangleq \nabla_{\Theta^{(i)}} L$, $E[g^2]_i$ is the moving average of squared gradients at iteration i, β is a constant moving average parameter and ϵ is a sufficiently small positive constant. RMSProp usually converges faster than Gradient Descent and SGD. It is especially used in training Wasserstein GAN (see Chapter 4).

Adam, short for Adaptive Moment Estimation [47], is an extension of SGD by including an adaptive moment. It is an efficient algorithm, which combines the advantages of Adagrad and RMSProp, and is popular in training neural networks.

$$
\begin{aligned}
m_i &= \beta_1 m_{i-1} + (1-\beta_1) g_i, \\
v_i &= \beta_2 v_{i-1} + (1-\beta_2) g_i^2, \\
\Theta^{(i+1)} &:= \Theta^{(i)} - \lambda \frac{\hat{m}_i}{\sqrt{\hat{v}_i} + \epsilon}
\end{aligned}
\tag{2.3.12}
$$

where $\hat{m}_i = \frac{m_i}{1-\beta_1^i}, \hat{v}_i = \frac{v_i}{1-\beta_2^i}$ are bias corrections, $\beta_1, \beta_2 \in [0, 1)$ are constants, λ is the learning rate and ϵ is a sufficiently small positive number. In brief, instead of updating parameters based on the gradient itself, Adam algorithm also considers the first two moments of the gradient.

Summary
The ANNs are the essentials in the machine learning methods. When establishing an ANN, the choice of the neural network architecture, the activation functions and the optimization algorithm are important, which can influence the model performance significantly.

2.4 Anomaly Detection

Anomaly detection deals with the problem of identifying abnormal patterns in a given dataset. Usually, the abnormal patterns are very few and behave differently than most data. Such patterns refer to anomalies, rare events, outliers, or others, based on the problem setting and the application area [48]. Even though anomalies are rare, they can significantly influence a system, raising the interest and necessity to dig out the dynamics behind them. The first and crucial step is to detect the anomalies precisely. All kinds of anomaly detection methods are then proposed, and they have been applied in various fields, such as fraud detection [49], cyber-security [50] and medical imaging [51].

In this section, we first briefly present the definition of anomalies. Then we conclude that the general procedure of detecting anomalies can be summarized into two steps, that is, learning the normal patterns and calculating the anomaly scores of all data. To support the conclusion, we shortly describe different techniques and give an overview of the related works.

What Are Anomalies
Anomalies are easier to identify through descriptions and diagrams, and there is no strict mathematical definition of the concept. An anomaly, as the name implies, is a pattern in the data inconsistent with most behavior (i.e., normal patterns), as is shown in Figure 2.13. Usually, anomalies are of extremely low probability and are also called rare events in time series [52].

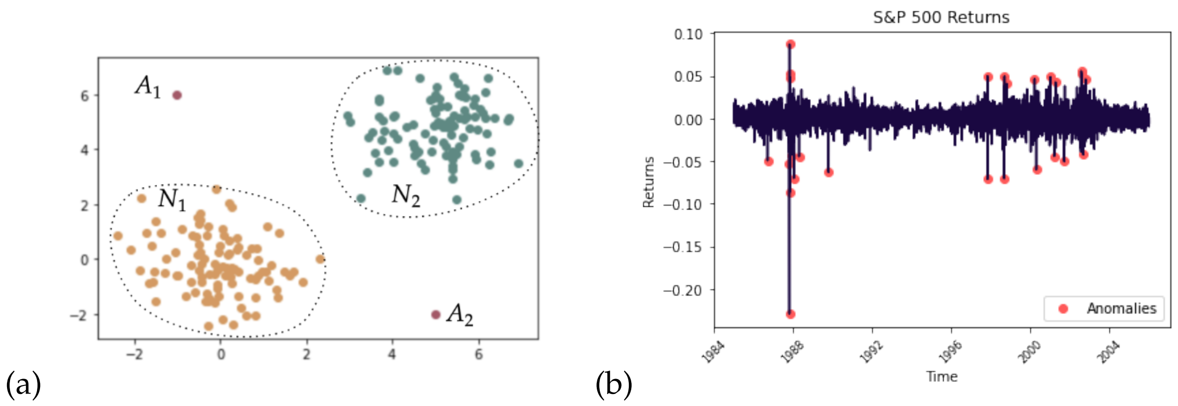


Figure 2.13: Two illustrations of anomalies. Left: A_1, A_2 are anomalies in a 2-dimensional dataset, while N_1 and N_2 are the regions of normal data; Right: The plot of S&P 500 returns between 1985 and 2005, where the red points are anomalies with extreme returns.

Anomaly Detection Techniques
Anomaly detection aims to find out abnormal samples from the given dataset. In general, the detection procedure consists of two components: 1) A mechanism is designed to figure out the "puzzles" of normal patterns; 2) The so-called anomaly score is usually constructed to measure how anomalous the samples in the dataset are.

According to the methods of learning normal data, anomaly detection techniques can be grouped into several categories: classification based, distance based, clustering based, reconstruction based, and so on [48; 53].

Classification based anomaly detection constructs a classifier to distinguish the normal and abnormal data. Usually, the classifier is trained using the available labeled training data and returns the classification results on the test data. Various classification algorithms, for example, classification neural networks and Support Vector Machines can be introduced to be the classifier and have been applied with excellent results in practice. The anomaly score is often built inside the classifier. For instance, the output of the neural network can be viewed as an anomaly score, presenting the probability of being abnormal.

In a distance based detection method, a similarity measure is introduced to quantify the distance between points or neighbors. The distance is used as an anomaly score and a threshold is set to separate the anomalies from normal data. The mechanism of distance based techniques is to assume that normal samples appear in dense neighborhoods, while anomalies are far away from their nearest neighbors [53].

Clustering based anomaly detection uses clustering algorithms to group data, and the samples that do not belong to any cluster are assumed anomalies. Similar to the distance based method, a similarity measure or distance is commonly involved in clustering data. Furthermore, the distance from a sample to its closest cluster represents the anomaly score.

The last category of anomaly detection techniques we discuss is based on reconstruction, which leads to a new direction of anomaly detection by learning and reconstructing the data. Usually, a deep-learning method is applied to regenerate the dataset, and the difference between the original sample and the reconstructed data is calculated as an anomaly score. A generative model, called generative adversarial networks (GANs), is popular in anomaly detection nowadays [54; 22; 55; 56]. The GANs-involved technique is also applied in this thesis and we will illustrate a GAN-related anomaly detection model called the AnoGAN in Section 3.5.2 as an application of GANs.

Chapter 3

Generative Adversarial Networks

3.1 Introduction

Generative adversarial networks (GANs) are a special kind of artificial intelligence algorithm proposed in the last decade [57], which has become a trend in various fields. The GAN techniques are especially appealing in image processing, where the new improved GAN techniques are usually tested on popular image datasets such as MNIST and CIFAR-10 [58; 59; 60]. Furthermore, with the applications on time series, GANs also give a new direction and developments in the financial industry, for example, [61] demonstrates impressive results regarding S&P 500 index simulations.

A GAN architecture consists of two artificial neural networks playing a zero-sum or a min-max game: one player called the generator G tries to learn the target data distribution as best as possible by generating fake examples whose distribution P_G approximates the real data distribution P_{data}; While the other player, the so-called a discriminator D aims to distinguish the examples generated by G from the real data samples [62]. D and G play against each other until convergence. Such technique is classified as a semi-supervised generative algorithm.

Since GANs are able to generate an approximate distribution with respect to the real data, they are well-suited for anomaly detection [54]. Consider a dataset that includes some rare and abnormal samples. The GANs are well-trained to capture the distribution of most samples (or the normal data), and they are not able to reproduce abnormal patterns. Therefore, there is an obvious difference between an abnormal sample and its GAN-generated pattern.

GANs, like other artificial neural network methods, are useful for tackling higher-dimensional problems and overcoming the curse of dimensionality [63]. [57] also points out that GANs can approximate sharp, even degenerate distributions, making them broadly applicable. However, GANs show a significant shortcoming of unstable training, which results in failure to fit the real data. Regarding the GANs failure modes, plenty of improved algorithms have been proposed in the recent years [64; 59; 60; 61] and remain appealing.

In this chapter, we first describe the structure and training process of a so-called vanilla

GAN in detail, and then show a straightforward extension called conditional GAN. Next, two applications using GANs are illustrated: path simulation for SDEs and anomaly detection. After that, We explain common problems of GANs during training with examples, and in this thesis, we mainly resort to Wasserstein GANs with gradient penalty (WGAN-GP) to resolve the failures (see Chapter 4).

3.2 Vanilla GAN

This section is inspired by [57], where the initial GAN model, which is usually called a vanilla GAN, was first illustrated. Generally, the GAN technique can be classified as a generative model from which the distribution of the training data is deduced based on density estimation [62]. Compared with traditional generative approaches, such as maximum likelihood estimation (MLE), GANs are implicit methods in the sense that the density function of the training examples is not explicitly learned.

The goal of a GAN model is to generate synthetic samples that follow the same probability distribution as the given data. Essentially, a GAN is made of two artificial neural networks: a generator G and a discriminator D. To achieve the goal, the generator G and the discriminator D work as follows:

Suppose the probability distribution of the given samples (the training data) is P_{data} with density function p_{data}. The generator G outputs fake samples $\hat{\mathbf{x}}$ by mapping latent variables $\mathbf{z}$ (with distribution P_z and density p_z) to the training data space, and we denote the distribution of the fake examples as P_G with density p_G. The discriminator D then plays a role to ensure that P_{data} is close to P_G or even equal. D gets inputs both from the real data $\mathbf{x}_d$ and the fake samples $\hat{\mathbf{x}}$ generated by G. If the input $\mathbf{x}$ is recognized as a real example by D, it is then labeled as real; otherwise, $\mathbf{x}$ is labeled as fake if it is recognized as a generated sample.

D and G are designed to compete each other by playing a minimax game[1]. The payoffs of both players can then be expressed via binary cross entropy functions [66], which are also called loss functions in the GANs theory. D and G keep updating their strategies based on the feedback (i.e. loss functions), until the game arrives at a Nash equilibrium where two players obtain their best possible payoffs. Such procedure is referred to as training in GANs techniques.

The mathematical mappings with respect to the discriminator D and the generator G can be presented as follows:

$$\begin{array}{rlcl} G_\eta : & \mathcal{Z} \subseteq \mathbb{R}^m & \to & G(\mathcal{Z};\eta) \subseteq \mathcal{X} \subseteq \mathbb{R}^n \\ & \mathbf{z} \sim p_z & \mapsto & \hat{\mathbf{x}} \sim p_G, \\ D_\theta : & \mathcal{X} \subseteq \mathbb{R}^n & \to & D(\mathcal{X};\theta) \subseteq [0,1] \\ & \mathbf{x} & \mapsto & D(\mathbf{x};\theta), \end{array} \tag{3.2.1}$$

where G and D are essentially differentiable functions with parameters η and θ respectively; the latent variable $\mathbf{z}$, which is also called as noise, is usually normally distributed; $\hat{\mathbf{x}} = G(\mathbf{z};\eta)$ is a generated sample following distribution P_G; $\mathbf{x}$ is a sample

[1]Recently, researchers point out that game-theory setting is incoherent with GANs because Nash equilibrium may not exist. [65]. Here, we use the game theory setting as a vivid intuition.

that is either from the real data or generated by G; $D(\mathbf{x};\theta)$ is a single scalar measuring the possibility of $\mathbf{x}$ coming from P_{data} rather than P_G. Thoroughly, $\mathbf{x}$ is assigned by value 1 if it is recognized as a real example, while assigned by value 0 if it is labeled as a generated sample.

During the training, the strategy of D is given by $\max\limits_{D_\theta} D_\theta(\mathbf{x}_d)$ and $\max\limits_{D_\theta} (1 - D_\theta(\hat{\mathbf{x}}))$, that is, maximize the probability of assigning the correct label to the inputs; While the strategy of G can be written as $\min\limits_{G_\eta} \left(1 - D_\theta\left(G_\eta(\mathbf{z})\right)\right)$, that is, minimize the probability of D making correct decisions. By introducing the binary cross entropy function, the loss functions of D and G are given by:

$$L_{D_\theta} = -\mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}} \left[\log D_\theta(\mathbf{x}_d)\right] - \mathbb{E}_{\mathbf{z}\sim p_z}\left[\log\left(1 - D_\theta\left(G_\eta(\mathbf{z})\right)\right)\right], \tag{3.2.2}$$

$$L_{G_\eta} = \mathbb{E}_{\mathbf{z}\sim p_z}\left[\log\left(1 - D_\theta\left(G_\eta(\mathbf{z})\right)\right)\right]; \tag{3.2.3}$$

The objective function corresponding to the minimax problem is defined by:

$$\min_{G_\eta}\max_{D_\theta} V\left(G_\eta, D_\theta\right) = \mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}} \left[\log D_\theta(\mathbf{x}_d)\right] + \mathbb{E}_{\mathbf{z}\sim p_z}\left[\log\left(1 - D_\theta\left(G_\eta(\mathbf{z})\right)\right)\right]. \tag{3.2.4}$$

See Figure 3.1 for a high-level framework of the training process described above.

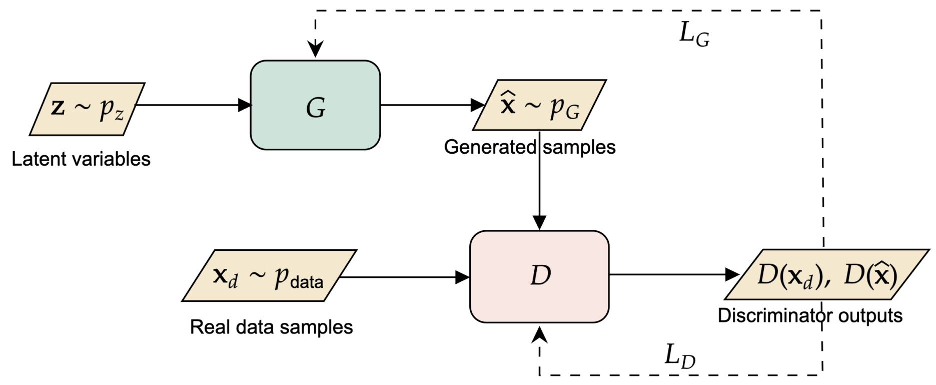


Figure 3.1: A high-level framework of GAN's training, where G and D are generally two independent artificial neural networks.

3.2.1 Theoretical Results

The existence and uniqueness of the optimal solution regarding the minimax problem (3.2.4) are interesting, and in the following part, we would like to present some theoretical analysis of the vanilla GAN.

Proposition 3.2.1 (Optimal discriminator)**.** *When the generator G is fixed, the optimal discriminator D^*_θ is given by*

$$D^*_\theta(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_G(\mathbf{x})}. \tag{3.2.5}$$

Proof. With fixed generator G, the discriminator D_θ is trained to maximize

$$\begin{aligned} V(G, D_\theta) &= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \log\left[D_\theta(\mathbf{x})\right] \mathrm{d}\mathbf{x} + \int_{\mathbf{z}} p_z(\mathbf{z}) \log\left[1 - D_\theta\left(G(\mathbf{z})\right)\right] \mathrm{d}\mathbf{z} \\ &= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \log\left[D_\theta(\mathbf{x})\right] \mathrm{d}\mathbf{x} + \int_{\hat{\mathbf{x}}} p_G(\hat{\mathbf{x}}) \log\left[1 - D_\theta(\hat{\mathbf{x}})\right] \mathrm{d}\hat{\mathbf{x}} \\ &= \int_{\mathbf{x}} p_{\text{data}}(\mathbf{x}) \log\left[D_\theta(\mathbf{x})\right] + p_G(\mathbf{x}) \log\left[1 - D_\theta(\mathbf{x})\right] \mathrm{d}\mathbf{x}. \end{aligned} \tag{3.2.6}$$

We now focus on the integrand. Consider a function $f(y) = a \log y + b \log(1-y)$ with $y \in [0,1]$ and $(a,b) \in \mathbb{R}^2 \backslash \{0,0\}$, $\arg\max_y f(y) = \frac{a}{a+b}$. Since D_θ is not required to be defined outside of $Supp(p_{\text{data}}) \cup Supp(p_G)$, we have $D^*_\theta = \arg\max_\theta V(G, D_\theta) = \frac{p_{\text{data}}}{p_{\text{data}} + p_G}$, which concludes the proof. □

Theorem 3.2.2 (Uniqueness). *The optimal discriminator D^*_θ is unique* [2].

Proof. For the proof we refer to [67]. □

The minimax problem (3.2.4) with D^*_θ is then reformulated by

$$\begin{aligned} \min_{G_\eta} C(G_\eta) &= V(G_\eta, D^*_\theta) \\ &= \mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}}\left[\log D^*_\theta(\mathbf{x}_d)\right] + \mathbb{E}_{\hat{\mathbf{x}} \sim p_G}\left[\log\left(1 - D^*_\theta(\hat{\mathbf{x}})\right]\right. \\ &= \mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\mathbf{x}_d)}{p_{\text{data}}(\mathbf{x}_d) + p_G(\mathbf{x}_d)}\right] + \mathbb{E}_{\hat{\mathbf{x}} \sim p_G}\left[\log \frac{p_G(\hat{\mathbf{x}})}{p_{\text{data}}(\hat{\mathbf{x}}) + p_G(\hat{\mathbf{x}})}\right], \end{aligned} \tag{3.2.7}$$

which is found having a tight relation with the the Kullback-Leibler (KL) divergence [68] and the Jensen-Shannon (JS) divergence [69].

The definition of KL divergence and JS divergence between two probability distributions P and Q are formulated as follows:

$$\text{KL}(P \| Q) = \int p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x, \tag{3.2.8}$$

$$\text{JS}(P \| Q) = \frac{1}{2}\text{KL}\left(P \| \frac{P+Q}{2}\right) + \frac{1}{2}\text{KL}\left(Q \| \frac{P+Q}{2}\right), \tag{3.2.9}$$

where $p(x)$ and $q(x)$ are the densities of p and Q respectively.

Equation(3.2.7) can then be written with KL divergence terms or a JS divergence com-

[2]In this thesis, we assume that all possible D_θ are available and $L_{D_\theta} < \infty$ for all D_θ. Besides that, $p_G(x) > 0$ at almost everywhere.

ponent, given by:

$$\begin{aligned}
C(G_\eta) &= \int p_{\text{data}}(x) \log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)} \mathrm{d}x + \int p_G(x) \log \frac{p_G(x)}{p_{\text{data}}(x) + p_G(x)} \mathrm{d}x \\
&= -2\log 2 + \int p_{\text{data}}(x) \log \frac{2p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)} \mathrm{d}x + \int p_G(x) \log \frac{2p_G(x)}{p_{\text{data}}(x) + p_G(x)} \mathrm{d}x \\
&= -2\log 2 + \mathrm{KL}\left(P_{\text{data}} \| \frac{P_{\text{data}} + P_G}{2}\right) + \mathrm{KL}\left(P_G \| \frac{P_{\text{data}} + P_G}{2}\right) \\
&= -2\log 2 + 2\mathrm{JS}(P_{\text{data}} \| P_G).
\end{aligned} \tag{3.2.10}$$

Theorem 3.2.3 (Global optimality). *$C(G_\eta)$ in (3.2.7) is globally minimized if and only if $P_{data} = P_G$, and the minimum value of $C(G_\eta)$ is $-2\log 2$.*

Proof. If $P_{\text{data}} = P_G$, then $D_\theta^* \equiv \frac{1}{2}$. $C(G_\eta) = \mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}}\left[\log \frac{1}{2}\right] + \mathbb{E}_{\hat{\mathbf{x}} \sim p_G}\left[\log \frac{1}{2}\right] = -2\log 2$.

Since for any two probability distributions, their JS divergence has the lower bound 0 which is arrived only when the two distributions are equal, from (3.2.10), we have $C(G_\eta) \geq -2\log 2$ and $C(G_\eta) = -2\log 2$ only if $P_{\text{data}} = P_G$. □

Therefore, the global optimization regarding the minimax problem (3.2.4) is achieved when G is a perfect replicator. However, the uniqueness of the optimal generator is more restricted, and the reader is referred to [67] for details.

3.3 Conditional GAN

The conditional GAN is one of the popular and straightforward extensions of the adversarial nets, which was proposed by [70]. As the name implies, the conditional GAN receives extra information $\mathbf{y}$ during training. Figure 3.2 displays a high-level architecture of the conditional GAN.

When defining the corresponding minimax problem, [70] treat $\mathbf{y}$ as conditions with respect to the real distribution P_{data} and the fake distribution P_G respectively. However, during the model training, $\mathbf{y}$ is processed as an additional input to both D and G. In order to avoid misunderstanding the concept of conditional distribution in statistics, we write the objective function as follows:

$$\min_{G_\eta} \max_{D_\theta} V\left(G_\eta, D_\theta; \mathbf{y}\right) = \mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}}\left[\log D_\theta\left(\mathbf{x}_d, \mathbf{y}\right)\right] + \mathbb{E}_{\mathbf{z} \sim p_z}\left[\log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}, \mathbf{y}\right)\right)\right)\right]. \tag{3.3.1}$$

Here, G and D are formulated by:

$$\begin{array}{llcl}
G_\eta: & \mathcal{Z} \times \mathcal{Y} \subseteq \mathbb{R}^m \times \mathbb{R}^l & \to & G(\mathcal{Z}, \mathcal{Y}; \eta) \subseteq \mathcal{X} \subseteq \mathbb{R}^n \\
& (\mathbf{z}, \mathbf{y}) \sim p_z & \mapsto & (\hat{\mathbf{x}}, \mathbf{y}) \sim p_{G,\mathbf{y}}, \\
D_\theta: & \mathcal{X} \times \mathcal{Y} \subseteq \mathbb{R}^n \times \mathbb{R}^l & \to & D(\mathcal{X}, \mathcal{Y}; \theta) \subseteq [0,1] \\
& (\mathbf{x}, \mathbf{y}) & \mapsto & D(\mathbf{x}, \mathbf{y}; \theta).
\end{array} \tag{3.3.2}$$

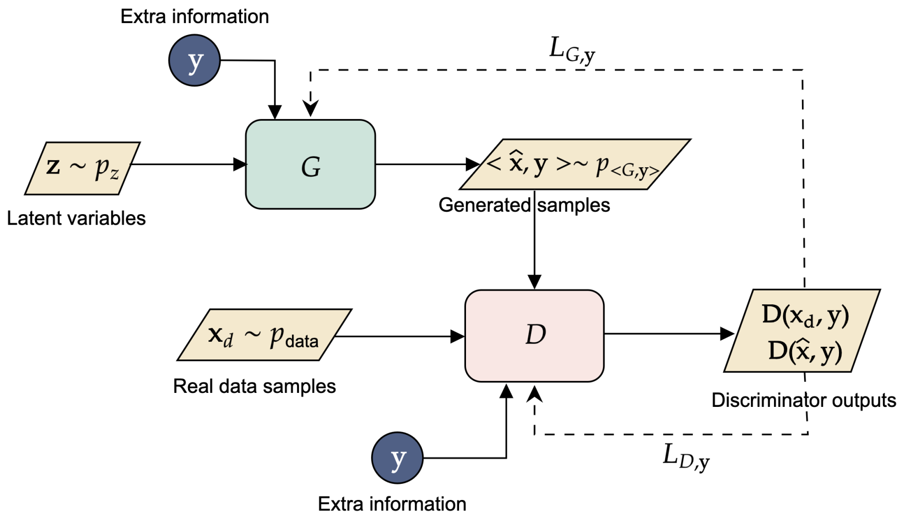


Figure 3.2: A high-level architecture of conditional GAN.

The loss functions of D and G are then represented by:

$$\begin{aligned} L_{(D,\mathbf{y})} &= -\mathbb{E}_{\mathbf{x}_d \sim p_{\text{data}}} \left[\log D_\theta \left(\mathbf{x}_d, \mathbf{y}\right)\right] - \mathbb{E}_{\mathbf{z}_d \sim p_z} \left[1 - \log D_\theta \left(G_\eta(\mathbf{z}, \mathbf{y})\right)\right], \\ L_{(G,\mathbf{y})} &= \mathbb{E}_{\mathbf{z}_d \sim p_z} \left[1 - \log D_\theta \left(G_\eta(\mathbf{z}, \mathbf{y})\right)\right]. \end{aligned} \tag{3.3.3}$$

All in all, the conditional GAN is not very special, except the extra information, which can be viewed as extended dimensions of the both latent variables and real samples. Similar theoretical results can therefore be directly derived from the vanilla GAN and we leave them to the reader.

3.4 GANs Training in Practice

In this section, the pseudocodes for training the vanilla GAN and the conditional GAN are illustrated respectively (see Algorithm 3 and Algorithm 4). Here, we resort to mini-batch gradient descent to update the parameters of D and G with Adam algorithm for optimizing the stochastic loss functions.

Moreover, [57] shows an interesting result related to the reliability of the training algorithm, stated as follows:

Proposition 3.4.1 (Convergence of Algorithm 3). *If D and G have enough capacity, and at each step of Algorithm 3, D is allowed to reach its optimum D^*_θ given G, and P_G is updated so as to improve the criterion*

$$\mathbb{E}_{\mathbf{x} \sim P_{data}} \left[\log D^*_\theta(\mathbf{x})\right] + \mathbb{E}_{\hat{\mathbf{x}} \sim P_G} \left[\log D^*_\theta \left(\hat{\mathbf{x}}\right)\right],$$

then P_G converges to P_{data}.

For a proof of training convergence, we refer to [57]. Similarly, the convergence of Algorithm 4 can be directly derived from Proposition 3.4.1.

Algorithm 3: Training algorithm of vanilla GAN.

```
Input  : Training epochs $N$, mini-batch size $L$, mini-batch number $M$, initial
         discriminator parameters $\theta_0$, initial generator parameters $\eta_0$.
Output: Discriminator parameters $\theta$, generator parameters $\eta$.
for Training epoch $n = 1, \dots, N$ do
    for Mini-batch $m = 1, \dots, M$ do
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        Choose $L$ random samples $\left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\right\}$ from the training data.
        /* Discriminator loss.                                              */
        $L_D^m \leftarrow \frac{1}{L} \sum_{l=1}^{L} - \left[\log D_\theta\left(\mathbf{x}^{(l)}\right) + \log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}^{(l)}\right)\right)\right)\right]$
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        /* Generator loss.                                                  */
        $L_G^m \leftarrow \frac{1}{L} \sum_{l=1}^{L} \log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}^{(l)}\right)\right)\right)$
        /* Update parameters.                                               */
        $\theta \leftarrow \text{Adam}\left(\nabla_\theta \sum_{m=1}^{M} L_D^m\right)$
        $\eta \leftarrow \text{Adam}\left(\nabla_\eta \sum_{m=1}^{M} L_G^m\right)$
    end
end
```

However, GANs are well-known for their training instability in practice, and we will discuss GANs failure modes extensively in Section 3.6. In the following part, we make a brief summary of different methods to improve GANs' stability by modifying the above algorithms.

There are mainly two ways to improve GANs training in practice. First of all, the generator G is trained to maximize $\log D_\theta\left(G_\eta(\mathbf{z})\right)$ instead of minimizing $\log\left(1 - D_\theta\left(G_\eta(\mathbf{z})\right)\right)$. With such modification, D and G can provide stronger gradients early [57]. Besides that, in each training epoch, D is set to update multiple times per G update, since the ratio of P_{data} and P_G should be estimated correctly. However, it usually does not lead to a clear improvement in practice [71]. The modified GAN training process is then illustrated in Algorithm 5, where $\alpha = 0$ means no extra information is provided, i.e. the vanilla GAN, and $\alpha = 1$ corresponds to the conditional GAN.

3.5 GANs Applications

GANs are powerful techniques and can be applied to various fields. Furthermore, researchers are interested in introducing GANs framework to problems with non-deep learning solutions. Advantages are usually gained by using GANs. In this section, we display two examples regarding GANs applications: path simulation of SDEs and anomaly detection, where the corresponding non-GAN methods are already described

Algorithm 4: Training algorithm of conditional GAN.

```
Input  : Extra information $\mathbf{y}$, training epochs $N$, mini-batch size $L$, mini-batch
         number $M$, initial discriminator parameters $\theta_0$, initial generator
         parameters $\eta_0$.
Output: Discriminator parameters $\theta$, generator parameters $\eta$.
for Training epoch $n = 1, \dots, N$ do
    for Mini-batch $m = 1, \dots, M$ do
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        Choose $L$ random samples $\left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\right\}$ from the training data.
        /* Discriminator loss.                                    */
        $L^m_{(D,\mathbf{y})} \leftarrow \frac{1}{L}\sum_{l=1}^{L} -\left[\log D_\theta\left(\mathbf{x}^{(l)}, \mathbf{y}\right) + \log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}^{(l)}, \mathbf{y}\right)\right)\right)\right]$
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        /* Generator loss.                                        */
        $L^m_{(G,\mathbf{y})} \leftarrow \frac{1}{L}\sum_{l=1}^{L} \log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}^{(l)}, \mathbf{y}\right)\right)\right)$
        /* Update parameters.                                     */
        $\theta \leftarrow \text{Adam}\left(\nabla_\theta \sum_{m=1}^{M} L^m_{(D,\mathbf{y})}\right)$
        $\eta \leftarrow \text{Adam}\left(\nabla_\eta \sum_{m=1}^{M} L^m_{(G,\mathbf{y})}\right)$
    end
end
```

in Section 2.2 and Section 2.4.

3.5.1 SDE-GAN: Path Simulation of SDEs Using GANs

Path simulation of SDEs is a methodology to solve the SDEs numerically. It uses a time-discretization method to approximate the differential equations (or integrals) and applies Monte Carlo simulation to introduce randomness. Euler and Milstein schemes are popular path simulation methods, and many extensions have been proposed to improve the accuracy. Instead of applying mathematics, in recent years, deep-learning algorithms have been introduced to solve SDEs. In the following part, we focus on a GAN-based methodology proposed in [15], which we will refer to as SDE-GAN[3].

The SDE-GAN is applicable for path simulations of SDEs, which in principle can be used for all kinds of SDEs. [15] presents excellent results with respect to the CIR model, and we will describe the GBM simulation process in detail.

Suppose we want to simulate a path during time period $[0, T]$, with time partition $0 = t_0 < t_1 < \cdots < t_m = T$ and $\Delta t = \frac{T}{m}$. The path is denoted by $\{S(t_0), S(t_1), \dots, S(t_m)\}$. From the Markov property, $S(t_k)|S(t_{k-1})$ is independent of $\mathcal{F}(t_{k-1})$ for $k \in \{1, \dots, m\}$.

[3]SDE-GAN is the unsupervised conditional GAN in [15]

```
Algorithm 5: Modified training algorithm of vanilla and conditional GANs.
Input  : Extra information $\mathbf{y}$, training epochs $N$, mini-batch size $L$, mini-batch
         number $M$, initial discriminator parameters $\theta_0$, initial generator
         parameters $\eta_0$, discriminator iteration number $N_D$, GAN model index $\alpha$.
Output: Discriminator parameters $\theta$, generator parameters $\eta$.
for Training epoch $n = 1, \dots, N$ do
    for Mini-batch $m = 1, \dots, M$ do
        for Discriminator iteration $n_D = 1, \dots, N_D$ do
            Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
            Choose $L$ random samples $\left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\right\}$ from the training data.
            /* Discriminator loss. */
            $L^m_{(D,\alpha\mathbf{y})} \leftarrow \frac{1}{L}\sum_{l=1}^{L} -\left[\log D_\theta\left(\mathbf{x}^{(l)}, \alpha\mathbf{y}\right) + \log\left(1 - D_\theta\left(G_\eta\left(\mathbf{z}^{(l)}, \alpha\mathbf{y}\right)\right)\right)\right]$
            /* Update discriminator parameters. */
            $\theta \leftarrow \text{Adam}\left(\nabla_\theta \sum_{m=1}^{M} L^m_{(D,\alpha\mathbf{y})}\right)$
        end
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        /* Generator loss. */
        $L^m_{(G,\alpha\mathbf{y})} \leftarrow \frac{1}{L}\sum_{l=1}^{L} -\log\left(D_\theta\left(G_\eta\left(\mathbf{z}^{(l)}, \alpha\mathbf{y}\right)\right)\right)$
        /* Update generator parameters. */
        $\eta \leftarrow \text{Adam}\left(\nabla_\eta \sum_{m=1}^{M} L^m_{(G,\alpha\mathbf{y})}\right)$
    end
end
```

Therefore, the relation between $S(t_{k-1})$ and $S(t_k)$ can be learned as long as $S_{t_{k-1}}$ is given. A path then can be generated by repetitively learning the maps from $S(t_{k-1})$ to $S(t_k)$ for $k \in \{1, \dots, m\}$. Because of the stochasticity, the relation between $S(t_{k-1})$ and $S(t_k)$ forms a conditional distribution $P_{S(t_k)|S(t_{k-1})}$. Therefore, instead of generating one point, multiple samples are drawn from $P_{S(t_k)|S(t_{k-1})}$. Plenty of paths are then constructed by iteratively sampling from the distribution $P_{S(t_k)|S(t_{k-1})}$, given the initial $S(t_0) = S_0$. Figure 3.3 from [15] gives a sketch of simulated paths.

The machine learning method to approximate the conditional distribution $P_{S(t_k)|S(t_{k-1})}$ is a conditional GAN, i.e. the so-called SDE-GAN, and the learning process can be formulated by:

$$\begin{aligned} \hat{S}(t_1)|S(t_0) &= G_\eta(Z, \Delta t, S(t_0)), \\ \hat{S}(t_{k+1})|\hat{S}(t_k) &= G_\eta(Z, \Delta t, \hat{S}(t_k)), k = 1, \dots, m-1. \end{aligned} \tag{3.5.1}$$

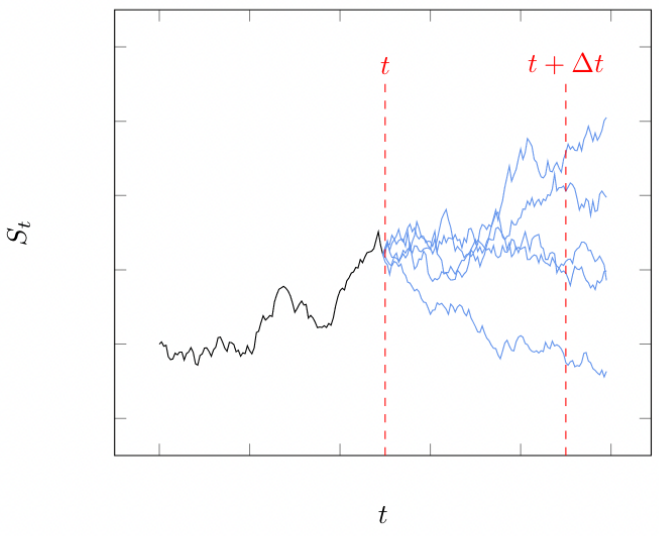


Figure 3.3: Illustration of path simulation [15].

where $Z \sim \mathcal{N}(0,1)$, $S(t_0) = S_0$ is the initial price, $\hat{S}(t_k)$ is the approximation of the exact $S(t_k)$ and G_η is the generator of SDE-GAN. In [15], SDE-GAN is trained on a dataset of tuples $((S(t_{k+1})|S(t_k), S(t_k), \Delta t)$ with varying Δt and $S(t_k)$, and thus Δt and $S(t_k)$ are set as extra information to the conditional GAN.

Path Simulation of GBM Using SDE-GAN
The methodology proposed in [15] can be summarized into five components: 1) Training dataset construction; 2) Data pre-processing; 3) SDE-GAN training; 4) Approximation; 5) Data post-processing. The schematic diagram of the methodology is illustrated in Figure 3.4.

The training dataset consists of N GBM paths with m time steps, denoted by $\{S(t_k)^i\}$, where $i = 1, \ldots, N$ and $k = 0, \ldots, m$. Each path starts at some fixed $S_0 \in \mathbb{R}$ and for any path i, $S(t_k)^i$ follows (2.2.11), that is,

$$S(t_k)^i = S_0 \exp\left(\left(\mu - \frac{1}{2}\sigma^2\right) t_k + \sigma W(t_k)\right). \tag{3.5.2}$$

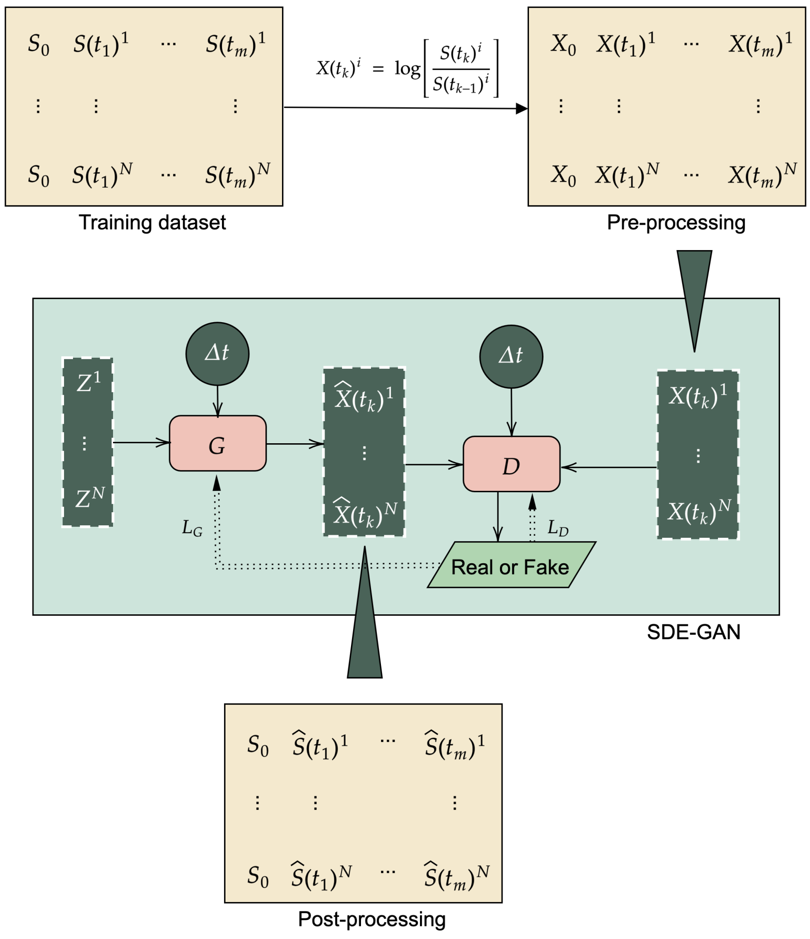
When training the conditional GAN, log-return transformation is applied to the dataset, formulated by

$$X(t_k)^i := \log\left[\frac{S(t_k)^i}{S(t_{k-1})^i}\right], \tag{3.5.3}$$

where $k = 1, \ldots, m$ and $X(t_0)^i := 0$.

Both generator G and discriminator D in the SDE-GAN are fully connected artificial neural networks, especially MLPs in [15]. Because of the data-preprocessing in [15], SDE-GAN is only conditioned on Δt regarding the GBM model. When SDE-GAN is well-trained following Algorithm 5, the log-price $X(t_k)^i$ is approximated by

$$\hat{X}(t_k)^i = G_\eta^*(Z, \Delta t), k = 1, \ldots, m \tag{3.5.4}$$



S_0 $S(t_1)^1$ ··· $S(t_m)^1$
S_0 $S(t_1)^N$ ··· $S(t_m)^N$
Training dataset
$X(t_k)^i = \log\left[\frac{S(t_k)^i}{S(t_{k-1})^i}\right]$
X_0 $X(t_1)^1$ ··· $X(t_m)^1$
X_0 $X(t_1)^N$ ··· $X(t_m)^N$
Pre-processing
Δt
Z^1
Z^N
G
$\widehat{X}(t_k)^1$
$\widehat{X}(t_k)^N$
D
$X(t_k)^1$
$X(t_k)^N$
L_G
L_D
Real or Fake
SDE-GAN
S_0 $\widehat{S}(t_1)^1$ ··· $\widehat{S}(t_m)^1$
S_0 $\widehat{S}(t_1)^N$ ··· $\widehat{S}(t_m)^N$
Post-processing

Figure 3.4: An overview of the methodology proposed in [15].

where G^*_η represents the well-trained generator and $Z \sim \mathcal{N}(0,1)$. The simulated price is then give by

$$\hat{S}(t_k)^i = \hat{S}(t_{k-1})^i \exp\left(\hat{X}(t_k)^i\right) = \hat{S}(t_{k-1})^i \exp\left(G^*_\eta(Z, \Delta t)\right). \tag{3.5.5}$$

Results
The performance of SDE-GAN for the GBM model is evaluated via two criteria: 1) The approximated conditional distribution $P_{S(t_k)|S(t_{k-1})}$ and 2) Comparison of simulated GBM paths to the exact solutions. [15] also presents the Monte Carlo error convergence compared with non-deep learning schemes. However, convergence issues of the SDE-GAN occur often in practice and the performance of SDE-GAN depends on Δt. It is therefore difficult to give definite statements about the quality of the convergence results.

Figure 3.5 illustrates the conditional distribution $P_{S(t_k)|S(t_{k-1})}$ learned by SDE-GAN, where $S(t_{k-1}) = S_0$ is fixed.

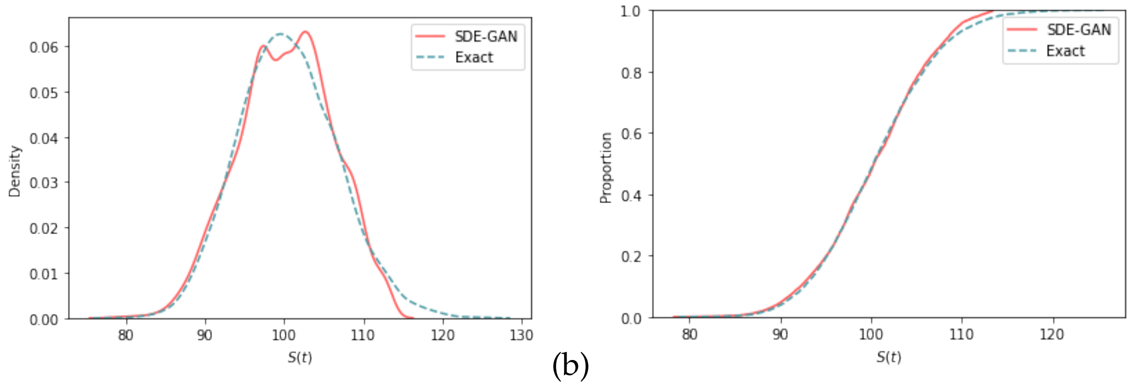


Figure 3.5: The conditional distribution $P_{S(t_k)|S(t_{k-1})}$ learned by SDE-GAN, compared with the exact solution. Left: the empirical probability density distribution function (EPDF) plot of $P_{S(t_k)|S(t_{k-1})}$; Right: the empirical cumulative distribution function (ECDF) plot. Here, we set $S_0 = 100$, $\Delta t = 0.1$, $\mu = 0.05$ and $\sigma = 0.2$.

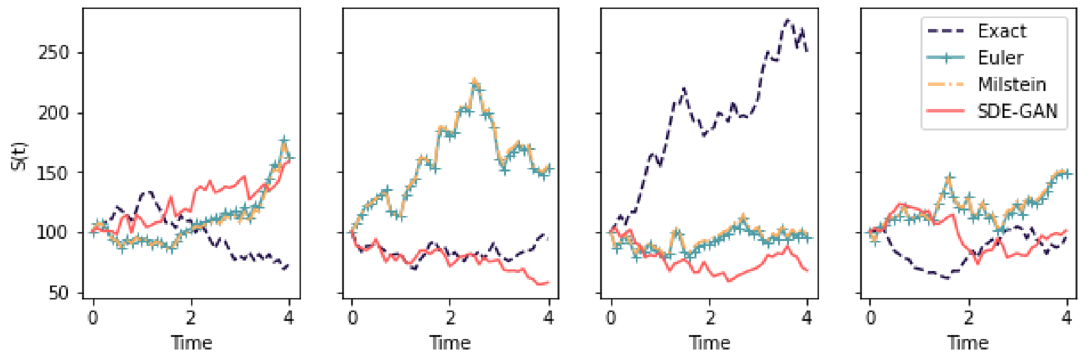


Figure 3.6: Four random paths generated by SDE-GAN, exact solution, Euler and Milstein schemes respectively, where $S(t_0) = 0$, $\Delta t = 0.1$, $T = 4$, $\mu = 0.05$ and $\sigma = 0.2$.

Figure 3.6 displays several GBM paths simulated by SDE-GAN, compared with the exact paths and paths constructed by Euler and Milstein schemes. As is highlighted in [15], SDE-GAN can only simulate weak solutions of the GBM model, that is, the generated path is not path-wise equivalent to the exact one.

From the results, we conclude that SDE-GAN can capture the relation between two adjacent GBM prices $S(t_{k-1})$ and $S(t_k)$, and can also simulate GBM paths well. For the SDE-GAN architecture and training details, we refer to Appendix A in [15].

The Pros and Cons
SDE-GAN displays a general methodology for path simulation of SDEs, which can be widely applied and easily adapted to various dynamics. With the advantages of GANs techniques, SDE-GAN can be extended to simulate higher-dimensional dynamics and overcome the curse of dimensionality.

Regarding the path simulation for the GBM model, in general, the performance of SDE-GAN is successful.

However, in practice, we find it quite difficult to train a stable SDE-GAN for the GBM model, which is also a common problem of GANs techniques. We will explain the problem deeply in Section 3.6.

3.5.2 AnoGAN: Anomaly Detection Using GANs

Another popular application of GANs is associated with anomaly detection, and in this section, we discuss a model called the AnoGAN [54; 56]. Since a GAN is trained to learn the distribution of the given dataset, and the generator can provide the learning result, i.e., an approximation of the real distribution, it is natural to introduce GANs to reconstruction based anomaly detection methods.

Given a dataset including normal and abnormal data, the AnoGAN is only trained on the normal data, and we denote the distribution of the normal data as $\mathbb{P}_N$. The generator G learns the push-forward map $G : \mathcal{Z} \mapsto \mathcal{X}$, and the distribution of generated samples $\mathbb{P}_G$ should be close to $\mathbb{P}_N$. On the other hand, if there exists a pull-back map $G^{-1} : \mathcal{X} \mapsto \mathcal{Z}$, which acts as an encoder, then for each sample $\mathbf{x}$ in the dataset, we can find the corresponding latent variable $\tilde{\mathbf{z}} \in \mathcal{Z}$. The generated result $G(\tilde{\mathbf{z}})$ is the reconstruction of $\mathbf{x}$.

Since the generator can only output the pattern of normal data, $G(\tilde{\mathbf{z}})$ should be a sample from $\mathbb{P}_N$ for any $\tilde{\mathbf{z}}$. For an anomaly $\mathbf{a}$, we find its corresponding latent variable $\tilde{z}_a = G^{-1}(\mathbf{a})$, the reconstruction $G(\tilde{z}_a)$ is non-anomalous. The anomalies can thus be detected by measuring the difference between the reconstruction result and the sample itself.

However, the difficulty is to find the pull-back map G^{-1} as we cannot directly compute the inverse map of the generator. Given a sample $\mathbf{x}$, [54] proposes an optimization algorithm to find a point $\mathbf{z}$ in the latent space such that the generated sample $G(\mathbf{z})$ is visually most similar to $\mathbf{x}$. The loss function of the optimization is defined as the weighted sum of a residual loss and a discriminator loss:

$$\begin{aligned} \mathcal{L}_R(\mathbf{z}) &= \|\mathbf{x} - G(\mathbf{z})\|, \\ \mathcal{L}_D(\mathbf{z}) &= \|\mathbf{f}(\mathbf{x}) - \mathbf{f}(G(\mathbf{z}))\|, \\ \mathcal{L}(\mathbf{z}) &= (1-\lambda)\mathcal{L}_R(\mathbf{z}) + \lambda \mathcal{L}_D(\mathbf{z}), \end{aligned} \tag{3.5.6}$$

where the residual loss $\mathcal{L}_R$ measures the difference between the given sample and the generated sample; The discriminator loss $\mathcal{L}_D$ represents the feedback from the discriminator; $\mathbf{f}$ is the output before the output layer of the discriminator; $\lambda \in (0, 1)$ is a constant.

The best latent variable $\tilde{\mathbf{z}}$ is found by minimizing the loss function $\mathcal{L}$ through back-propagation steps with respect to $\mathbf{z}$. And optimizers, such as the Adam, can be used to descend the gradients.

The anomaly score of the sample $\mathbf{x}$ is formulated by value of the loss function at $\tilde{\mathbf{z}}$:

$$A(\mathbf{x}) = \mathcal{L}(\tilde{\mathbf{z}}). \tag{3.5.7}$$

A threshold is decided based on the anomaly scores of all the samples, and if an

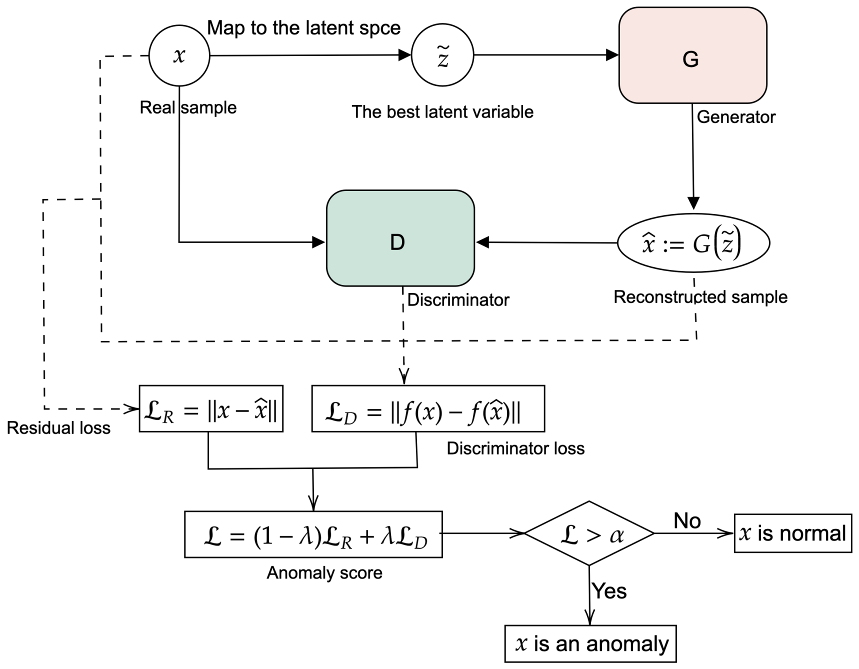


Figure 3.7: The methodology of AnoGAN, where $f(\cdot)$ is the output of an intermediate layer of the discriminator.

anomaly score is larger than the threshold, then the corresponding sample is an anomaly.

Figure 3.7 presents the methodology of AnoGAN, where the generator G and the discriminator D are well-trained on the normal data, and α is the threshold which is a constant.

3.6 GANs Problems and Improvements

Model training is one of the important stages in neural network construction. However, it is more difficult in GANs, since two competitive neural networks are involved and trained simultaneously. The equilibrium point is not easy to find or may not even exist. Sometimes, GANs training leads to poor performance, which is called GANs failure modes. Researchers would like to improve the GANs training stability.

In this section, we first identify GANs different failure modes by using SDE-GAN as examples. Next, various improvements are summarized besides the modifications in Algorithm 5.

3.6.1 GANs Problems

In general, GANs failure modes can be summarized into three aspects: 1) mode collapse, 2) vanishing gradients and 3) non-convergence. We describe these problems respectively in the following part, where parameters of SDE-GAN in examples are the same as Section 3.5.1.

Mode collapse usually relates to a generator which can only produce a small set of outputs. It happens when the generator learns an especially plausible distribution and only produce that output, then the best strategy for the discriminator is to reject. Therefore, the discriminator is trapped in this local minimum, resulting the generator oscillates in a small range. Figure 3.8 is an example of SDE-GAN mode collapse, where during the training, the generator can only generate prices near S_0.

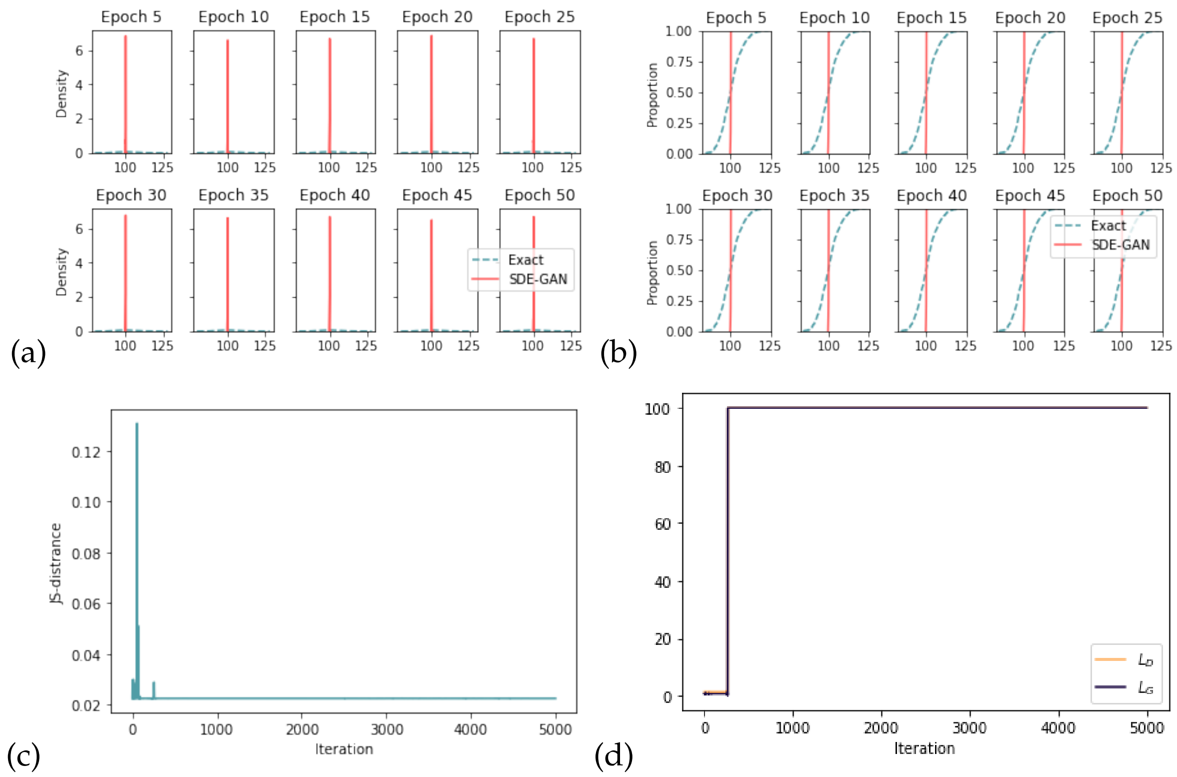


Figure 3.8: The Training process of SDE-GAN when SDE-GAN mode collapse. (a) The KDE plot of SDE-GAN generator output every five epochs; (b) The ECDF plot of generator output every five epochs; (c) The JS divergence between the generator output and the exact solution every network iteration; (d) The losses of generator and discriminator during training.

On the other hand, vanishing gradients is usually caused by the discriminator: When a discriminator performs too well, the generator then cannot receive useful information to make progress. The name vanishing gradients is associated with JS divergence. When the discriminator is close to optimal, it has the form of (3.2.5), and the minimax problem is reduced to

$$\min_{G_\eta} C\left(G_\eta\right) = -2\log 2 + 2\mathrm{JS}\left(P_{\text{data}} \| P_G\right). \tag{3.6.1}$$

When the generated distribution P_G is dissimilar to P_{data}, the slope of JS divergence is close to zero, as is illustrated in Figure 3.9. The generator therefore gets stuck and continuously produces bad outputs.

Diminishing gradients also appear when training SDE-GAN, see Figure 3.10. After training around 35 epochs, the generator learns nothing and the JS divergence between

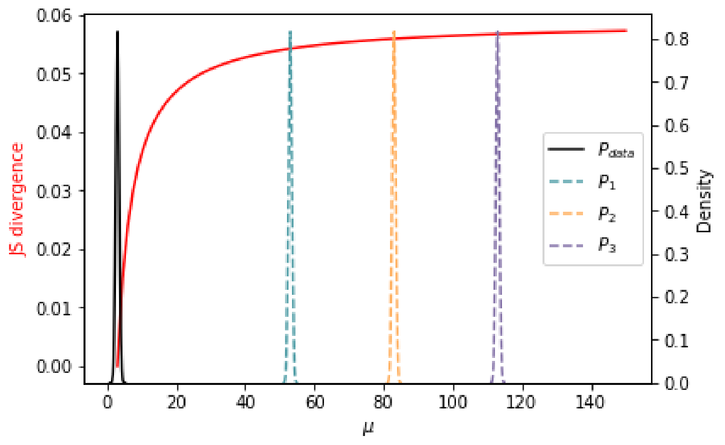


Figure 3.9: JS divergence between distributions P_n and P_{data} (the solid red line), where $P_{\text{data}} \sim \mathcal{N}(3, 0.5^2)$, $P_n \sim \mathcal{N}(\mu, 0.5^2)$ for $\mu \in [3, 150]$, in particular, $P_1 \sim \mathcal{N}(50, 0.5^2)$, $P_2 \sim \mathcal{N}(80, 0.5^2)$ and $P_3 \sim \mathcal{N}(110, 0.5^2)$.

P_{data} and P_G increases significantly and rapidly, indicating the occurrence of vanishing gradients.

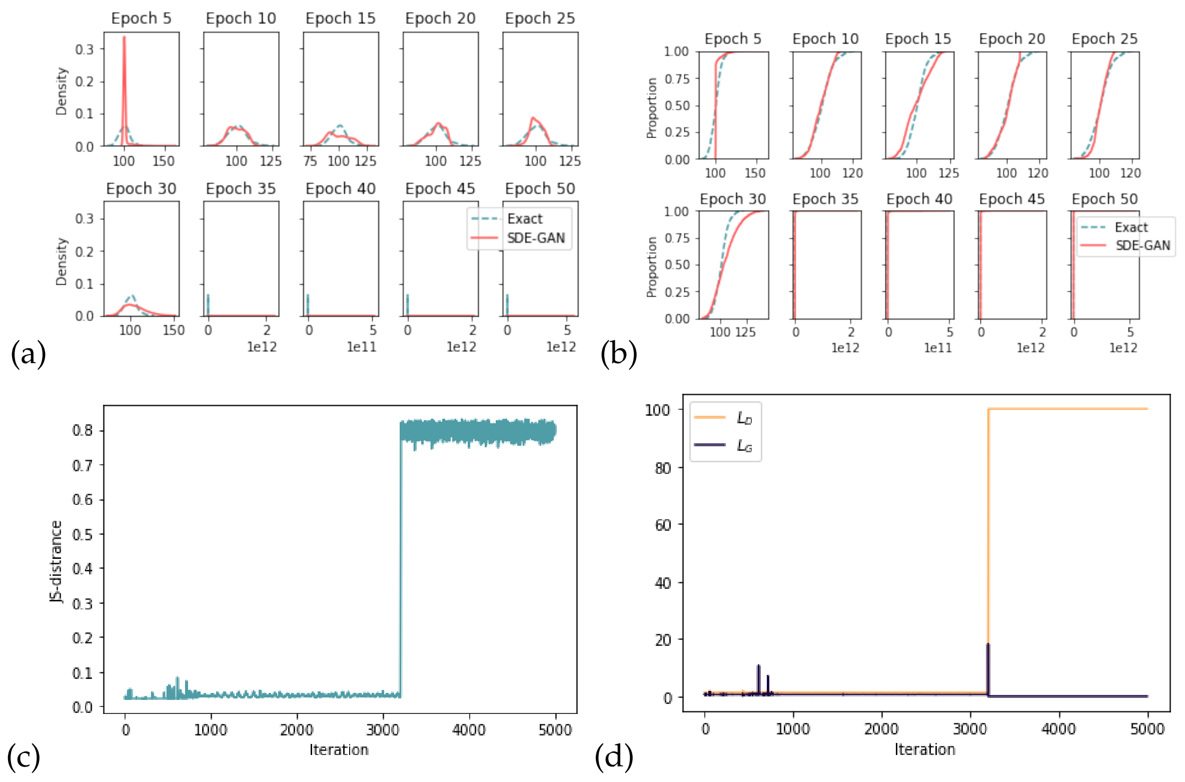


Figure 3.10: The Training process of SDE-GAN when vanishing gradients occurs. (a) The KDE plot of SDE-GAN generator output every five epochs; (b) The ECDF plot of generator output every five epochs; (c) The JS divergence between the generator output and the exact solution every network iteration; (d) The losses of generator and discriminator during training.

It is also common for GANs to fail to converge. In this situation, no equilibrium can be found between the discriminator and generator. One signal to identify GANs non-convergence failure is the discriminator loss decreasing to zero or close to zero. Because of that, the generator only provide low quality outputs which are easily be recognized as fake samples by the discriminator. For instance, in Figure 3.11, after training about 15 epochs, the discriminator loss decreases to zero and the generator produces samples near zero which is far from S_0.

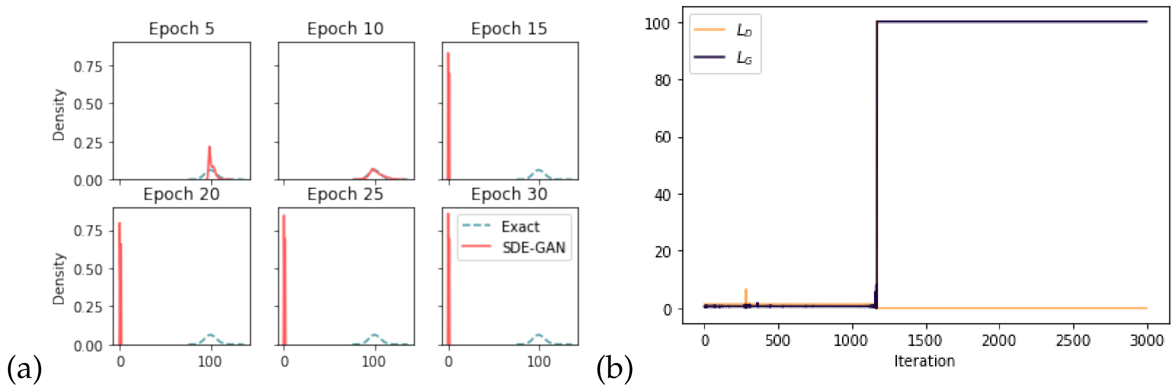


Figure 3.11: The Training process of SDE-GAN when SDE-GAN fails to converge. Left: The KDE plot of SDE-GAN generator output every five epochs; Right: The losses of generator and discriminator during training.

3.6.2 GANs Improvements

A variety of approaches have been proposed to avoid GANs failure modes. On the one hand, researchers adjust the training algorithm to enhance the stability of finding the equilibrium. For example, learning rate scheduling is used in [72], leading to faster convergence and higher accuracy. On the other hand, GANs architecture is modified, particularly in constructing different loss functions.

Among all kinds of GANs modifications, the so-called Wasserstein GAN (WGAN) is attractive, from which Wasserstein distance is introduced to replace JS divergence. The benefits of using Wasserstein distance is explained in [59]. Researches have shown that WGAN is a practical approach to remedy mode collapse and vanishing gradients.

Chapter 4

Wasserstein GAN

The Wasserstein GAN (WGAN) [59] is appealing as it offers a solution to remedy the GANs failure in vanishing gradients, which is crucial regarding our proposed framework. In this chapter, Wasserstein GAN is described thoroughly. First, we highlight the key difference between the GAN and WGAN, namely, the Wasserstein distance. Next, the architecture of the WGAN is described, explaining the details of how to involve the Wasserstein distance. After that, we list some related theoretical results, showing the usefulness of Wasserstein loss. In the next part, the training algorithm is illustrated, which is similar to GANs training, but with different parameter update criteria and additional constraints. However, as is mentioned in Section 3.6, the WGAN may still fail to converge. We then display an improved model, the so-called Wasserstein GAN with gradient penalty loss function (WGAN-GP).

4.1 Wasserstein Distance

Wasserstein distance has a close connection with optimal transport problems, which were first formalized by Monge in 1781 and developed significantly by Kantorovich in 1942. A natural motivation of the optimal transport problem is a source allocation problem: How to transfer raw materials from various warehouses to factories with different demands [73]. In mathematical terms, the so-called Kantorovich problem under probabilistic interpretation is defined by

$$\mathcal{L}_c(\mu,\nu) = \inf_{(X,Y)} \left\{ \mathbb{E}_{(X,Y)}[c(X,Y)] : X \sim \mu, Y \sim \nu \right\}, \tag{4.1.1}$$

where c is a continuous function, (μ,ν) are probability measures defined on compact spaces $(\mathcal{X},\mathcal{Y})$ and (X,Y) is a couple of random variables over the product space $\mathcal{X} \times \mathcal{Y}$. Moreover, if $\mathcal{X} = \mathcal{Y}$ and $c(x,y) = d(x,y)^p$ is a distance on $\mathcal{X}$, then

$$\mathcal{W}_p(\mu,\nu) \triangleq \mathcal{L}_{d^p}(\mu,\nu)^{1/p} \tag{4.1.2}$$

defines the p-Wasserstein distance on $\mathcal{X}$ [73]. In short, the Wasserstein distance is a distance metric defined between probability distributions on a given metric space $\mathcal{X}$.

4.1.1 1-Wasserstein Distance

We are interested in a special case of the p-Wasserstein metric, namely when $p = 1$ and $\mathcal{X} \in \mathbb{R}^n$, whose formula is given by:

$$W(\mu, \nu) = \inf_{(X,Y)} \left\{ \mathbb{E}_{(X,Y)}[\|X - Y\|] : X \sim \mu, Y \sim \nu \right\}. \tag{4.1.3}$$

The 1-Wasserstein distance is also called the earth's mover distance [74], which can be interpreted as the minimum energy cost of moving a pile of material from one form to another form (see Figure 4.1).

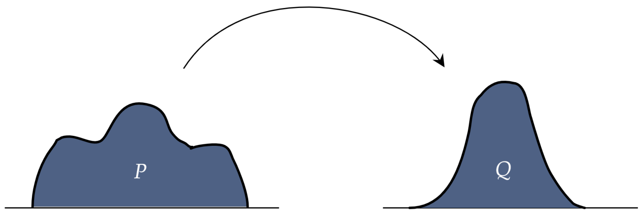


Figure 4.1: An interpretation of earth's mover distance or 1-Wasserstein distance.

In the discrete case, the 1-Wasserstein distance can be viewed as the minimum bin movement of transforming one histogram into another histogram. We refer to an example (see Figure 4.2) in [75] for details. Therefore, the 1-Wasserstein distance was introduced in computer vision and machine learning to compare histograms [73], and in this thesis, it is a choice to define the objective functions in GANs, leading to a more robust model, namely WGAN.

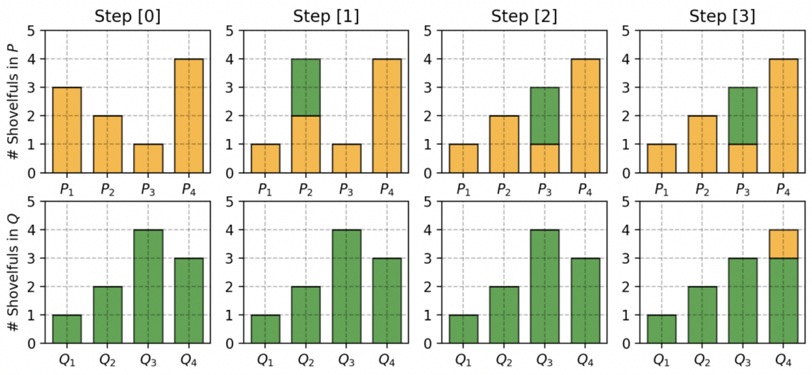


Figure 4.2: An example of the 1-Wasserstein distance in [75]. Here, the figure shows a step-by-step plan of matching two histograms P and Q, and the 1-Wasserstein distance between P and Q is 5.

4.1.2 Dual Problem

As the Kantorovich problem (4.1.1) is a constrained convex minimization problem, it admits a dual form which is a constrained concave maximization problem [73].

The dual problem of the 1-Wasserstein distance in (4.1.3) is formulated by:

$$W(\mu,\nu) = \sup_{\|\nabla f\|_\infty \leq 1} \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{x\sim\nu}[f(x)], \tag{4.1.4}$$

Here the 1-Lipschitz constraint $\|\nabla f\|_\infty \leq 1$ indicates that the norm of the gradient of f at any point x is upper bounded by 1. We refer to [73] for a thorough deduction. The dual problem is also called the Kantorovich-Rubenstein duality.

4.2 WGAN Architecture

In practice, it is intractable to exhaust all the joint possibilities (X, Y) to calculate the infimum in (4.1.3) [59]. Instead, the Kantorovich-Rubinstein duality (4.1.4) is used to train a GAN.

The objective function of a WGAN is constructed as follows:

$$\min_{G_\eta} \max_{D\in\mathcal{D}} \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim P_z}[D(G_\eta(z))], \tag{4.2.1}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. Note that the discriminator output $D(\mathbf{x})$ here does not signify the probability of real samples, and its value is not bounded by $[0,1]$. The discriminator, or the neural network used to distinguish the generated samples from the real samples, is called a critic in a Wasserstein GAN, and is denoted by f_w. The output of the critic is a scalar score, reflecting how real the input sample is.

To clearly show the difference between vanilla GANs and Wasserstein GANs, we rewrite the objective function (4.2.1) as:

$$\min_{G_\eta} \max_{w\in\mathcal{W}} \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[f_w(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim P_z}\left[f_w(G_\eta(\mathbf{z}))\right], \tag{4.2.2}$$

where $\{f_w\}_{w\in\mathcal{W}}$ is a parameterized family of 1-Lipschitz functions.

The loss functions of the critic f_w and the generator G are then given as follows:

$$L_C = -\left(\mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[f_w(\mathbf{x})] - \mathbb{E}_{\hat{\mathbf{x}}\sim P_G}\left[f_w(\hat{\mathbf{x}})\right]\right), \tag{4.2.3}$$

$$L_G = -\mathbb{E}_{\mathbf{z}\sim P_z}\left[f_w(G_\eta(\mathbf{z}))\right], \tag{4.2.4}$$

where $\hat{\mathbf{x}} := G_\eta(\mathbf{z})$ is the output of the generator G, following the generated distribution P_G.

Similar to Figure 3.1, we illustrate the architecture of a WGAN in Figure 4.3.

4.3 Theoretical Results

In this section that is heavily inspired by [59], we describe several mathematical results to show it is reasonable to use the 1-Wasserstein distance as compared to the JS

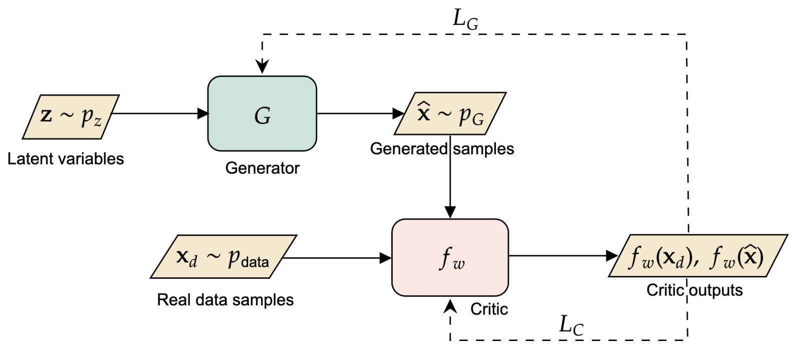


Figure 4.3: A high-level overview of WGAN.

divergence in GANs modeling.

Perhaps the most important property of the Wasserstein distance is that it is a weak distance [73]. Because of this, under some mild assumptions, we can state the continuity and differentiability of the 1-Wasserstein distance. The mathematical results are illustrated as follows, and we refer to Appendix C in [59] for the detailed proofs.

Assumption 4.3.1 (Regularity assumption)**.** *Let $G_\eta(z,\eta) : \mathcal{Z} \times \mathbb{R}^d \mapsto \mathcal{X}$ be a function whose output value is denoted by $G_\eta(z)$. We say that G satisfies a regularity assumption for a certain probability distribution P_z over $\mathcal{Z}$ if there are local Lipschitz constants $L(\eta,z)$ such that $\mathbb{E}_{z\sim P_z}[L(\eta,z)] < +\infty$.*

Theorem 4.3.2. *Let P_{data} be a fixed distribution over $\mathcal{X}$. Let $z \sim P_z$ be a random variable over another space $\mathcal{Z}$. Let $G_\eta(z,\eta) : \mathcal{Z} \times \mathbb{R}^d \mapsto \mathcal{X}$ be a function whose value is denoted by $G_\eta(z)$. Let P_G denote the distribution of $G_\eta(\mathcal{Z})$. Then*

1. *If G_η is continuous in η, so is $W(P_{data}, P_G)$, where $W(P_{data}, P_G)$ follows* (4.1.3).
2. *If G_η is locally Lipschitz and satisfies the regularity assumption, then $W(P_{data}, P_G)$ is continuous everywhere, and differentiable almost everywhere.*
3. *Statements 1-2 are false for $JS(P_{data}||P_G)$.*

Corollary 4.3.3. *Let G be any feedforward neural network parameterized by η, and P_z a latent distribution satisfying $\mathbb{E}_{z\sim P_z}[||z||] < \infty$. Then, $W(P_{data}, P_G)$ is continuous everywhere and differentiable almost everywhere.*

From the results above, we can also conclude that the 1-Wasserstein distance seems to make more sense than the JS divergence, which explains the practical use of the WGAN. Additionally, we reuse Figure 3.9 to display the sensibility, see Figure 4.4. We can see that in the region where the gradients based on the JS divergence vanish, the 1-Wasserstein distance still admits strong gradients.

Next, we would like to discuss the existence of WGANs' global optimality. Again, we refer to [59] for the proof details.

Theorem 4.3.4. *Let P_{data} be any distribution. Let P_G be the distribution of $G_\eta(z)$ with z a*

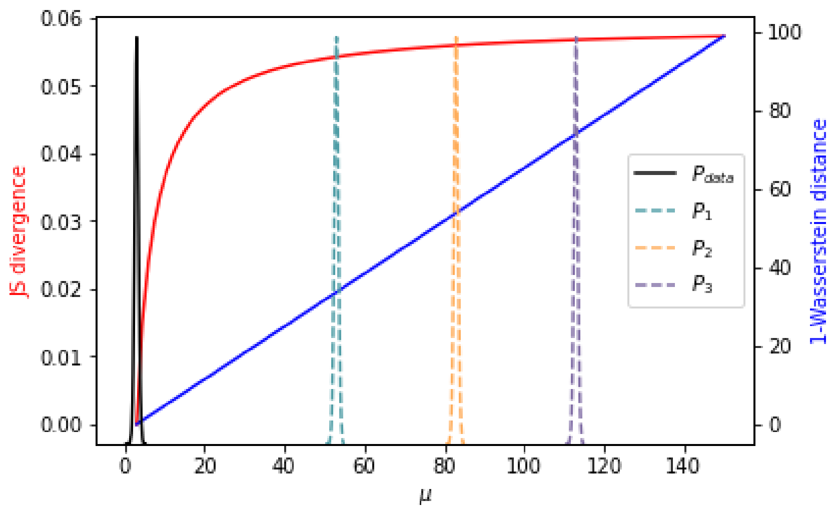


Figure 4.4: Reuse Figure 3.9 by adding the 1-Wasserstein distance between P_n and P_{data} (the solid blue line).

random variable following distribution P_z and G_η the satisfying regularity assumption. Then, there is a solution $f : \mathcal{X} \mapsto \mathbb{R}$ to the problem

$$\max_{\|\nabla f\|_\infty \le 1} \mathbb{E}_{x\sim P_{data}}[f(x)] - \mathbb{E}_{x\sim P_G}[f(x)] \tag{4.3.1}$$

and we have

$$\nabla_\eta W(P_{data}, P_G) = -\mathbb{E}_{z\sim P_z}\left[\nabla_\eta f(G_\eta(z))\right] \tag{4.3.2}$$

if both terms are well-defined.

Last but not least, we show some properties of the optimal critic in Theorem 4.3.4, and we refer to [60] for the details of proofs.

Proposition 4.3.5. *Let f^* be the solution of problem* (4.3.1). *Let π be the optimal coupling between P_{data} and P_G, defined as the minimizer of*

$$W(P_{data}, P_G) = \inf_{\pi\in\Pi(P_{data},P_G)} \mathbb{E}_{(x,y)\sim\pi}[\|x-y\|],$$

where $\Pi(P_{data}, P_G)$ is the set of joint distributions $\pi(x,y)$ whose marginals are P_{data} and P_G, respectively. Then, if f^* is differentiable, $\pi(x=y)=0$, and $x_t = tx + (1-t)y$ with $0 \le t \le 1$, we have

$$P_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y-x_t}{\|y-x_t\|}\right] = 1.$$

Corollary 4.3.6. *f^* has gradient norm 1 almost everywhere under P_{data} and P_G.*

4.4 Training Algorithm

Theorem 4.3.4 guarantees the existence of an optimal critic. Similar to GANs, we can train neural networks to approximate the solution of the Wasserstein GAN. Since the 1-

Wasserstein distance is continuous and differentiable almost everywhere, we can and we should train the critic till the optimality. That is, for every training iteration, the critic parameters are updated more frequently than the generator parameters.

The training structure of WGANs is similar to Algorithm 5. However, because of the Lipschitz constraint, a weight clipping method is applied to enforce the condition. It simply restricts the critic weights after each gradient update by setting a weight ceiling and a weight floor. Furthermore, the RMSProp optimizer is used in WGAN, which empirically performs better than the Adam optimizer. The detailed training algorithm of WGANs is illustrated in Algorithm 6.

Algorithm 6: Training algorithm of Wasserstein GAN, proposed in [59].

```
Input : Training epochs $N$, mini-batch size $L$, mini-batch number $M$, initial critic
        parameters $w_0$, initial generator parameters $\eta_0$, critic iteration number $N_c$,
        weight clipping parameter $c$.
Output: Critic parameters $w$, generator parameters $\eta$.
for Training epoch $n = 1, \dots, N$ do
    for Mini-batch $m = 1, \dots, M$ do
        for Critic iteration $n_c = 1, \dots, N_C$ do
            Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
            Choose $L$ random samples $\left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\right\}$ from the training data.
            /* Critic loss.                                                    */
            $L_C^m \leftarrow -\left[\frac{1}{L}\sum_{l=1}^{L} f_w\left(\mathbf{x}^{(l)}\right) - \frac{1}{L}\sum_{l=1}^{L} f_w\left(G_\eta\left(\mathbf{z}^{(l)}\right)\right)\right]$
            /* Update critic parameters.                                       */
            $w \leftarrow \text{RMSProp}\left(\nabla_w \sum_{m=1}^{M} L_C^m\right)$
            /* Weight clipping                                                 */
            $w \leftarrow \text{clip}\,(w, -c, c)$
        end
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        /* Generator loss.                                                     */
        $L_G^m \leftarrow -\frac{1}{L}\sum_{l=1}^{L} f_w\left(G_\eta\left(\mathbf{z}^{(l)}\right)\right)$
        /* Update generator parameters.                                        */
        $\eta \leftarrow \text{RMSProp}\left(\nabla_\eta \sum_{m=1}^{M} L_G^m\right)$
    end
end
```

4.5 WGAN Improvement: WGAN-GP

The authors in [59] admit that weight clipping is not a satisfactory method to satisfy the Lipschitz condition and when the clipping parameters are not carefully chosen, the

WGAN easily fails to provide promising performance, resulting in vanishing gradients when the clipping range is too small or slower convergence when the range is too big [75].

An alternative method to enforce the Lipschitz constraint is proposed in [60]. The authors of the so-called gradient penalty method propose to add an additional term to the critic loss (4.2.3), which is formulated as follows:

$$L_C = \underbrace{\mathbb{E}_{\hat{\mathbf{x}}\sim P_G}[f_w(\hat{\mathbf{x}}))] - \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[f_w(\mathbf{x})]}_{\text{Original critic loss}} + \underbrace{\lambda\mathbb{E}_{\tilde{x}\sim\ P_{\tilde{x}}}[(\|\nabla_{\tilde{x}} f_w(\tilde{x})\|_2 - 1)^2]}_{\text{gradient penalty}}, \tag{4.5.1}$$

where $P_{\tilde{x}}$ samples uniformly along the straight lines between pairs (x, y) with any $x \sim P_{\text{data}}$ and $y \sim P_G$; $\lambda \in \mathbb{R}$ is a hyperparameter which is a constant. The WGAN with gradient penalty added to the critic loss is called WGAN-GP for short.

As reason to construct such penalty term is as follows: First, a differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere; Second, from Proposition 4.3.5, the optimal critic is with gradient norm 1 on the straight lines between pairs of the points that are sampled from P_{data} and P_G. Therefore, the gradient penalty term in (4.5.1) not only provides a flexible constraint associated with inputs, but also results in highly satisfactory performance [60].

We illustrate the training process of a WGAN-GP in Algorithm 7. Note that Adam optimizer is applied in training the WGAN-GP. Actually, there is no theoretical criterion for choosing the proper optimization algorithm, and which to use is based on empirical results.

Experiments in [60] confirm that the WGAN-GP is more stable, which allows the construction of more complicated neural networks. WGAN-GP is accepted as a state-of the-art GAN variant in various fields for proposing a robust model. In this thesis, we also replace the training structure of the SDE-GAN by a WGAN-GP (see Chapter 5), and we find that no GANs failure mode occurs anymore during training.

Algorithm 7: Training algorithm of WGAN-GP, proposed in [60].

```
Input  : Training epochs $N$, mini-batch size $L$, mini-batch number $M$, initial critic
         parameters $w_0$, initial generator parameters $\eta_0$, critic iteration number $N_c$,
         gradient penalty coefficient $\lambda$.
Output: Critic parameters $w$, generator parameters $\eta$.
for Training epoch $n = 1, \dots, N$ do
    for Mini-batch $m = 1, \dots, M$ do
        for Critic iteration $n_c = 1, \dots, N_C$ do
            Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
            Choose $L$ random samples $\left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\right\}$ from the training data.
            /* Gradient penalty construction.                                  */
            Sample $L$ random numbers $\left\{\epsilon^{(1)}, \dots, \epsilon^{(L)}\right\}$ from uniform distribution
             $\mathcal{U}[0, 1]$.
            $\hat{\mathbf{x}}^{(l)} \leftarrow G_\eta\left(\mathbf{z}^{(l)}\right)$, for $l = 1, \dots, L$
            $\tilde{\mathbf{x}}^{(l)} \leftarrow \epsilon \mathbf{x}^{(l)} + (1 - \epsilon)\hat{\mathbf{x}}^{(l)}$, for $l = 1, \dots, L$
            /* Critic loss.                                                    */
            $L_C^m \leftarrow -\left[\frac{1}{L}\sum_{l=1}^{L} f_w\left(\mathbf{x}^{(l)}\right) - \frac{1}{L}\sum_{l=1}^{L} f_w\left(\hat{\mathbf{x}}^{(l)}\right)\right] + \lambda\left(\|\nabla_{\tilde{\mathbf{x}}} f_w(\tilde{\mathbf{x}}^{(l)})\| - 1\right)^2$
            /* Update critic parameters.                                       */
            $w \leftarrow \text{Adam}\left(\nabla_w \sum_{m=1}^{M} L_C^m\right)$
        end
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        /* Generator loss.                                                     */
        $L_G^m \leftarrow -\frac{1}{L}\sum_{l=1}^{L} f_w\left(G_\eta\left(\mathbf{z}^{(l)}\right)\right)$
        /* Update generator parameters.                                        */
        $\eta \leftarrow \text{Adam}\left(\nabla_\eta \sum_{m=1}^{M} L_G^m\right)$
    end
end
```

Chapter 5

Proposed Framework

In this thesis, the achievement is to propose a framework to simulate a jump-diffusion process by using GAN techniques. As the paths simulation of the jump-diffusion process consists of the diffusion part and the jump part, the framework can be divided into two parts: 1) diffusion learning and 2) jump detection.

Regarding the diffusion learning part, we first suppose that the jump information (for example, the jump size, the jump direction and the jump instances) is known. Our goal is to simulate the diffusion part, and the jump-diffusion paths are then generated by adding the jumps additionally. We propose an improved SDE-GAN to achieve the simulation, called SDE-WGAN.

Concerning the jump detection part, we introduce the AnoGAN to recognize jumps: Given a jump-diffusion path, we view the jumps as anomalies, while the non-jump prices (i.e., from the diffusion part) are the normal data. The jumps are then detected via a GAN-based anomaly detection method, namely, the AnoGAN. According to the detected jump samples, we can use the maximum likelihood estimation method (MLE) to approximate the parameters of the jumps.

By combining the learned diffusion part and the estimated jumps, a general framework for generating a jump-diffusion process is illustrated in Figure 5.1. We highlight that the generator and the critic are the well-trained neural networks from the diffusion learning part; that is, we only run the training algorithm of the SDE-WGAN once.

In this chapter, we describe the proposed framework part by part. It starts with a new idea about constructing the dataset, from which a so-called nested Monte Carlo method is applied. Next, a general GAN model of the SDE-WGAN is illustrated in detail. It is a conditional Wasserstein GAN with gradient penalty, which combines the ideas of the conditional GAN and the WGAN-GP. We then present the process for the diffusion learning part, that is, the Monte Carlo path simulation of the GBM model using the SDE-WGAN. We further simulate the jump-diffusion model with the assumed jump information. In the rest, we suppose the jump parameters are unknown and design a jump detection model to approximate the parameters. The jump detection part is further structured by detecting jumps and estimating jump information, where the SDE-WGAN combined with the AnoGAN methodology is applied to recognize jumps, and the MLE method is used to estimate jump parameters.

5.1 3D Dataset Construction

When using the SDE-GAN to simulate the GBM paths, the training dataset consists of n Monte Carlo simulation paths with m time steps, and the authors of [15] set $n = 10^5$. Therefore, it is a 2-dimensional dataset (2D dataset) with 10^5 GBM paths. In this thesis, we are interested in a new idea of constructing a 3-dimensional dataset (3D-dataset), based on the nested Monte Carlo simulation [76; 77]. Figure 5.3 displays the structures of a 2D dataset and a 3D dataset, respectively.

A sample $S(t_i)^j_k$ in the 3D dataset has three indexes: timestamp t_i, path j, and depth k. When using the nested Monte Carlo method to generate the GBM samples, two layers of simulation are looped: The outer layer simulates independent paths, which is the same as generating the 2D dataset; At each step of a path, the inner layer generates independent samples based on the former step. The inner samples usually locate in the neighborhood of the current state, formulating a distribution at the current timestamp. To simulate a 3D-dataset $S(t_i)^j_k$ with m time steps, n paths and d depths, the outer layer simulation is expressed by

$$S(t_i)^j_1 = f(S(t_{i-1})^j_1), \quad i = 1, \dots, m \text{ and } j = 1, \dots, n, \tag{5.1.1}$$

and the inner layer loop is given by

$$S(t_i)^j_k = f(S(t_{i-1})^j_1), \quad k = 1, \dots, d, \tag{5.1.2}$$

where f represents a discretization scheme to approximate the GBM dynamics.

For example, under the Euler scheme,

$$f(s_i) := s_{i-1} + \mu s_{i-1}\Delta t + \sigma s_{i-1}\Delta t Z,$$

where μ, σ are constants, Δt is the time step size and $Z \sim \mathcal{N}(0, 1)$ is a random variable.

Example 5.1.1 (2D dataset and 3D dataset for the GBM samples). *We present an example of the 2D and 3D datasets for the GBM samples and discover the convergence of the 3D dataset regarding the time step and the depth (see Figure 5.3).*

From the error convergence plots in Example 5.1.1, we conclude that the 3D dataset contains more accurate samples under the Euler scheme. On the other hand, increasing the depth cannot significantly decrease the approximation error.

In training practice, we also enjoy a speedup using the 3D dataset: fewer paths are required for sufficient training due to the storage of samples in the depth dimension.

5.2 Conditional Wasserstein GAN with Gradient Penalty

In this section, we describe the general GAN model of the SDE-WGAN, that is, a conditional Wasserstein GAN with gradient penalty (cWGAN-GP). Like the conditional GAN being an extension of the vanilla GAN, the cWGAN-GP is a Wasserstein GAN with gradient penalty, whose generator and critic receive auxiliary information as extra inputs. The section is structured as follows: First, we present the architecture of the cWGAN-GP, including the formula of loss functions; Next, we illustrate the training algorithm in detail.

5.2.1 cWGAN-GP Architecture

Similar to the extension idea about the conditional GAN proposed in [70], the cWGAN-GP is a naturally extended conditional model based on the WGAN-GP, where some extra information $\mathbf{y}$ is added to both the generator and critic.

Extending (4.2.2), the objective function of the cWGAN-GP is given by:

$$\min_{G_\eta} \max_{w \in \mathcal{W}} \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[f_w(\mathbf{x}, \mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim P_z}\left[f_w(G_\eta(\mathbf{z}, \mathbf{y}), \mathbf{y})\right]. \tag{5.2.1}$$

The loss functions of the generator and the critic are then formulated by

$$L_{(G,\mathbf{y})} = -\mathbb{E}_{\mathbf{z}\sim P_z}[f_w(G_\eta(\mathbf{z}), \mathbf{y})], \tag{5.2.2}$$

$$L_{(f_w,\mathbf{y})} = \mathbb{E}_{\hat{\mathbf{x}}\sim P_G}[f_w(\hat{\mathbf{x}}, \mathbf{y}))] - \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[f_w(\mathbf{x}, \mathbf{y})] + \underbrace{\lambda \mathbb{E}_{\tilde{x}\sim P_{\tilde{x}}}[(\|\nabla_{(\tilde{x},\mathbf{y})} f_w(\tilde{x}, \mathbf{y})\|_2 - 1)^2]}_{\text{gradient penalty}}, \tag{5.2.3}$$

where $\hat{\mathbf{x}} := G_\eta(\mathbf{z}, \mathbf{y})$ denotes the output of the generator with distribution $\mathbb{P}_{(G,\mathbf{y})}$. Regarding the Lipschitz constraints of the conditional loss functions, there is no deep mathematical justification, however, we refer to [78], a short note about the regularity of the critic. Figure 5.4 presents a high-level architecture of the cWGAN-GP.

5.2.2 cWGAN-GP in Practice

When building the architecture of the cWGAN-GP, the conditions $\mathbf{y} = \{y_1, y_2, \dots\}$ are concatenated to the input neurons of both the generator and critic, and Figure 5.5 displays a simple example about the training structure of the cWGAN-GP, where both the generator and critic are MLPs. The detailed training process of the cWGAN-GP is illustrated in Algorithm 8. Note that here we use the RMSProp method for the gradient descent, which performs better than the Adam in practice.

5.3 Monte Carlo Simulation Using SDE-WGAN

As is discussed in Section 3.5.1, the SDE-GAN often fails to learn the GBM dynamics in practice, and mainly because of the GAN mode collapse and vanishing gradients during the training. To improve the training stability and the simulation's robustness, we propose a so-called SDE-WGAN, where a cWGAN-GP replaces the GAN architecture in the SDE-GAN.

This section is associated with the diffusion learning part of the proposed framework. First, we describe the structure of the SDE-WGAN thoroughly. Then, we use the SDE-WGAN to simulate the log-GBM dynamics corresponding to the diffusion part in (2.2.2), namely

$$\mathrm{d}X(t) = \mu \mathrm{d}t + \sigma \mathrm{d}W^{\mathbb{P}}(t). \tag{5.3.1}$$

Furthermore, we use the provided jump information, for example the jump intensity

```
Algorithm 8: Training algorithm of cWGAN-GP.
Input : Training epochs $N$, mini-batch size $L$, mini-batch number $M$, initial critic
        parameters $w_0$, initial generator parameters $\eta_0$, critic iteration number $N_c$,
        gradient penalty coefficient $\lambda$, extra information $\mathbf{y}$.
Output: Critic parameters $w$, generator parameters $\eta$.
for Training epoch $n = 1, \dots, N$ do
    for Mini-batch $m = 1, \dots, M$ do
        for Critic iteration $n_c = 1, \dots, N_C$ do
            Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
            Choose $L$ random samples $\left\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(L)}\right\}$ from the training data.
            /* Gradient penalty construction. */
            Sample $L$ random numbers $\left\{\epsilon^{(1)}, \dots, \epsilon^{(L)}\right\}$ from uniform distribution
             $\mathcal{U}[0,1]$.
            $\hat{\mathbf{x}}^{(l)} \leftarrow G_\eta\left(\mathbf{z}^{(l)}, \mathbf{y}\right)$, for $l = 1, \dots, L$
            $\tilde{\mathbf{x}}^{(l)} \leftarrow \epsilon \mathbf{x}^{(l)} + (1-\epsilon)\hat{\mathbf{x}}^{(l)}$, for $l = 1, \dots, L$
            /* Critic loss. */
            $L^m_{f_w} \leftarrow$
             $-\left[\frac{1}{L}\sum_{l=1}^{L} f_w\left(\mathbf{x}^{(l)}, \mathbf{y}\right) - \frac{1}{L}\sum_{l=1}^{L} f_w\left(\hat{\mathbf{x}}^{(l)}, \mathbf{y}\right)\right] + \lambda\left(\|\nabla_{(\tilde{\mathbf{x}}, \mathbf{y})} f_w(\tilde{\mathbf{x}}^{(l)}, \mathbf{y})\| - 1\right)^2$
            /* Update critic parameters. */
            $w \leftarrow \text{RMSProp}\left(\nabla_w \sum_{m=1}^{M} L^m_{f_w}\right)$
        end
        Generate $L$ random latent variables $\left\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(L)}\right\}$ from distribution $P_z$.
        /* Generator loss. */
        $L^m_G \leftarrow -\frac{1}{L}\sum_{l=1}^{L} f_w\left(G_\eta\left(\mathbf{z}^{(l)}, \mathbf{y}\right), \mathbf{y}\right)$
        /* Update generator parameters. */
        $\eta \leftarrow \text{RMSProp}\left(\nabla_\eta \sum_{m=1}^{M} L^m_G\right)$
    end
end
```

and the distribution of the jump size, to generate the paths under the dynamics (2.2.2).

5.3.1 SDE-WGAN

The SDE-WGAN is adapted from the SDE-GAN, which aims to improve the model stability and robustness during the training. Compared to the SDE-GAN, the modifications of the SDE-WGAN are mainly related to three components: First and most importantly, the cWGAN-GP is in place of the conditional GAN to learn the GBM dynamics; Second, two conditions, the time step Δt and the precious state $X(t_{i-1})$ are added when simulating $X(t_i)$; Third, the pre-and post-processing is eliminated, and the log-prices are directly input as the real samples.

As the SDE-WGAN is essentially a cWGAN-GP, it is trained by following Algorithm 8. We construct a 3D dataset $\{X(t_i)_k^j\}$ for the sufficient training, where $i = 0, 1, \ldots, m$, $j = 1, \ldots, n$ and $k = 1, \ldots, d$. During the training, both the generator and critic are conditioned by the extra information $\mathbf{y} := (\Delta t, X(t_{i-1})_1, \ldots, X(t_{i-1})_d)$. The generator learns a map from a d-dimensional latent variable $\mathbf{Z} := (Z_1, \ldots, Z_d)$ combining the conditions $\mathbf{y}$ and outputs a d-dimensional approximation $\hat{\mathbf{X}}(t_i) = \big(\hat{X}(t_i)_1, \ldots, \hat{X}(t_i)_d\big)$. The critic gives the feedback by judging the real sample $\mathbf{X}(\mathbf{t_i}) := (X(t_i)_1, \ldots, X(t_i)_d)$ and the fake sample $\hat{\mathbf{X}}(t_i)$. The detailed structure of the SDE-WGAN is illustrated in Figure 5.6, where both the generator and critic are multilayer perceptrons.

5.3.2 GBM Path Simulation

Suppose to simulate a GBM log-price path $\{X(t_0), X(t_1), \ldots, X(t_m)\}$. The dynamics of $\mathrm{d}X(t)$ are

$$\mathrm{d}X(t) = \mu \mathrm{d}t + \sigma \mathrm{d}W^{\mathbb{P}}(t), \text{ with } X(0) = \log S(0), \tag{5.3.2}$$

and the time discretization formula under the Euler scheme is given by

$$X(t_i) = X(t_{i-1}) + \mu\Delta t + \sigma\sqrt{\Delta t}Z, \tag{5.3.3}$$

where $Z \sim \mathcal{N}(0,1)$, the time period is $[0, T]$ and the time step $\Delta t := t_i - t_{i-1} = \frac{T}{m}$.

Similar to Section 3.5.1, the SDE-WGAN is trained to learn the relation between the previous state $\mathbf{X}(t_{i-1})$ and the current conditional state $\mathbf{X}(t_i)$. More precisely, the SDE-WGAN approximates the conditional distribution $\mathbb{P}_{\mathbf{X}(t_i)|\mathbf{X}(t_{i-1})}$. Note that $\mathbf{X}(t_i) := (X(t_i)_1, \ldots, X((t_i)_d))$, where d is the depth of the 3D training dataset. The GBM log-price path is then generated by iteratively sampling points $\mathbf{X}(t_i) \in \mathbb{R}^d$ from the approximated conditional distributions.

The mathematical formulas regarding the path simulation are given by:

$$\begin{aligned} \hat{\mathbf{X}}(t_1) &= G_\eta^*(\mathbf{Z}, \Delta t, \mathbf{X}(t_0)), \\ \hat{\mathbf{X}}(t_i) &= G_\eta^*(\mathbf{Z}, \Delta t, \hat{\mathbf{X}}(t_{i-1})), \quad k = 2, \ldots, m, \end{aligned} \tag{5.3.4}$$

where $\mathbf{Z}$ consists of d i.i.d random variables Z_k sampled from the standard normal distribution, $\mathbf{X}(t_0) = (\log(S0), \ldots, \log(S0))$ is the fixed initial log-price vector, G_η^* is the well-trained generator and $\hat{\mathbf{X}}(t_i)$ is the approximation by the SDE-WGAN.

Therefore, the Monte Carlo paths generated by the SDE-WGAN are also "nested", that is, the generated samples are also three-dimensional.

5.3.3 Jump-Diffusion Path Simulation

Recall in Section 2.2.2, the Monte Carlo jump-diffusion paths $\{X_J(t_i)\}_{i=0}^m$ are simulated by adding extra jump components to the GBM log-price paths $\{X(t_i)\}_{i=0}^m$. Moreover, regarding the jump instances, the Bernoulli random variables are introduced to simulate the jump occurrences.

In the diffusion learning part, we still approach the same simulation method for the jump component: First, at each timestamp t_i, a Bernoulli random variable $B(t_i)$ with the jump intensity parameter λ_p is generated to decide the occurrence of a jump; Second, the jump size J is randomly sampled from the assumed distribution, for example, J follows a normal distribution in Merton's model; Next, the jump size J is added to the diffusion part if $B(t_i) = 1$.

Combining the diffusion component simulated by the SDE-WGAN, the jump-diffusion path simulation is given by

$$\begin{aligned} \hat{\mathbf{X}}_J(t_1) &= G^*_\eta(\mathbf{Z}, \Delta t, \mathbf{X}_J(t_0)) \\ \hat{\mathbf{X}}_J(t_i) &= G^*_\eta(\mathbf{Z}, \Delta t, \hat{\mathbf{X}}_J(t_{i-1})) + JB(t_i), i = 2, \ldots, m, \end{aligned} \tag{5.3.5}$$

where $\mathbf{X}_J(t_0)$ is the initial state satisfying $\mathbf{X}_J(t_0) = \mathbf{X}(t_0)$. Note that the jump component is added to each component of the vector $G^*_\eta(\mathbf{Z}, \Delta t, \hat{\mathbf{X}}(t_{i-1}))$. We illustrate a schematic diagram for a clear description of the simulation process, see Figure 5.7.

5.4 Jump Detection Model

In the diffusion learning part, we propose an SDE-WGAN, giving a general methodology for simulating the diffusion part: The parameters μ and σ, which are usually implied by the market data, are not required. On the other hand, the jump information is usually also unknown. In this section, we discuss the second part of the proposed framework, from which a GAN-based anomaly detection model is introduced to figure out the jump information.

In the following part, we assume that no jump occurs at the initial time t_0, and our goal is to recognize the jumps in a jump-diffusion path $\{X(t_0), X(t_1), \ldots, X(t_m)\}$. To achieve the goal, we first introduce an AnoGAN-based method to find out the jump instances. Next, based on the detected jumps, we apply the MLE method to estimate the jump parameters. For example, in Merton's model ($J \sim \mathcal{N}(\mu_J, \sigma_J^2)$), our purpose is to estimate the jump intensity λ_p, the mean jump size μ_J and the jump size volatility σ_J.

5.4.1 Jump Instances Detection

The jumps can be viewed as anomalies. On the one hand, jumps, especially big jumps, are rare to happen in the real markets. The jumps are usually recognized as sudden price changes, reflecting unexpected market information. On the other hand, regarding the jump-diffusion discretization formula, the jump-diffusion price reduces to a GBM log-price when the Bernoulli variable is equal to 0. It is then natural to consider the jumps as abnormal prices, while the diffusion part indicates normal market behavior. Therefore, anomaly detection techniques can be applied to distinguish the jumps from the diffusion prices. And the difference between a jump and the corresponding diffusion price can be introduced to measure the anomalousness.

Moreover, when the SDE-WGAN can produce excellent diffusion prices, the AnoGAN methodology can be applied for detecting jumps. Recall in Section 3.5.2, the GAN model is only trained on the normal data, and the anomalies are also reconstructed as normal samples. The anomalousness is then evaluated by comparing the reconstructed pattern to the original one. The idea of bringing in the AnoGAN is expressed as follows:

Consider (5.3.5), the SDE-WGAN produces the diffusion part. When a jump occurs at t_i, that is, $B(t_i) = 1$, the jump J is additionally added to the approximated diffusion part. Therefore, when using the SDE-WGAN to reconstruct the jump price $\mathbf{X}_J(t_i)$, it cannot capture the jump J [1]. The difference between the reconstructed price $\hat{\mathbf{X}}(t_i)$and $\mathbf{X}_J(t_i)$ then indicates the jump. Similarly, we can construct an anomaly score to measure the difference, which contains the residual loss and the critic loss.

In summary, the detection procedure of the jump instances is illustrated as follows:

Suppose the SDE-WGAN is well-trained, and the generator G^*_η can produce the diffusion part with satisfactory. Given a jump diffusion path during the time period $[0, T]$ with time step $\Delta t = \frac{T}{m}$, which is denoted by $\{X_J(t_0), X_J(t_1), \ldots, X_J(t_m)\}$, we first find the corresponding latent variable $\tilde{\mathbf{Z}}_i \in \mathbb{R}^d$ of each state $X_J(t_i)$ in the latent space. $\tilde{\mathbf{Z}}_i$ is obtained by an optimization algorithm (see Section 3.5.2) such that $\hat{\mathbf{X}}(t_i) := G^*_\eta(\tilde{\mathbf{Z}}_i, \Delta t, \mathbf{X}_J(t_{i-1}))$ is visually most similar to $\mathbf{X}_J(t_i))$[2]. Second, we use the generator to reconstruct the jump price, which is formulated by $\hat{\mathbf{X}}(t_i) := G^*_\eta\left(\tilde{\mathbf{Z}}_i, \Delta t, \hat{\mathbf{X}}(t_{i-1})\right)$. The residual loss is then given by

$$\mathcal{L}_R(t_i) = \|\mathbf{X}_J(t_i) - \hat{\mathbf{X}}(t_i)\|_1. \tag{5.4.1}$$

We also think of the critic error, and the critic loss is defined by

$$\mathcal{L}_{f^*_w}(t_i) = \|f^*_w(\mathbf{X}_J(t_i)) - f^*_w(\hat{\mathbf{X}}(t_i))\|_1. \tag{5.4.2}$$

Here $f^*_w(\cdot)$ is the output of the critic. In the end, we calculate the anomaly score of each state which is given by

$$A(t_i) = (1-\lambda)\mathcal{L}_R(t_i) + \lambda\mathcal{L}_{f_w}(t_i), \tag{5.4.3}$$

where $\lambda \in (0, 1)$ is a constant. When the anomaly score $A(t_i)$ is larger than the threshold α, then a jump happens at t_i. In this thesis, α is decided visually based on the anomaly score pattern.

We formulate the process above as a pseudocode, illustrated in Algorithm 9. Note that the reconstructed pattern consists of d samples, and we use the mean sample price as the reconstructed price. Because of this, less reconstruction error, which is associated with the accuracy of SDE-WGAN, would be involved.

5.4.2 Jump Parameters Estimation

When the jump instances are detected, for example at t_i, we can use the difference between the reconstructed price and the input price to approximate the jump size, namely

$$J := X_J(t_i) - \overline{\hat{\mathbf{X}}}(t_i), \tag{5.4.4}$$

[1]Here, the jump J includes the jump direction, that is, $J < 0$ if the price jumps down.

[2]$\mathbf{X}_J(t_i) := (X_J(t_i), \ldots, X_J(t_i))$ is a vector of length d.

Algorithm 9: Jump instances detection.

```
Input  : A jump-diffusion path $\{X_J(t_0), \dots, X_J(t_m)\}$, where
         $0 = t_0 < t_1 < \cdots < t_m = T$ and $\Delta t := t_i - t_{i-1} = \frac{T}{m}$; The welled-trained
         SDE-WGAN with the generator $G^*_\eta$ and the critic $f^*_w$; Anomaly score
         coefficient $\lambda$; Iteration number $N$.
Output: The jump instances $t^J_1, \dots, t^J_K$.
for $i = 1, \dots, m$ do
    Sample $\mathbf{Z}$ from the distribution $\mathbb{P}_z$
    for $n = 1, \dots, N$ do
        /* Map the jump price to the latent space.                */
        $\hat{\mathbf{X}} \leftarrow G^*_\eta(\mathbf{Z}, \Delta t, X_J(t_{i-1}))$
        $l_R \leftarrow \|\hat{\mathbf{X}} - X_J(t_{i-1})\|_1$
        $l_{f^*_w} \leftarrow \|f^*_w(\hat{\mathbf{X}}) - f^*_w(X_J(t_{i-1}))\|_1$
        $l \leftarrow (1-\lambda) l_R + \lambda l_{f^*_w}$
        $\mathbf{Z} \leftarrow \text{Adam}(\nabla_{\mathbf{Z}} l)$
    end
    /* Calculate the anomaly score.                               */
    $\hat{\mathbf{X}}(t_i) \leftarrow G^*_\eta(\mathbf{Z}, \Delta t, X_J(t_{i-1}))$
    $\mathcal{L}_R \leftarrow \|\hat{\mathbf{X}}(t_i) - X_J(t_{i-1})\|_1$
    $\mathcal{L}_{f^*_w} \leftarrow \|f^*_w(\hat{\mathbf{X}}(t_i)) - f^*_w(X_J(t_{i-1}))\|_1$
    $A(t_i) \leftarrow (1-\lambda)\mathcal{L}_R + \lambda \mathcal{L}_{f^*_w}$
    /* Save the detected jump instance.                           */
    if $A(t_i) > \alpha$ then
        $t^J_k \leftarrow t_i$
    end
end
```

where $\{X_J\}$ denotes the jump-diffusion path for detection, $\hat{\mathbf{X}}(t_i) := G^*_\eta\left(\tilde{\mathbf{Z}}_i, \Delta t, \hat{\mathbf{X}}(t_{i-1})\right)$ is the price approximated by the well-trained SDE-WGAN, and $\overline{\hat{\mathbf{X}}}$ represents the mean value.

Since the jump instances and the jump sizes are decided, we can use the Maximum Likelihood Estimation (MLE) method [79] to approximate the parameters of the jump part, a compound Poisson process $\sum\limits_{k=1}^{X_{\mathcal{P}}(T)} J_k$.

With respect to the Poisson process $\{X_{\mathcal{P}}(t), t \geq 0\}$, our goal is to estimate the rate or the jump intensity λ_p. The detected jump instances which are denoted by $\left\{t^J_1, \dots, t^J_K\right\}$ are then related. Based on the path simulation method of the jump-diffusion process, only one event or jump occurs at each jump instance t^J_k, $k = 1, \dots, K$, and we have

$X_{\mathcal{P}}(T) = K$. The corresponding likelihood function is then given by

$$L(\lambda_p) = \prod_{k=1}^{K} \frac{\left(\lambda_p \left(t_k - t_{k-1}\right)\right)^1 e^{-\lambda_p (t_k - t_{k-1})}}{1!}, \quad t_0 = 0, \tag{5.4.5}$$

and the log-likelihood function is

$$l(\lambda_p) = \sum_{k=1}^{K} \ln \lambda_p + \ln(t_k - t_{k-1}) - \lambda_p (t_k - t_{k-1}). \tag{5.4.6}$$

Setting $\frac{\mathrm{d}l(\lambda_p)}{\lambda_p} = 0$, we have $\frac{K}{\lambda_p} - t_K = 0$. Hence, the MLE estimator is

$$\hat{\lambda}_p = \frac{K}{t_K}. \tag{5.4.7}$$

Regarding the jumps $\{J_k, k \geq 1\}$, the parameters to estimate and the corresponding likelihood function are associated with the jump-diffusion model setting. Take the Merton's model as an instance, where $J_k \sim \mathcal{N}(\mu_J, \sigma_J^2)$. We need to estimate μ_J and σ_J. The likelihood function is written as

$$L(\mu_J, \sigma_J^2; J_1, \ldots, J_K) = \left(2\pi\sigma_J^2\right)^{-K/2} \exp\left(-\frac{1}{2\sigma_J^2} \sum_{k=1}^{K} (J_k - \mu_J)^2\right), \tag{5.4.8}$$

and the log-likelihood function is then given by

$$l(\mu_J, \sigma_J^2; J_1, \ldots, J_K) = -\frac{K}{2} \ln 2\pi - \frac{K}{2} \ln \sigma_J^2 - \frac{1}{2\sigma_J^2} \sum_{k=1}^{K} (J_k - \mu_J)^2. \tag{5.4.9}$$

Calculating $\frac{\partial l(\mu_J, \sigma_J^2; J_1, \ldots, J_K)}{\partial \mu_J} = 0$ and $\frac{\partial l(\mu_J, \sigma_J^2; J_1, \ldots, J_K)}{\partial \sigma_J^2} = 0$, respectively, we obtain the MLE estimators

$$\begin{aligned} \hat{\mu}_J &= \frac{1}{K} \sum_{k=1}^{K} J_k, \\ \hat{\sigma}_J^2 &= \frac{1}{K} \sum_{k=1}^{K} (J_k - \hat{\mu}_J)^2. \end{aligned} \tag{5.4.10}$$

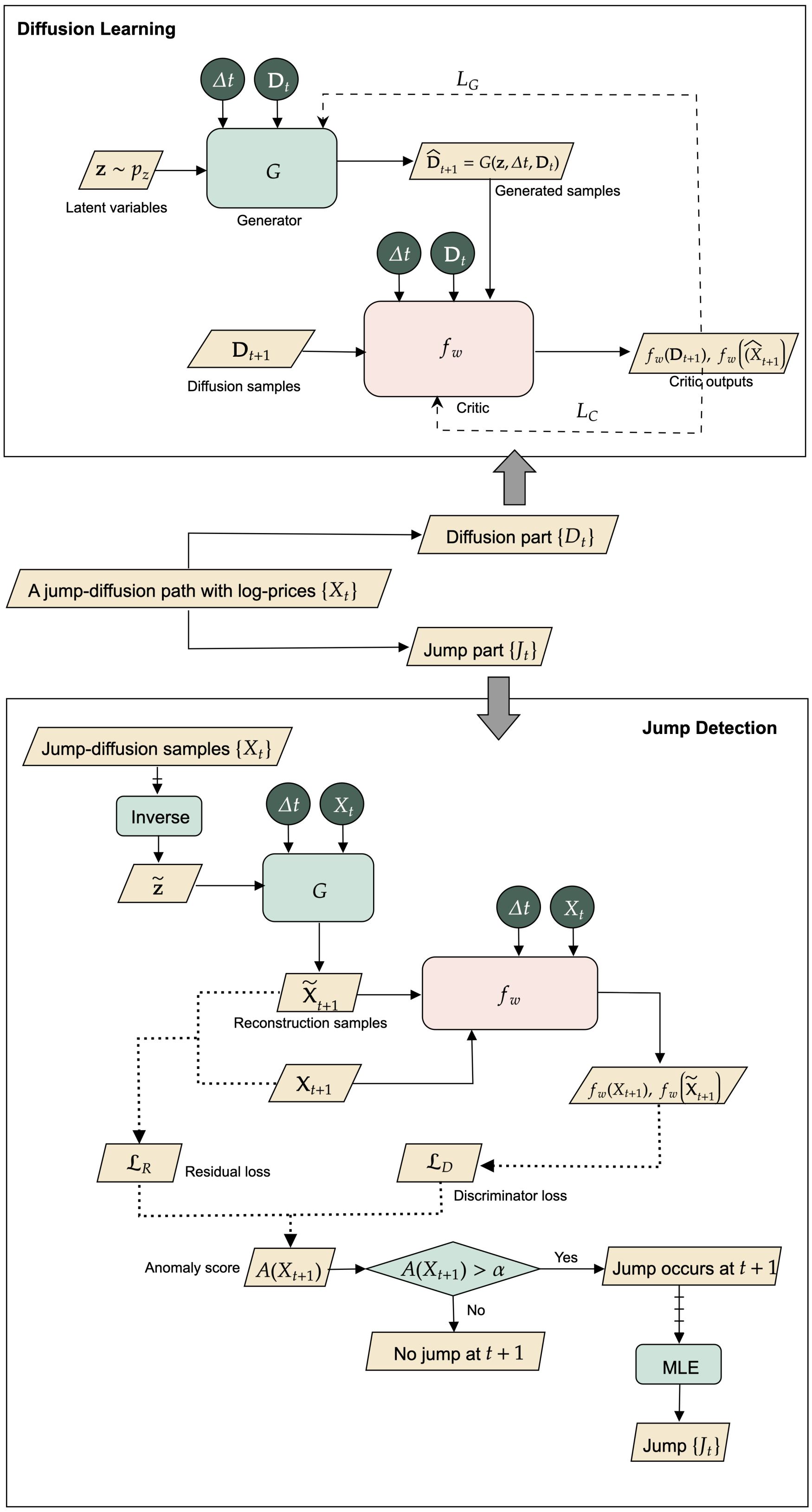


Figure 5.1: An overview of the proposed framework. In the diffusion learning part, the SDE-WGAN is illustrated, and it is trained to reproduce the diffusion part of a jump-diffusion path. In the jump detection part, the adapted AnoGAN is displayed, where the details of finding the corresponding latent variables are eliminated.

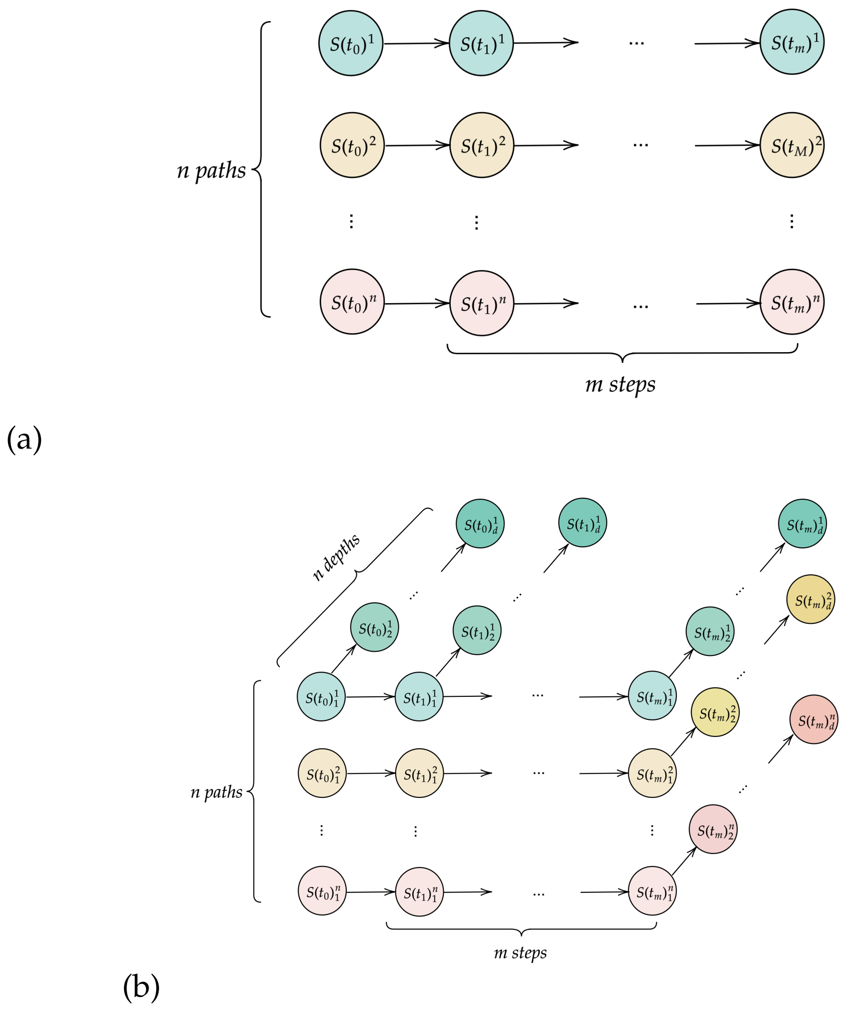

n paths
$S(t_0)^1$ $S(t_1)^1$ ... $S(t_m)^1$
$S(t_0)^2$ $S(t_1)^2$ ... $S(t_M)^2$
$S(t_0)^n$ $S(t_1)^n$ ... $S(t_m)^n$
m steps
(a)
n depths
$S(t_0)^1_d$ $S(t_1)^1_d$ $S(t_m)^1_d$
$S(t_0)^1_2$ $S(t_1)^1_2$ $S(t_m)^1_2$ $S(t_m)^2_d$
$S(t_0)^1_1$ $S(t_1)^1_1$ ... $S(t_m)^1_1$ $S(t_m)^2_2$ $S(t_m)^n_d$
n paths
$S(t_0)^2_1$ $S(t_1)^2_1$ ... $S(t_m)^2_1$
$S(t_m)^n_2$
$S(t_0)^n_1$ $S(t_1)^n_1$ ... $S(t_m)^n_1$
m steps
(b)

Figure 5.2: The dataset structure of a 2D-dataset (a) and a 3D dataset (b), where the arrows show the sampling directions

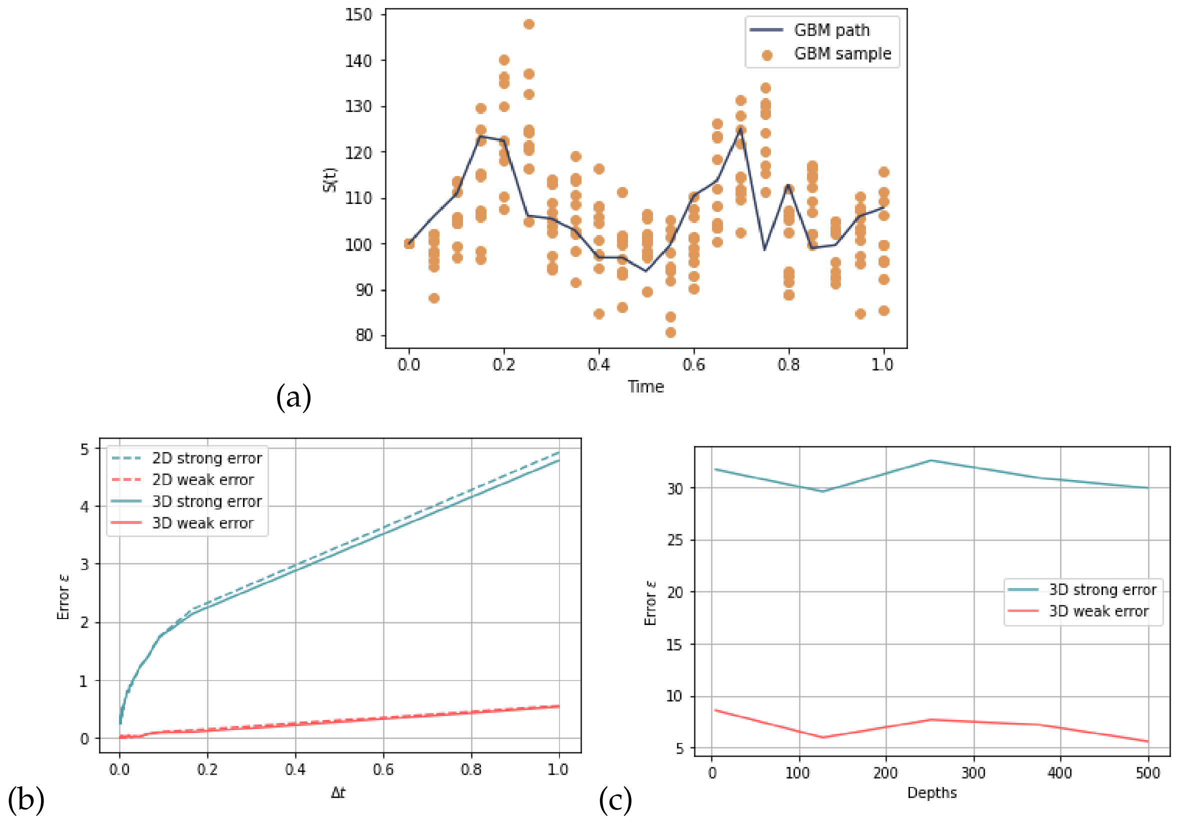


Figure 5.3: An example of constructing the GBM datasets with 2D and 3D structure, respectively. The parameters of the GBM model are the same as Example 2.2.4. (a) A plot of a random path in both datasets. In the 2D dataset, there is only one sample at each timestamp; In the 3D dataset with 10 depths, ten random states are sampled at each timestamp based on the previous state in the 2D dataset. (b) The error convergence plot of the 2D dataset and 3D dataset with 10 depths. (c) The weak and strong errors of the 3D dataset with respect to various depths.

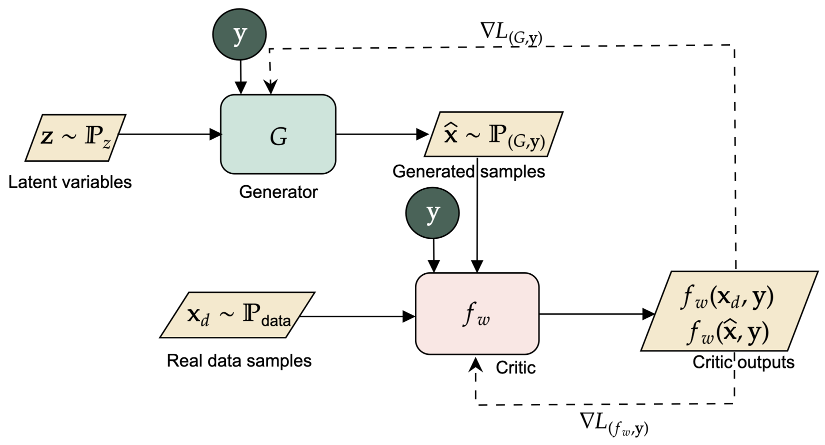


Figure 5.4: A high-level overview of the cWGAN-GP, where $\nabla L_{(G,\mathbf{y})}$ and $\nabla L_{(f_w,\mathbf{y})}$ means the gradient descent.

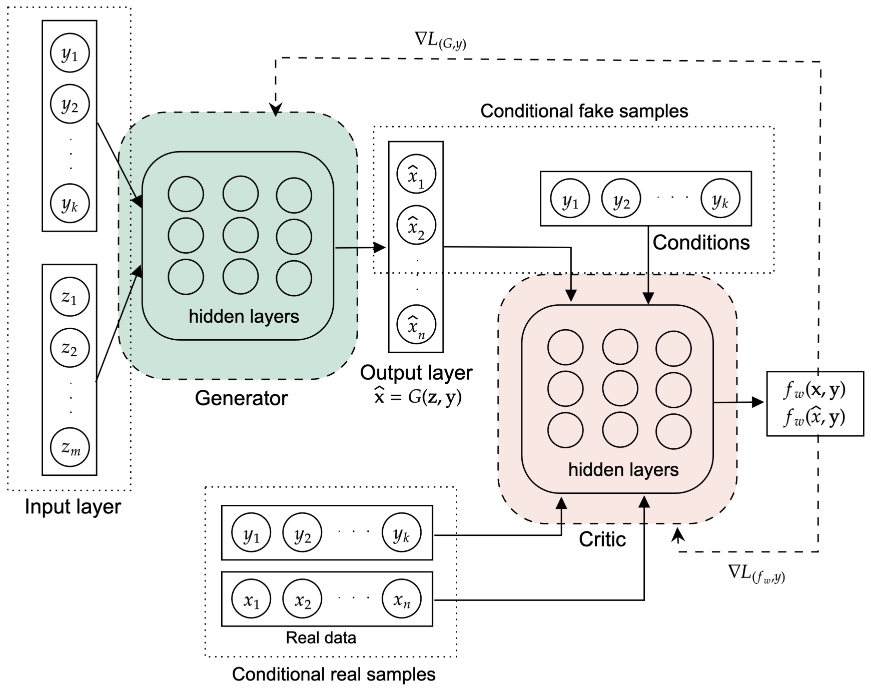


Figure 5.5: A simple example of the cWGAN-GP structure in practice, where both the generator and critic are MLPs.

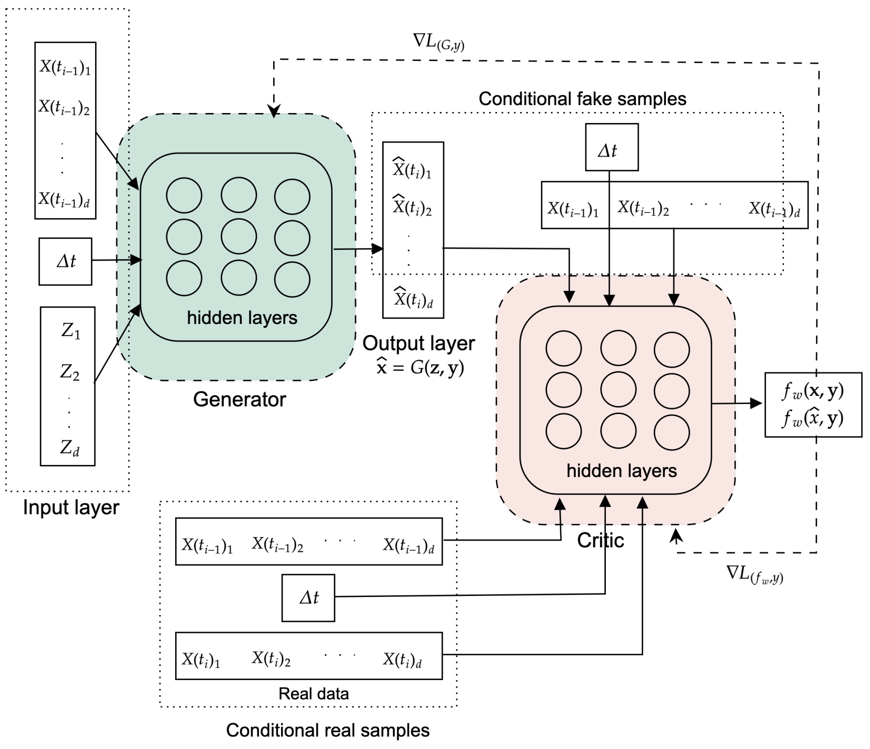


Figure 5.6: The detailed structure of the SDE-WGAN, where both the generator and critic are MLPs. In empirical experiments, the time step condition Δt is also expanded to a d-dimensional vector $(\Delta t, \ldots, \Delta t)$.

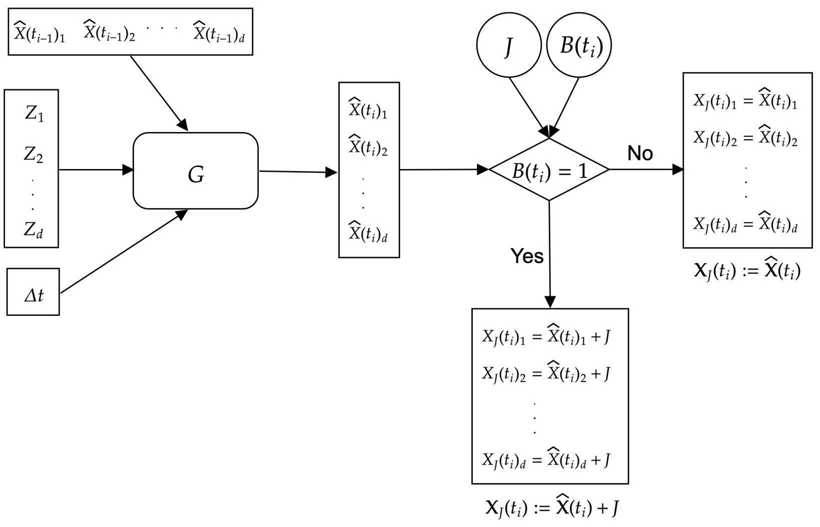

$\widehat{X}(t_{i-1})_1$ $\widehat{X}(t_{i-1})_2$ $\widehat{X}(t_{i-1})_d$
Z_1
Z_2
Z_d
Δt
G
$\widehat{X}(t_i)_1$
$\widehat{X}(t_i)_2$
$\widehat{X}(t_i)_d$
J
$B(t_i)$
$B(t_i) = 1$
No
Yes
$X_J(t_i)_1 = \widehat{X}(t_i)_1$
$X_J(t_i)_2 = \widehat{X}(t_i)_2$
$X_J(t_i)_d = \widehat{X}(t_i)_d$
$X_J(t_i) := \widehat{X}(t_i)$
$X_J(t_i)_1 = \widehat{X}(t_i)_1 + J$
$X_J(t_i)_2 = \widehat{X}(t_i)_2 + J$
$X_J(t_i)_d = \widehat{X}(t_i)_d + J$
$X_J(t_i) := \widehat{X}(t_i) + J$

Figure 5.7: The detailed jump-diffusion path simulation process in the diffusion learning part, where the generator G is the well-trained generator in the SDE-WGAN.

Chapter 6

Experiment Results

In this chapter, we show the empirical experiment results of the proposed framework. We describe the detailed setups of the experiments and introduce several metrics to evaluate the performance. We also process a brief test regarding the robustness of the proposed framework.

6.1 Experimental Setup

The experiments are performed using NVIDIA®Tesla®T4 GPU on the online platform Google Colaboratory[1]. We use the PyTorch [2], a Python based library for machine learning, to implement the SDE-WGAN. We also use Numpy [3], Scipy.stats [4], and Seaborn [5] to generate, analyse and visualize statistic results.

6.1.1 Data Setup

SDE-WGAN Training Dataset

We construct a 3D dataset with 10000 paths, 500 steps and 10 depths to train the SDE-WGAN, which is denoted by $\left\{X(t_i)_k^j\right\}$. The dataset consists of the samples from the GBM log-price dynamics, that is

$$X(t_i)_k^j) = X(t_{i-1})_1^j) + \mu\Delta t + \sigma Z\sqrt{\Delta t}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \text{ and } k = 1, \ldots, d, \tag{6.1.1}$$

where $\mu = 0.05$, $\sigma = 0.2$, $\Delta t = 0.1$ and $Z \sim \mathcal{N}(0, 1)$. In order to guarantee that $\left\{X(t_i)_k^j\right\}$ contains sufficient samples of different states, we further divide the dataset

[1] https://research.google.com/colaboratory/faq.html

[2] https://pytorch.org/

[3] https://numpy.org/

[4] https://docs.scipy.org/doc/scipy/reference/stats.html

[5] https://seaborn.pydata.org/

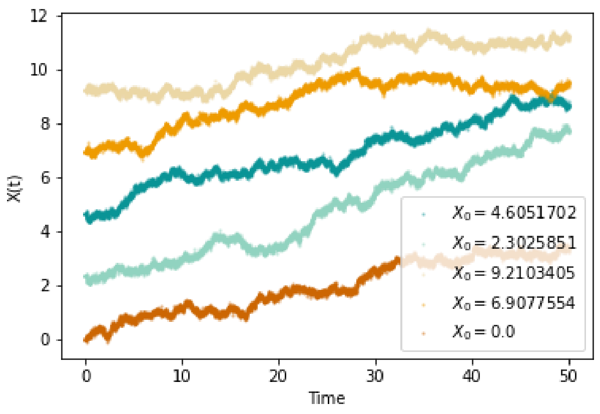


Figure 6.1: An example of the training dataset, where the samples have different initial states.

into five parts, and each part has a different initial state. Precisely,

$$X(t_0)_k^j = \begin{cases} 0, & j = 1, \dots, 2000 \\ \log 10, & j = 2001, \dots, 4000 \\ \log 100, & j = 4001, \dots, 6000 \\ \log 1000, & j = 6001, \dots, 8000 \\ \log 10000, & j = 8001, \dots, 10000 \end{cases},$$

for $k = 1, \dots, d$. Figure 6.1 displays an example of the training dataset.

Jump-Diffusion Model
In this thesis, we implement the proposed framework on the Merton's model, namely, the jump $J \sim \mathcal{N}(\mu_J, \sigma_J^2)$. When simulating the jump-diffusion paths, we set the jump intensity $\lambda_p = 1$, $\mu_J = -0.2$ and $\sigma_J = 0.5$.

6.1.2 SDE-WGAN Setup

Both the generator and critic have similar structures to the neural networks in the SDE-GAN. Precisely, both of them are multilayer perceptrons. The SDE-WGAN is trained following Algorithm 8, and a learning rate scheduler is resorted to increasing the convergence speed and learning accuracy. The implementation details are illustrated in Appendix B.

6.2 Evaluation Metrics

In this section, we describe several metrics to evaluate the performance of the proposed framework. In the diffusion learning part, we focus on the simulation performance of the SDE-WGAN. We introduce nonparametric statistics to measure the similarity between the approximated and the exact distributions, where the approximated distribution is learned by the SDE-WGAN. Inspired by [15], we apply the Kolmogorov–Smirnov test (KS test) [80] and the 1-Wasserstein distance to assess the simulation. In the jump detection part, we would like to evaluate the accuracy of the detection. As each jump-diffusion state $X_J(t_i)$ is classified as a normal sample (negative sample) or an anomaly (positive sample), we can compute the confusion matrix to evaluate the accuracy of the classification.

6.2.1 Empirical Probability Distribution

The SDE-WGAN does not output the formula of the conditional distribution directly, however, we can calculate the empirical cumulative distribution function (ECDF) and estimate the probability density [81] according to the generated samples.

Definition 6.2.1 (Empirical cumulative distribution function)**.** *Let $(X_1, \dots, X_n)$ be i.i.d real random variables with the common cumulative distribution function $F(x)$, then, the ECDF is defined as*

$$\hat{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{X_i \le x}(x), \tag{6.2.1}$$

where $\mathbb{1}$ is the indicator function.

Remark 6.2.2. *$\hat{F}_n(x)$ is an unbiased estimator for $F(x)$ [81].*

The empirical probability density function (EPDF) can also be estimated by a nonparametric method, called kernel density estimation (KDE). The kernel density estimate is closely linked to the histogram, but can be smooth or continuous.

Definition 6.2.3. *Let $(X_1, \dots, X_n)$ be i.i.d real random variables sampled from a univariate distribution with an unknown density f at any given point x. The kernel density estimator is defined as*

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right), \tag{6.2.2}$$

where K is a non-negative function called the kernel and $h > 0$ is a smoothing parameter called the bandwith.

In practice, both the ECDF and EPDF of the generated samples are easy to implement, and we can visually compare the estimators with the exact functions.

6.2.2 KS Test and 1-Wasserstein Distance

Suppose the generator of the SDE-WGAN outputs N samples $\{X_i\}_{i=1}^{N}$. When the ECDF of the generator output is estimated which is denoted by $\hat{F}_N$, we can compare it to the exact cumulative distribution function F. The formulas of two comparison method, the KS test and the 1-Wasserstein distance, is given as follows:

The 2-sided 1-sample KS metric is written as

$$KS = \max_{i \in \{1,\dots,N\}} |F_N(X_i) - F(X_i)|. \tag{6.2.3}$$

The 1-Wasserstein distance is expressed by

$$W_1 = \int_{\mathbb{R}} |F_N(x) - F(x)| \mathrm{d}x. \tag{6.2.4}$$

Both the metrics can be efficiently calculated by applying the Scipy.stats package. Moreover, we can obtain the p-value of the KS test, where the null hypothesis is that the two distributions $\hat{F}_N$ and F are identical.

6.2.3 Classification Performance

The jump detection results can be divided into four groups: true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The true positive presents that the detected jumps are the actual jump in the jump-diffusion path; The true negative means the classified normal data is also a non-jump state in the detected path; The false positive and the false negative indicate the wrong classification, that is, the situation that a normal data is labeled as a jump or a jump is not detected respectively. Counting the four kinds of results leads to a so-called confusion matrix, which is a table layout that is specifically used in the statistic classification problem [82], see Table 6.1.

Table 6.1: The confusion matrix

		Predicted results	
	Total (P+N)	Negative (N)	Positive (P)
Actual results	Negative (N)	True Negative (TN)	False Positive (FP)
	Positive (P)	False Negative (FN)	True Positive (TP)

Furthermore, several metrics are formulated to analysis the classification performance:

$$\begin{aligned} \text{Accuracy} &= \frac{TP+TN}{P+N}, \\ \text{Precision} &= \frac{TP}{TP+FP}, \\ \text{Recall} &= \frac{TP}{TP+FN}, \\ \text{F1 score} &= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \end{aligned} \tag{6.2.5}$$

In an imbalanced test dataset, for example, the jump-diffusion path, the accuracy could be misleading and insufficient to evaluate the performance. The precision and the recall are then calculated as complements, where the precision indicates the reliability of the jump detection model in identifying the jumps, and the recall reflects the model's ability to detect the jumps. The F1 score combines the precision and the recall, giving a general assessment of the model performance.

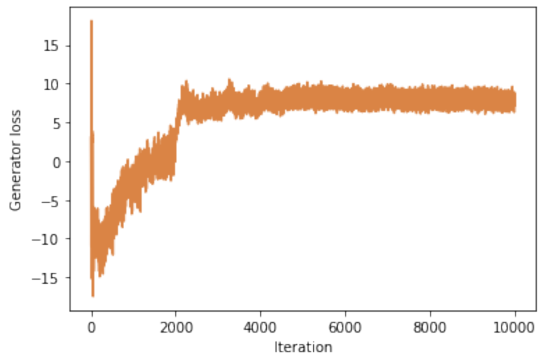
6.3 Results

6.3.1 SDE-WGAN Performance

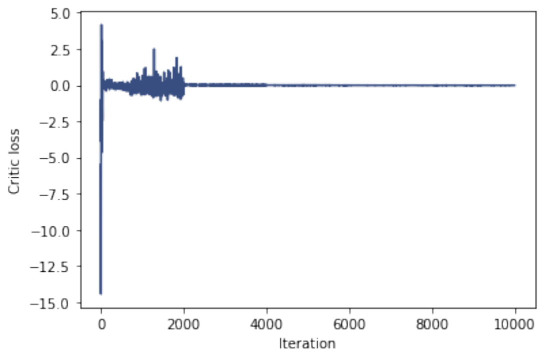
The SDE-WGAN is the key point of the proposed framework, and its performance will directly influence the results of the framework. Unlike the SDE-GAN, the SDE-WGAN is more easier to train and never traps in the failure modes in our experiments.

Figure 6.2 presents the training process of the SDE-WGAN on the constructed 3D training dataset, where we use the KS metric and the 1-Wasserstein distance to measure the similarity between the generated samples and the exact GBM samples. We can say that the SDE-WGAN's training is sufficient.

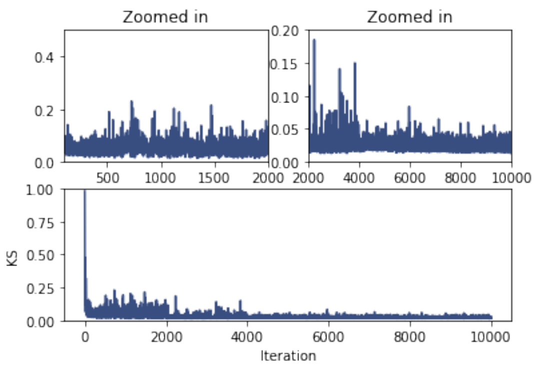
The test dataset is constructed by re-sampling the training dataset, and we fix the initial



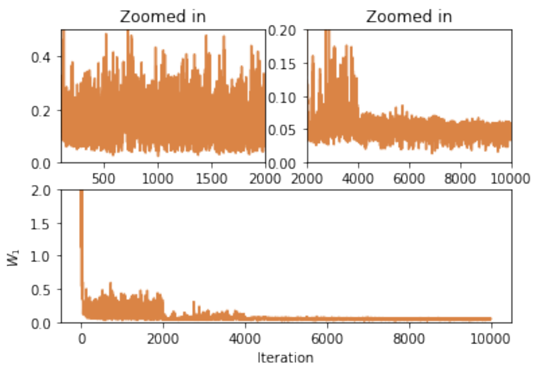
(a)



(b)



(c)



(d)

Figure 6.2: The training process of the SDE-WGAN, where the training epochs $= 100$ and batch size $= 100$. Each mini-batch training is one iteration. (a) The generator loss of each batch during the training; (b) The critic losses; (c) The KS metric between the generator outputs and the real data; (d) The 1-Wasserstein distance between the generator outputs and the real data.

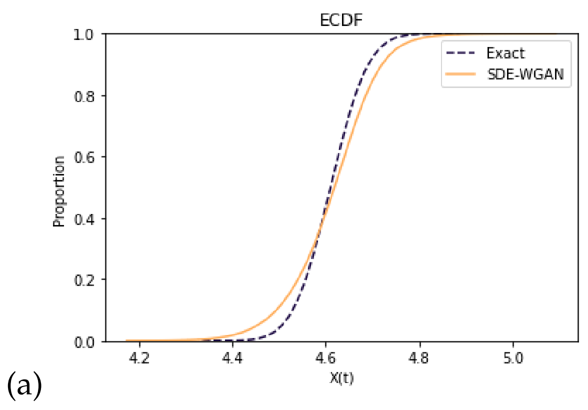
states $\{X(t_0)_k^j\}$ being $\log 100$. We use the well-trained SDE-WGAN to perform on the test dataset: at each timestamp t_{i-1}, the SDE-WGAN outputs the states $\hat{\mathbf{X}}(t_i)$ based on the time step Δt and the exact GBM samples $\mathbf{X}(t_{i-1})$, that is,

$$\hat{\mathbf{X}}(t_i) = G_\eta^*(\mathbf{Z}, \Delta t, \mathbf{X}(t_{i-1})), i = 1, \ldots, m \tag{6.3.1}$$

where G_η^* is the well-trained generator.

We visualize the performance of the SDE-WGAN by plotting the ECDF and the EPDF of the approximated distribution $\mathbb{P}_{\hat{\mathbf{X}}(t_i)|\mathbf{X}(t_{i-1})}$, compared to the exact results of $\mathbb{P}_{\mathbf{X}(t_i)|\mathbf{X}(t_{i-1})}$. Figure 6.3 shows the empirical results of the conditional distribution $\mathbb{P}_{\hat{\mathbf{X}}(t_1)|\mathbf{X}(t_0)}$. Contrary to Figure 3.5, the SDE-WGAN seems cannot approximate the conditional distribution well. However, it presents excellent results regarding the conditional distribution $\mathbb{P}_{\hat{\mathbf{X}}(t_m)|\mathbf{X}(t_{m-1})}$, see Figure 6.4. The KS metric and the 1-Wasserstein distance regarding the first and the last timestamps are shown in Table 6.2.

We further explore the approximation of each timestamp $t_1, \ldots, t_m$ by calculating the KS metric and the 1-Wasserstein distance (see Figure 6.5), we conclude that except the first few timestamps, the SDE-WGAN can approximate the conditional distribution $\mathbb{P}_{\mathbf{X}(t_i)|\mathbf{X}(t_{i-1})}$ with satisfactory.



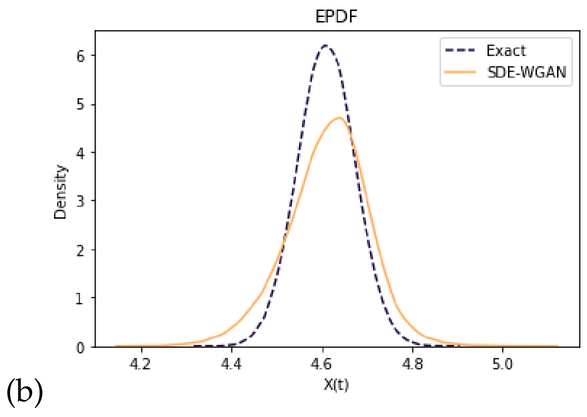
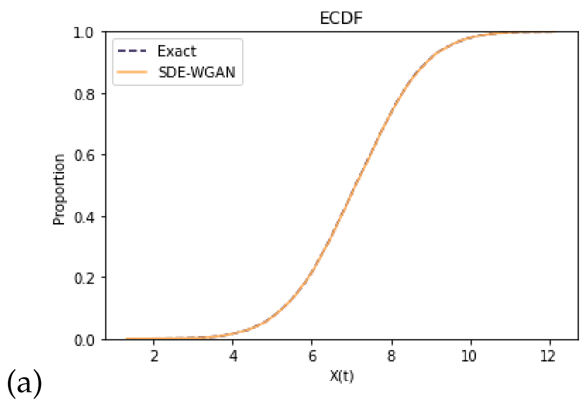


Figure 6.3: The ECDF and the EPDF plots of the conditional distribution $\mathbb{P}_{\hat{\mathbf{X}}(t_1)|\mathbf{X}(t_0)}$, where $\mathbf{X}(t_0) = \log 100$ is fixed.



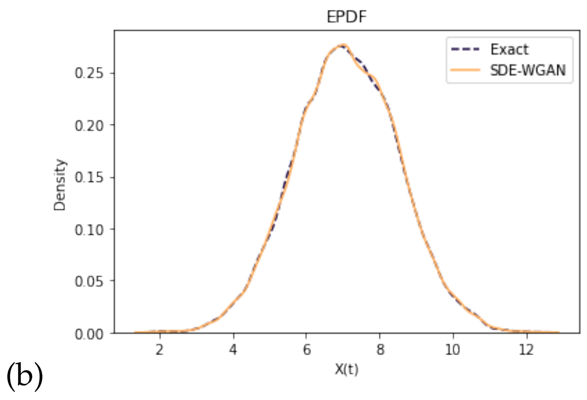


Figure 6.4: The KS metric and the 1-Wasserstein distance between $\mathbb{P}_{\hat{\mathbf{X}}(t_i)|\mathbf{X}(t_{i-1})}$ and $\mathbb{P}_{\mathbf{X}(t_i)|\mathbf{X}(t_{i-1})}$, for $i = 1, \ldots, m$.

Table 6.2: The metrics of the comparison between the exact and the approximated conditional distribution at the first and the last timestamps, respectively.

Timestamp	KS metric	p-value	1-Wasserstein
t_1	0.091510	0.000000	0.021792
t_m	0.003110	0.717617	0.006066

The results are reasonable because of the 3D dataset construction. In the training dataset, only one-fifth of the paths start at $\log 100$, while in the test dataset, all paths have the initial state $\log 100$. The insufficient training samples lead to a bias on approximation. Such influence is reduced when the previous states are various.

All in all, the SDE-WGAN can learn the diffusion part with excellent performance.

6.3.2 Jump-Diffusion Path Simulation

When the diffusion part is simulated by the SDE-WGAN, we add the jump part under Merton's model to simulate jump-diffusion paths, following (5.3.5). We display a random simulated jump-diffusion path in Figure 6.6. The simulation is not path-wise

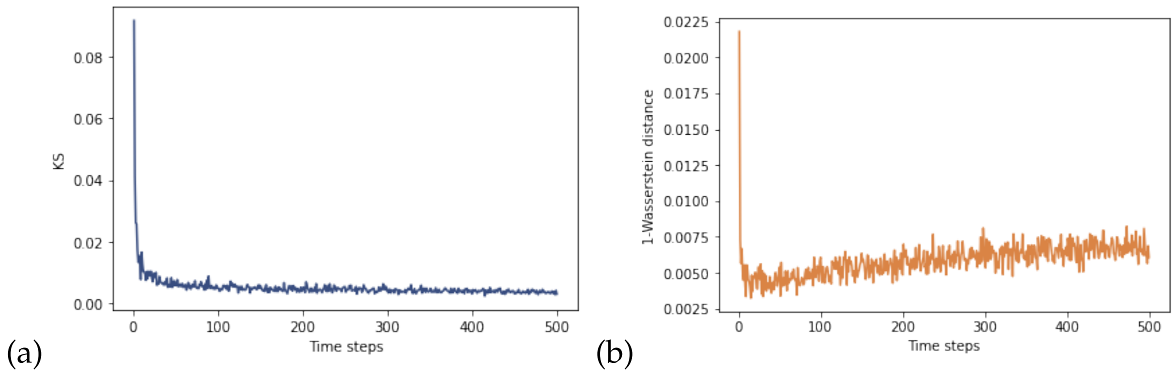


Figure 6.5: The ECDF and the EPDF plots of the conditional distribution $\mathbb{P}_{\hat{\mathbf{X}}(t_m)|\mathbf{X}(t_{m-1})}$, that is, the conditional distribution at the last timestamp.

compared to the exact path, since the SDE-WGAN can only provide the weak solutions of the SDEs [15], that is, the SDE-WGAN cannot provide the path-wise diffusion part. We further calculate the weak error and the strong error based on a dataset consists of 10000 jump-diffusion paths with 500 timestamps, and the results are 0.279908 and 0.823819 respectively.

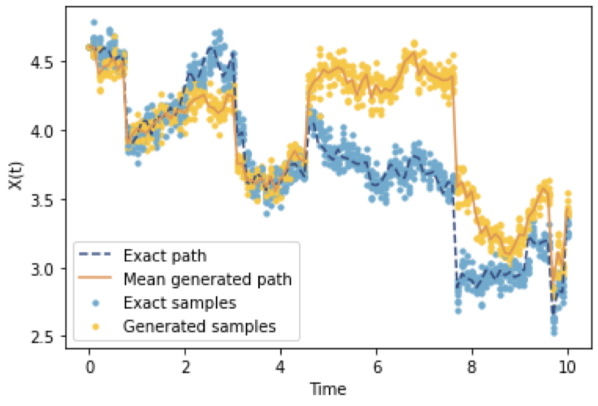


Figure 6.6: A jump-diffusion path simulated by following (5.3.5).

The experiment shows a success in practicing the diffusion learning part of the proposed framework. We can also conclude that the SDE-WGAN is able to simulate jump-diffusion paths.

6.3.3 Jump Detection

In this part, we shows the performance in practicing the jump detection part of the proposed framework. The jump-diffusion path $\{X_J(t_i)\}$ to be detected is a path with 505 timestamps, where the initial state $X_J(t_0) = \log 100$, and other parameters are the same as the ones mentioned above. In case of the prices being extreme, we reset $X_J(t_i) = \log 100$ after every 100 steps. Essentially, it consists of five paths with 100 time steps, see Figure 6.7. Since we assume that no jump occurs at the initial states, we eliminate the initial states and detect the jump occurrence in the rest 500 timestamps.

Following Algorithm 9 with anomaly score coefficient $\lambda = 0.4$, the anomaly score of

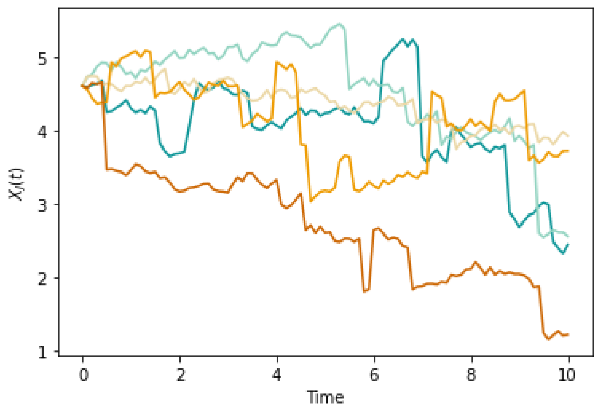


Figure 6.7: The jump-diffusion paths to be detected.

each state is illustrated in Figure 6.8. We roughly set the threshold $\alpha = 0.2$, and obtain the confusion matrix of the detection results, see Table 6.3. With this threshold, there are 33 actual jumps truly detected, and 13 actual jumps are recognized as the normal data. In addition, Table 6.4 shows the evaluations with respect to the detection results. Overall, we can say that the jump detection method performs well regarding this jump-diffusion path.

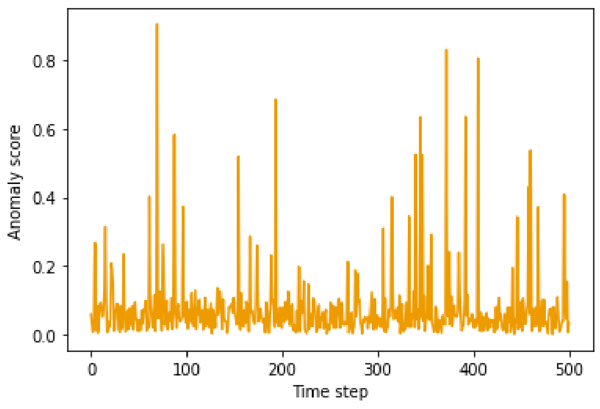


Figure 6.8: The anomaly scores of the jump-diffusion path.

Table 6.3: The confusion matrix of the jump detection results

	Normal data	Jumps
Normal data	454	0
Jumps	13	33

Table 6.4: The evaluation metrics of the jump detection results.

Accuracy	Recall	Precision	F1 score
97.40%	97.22%	100%	98.59

Based on the 33 detected jumps, we apply the MLE method to estimate the jump intensity λ_p and the Merton parameters μ_J and σ_J. When estimate the jump intensity, we divide the jump-diffusion path into five parts, and each part has 100 timestamps. The estimated jump intensity is the average of the estimations from the five parts. Table 6.5 displays the estimated results.

Table 6.5: The results of the estimated jump parameter.

	λ_p	μ_J	σ_J
Estimated results	0.715834	-0.324300	0.638275
Exact results	1.0	-0.2	0.5

To visualize the performance of the estimation, we use the estimated parameters to generate jump-diffusion samples, and Figure 6.9 shows a comparison regarding the empirical distributions. In conclusion, the estimated distribution is almost fit the exact one generally, and the proposed jump detection model is able to estimate the jump parameters.

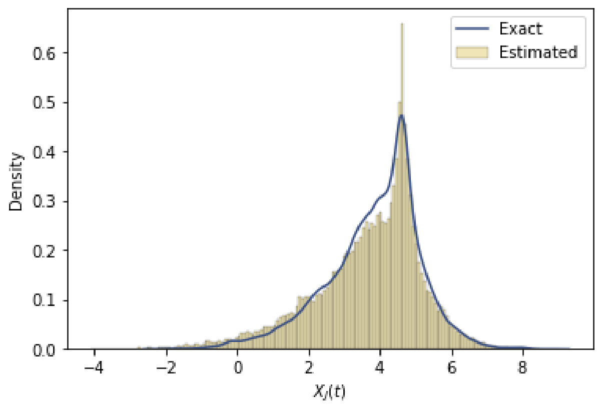


Figure 6.9: The histogram of the jump-diffusion samples generated by the estimated parameters, comparing with the empirical distribution generated by the actual parameters.

6.4 Robustness

The experiments above show a successful application of the proposed framework. However, we are interested in the robustness of the framework. In this section, we focus on two problems: 1) When the parameters change, can the SDE-WGAN still learn the conditional distribution? 2) When the jump parameters change, can the jump detection model still recognize the occurrence of the jumps?

Regarding the robustness of the SDE-WGAN, we train the model on various Δt, μ and σ, and calculate the KS metric and the 1-Wasserstein distance between the generated conditional distribution $\mathbb{P}_{\hat{X}(t_m)|X(t_{m-1})}$ and the exact distribution $\mathbb{P}_{X(t_m)|X(t_{m-1})}$. The results are illustrated in Table 6.6. We can summarize that the SDE-WGAN is robust and can still approximate the conditional distribution when the parameters change.

Regarding the robustness of the jump detection model, we apply the jump detection model to different jump-diffusion paths, and the results are summarized in Table 6.7.

Table 6.6: The KS test and the 1-Wasserstein distance between $\mathbb{P}_{\hat{\mathbf{X}}(t_m)|\mathbf{X}(t_{m-1})}$ and $\mathbb{P}_{\mathbf{X}(t_m)|\mathbf{X}(t_{m-1})}$ with respect to different diffusion parameters.

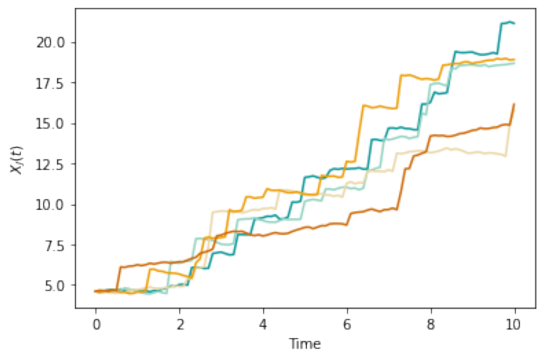
Δt	μ	σ	KS	KS p-value	W_1
1.0	0.05	0.2	0.002700	0.858293	0.014771
0.5	0.05	0.2	0.002310	0.951809	0.010555
0.01	0.05	0.2	0.005040	0.157102	0.004024
0.001	0.05	0.2	0.003090	0.725023	0.000633
0.1	0.1	0.2	0.003510	0.567733	0.004405
0.1	-1.0	0.2	0.003190	0.687738	0.005754
0.1	0.05	1.0	0.002830	0.816995	0.025889

Table 6.7: The jump detection results with respect to different jump parameters.

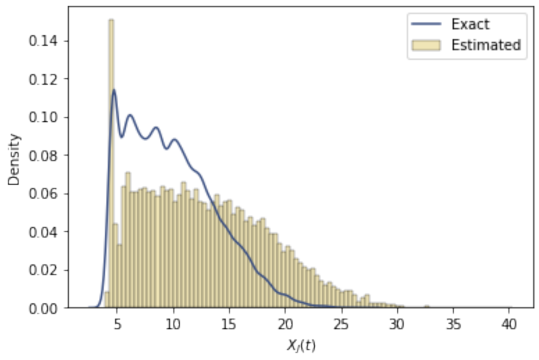
	λ_p	μ_J	σ_J	Confusion matrix	F1 score
Exact	1	-1.0	0.5	$\begin{pmatrix} 447 & 3 \\ 3 & 47 \end{pmatrix}$	99.44
Estimated	1.099436	-0.979969	0.527999		
Exact	1	0.0	0.5	$\begin{pmatrix} 444 & 0 \\ 21 & 35 \end{pmatrix}$	97.69
Estimated	0.824069	0.009215	0.651517		
Exact	1	0.0	0.2	$\begin{pmatrix} 432 & 3 \\ 44 & 21 \end{pmatrix}$	94.84
Estimated	0.546952	-0.034642	0.328174		
Exact	1	1.0	0.5	$\begin{pmatrix} 440 & 0 \\ 6 & 54 \end{pmatrix}$	99.32
Estimated	1.328758	1.192697	0.426563		

Here, we fix $\Delta t = 0.1$, $\mu = 0.2$ and $\sigma = 0.05$, and repeat the method with respect to different jump parameters.

When the jump magnitude is obvious, namely is easy to recognize visually, then the jump detection model can detect the jumps splendidly. However, when the jump-diffusion path increases to an extreme price, for example, the log price is larger than 10, then the accuracy of the MLE method is not enough, which means the estimation error leads to an obvious bias in simulation, see Figure 6.10. On the other hand, when the jump magnitude can be covered by the diffusion (see Figure 6.11), then the jump detection model fails to recognize jumps.



(a)



(b)

Figure 6.10: The detected jump-diffusion paths (left) and the histogram (right) of the jump-diffusion samples generated by the estimated parameters, where the exact parameters are $\mu_J = 1.0$ and $\sigma = 0.5$

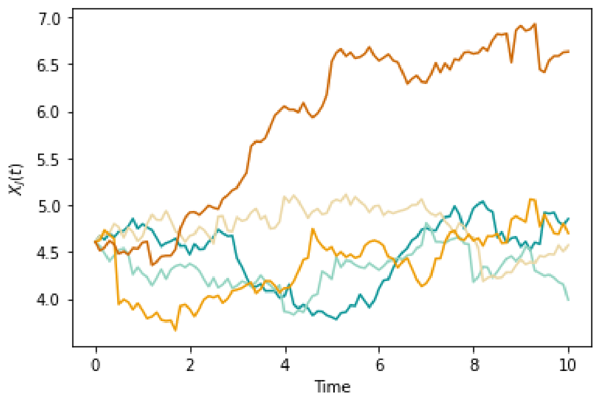


Figure 6.11: The jump-diffusion paths to be detected, where $\mu_J = 0$ and $\sigma = 0.2$.

Chapter 7

Discussion and Conclusions

7.1 Discussion

7.1.1 The 3D Dataset

We enjoy the speedup because the 3D dataset: The SDE-WGAN trains about 10 minutes on the 3D dataset, but it takes around 30 minutes to train on the 2D dataset, where the amount of the training samples are the same. however, it raises problems when applying the proposed framework to the real market data. Since we cannot directly obtain the 3D dataset, we can estimate the diffusion parameters μ and σ to construct a 3D dataset artificially. However, this approach introduces error in simulation and obeys the intention of using the SDE-WGAN. Maybe using the 2D dataset is an alternative way when applying the framework to the empirical market data.

7.1.2 Jump Detection: BiGAN

In the jump detection model, we follow the methodology of the AnoGAN to reconstruct the jumps. Instead of using the optimization method to find the "inverse image" of the jumps in the latent space, the methodology of the BiGAN [58] can be applied, where the encoder is introduced to find the "inverse map" from the generated space to the latent space. The AnoGAN algorithm could be replaced by the BiGAN to reconstruct the jumps, as the BiGAN shows better results among the experiments in [58].

7.1.3 Anomaly Score Threshold

In the jump detection model, the anomaly score threshold is decided visually, which is inaccurate and unreliable. [51] proposes an algorithm that automatically selects the anomaly threshold. The algorithm has theoretical support from the Extreme Value Theory, leading to a more reliable threshold.

7.1.4 SDE-GAN and SDE-WGAN

The authors in [15] mention that no GAN failure modes occur in the experiments. This is against to our experiments. The reason may be the difference in implementing the GAN model in practice, even though we follow the same methodology. Whatever, the Wasserstein GAN is the forefront technique and it is better to have a stable model when extending to other SDEs.

7.2 Conclusions

In this thesis, we proposed a general framework about the jump-diffusion path simulation, where the diffusion part is simulated by an conditional Wasserstein GAN with gradient penalty and the jump part involves an anomaly detection method. The framework consists of two parts: the diffusion learning part and the jump detection part. The diffusion learning part focuses on the simulation of the diffusion part by using a GAN model, which is inspired the methodology proposed in [15] (we call it the SDE-GAN). We adapt and improve the SDE-GAN by introducing the Wasserstein loss and the gradient penalty constraint, and the so-called SDE-WGAN is stable and never raises failure modes. The jump detection part is designed to detect the jumps and estimate the jump parameters, the jump part is then simulated based on the sufficient estimated values. We view the jumps are the anomalies in the jump-diffusion paths while the non-jump prices are the normal data. We then introduce a GAN-based anomaly detection technique to recognize jumps, where the SDE-WGAN is well-combined. We perform the proposed framework on the artificial data, and obtain promising results. However, the framework may fail when the jump magnitude is small. In this work, we only take the Merton's model as an instance. However, the proposed framework can be easily adapted to simulate various jump-diffusion models, as long as we can estimate the jump parameters based on the detected jump instances.

List of Figures

List of Tables

Bibliography

[1] Elton, E., Gruber, M., Brown, S. & Goetzmann, W. *Modern Portfolio Theory and Investment Analysis* (Wiley, 2014). URL `https://books.google.nl/books?id=181CEAAAQBAJ`.

[2] Black, F. & Scholes, M. The pricing of options and corporate liabilities. *Journal of Political Economy* **81**, 637–654 (1973).

[3] Kumar, P., Mallieswari, R. *et al.* Predicting stock market price movement using machine learning technique: Evidence from india. In *2022 Interdisciplinary Research in Technology and Management (IRTM)*, 1–7 (IEEE, 2022).

[4] Reddy, K. & Clinton, V. Simulating stock prices using geometric brownian motion: Evidence from australian companies. *Australasian Accounting, Business and Finance Journal* **10**, 23–47 (2016).

[5] Kou, S. G. Jump-diffusion models for asset pricing in financial engineering. *Handbooks in operations research and management science* **15**, 73–116 (2007).

[6] Sepp, A. & Skachkov, I. Option pricing with jumps. *Wilmott magazine* 50–58 (2003).

[7] Bass, R. F. Stochastic differential equations with jumps. *Probability Surveys* **1**, 1 – 19 (2004). URL `https://doi.org/10.1214/154957804100000015`.

[8] Ramezani, C. A. & Zeng, Y. Maximum likelihood estimation of the double exponential jump-diffusion process. *Annals of Finance* **3**, 487–507 (2007).

[9] Kou, S. G. A jump-diffusion model for option pricing. *Management science* **48**, 1086–1101 (2002).

[10] Tang, F. Merton jump-diffusion modeling of stock price data (2018).

[11] Brandimarte, P. *Handbook in Monte Carlo simulation: applications in financial engineering, risk management, and economics* (John Wiley & Sons, 2014).

[12] Glasserman, P. *Monte Carlo methods in financial engineering*, vol. 53 (Springer, 2004).

[13] Bally, V. & Talay, D. The law of the euler scheme for stochastic differential equations. *Probability theory and related fields* **104**, 43–60 (1996).

[14] Rouah, F. D. Euler and milstein discretization. *Documento de trabajo, Sapient Global Markets, Estados Unidos. Recuperado de www. frouah. com* (2011).

[15] van Rhijn, J., Oosterlee, C. W., Grzelak, L. A. & Liu, S. Monte carlo simulation of sdes using gans. *arXiv preprint arXiv:2104.01437* (2021).

[16] Berner, J., Grohs, P. & Jentzen, A. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of black–scholes partial differential equations. *SIAM Journal on Mathematics of Data Science* **2**, 631–657 (2020).

[17] Broadie, M. & Kaya, Ö. Exact simulation of stochastic volatility and other affine jump diffusion processes. *Operations research* **54**, 217–231 (2006).

[18] Seydel, R. & Seydel, R. *Tools for computational finance*, vol. 3 (Springer, 2006).

[19] Merton, R. C. Option pricing when underlying stock returns are discontinuous. *Journal of financial economics* **3**, 125–144 (1976).

[20] Ramezani, C. A. & Zeng, Y. Maximum likelihood estimation of asymmetric jump-diffusion processes: Application to security prices. *Available at SSRN 606361* (1998).

[21] Hanson, F. B., Westman, J. J. & Zhu, Z. Market parameters for stock jump-diffusion models. In *Mathematics of Finance: Proceedings of an AMS-IMS-SIAM Joint Summer Research Conference on Mathematics of Finance*, 155 (2004).

[22] Zenati, H., Romain, M., Foo, C.-S., Lecouat, B. & Chandrasekhar, V. Adversarially learned anomaly detection. In *2018 IEEE International conference on data mining (ICDM)*, 727–736 (IEEE, 2018).

[23] Bacry, E., Mastromatteo, I. & Muzy, J.-F. Hawkes processes in finance. *Market Microstructure and Liquidity* **1**, 1550005 (2015).

[24] Lamperti, J. *Stochastic processes: a survey of the mathematical theory*, vol. 23 (Springer Science & Business Media, 2012).

[25] Oosterlee, C. W. & Grzelak, L. A. *Mathematical modeling and computation in finance: with exercises and Python and MATLAB computer codes* (World Scientific, 2019).

[26] Klebaner, F. C. *Introduction to stochastic calculus with applications* (World Scientific Publishing Company, 2012).

[27] Brătian, V., Acu, A.-M., Mihaiu, D. M. & Șerban, R.-A. Geometric brownian motion (gbm) of stock indexes and financial market uncertainty in the context of non-crisis and financial crisis scenarios. *Mathematics* **10**, 309 (2022).

[28] Matsuda, K. Introduction to merton jump diffusion model. *Department of Economics, The Graduate Center, The City University of New York, New York* (2004).

[29] Hawkes, A. G. Hawkes jump-diffusions and finance: a brief history and review. *The European Journal of Finance* **28**, 627–641 (2022).

[30] Runggaldier, W. J. Jump-diffusion models. In *Handbook of heavy tailed distributions in finance*, 169–209 (Elsevier, 2003).

[31] Kroese, D. P., Brereton, T., Taimre, T. & Botev, Z. I. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics* **6**, 386–392 (2014).

[32] Press, W. H. & Farrar, G. R. Recursive stratified sampling for multidimensional monte carlo integration. *Computers in Physics* **4**, 190–195 (1990).

[33] Kloeden, P. E., Platen, E. & Schurz, H. *Numerical solution of SDE through computer experiments* (Springer Science & Business Media, 2012).

[34] McQuighan, P. Simulating the poisson process. *Department of Mathematics-University of Chicago* **23** (2010).

[35] Glasserman, P. & Merener, N. Convergence of a discretization scheme for jump-diffusion processes with state–dependent intensities. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **460**, 111–127 (2004).

[36] Bishop, C. M. & Nasrabadi, N. M. *Pattern recognition and machine learning*, vol. 4 (Springer, 2006).

[37] Nair, V. & Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Icml* (2010).

[38] Maas, A. L., Hannun, A. Y., Ng, A. Y. *et al.* Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, vol. 30, 3 (Atlanta, Georgia, USA, 2013).

[39] Stone, M. H. The generalized weierstrass approximation theorem. *Mathematics Magazine* **21**, 237–254 (1948).

[40] Csáji, B. C. *et al.* Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary* **24**, 7 (2001).

[41] Lu, Z., Pu, H., Wang, F., Hu, Z. & Wang, L. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems* **30** (2017).

[42] Yu, X., Efe, M. O. & Kaynak, O. A general backpropagation algorithm for feedforward neural networks learning. *IEEE transactions on neural networks* **13**, 251–254 (2002).

[43] Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, 177–186 (Springer, 2010).

[44] Hinton, G., Srivastava, N. & Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on* **14**, 2 (2012).

[45] Igel, C. & Hüsken, M. Improving the rprop learning algorithm. In *Proceedings of the second international ICSC symposium on neural computation (NC 2000)*, vol. 2000, 115–121 (2000).

[46] Duchi, J., Hazan, E. & Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* **12** (2011).

[47] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization (2014). `1412.6980`.

[48] Chandola, V., Banerjee, A. & Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)* **41**, 1–58 (2009).

[49] Maes, S., Tuyls, K., Vanschoenwinkel, B. & Manderick, B. Credit card fraud detection using bayesian and neural networks. In *Proceedings of the 1st international naiso congress on neuro fuzzy technologies*, vol. 261, 270 (2002).

[50] Siddiqui, M. A. *et al.* Detecting cyber attacks using anomaly detection with explanations and expert feedback. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2872–2876 (IEEE, 2019).

[51] Liu, S. *et al.* Time series anomaly detection with adversarial reconstruction networks. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[52] Carreño, A., Inza, I. & Lozano, J. A. Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework. *Artificial Intelligence Review* **53**, 3575–3594 (2020).

[53] Mehrotra, K. G., Mohan, C. K. & Huang, H. *Anomaly detection principles and algorithms*, vol. 1 (Springer, 2017).

[54] Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U. & Langs, G. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, 146–157 (Springer, 2017).

[55] Zenati, H., Foo, C. S., Lecouat, B., Manek, G. & Chandrasekhar, V. R. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222* (2018).

[56] Di Mattia, F., Galeone, P., De Simoni, M. & Ghelfi, E. A survey on gans for anomaly detection. *arXiv preprint arXiv:1906.11632* (2019).

[57] Goodfellow Ian, J. *et al.* Generative adversarial nets. In *Proceedings of the 27th international conference on neural information processing systems*, vol. 2, 2672–2680 (2014).

[58] Donahue, J., Krähenbühl, P. & Darrell, T. Adversarial feature learning. *arXiv preprint arXiv:1605.09782* (2016).

[59] Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein generative adversarial networks. In *International conference on machine learning*, 214–223 (PMLR, 2017).

[60] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. & Courville, A. C. Improved training of wasserstein gans. *Advances in neural information processing systems* **30** (2017).

[61] Eckerli, F. & Osterrieder, J. Generative adversarial networks in finance: an overview. *arXiv preprint arXiv:2106.06364* (2021).

[62] Goodfellow, I. *et al.* Generative adversarial networks. *Communications of the ACM* **63**, 139–144 (2020).

[63] Grohs, P., Hornung, F., Jentzen, A. & Von Wurstemberger, P. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362* (2018).

[64] Metz, L., Poole, B., Pfau, D. & Sohl-Dickstein, J. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163* (2016).

[65] Farnia, F. & Ozdaglar, A. Do gans always have nash equilibria? In *International Conference on Machine Learning*, 3029–3039 (PMLR, 2020).

[66] Mannor, S., Peleg, D. & Rubinstein, R. The cross entropy method for classification. In *Proceedings of the 22nd international conference on Machine learning*, 561–568 (2005).

[67] Biau, G., Cadre, B., Sangnier, M. & Tanielian, U. Some theoretical properties of gans. *arXiv preprint arXiv:1803.07819* (2018).

[68] Kullback, S. & Leibler, R. A. On information and sufficiency. *The annals of mathematical statistics* **22**, 79–86 (1951).

[69] Lin, J. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory* **37**, 145–151 (1991).

[70] Mirza, M. & Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).

[71] Goodfellow, I. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).

[72] Senior, A., Heigold, G., Ranzato, M. & Yang, K. An empirical study of learning rates in deep neural networks for speech recognition. In *2013 IEEE international conference on acoustics, speech and signal processing*, 6724–6728 (IEEE, 2013).

[73] Peyré, G., Cuturi, M. *et al.* Computational optimal transport. *Center for Research in Economics and Statistics Working Papers* (2017).

[74] Levina, E. & Bickel, P. The earth mover's distance is the mallows distance: Some insights from statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, 251–256 (IEEE, 2001).

[75] Weng, L. From gan to wgan. *arXiv preprint arXiv:1904.08994* (2019).

[76] Rainforth, T., Cornish, R., Yang, H., Warrington, A. & Wood, F. On nesting monte carlo estimators. In *International Conference on Machine Learning*, 4267–4276 (PMLR, 2018).

[77] Li, P. & Feng, R. Nested monte carlo simulation in financial reporting: a review and a new hybrid approach. *Scandinavian Actuarial Journal* **2021**, 744–778 (2021).

[78] Martin, J. About exchanging expectation and supremum for conditional wasserstein gans. *arXiv preprint arXiv:2103.13906* (2021).

[79] Le Cam, L. Maximum likelihood: an introduction. *International Statistical Review/Revue Internationale de Statistique* 153–171 (1990).

[80] Massey Jr, F. J. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association* **46**, 68–78 (1951).

[81] Van der Vaart, A. W. *Asymptotic statistics*, vol. 3 (Cambridge university press, 2000).

[82] Stehman, S. V. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment* **62**, 77–89 (1997).

Appendix A

Stochastic Preliminary

A.1 Random Variables

Definition A.1.1 (Bernoulli random variable). *A Bernoulli random variable X can only take on two values, 1 and 0. For a random experiment with two possible outcomes, success with probability p and failure with probability $1-p$, X takes on 1 if the experiment results in success and 0 otherwise. Such random variable can be denoted by $X \sim Ber(p)$, and the experiment is so-called a Bernoulli trail.*

The probability mass function of a Bernoulli random variable $X \sim \mathrm{Ber}(p)$ is

$$\mathbb{P}(X=1) = p, \tag{A.1.1}$$
$$\mathbb{P}(X=0) = 1-p. \tag{A.1.2}$$

Definition A.1.2 (Binomial random variable). *For n independent Bernoulli trails, a binomial random variable X represents the number of successes in those n trials, and is determined by the values of n and p, denoted by $B(n,p)$. Bernoulli random variable is a special case of the binomial random variable with $n = 1$.*

The probability mass function of a binomial random variable $X \sim \mathrm{B}(n,p)$ is

$$\mathbb{P}(X=k) = \binom{n}{k} p^k (1-p)^{n-k}, \tag{A.1.3}$$

where k is the number of successful trails ($k = 0,1,2,\dots,n$).

The expected value of $X \sim \mathrm{B}(n,p)$ is $\mathbb{E}[X] = \lambda = np$. When $n \to \infty$, we can find that (A.1.3) converges to $\frac{\lambda^k}{k!}e^{-\lambda}$, which is known as the probability mass function of a Poisson random variable, with parameter $\lambda > 0$.

Remark A.1.3 (Approximation of binomial random variable). *Let $\lambda = np$, when p is small, $\binom{n}{k} p^k (1-p)^{n-k} \to \frac{\lambda^k}{k!}e^{-\lambda}$ as $n \to \infty$, for any fixed $k \in \{0,1,2,\dots,n\}$.*

Proof. Since $\lambda = np$, we substitute $p = \frac{\lambda}{n}$ into $\binom{n}{k} p^k (1-p)^{n-k}$:

$$\begin{aligned}
\binom{n}{k} p^k (1-p)^{n-k} &= \frac{n(n-1)(n-2)\dots(n-k+1)}{k!} (\frac{\lambda}{n})^k (1-\frac{\lambda}{n})^{n-k} \\
&= \frac{n}{n}\frac{n-1}{n}\dots\frac{n-k+1}{n}\frac{\lambda^k}{k!}(1-\frac{\lambda}{n})^{n-k} \\
&= \frac{n}{n}\frac{n-1}{n}\dots\frac{n-k+1}{n}\frac{\lambda^k}{k!}(1-\frac{\lambda}{n})^{n}(1-\frac{\lambda}{n})^{-k} \\
&\overset{n\to\infty}{=} \frac{\lambda^k}{k!}e^{-\lambda}
\end{aligned} \tag{A.1.4}$$

□

Therefore, a Poisson random variable can be viewed as an approximation of the corresponding binomial random variable. In the following, we give the formal definition of a Poisson random variable.

Definition A.1.4 (Poisson random variable)**.** *A Poisson random variable X counts the number of occurrences of an event during a given time period, denoted by Pois(λ). Here, λ is equal to the expected value of X, meaning the average number of occurrences of the event, and also to its variance.*

The probability mass function of a Poisson random variable $X \sim \text{Pois}(\lambda)$ is

$$\mathbb{P}(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{A.1.5}$$

where k is the number of occurrences ($k = 0, 1, 2, \dots$).

A.2 Stochastic processes: Markov Property and Itô's Lemma

We give brief proofs with respect to the theoretical results in Section 2.1.

Poof of Theorem 2.1.6.
We only need to show that the moment generating function corresponding to the conditional distribution of $W(t+s)|\mathcal{F}_t$ is the same as $W(t+s)|W(t)$.

1. From the properties of Wiener increments, we have:

$$\begin{aligned}
\mathbb{E}[e^{uW(t+s)}|\mathcal{F}_t] &= e^{uW(t)}\mathbb{E}[e^{u(W(t+s)-W(t))}|\mathcal{F}_t] \\
&= e^{uW(t)}\mathbb{E}[e^{u(W(t+s)-W(t))}] \\
&= e^{uW(t)}e^{u^2 s/2} \\
&= e^{uW(t)}\mathbb{E}[e^{u(W(t+s)-W(t))}|W(t)] = \mathbb{E}[e^{uW(t+s)}|W(t)]
\end{aligned}$$

2. A similar strategy can be used for the Markov property of a Poisson process.

□

Proof of Itô's Lemma.
Consider an arbitrary point (t_0, X_0), and let $\Delta t = t - t_0$, $\Delta X = X - X_0$, the 2D Taylor series expansion around (t_0, X_0) is given by:

$$\begin{aligned} g(t,X) &= g(t_0,X_0) + \frac{\partial g}{\partial t}(t_0,X_0)\Delta t + \frac{\partial g}{\partial X}(t_0,X_0)\Delta X \\ &+ \frac{1}{2}\frac{\partial^2 g}{\partial t^2}(t_0,X_0)\Delta t^2 + \frac{1}{2}\frac{\partial^2 g}{\partial X^2}(t_0,X_0)\Delta X^2 + \frac{\partial^2 g}{\partial t \partial X}(t_0,X_0)\Delta t \Delta X \\ &+ o(\Delta t^3 + \Delta X^3) \end{aligned}$$

For $\Delta t \to 0$, $\Delta X \to 0$, and $\mathrm{d}t = \lim_{t \to t_0} t - t_0$, $\mathrm{d}X = \lim_{X \to X_0} X - X_0$, we have

$$\mathrm{d}g(t,X) = \frac{\partial g}{\partial t}\mathrm{d}t + \frac{\partial g}{\partial X}\mathrm{d}X + \frac{1}{2}\frac{\partial^2 g}{\partial t^2}\mathrm{d}t^2 + \frac{1}{2}\frac{\partial^2 g}{\partial X^2}\mathrm{d}X^2 + \frac{\partial^2 g}{\partial t \partial X}\mathrm{d}t\mathrm{d}X + o(\mathrm{d}t^3 + \mathrm{d}X^3).$$

For any $n > 1$, $\mathrm{d}t^n$ goes to 0 faster than $\mathrm{d}t$, we can therefore neglect the higher order $\mathrm{d}t$-terms.

Regarding $\mathrm{d}X^2 = (\bar{\mu}(t, X(t))\mathrm{d}t + \bar{\sigma}(t, X(t))\mathrm{d}W(t))^2$, we need to determine two terms: $\mathrm{d}t\mathrm{d}W$ and $\mathrm{d}W^2$. Since $\mathbb{E}[\mathrm{d}t\mathrm{d}W] = 0$ and $\mathbb{V}\mathrm{ar}[\mathrm{d}t\mathrm{d}W] = (\mathrm{d}t)^{\frac{3}{2}}$, $\mathrm{d}t\mathrm{d}W$ goes to 0 rapidly when $\mathrm{d}t \to 0$, we can also neglect this term. With respect to $\mathrm{d}W^2$, we claim that $(\mathrm{d}W)^2$ is of order $\mathrm{d}t$.

By the Gaussian property of Wiener increment, we have

$$\mathbb{E}[\mathrm{d}W^2] = \lim_{\Delta t \to 0} \mathbb{E}[(W(t+\Delta t) - W(t))^2] = \lim_{\Delta t \to 0} \Delta t = \mathrm{d}t,$$

and

$$\begin{aligned} \mathbb{V}\mathrm{ar}[\mathrm{d}W^2] &= \lim_{\Delta t \to 0} \mathbb{E}[(W(t+\Delta t) - W(t))^4] - (\mathbb{E}[(W(t+\Delta t) - W(t))^2])^2 \\ &= \lim_{\Delta t \to 0} 3(\Delta t)^2 - (\Delta t)^2 = 2(\mathrm{d}t)^2 \end{aligned}$$

As the variance of $\mathrm{d}W^2$ converges to 0 faster than the expectation, we conclude that $(\mathrm{d}W)^2$ is of order $\mathrm{d}t$.

Combining all together, we have

$$\mathrm{d}g(t,X) = (\frac{\partial g}{\partial t} + \bar{\mu}(t, X(t))\frac{\partial g}{\partial X} + \frac{1}{2}\bar{\sigma}^2(t, X(t))\frac{\partial^2 g}{\partial X^2})\mathrm{d}t + \bar{\sigma}(t, X(t))\frac{\partial g}{\partial X}\mathrm{d}W(t).$$

□

Proof of Itô's Lemma for Poisson processes.
We first claim the product rule of $\mathrm{d}t$ and $\mathrm{d}X_{\mathcal{P}}(t)$, as is shown in Table A.1. From the

property of Poisson process, we have

$$\mathrm{d}X_{\mathcal{P}}(t)=\begin{cases}1, & \text{with probability } \lambda_p \mathrm{d}t\\ 0, & \text{with probability } 1-\lambda_p \mathrm{d}t\end{cases}. \tag{A.2.1}$$

Since the expectation of $\mathrm{d}t\mathrm{d}X_{\mathcal{P}}(t)$ equals $\lambda_p(\mathrm{d}t)^2$ and the standard deviation equals $\sqrt{\lambda_p}(\mathrm{d}t)^{\frac{3}{2}}$, $\mathrm{d}t\mathrm{d}X_{\mathcal{P}}(t)$ goes to 0 rapidly when $\mathrm{d}t \mapsto 0$. Regarding the item $(\mathrm{d}X_{\mathcal{P}}(t))^2$, we have

$$\begin{aligned}(\mathrm{d}X_{\mathcal{P}}(t))^2 &= \begin{cases}(1)^2, & \text{with probability } \lambda_p \mathrm{d}t\\ (0)^2, & \text{with probability } 1-\lambda_p \mathrm{d}t\end{cases}' \\ &= \mathrm{d}X_{\mathcal{P}}(t).\end{aligned} \tag{A.2.2}$$

Table A.1: Itô multiplication table for Poisson process.

	$\mathrm{d}t$	$\mathrm{d}X_{\mathcal{P}}(t)$
$\mathrm{d}t$	0	0
$\mathrm{d}X_{\mathcal{P}}(t)$	0	$\mathrm{d}X_{\mathcal{P}}(t)$

We now return to the proof of Equation (2.1.15). By using (A.2.1), we can write

$$\begin{aligned}\mathrm{d}g(t,X(t)) &= g(t+\mathrm{d}t, X(t\_)+\mathrm{d}X)-g(t,X(t\_))\\ &= g(t+\mathrm{d}t, X(t\_)+\bar{\mu}\mathrm{d}t+\bar{J}\mathrm{d}X_{\mathcal{P}}(t))-g(t,X(t\_))\\ &= g(t+\mathrm{d}t, X(t\_)+\bar{\mu}\mathrm{d}t+\bar{J})\mathrm{d}X_{\mathcal{P}}(t)\\ &\quad + g(t+\mathrm{d}t, X(t\_)+\bar{\mu}\mathrm{d}t)(1-\mathrm{d}X_{\mathcal{P}}(t))-g(t,X(t\_))\\ &= \left[g(t,X(t\_)+\bar{J})+\frac{\partial g(t,X(t\_)+\bar{J})}{\partial X(t)}\bar{\mu}\mathrm{d}t+\frac{\partial g(t,X(t\_)+\bar{J})}{\partial t}\mathrm{d}t\right]\mathrm{d}X_{\mathcal{P}}(t) \\ &\quad + \left[g(t,X(t\_))+\frac{\partial g(t,X(t\_))}{\partial X}\bar{\mu}\mathrm{d}t+\frac{\partial g(t,X(t\_))}{\partial t}\mathrm{d}t\right](1-\mathrm{d}X_{\mathcal{P}}(t))-g(t,X(t\_))\\ &= \left[\frac{\partial g}{\partial t}+\bar{\mu}\frac{\partial g}{\partial X}\right]\mathrm{d}t+\left[g(t,X(t\_)+\bar{J})-g(t,X(t\_))\right]\mathrm{d}X_{\mathcal{P}}(t)\end{aligned},$$

here, $\bar{\mu}$ and $\bar{J}$ represent $\bar{\mu}(t,X(t))$ and $\bar{J}(t,X(t\_))$ respectively. □

Appendix B

SDE-WGAN

B.1 SDE-WGAN Architecture

Both the generator and critic are MLPs, and their architectures are formed in Table B.1 and Table B.2, where d is the depth of the 3D training dataset. During the training of SDE-WGAN, a learning rate scheduler is applied, that is, the learning rate lr decays in every p training epochs with rate γ, and minimum learning rate is $\min lr$.

Table B.1: Generator architecture

Optimiser	RMSProp lr = 0.0003, $\beta = 0.99$	
Layer	Nodes	Activation
Input layer	$3d$	LeakyReLU, negative slope=0.1
Hidden 1	100	LeakyReLU, negative slope=0.1
Hidden 2	100	LeakyReLU, negative slope=0.1
Hidden 3	100	LeakyReLU, negative slope=0.1
Hidden 4	100	LeakyReLU, negative slope=0.1
Output layer	d	None

Table B.2: Critic architecture

Optimiser	RMSProp lr = 0.0003, $\beta = 0.99$	
Layer	Nodes	Activation
Input layer	$3d$	LeakyReLU, negative slope=0.1
Hidden 1	100	LeakyReLU, negative slope=0.1
Hidden 2	100	LeakyReLU, negative slope=0.1
Hidden 3	100	LeakyReLU, negative slope=0.1
Hidden 4	100	LeakyReLU, negative slope=0.1
Output layer	1	None

B.2 Hyperparameter Tuning

We process a hyperparameter tuning regarding the following hyperparameters: batch size M, the iteration number of the critic N_c, the gradient penalty coefficient λ, the minimum learning rate $\min lr$, the learning rate scheduler parameters (m, γ). The tuning results are shown in Figure B.1, where $M = 100$, $N_c = 9$, $\lambda = 60$, $\min lr = 10^{-8}$, and $(m, \gamma) = (20, 0.3)$.

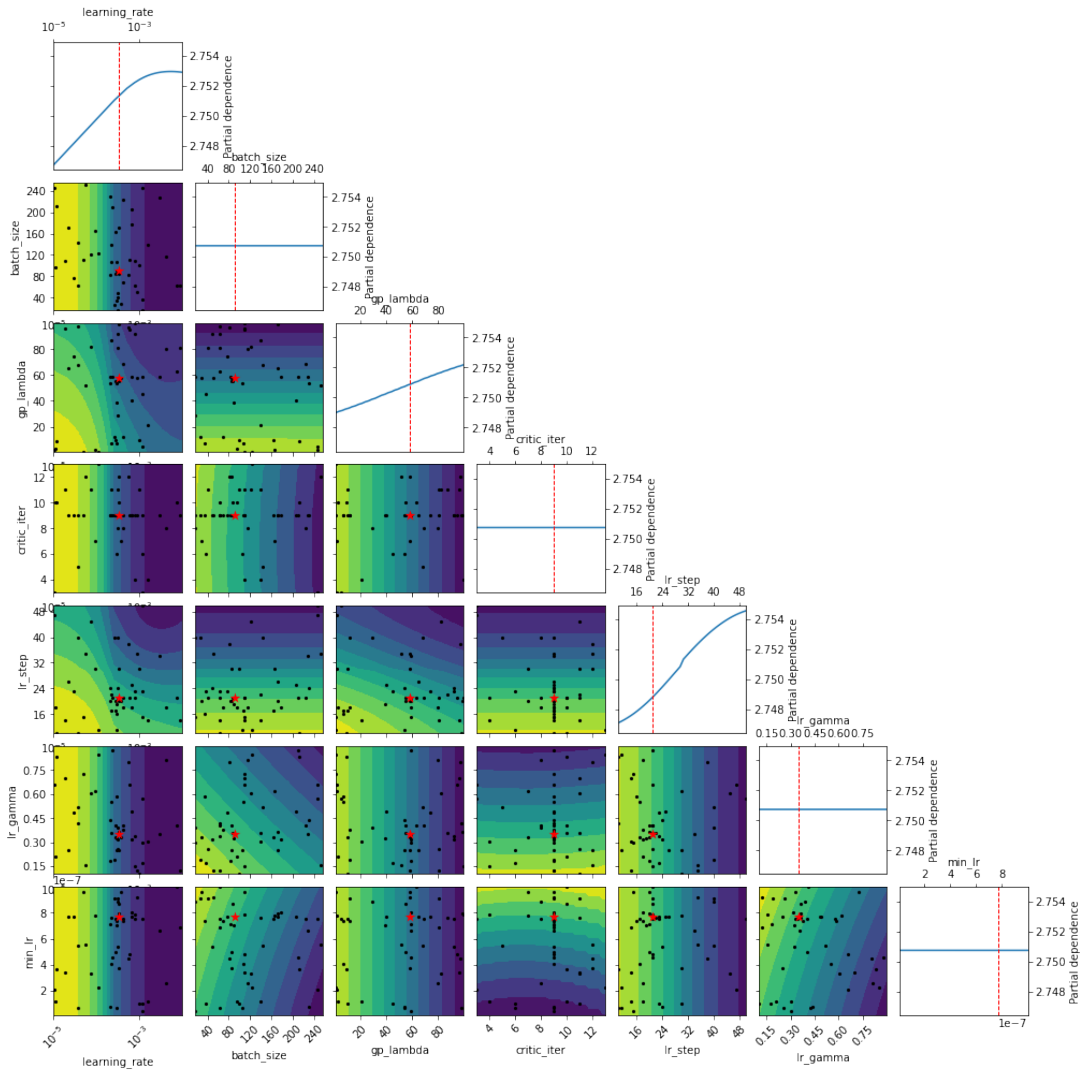


Figure B.1: The results of the hyperparameter tuning.