University of Europe for Applied Sciences

Software Engineering

Thesis

Investigating the effectiveness of synthetic data in financial applications

Aniket Dere

Matriculation Number: 60334138

July 31, 2024

Supervised by
Prof. Dr. Iftikhar Ahmed

Assistant Supervisor
Prof. Dr. Talha Ali Khan

Statutory Declaration:

I hereby declare that I have completed the enclosed Master Thesis entirely by myself and without unauthorized assistance. I affirm that I have not used any sources or means other than those indicated. All literature and other sources that I have employed, whether quoted directly or paraphrased, are clearly marked and separately listed in the references section. I understand that any violation of these rules will result in the failure of my thesis.

Berlin, July 31, 2024

[illegible]

. . . .

(Signature Aniket Dere)

Abstract

The financial sector increasingly relies on data-driven decision-making, yet the sensitive nature of financial data poses significant privacy concerns. This thesis addresses the critical problem of balancing data utility and privacy by investigating the effectiveness of synthetic data in financial applications, particularly in the stock market. The study evaluates synthetic data generated using advanced methodologies such as TimeGAN, Long Short-Term Memory (LSTM) networks, and block bootstrapping. The primary objective is to assess the quality of synthetic financial datasets in replicating real data through statistical similarity, predictive accuracy, visual congruence, and discriminative metrics.

Methodologically, the thesis focuses on generating synthetic time series data from Gold stock prices spanning January 2014 to January 2024. The generated synthetic data are compared against real data to evaluate the performance and utility of different synthetic data generation techniques. Results indicate that while synthetic data can enhance privacy and provide cost-effective alternatives, they may struggle with replicating complex temporal dependencies and can introduce biases for some algorithms.

The findings emphasize the importance of careful methodological selection in synthetic data generation, which can significantly impact the usability of synthetic datasets in real-world financial applications. This research contributes to the broader understanding of synthetic data's role in finance, highlighting both its potential and the caution required in its application. The study concludes with recommendations for improving synthetic data generation techniques to better serve the financial sector's needs.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In today's era of data and data based decisions, financial industry is intrinsically data-driven, using huge datasets for data driven decision making processes, optimize strategies and keeping regulatory compliance. Financial organizations use data for multitude applications, including but not limited to, risk management, algorithmic trading, fraud detection, portfolio management and analysis of customer behaviour. These operations requires high quality datasets, using which foundation of models and strategies are designed.

Although, use of real financial data struggle with significant challenges. One of the main challenge is data privacy. Financial datasets often include highly private and sensitive information of individual from the corporations. This information includes personal identification details, transaction history, investment records and institutions private information. This sensitive information exposure can lead to severe damage to organizations as well as person and result into issues like financial loss, identity theft and reputational damange. To minimize problems like this stringent regulations as "The General Data Protection Regulation(GDPR)" in Europe GDPR (2016) and "The California Consumer Privacy Act(CCPA)" in United States CCPA (2018) has been sanctioned. The regulations helps in achieving the strict policies against accessing, using and sharing private personal records which ultimately result into less availability of real financial datasets for analysis and research (Künsch, 1989).

Data scarcity is another such challenge faced by financial institutes. In some scenarios, gathering enough historical data for specific scenario is difficult job. The scarcity can be caused by different scenarios like new emergence of financial instruments , regulatory challenges which makes market dynamics and data required for specific events. This problems result into limited data availability and hampers the ability to develop robust models to analyze the patterns in data, particularly where specific scenario requires extensive historical data (Ariyo et al., 2014).

Additionally, the cost of acquiring such high-quality financial data is huge and often prohibitive. Many financial organizations offer this kind of data with substantial fees which make small scale organizations, researchers and start ups hard to get their hands on it. This financial burden restrictions often become reason for less innovations and democratization of financial data in the industry.

Synthetic data generation answers most of the research questions across many domains like healthcare, finance and autonomous system. Artificial algorithms are used to generate synthetic data, mimicking the statistical properties of original data.Particularly in finance sector, synthetic data can help with privacy concerns, regulatory compliance and data needed for training machine learning models for under different situations (Patki et al., 2016).

Recent developments in machine learning have introduced advanced techniques for creating synthetic data. The Long Short-Term Memory (LSTM) networks, first described by Hochreiter and Schmidhuber (1997), are a type of recurrent neural network (RNN) designed to maintain long-term dependencies in sequential datasets (Hochreiter and Schmidhuber, 1997). Another notable approach is Generative Adversarial Networks (GANs), as outlined by Goodfellow et al. (2014), which generate realistic synthetic data by employing a competitive training process between two neural networks (Goodfellow et al., 2014). For handling time-dependent data, Time-Series GANs (TimeGAN), introduced by Yoon et al. (2019), extend GANs to address the unique challenges of temporal dependencies in time series (Yoon et al., 2019). Furthermore, the block bootstrap technique, a non-parametric method for resampling, has been widely adopted to create synthetic time series data by resampling blocks of the original data (Künsch, 1989).

Given above challenges, there is constant need of alternative solution for increasing demand of datasets which can provide financial data without the associated limitations. Synthetic data promises solution to these problems without consequences. Synthetic data is artificially generated data which has same statistical properties and patterns as real data. By using synthetic data we can have the characteristics of real financial data with possible solutions over privacy concerns, data scarcity and affordable cost.

1.2 Problem Statement

Investigate the effectiveness of synthetic data in improving fraud detection and algorithmic trading outcomes in the financial domain, with a focus on evaluating performance, comparing approaches, identifying benefits, analyzing limitations, exploring use cases, proposing recommendations, and contributing to knowledge advancement.

Though synthetic data holds strong promise for data, its effectiveness to replicate the behavior and patterns in real financial data need promising evaluation metrics. The primary problem addressed in this thesis is: How effective is synthetic data for various financial applications?

1.3 Objective

The objective of this research is to evaluate the effectiveness of synthetic data in financial applications. Specifically, the study aims to:

1. Generate synthetic financial data: Employ various algorithms such as TimeGAN, LSTM, and block bootstrapping to generate synthetic financial datasets.
2. Compare synthetic and real data: Evaluate the synthetic data against real data in terms of statistical similarity, predictive performance, Visual similarities, and discriminative metrics in financial models.
3. Assess advantages and limitations: Identify the potential benefits and drawbacks of using synthetic data in financial contexts.

This research paper provide a extensive understanding of how synthetic time series data can be utilized effectively in the financial industry and to what extent it can replace or augment real data along with its use cases in finance industry.

1.4 Scope

This thesis mainly focuses on the generation and evaluation of synthetic financial time-series data using three main methodologies:

1. **TimeGAN (Time-series Generative Adversarial Networks):** "TimeGAN is a generative model specifically designed for time-series data. TimeGAN has combined strength of Generative Adversarial Network(GANs) as well as Recurrent Neural Networks (RNNs) which help it in capturing the temporal dynamics of time series data very well. TimeGAN has been shown to generate high-fidelity synthetic data having high ability to preserve the temporal correlations present in real datasets (Yoon et al., 2019)."
2. **LSTM (Long Short-Term Memory networks):** "LSTMs are a type of RNN capable of learning long-term dependencies in sequential data. They are widely used in time-series forecasting and sequence generation tasks. In this research paper, LSTMs are used

to model and generate synthetic financial time-series data, capturing the sequential patterns inherent in the data (Hochreiter and Schmidhuber, 1997)."

3. **Block Bootstrapping:** "Block bootstrapping is a resampling technique used to generate synthetic data by preserving the auto-correlation structure of the original time series. Unlike simple bootstrapping, which samples individual data points separately, samples of contiguous blocks of data, maintaining the temporal dependencies and correlations present in the original series is used by block bootstrapping (Künsch, 1989)."

The scope of this research includes but not limited to detailed comparison of above methodologies with respect to ability to replicate real financial times series data characteristics and their daily usage in financial applications. Synthetic time series data generated by above algorithms will be carefully evaluated across various metrics like statistical similarity with real data, predictive performance in financial models , descriminative metrics with respect to real world time series data and possible applicability in applications of risk assessment, fraud detection or anomaly detection.

This thesis also explores the potential advantages and limitations of synthetic financial time-series data. Advantages like enhanced data privacy, the ability to augment real datasets for better model training, and cost-effective access to high-quality data. Also some limitations like poor ability to accurately replicating the complex patterns, possible biases in generated synthetic data and the computational cost associated with generation of high-fidelity synthetic datasets.

1.5 Thesis Organization

Chapter 2 delves into a comprehensive literature review, exploring existing research done on synthetic data in past years and its applications in financial contexts. Chapter 3 outlines the methodology, detailing the processes and tools used for generating and evaluating synthetic financial data. Chapter 4 presents the results, analyzing the effectiveness of different synthetic data generation techniques and discussing their implications in the financial sector. Finally, chapter 5 concludes the thesis, summarizing the findings, discussing their broader impact, and suggesting directions for future research.

2 Literature Review

2.1 Introduction

Since last few decades, financial industries are very much interested in implementing Artificial Intelligence (AI) and Machine Learning(ML) techniques to understand complex data and forecast the decision based on result. These machine learning models or techniques requires huge quantity of data to train the models to make valid and highly accurate decisions. Unfortunately, banks can not consider sharing data generated by their customers. Most of the data generated in banks is private and crucial if leaked or shared. Customers also can not allow banks to share their private data therefore, data needs to be generated synthetically. Financial applications generally generate time series data over the period of time therefore time series data need to feed to deep learning or statistical machine learning algorithms. Synthetic time series data has lots of advantages like augmenting incomplete and limited datasets, mimic same datasets considering different market scenarios, provide data without private or confidential information. This provides alternatives for research and analysis purpose (Patki et al., 2016).

This literature review aims to provide detailed overview of techniques and methods to generate synthetic time series datasets in the area of financial applications using deep learning and statistical learning approaches. After studying and carefully examining these various techniques such as Recurrent Neural Networks(RNNs), Generative Adversarial Networks(GANs) and Bootstrapping methods have been used for synthetic data generation. I aim to elucidate respective strengths and weaknesses of deep learning and statistical learning techniques in different financial applications. Furthermore the discussion on trends, challenges and future directions in the field of synthetic time series data.

Our fundamental aim is to resolve below research questions in the field of synthetic time series data with financial application perspective.

- What are the techniques to generate time series financial data such as stock prices, fraud detection, bitcoin price prediction and gold prices prediction etc?
- Which evaluation matrices we can use to calculate the performance of the different techniques and algorithms?
- How to implement most effective method to generate the synthetic data ?
- What are the future directions to work in the field of synthetic time series financial data?

Sezer et al. (2020) has studied similar financial time series forecasting algorithms with focus on deep learning algorithms and forecasting the financial time series data (Sezer et al., 2020). Similarly, this literature review intent to study deep learning.

2.1.1 Overview of Synthetic Time Series Data

Synthetic time series data means data which is generated by machine learning or deep learning algorithms over the period of certain time intervals which mimic the properties, pattern

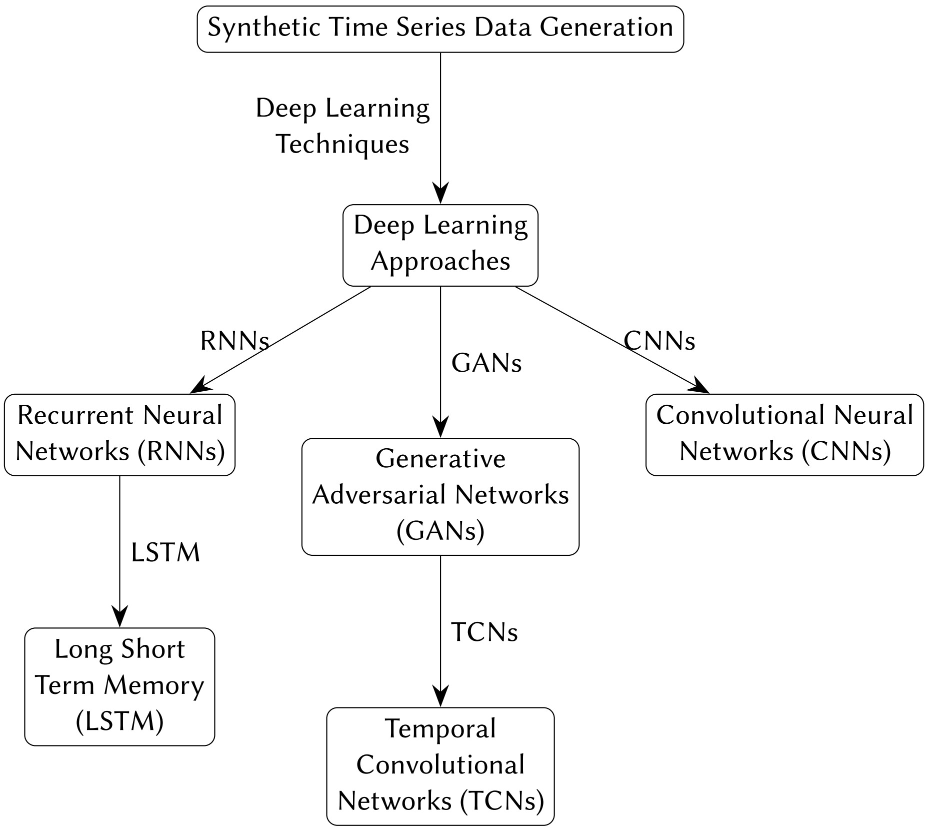

Synthetic Time Series Data Generation
Deep Learning Techniques
Deep Learning Approaches
RNNs
GANs
CNNs
Recurrent Neural Networks (RNNs)
Generative Adversarial Networks (GANs)
Convolutional Neural Networks (CNNs)
LSTM
TCNs
Long Short Term Memory (LSTM)
Temporal Convolutional Networks (TCNs)

Figure 2.1: Illustration of deep learning approaches for synthetic time series data generation.

and behaviour of the real world time series data. We use this synthetically created time series data for various purposes like analytics or predictive tasks. Real world time series data is often rare, limited, incomplete or expensive to obtain in such scenarios, we can use this synthetically generated data to train our algorithms. Real world data also has privacy vulnerable insights which might cause trouble as per different laws and regulations therefore it is very hard for analysts to use real world data and that is where synthetic data make perfect use. Synthetic data is realistic,versatile and scalability as per users need, however some concerns arrives with authenticity of synthetic data also evaluation of such data is challenging task. Generation of synthetic data may induce biases while generating data along with implications on real world applications of that data. We will discuss, how we can use financial synthetic time series data for various applications. Potluru et al. (2023) from JP Morgan research team describes various methods used to generating synthetic data in today's AI world. ChatGPT4, DALLE these generative models in the field of synthetic text generation or image generation are already using real world data as well as creating vast amount of synthetic data based on the results and existing data.

Yoon et al. (2019) In his paper, he discussed how to preserve the 'temporal dynamics in the sense that new sequences respect the original relationships between variables across time.' According to him existing old algorithms do not help in achieving temporal correlations to uniquely generate the time series data as much as General Adversarial Networks(GANs).

2.1.2 Importance of Synthetic Data in Financial Applications

Synthetic data in financial industry is growing at increasing scale to make its dependability by enhancing the analytical and predictive capabilities of the models. It resolves lots of problems faced by researches and practitioners by mimicking the characteristics of the real world financial time series data.

Synthetic data is most important component in training the machine learning or deep learning models especially when data is related to financial applications and can be generated over time period with privacy constraints. It helps model to learn from diverse set of data available for different scenarios or market conditions. For instance VAN PARIDON (2022) highlights the use of Time-Series Generative Adversarial Network (T-GANs) in generation of stock market time series data as T-GANs gives realistic data compared to other algorithms. This data is used to train predictive models, enhance the performance of the model and robustness by making vast variaty of data available. Similarly Time series Convolutional Generative Adversarial Network (TCGANs) demonstrate the use of convolutional GANs for generating time series synthetic data which improves accuracy of classification and clustering tasks. This method helps in dealing with the impurity or noise added in real world data.

Furthermore, this data helps financial industries to do the stress testing and risk management simulations as you can create data with extreme market conditions and rare scenarios, strategies without relying on past historical rare events. Synthetic data will give same statistical properties and correlation of real data, providing challenging and realistic scenarios for testing. Creating synthetic data is also cost effective than gathering and maintaining huge volume of data. it addresses the privacy concerns as synthetic data does not have any private sensitive information which could lead the breaches. This helps in the area of institutes where data privacy and protection is more important. They can use synthetic data to comply with the regulations and still do the analysis and model training. Moreover it fosters innovation by providing flexibility for researchers innovations by providing various scenarios and model configurations. Synthetic data also allows the rapid prototyping and testing new algorithms and strategies in risk free environment. Sezer et al. (2020) notes the increasing use of deep learning models, require huge amounts of data for training. Synthetic data satisfy this demand, providing advancements in financial forecasting and analysis.

Continuous improvements in deep learning algorithms specially GANs ensure that the synthetic data becomes increasingly realistic and useful for researchers. These inventions will lead better performances in the financial applications by pushing the boundaries what synthetic data actually can do.Incorporating synthetic data into financial applications improves the robustness, reliability, and fairness of financial models, resulting into better decision-making and risk management in the finance industry.

2.2 Theoretical Background

2.2.1 Definition of Time Series Data

Time series data is the sequence of data gathered over certain time period with uniform interval of data points. This type of data records the temporal dynamics of a variable and is used to analyze trends, seasonal patterns, and cyclical movements. In finance, time series data typically records it for historical records of prices, volumes, interest rates, and other financial metrics, allowing them the modeling and prediction of market behaviors based on past trends and behaviours. Each observation in a time series is basically dependent on past values, making it important for forecasting and decision-making for futuristic decistions.

2.2.2 Types of Time Series Data in Finance

Time series data in finance consists of a wide range of types that are most important for analyzing and predicting market behavioral output. This data is collected over period of time and consists of various financial indicators which reflect the state and movements of financial markets. Different types of financial time series data is most important factor for developing accurate financial models and strategies for its applications.

Stock Prices
Stock prices are perhaps the most commonly analyzed type of financial time series data. This

data tell information about the prices at which stocks were purchased and sold over the time period. This data is important for technical analysis to study the past trends and predict future prices from the trends and factor present. Stock prices are recorded with same kind of intervals, such as daily, hourly, or minute-by-minutes this shows trends of current market, flexibility of market and how investors are looking at the market. VAN PARIDON (2022) has explored various types of stock exchange data available in market and tried to mimic with same market trends possible.

Exchange Rates
Exchange rates means the currency value of one country against another country. Mostly it is calculated for the most of the other countries currencies. Time series data of exchange rates can help in predicting future currency values which can help in making informed decisions in forex trading, risk management, and economic policy-making. The rapid changes in exchange rates can be seen because of variety of factors like countries policy for foreign currencies, interest rate of countries and political events happening all over.

Market Indices
Time series data of market indices shows trend of overall market and are used to measure market performance, by comparing with different time periods, and conduct sector analysis for specific time interval. These indices are essential for portfolio management and benchmarking individual investment in the broad market. The performance of market indices tells about investor's sentiment and bigger picture of economic trends. Various factors show shows how stocks are changing for particular stock market like S&P 500, Nifty 50 and NASDAQ.

Commodity Prices
Commodity prices is the historical data of the prices of commodities like gold, oil, and agricultural products. This data plays important role in understanding supply and demand of products, inflationary trends, and economic impact because of the commodity price fluctuations. Investors and analysts using the time series data on commodity prices to forecast future price fluctuations and derive educated trading decisions for the commodity markets. For example, Gold prices are influenced by factors such as geopolitical stability, inflation, currency fluctuations and demand in market etc.

Volatility Measures
Volatility measures, examples the VIX (Volatility Index), helps in providing insights into markets uncertainty and risk. Time series data of volatility measures can help in predicting market sentiment and the possible feasibility of significant price changes. This data is used in risk management, derivative pricing, and constructing strategies to determine potential losses because of the market volatility. High level volatility frequently indicates increased uncertainty and potential market corrections.

2.2.3 Challenges in Generating Synthetic Time Series Data

While generating synthetic time series data, mainly in finance, faces many challenges which should be addressed individually and try to resolve them correctly to avoid any future miss leading.

Capturing Complex Dynamics and Statistical Properties:
Potluru et al. (2023) highlight that financial time series data frequently contains complex distributions and temporal correlations, making it challenging for synthetic data generation models to accurately replicate these intricacies. The main problem in creating synthetic time series data is to make sure it accurately mirrors the statistics and patterns which are present in real-world data. Financial time series data, example stock prices, exchange rates, has com-

plex behaviors such as trends, seasonality, unpredictability, clustering, and sudden changes caused by some external market events. It's hard to capture these features in synthetic data, and all these differences from real-world data can make the synthetic data unsafely or even misleading for analysis and model training for researchers.Huang and Deng (2023) emphasized that financial time series exhibit long-term dependencies, which are difficult to capture accurately in synthetic data.However Wiese et al. (2020) discuss the challenges which are facing in generating synthetic financial time series that replicate inner trends, such as volatility clustering, fat-tailed distributions, and long-range dependencies.

Data Scarcity and Quality:
Data scarcity is a noteworthy challenge in the field of financial time series data, specially while generating synthetic data. Potluru et al. (2023) mention that 'financial data can be scarce or have quality issues, making it difficult to train generative models effectively'. Raghunathan (2021) points out that data scarcity, mainly for rare events like market crashes, where it can be significantly challenging to generate synthetic time series data. Huang and Deng (2023) discuss the issue of generated data quality as limited labeled data in time series classification tasks, which later can be addressed by using generative models like GANs.

Privacy and Confidentiality Concerns:
Potluru et al. (2023) underscore the necessity for synthetic data generation in finance to address privacy concerns due to the inherently sensitive nature of financial information. He proposed 6 levels defence guide to generate synthetic time series data in financial application which doesn't affect the privacy.

Computational Resources and Efficiency:
Huang and Deng (2023) discuss the challenges faced in training deep learning models, mainly with huge datasets and intricate architectures. Authors highlight that insufficient memory can be a significant issue when dealing with extensive financial time series data. The authors had to limit the training period and data size due to memory constraints. Therefore the proposed Time-Series Convolutional Generative Adversarial Network (TCGAN) model demonstrates significant gains over other time-series GAN architectures such as TimeGAN and CotGAN, mostly for both training and hypothesis speed. This improved efficiency is basically attributed to the use of convolutional layers in the TCGAN framework, which provides parallelization and connect the computational burden with recurrent architectures.

Evaluation and Validation of Synthetic Time Series Data:
Evaluating the quality of synthetic time series data is important challenge which is highlighted by multiple sources.Potluru et al. (2023) emphasized the need for a comprehensive method in evaluating synthetic data. While conventional evaluation metrics such as the Kolmogorov-Smirnov and Chi-Squared tests can assess distributional likeness, they might fall short in assessing the data's suitability for precise financial tasks. The authors advocate for the incorporation of domain-specific indicators such as stylized fact analysis for financial time series and the utilization of tools like T-SNE plots for visual evaluation. also Shukla and Poornima (2023)acknowledged the problem in evaluating synthetic stock market data and use error metrics like Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE) to compare the performance of neural networks trained on real and synthetic data to evaluate and validate the synthetic data generated. **Nearest Neighbor Distance (NND):** Calculates the average distance between each synthetic sample and its nearest neighbor in the real dataset

$$\text{NND} = \frac{1}{n} \sum_{i=1}^{N} \min_{j=1,\ldots,N} \|\hat{x}_i - x_j\|_2, \tag{2.1}$$

where:

- $\hat{x}_i$ represents a point in the generated distribution,
- x_j represents a point in the historical distribution, and
- n is the total number of points in the generated distribution.

Wiese et al. (2020) focused on evaluating the distributional and dependant properties of generated financial time series. They used metrics like the Earth Mover Distance (EMD) and DY metric to assess distributional similarities and employed auto correlation function (ACF) scores and leverage effect scores to evaluate architecture.

Earth Mover Distance (EMD): Quantifies the minimum cost of transforming the distribution of real data into the distribution of synthetic data

$$\mathrm{EMD}(P_h, P_g) = \inf_{\pi \in \Pi(P_h, P_g)} \mathbb{E}_{(X,Y)\sim\pi}[\|X - Y\|], \tag{2.2}$$

where:

- P_h represents the historical distribution,
- P_g represents the generated distribution, and
- $\Pi(P_h, P_g)$ is the set of all joint distributions with marginals P_h and P_g.

Mean Maximum Discrepancy (MMD): Measures the distance between distributions of real and synthetic data.

$$\mathrm{MMD}^2 = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j\neq i}}^{N} K(x_i, x_j) + \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j\neq i}}^{N} K(\hat{x}_i, \hat{x}_j) - \frac{2}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} K(x_i, \hat{x}_j) \tag{2.3}$$

where:

- $K(\cdot, \cdot)$ is a kernel function (e.g., RBF kernel).
- x_i are real samples.
- $\hat{x}_i$ are synthetic samples.

2.3 Deep Learning Approaches for Generating Synthetic Time Series Data

2.3.1 Introduction to Deep Learning Techniques

Deep learning is a subset of machine learning where many layers of neural networks are involve that is why it is called 'Deep'. These neural networks are able to learn and model complex patterns and relationships present in data present. Unlike traditional machine learning techniques, deep learning models are capable of extracting features from raw data automatically through many layers of abstraction. This ability makes them more suitable to use against time series data , which often involves complex temporal dependencies and patterns (Wang and Liu, 2018).

Advantages of Deep Learning for Time Series Data

Deep learning overcome many challenges faced by traditional methods resulting into many advantages. Khandani et al. (2010) discussed below challenges:

- **Automatic Feature Extraction:** Deep learning models become capable of extracting features from raw data minimizing the need for manual feature extraction.
- **Nonlinear Relationships:** They can capture nonlinear relationship between the raw data which make them more flexible, accurate in creating the model for real world problems.
- **Temporal Dependencies:** Recurrent Neural Networks (RNNs) and their variants (LSTMs and GRUs) are designed mainly for handling sequential data while preserving the temporal dependencies which plays vital role in analysis of time series data.
- **Scalability:** Deep learning models are designed to handle huge data when supplied with so they are highly scalable and high-dimensional datasets, this quality make them perfect for generating large quantity of synthetic data.
- **Transfer Learning and Pretrained Models:** Using Transfer learning techniques, pretrained models can be train with similar related work which improve efficiency and performance in time series analysis along with saving time.

Throughout the literature review, we will be going through Multiple deep learning models which are discussed by researchers in various research papers.
Below is the list of Deep learning algorithms being discussed.

1. Time Series Generative Adversarial Networks(TimeGANs)
2. Recurrent Neural Networks (RNNs)
3. Temporal Convolutional Networks (TCNs)
4. Autoencoders (AEs)
5. Diffusion Models
6. Long Short-Term Memory(LSTM)

2.3.2 Generative Adversarial Networks(GANs)

Generative Adversarial Networks

Introduction

Generative Adversarial Networks (GANs) became as a groundbreaking advancement and got widely accepted in meantime all over by machine learning and artificial intelligence researchers since 2014 when it was introduced by Goodfellow et al..They represented special area of neural networks mostly in generating synthetic data which mimic the real data. It is comprising of two main components: a Generator and a Discriminator. Both of them have simple functionality, Generator creates synthetic data samples with some defects where discriminator evaluates those created samples if they are authentic and this process will carry on till discriminator is unable to distinguish between real data and synthetic data generated by generator. This process make it highly effective to generate highly realistic data and it make it valuable for wide range of applications including images creation, data creation for banking application, data augmentation and beyond.

The traditional machine learning algorithms mostly focused on labelled data for supervised learning and this process is quite expensive as this requires too much efforts to acquire labeled data. GANs help in solving this challenge by learning to generate data by underlying data distribution without requiring only labeled data. This ability make it special in field of research and applications of computer vision, natural language processing and even scientific research.

Types of GANs

Since their inception, numerous variants of GANs have been developed to address specific challenges and enhance their performance. This section will explore some of the prominent types of GANs, including Conditional GANs, Deep Convolutional GANs, Wasserstein GANs, CycleGANs, and Time-series GANs such as TimeGAN.

1. **Conditional GANs (cGANs)**:

   This GANs framework was introduced by Mirza and Osindero (2014) which demonstrated effectiveness in generating MNIST digits conditioned on class labels. This was considered as important improvement on original GANs and generated data without controlling the output data.Conditional GANs extend the original GAN framework by conditioning the generator and discriminator on additional information, such as class labels or data from other modalities. This approach allows for controlled generation of data, making it possible to generate specific classes or styles of data.

2. **Deep Convolutional GANs (DCGANs):**

   Radford et al. (2016) worked on this GAN research for very first time. Authors demonstrated that convolutional layers addition into GAN, can significantly improve GAN performance which can lead to generation of high quality images with improved stability during training, therefore authors added CNNs layer in both of the discriminator and generator. This architecture gave stabled training process and increased the quality of generated images by utilizing convolutional layers, batch normalization and strided convolutions. Radford et al. (2016)'s work on DCGANs is widely accepted for generating the image. The model designed by authors successfully generated high quality images of bedrooms, faces and other objects from the LSUN dataset. This models has architecture based on convolutional layers and batch normalization which significantly improved the stability of GAN training. Currently, its applications can be seen in field of image super resolution with low quality images, repainting the image with missing parts from the image and also in the field of generative artwork.

3. **Wasserstein GANs (WGANs):** Arjovsky et al. (2017) replaced original loss function of GANs with Wasserstein distance which significantly enhanced the GAN training stability and reduced mode collapse. This distance is also called as Earth Mover's distance. This approach provided a more meaningful and useful gradient for the generator which resulted into improved training stability. Authors subsequently introduced WGAN-GP(WGAN with Gradiend Penalty which further improved the performance with improving the Lipschitz continuity more effectively. Gulrajani et al. (2017) further improved WGAN-GP, incorporating gradient penalties to enforce Lipschitz continuity more effectively.

4. **CycleGANs:** Zhu et al. (2017) introduced Cycle GANs in 2017, having advance abilities such as photo enhancement, artistic style transfer and object transfiguration using unpaired datasets. The key innovation in this GANs is cycle consistency loss which guaranteed the translating images from one domain to another and back result to original domain image. It was designed for unpaired image to image translation tasks. CycleGANs also doesn't require paired datasets like original GAN therefore it make them

perfect for tasks where paired data is unavailable or very expensive. Zhu et al. (2017) demonstrated the unsupervised learning ability of the cycle GAN by translating the image between diffirent domains like converting images into paintings or transforming hourses into zebras, without the need of paired examples. Cycle GAN have been used to convert low-resolution MRI scans into high resolution images which helps in more accurate diagnosis.

5. **SeqGANs:** Yu et al. (2017) introduced this algorithm in 2017, corroborating the ability to generate realistic sequences of text by creating sequence of action as a process with reinforcement learning. This ability overcome the issue faced by traditional GANs to generate the text due to discrete data. SeqGANs resolved it by using policy gradient methods from reinforcement learning to train the generator. SeqGANs has been getting used for natural language processing, text generation, dialogue systems and machine translation.

6. **Time series GANs (TimeGANs):** Yoon et al. (2019) proposed TimeGAN which uniquely addressed challenges of time series data generation. TimeGAN combines autoregressive and recurrent networks with adversarial training and generate high quality synthetic time series data. it integrates recurrent neural networks and an embedding network to capture temporal dependencies, which help it in particularly in time series data generation.

7. **MedGANs:** Choi et al. (2017) introduced MedGAN in 2017, showcasing its ability to generate high-dimensional binary data that represents patient records. A specialized type of Generative Adversarial Network (GAN) called MedGAN creates realistic, synthetic electronic health records (EHRs). This allows for privacy-preserving data sharing in medicine by mimicking real patient data. MedGAN has applications in various fields like medical researchers to work with realistic synthetic data without compromising patient privacy. This give more data and smoothing the development of new diagnostic tools and treatments.

TimeGAN

Introduction to TimeGAN

In the world of finance, generating synthetic time series data can be prune to significant potential for risk management, algorithmic trading, and stress testing. Traditional machine learning and generative models many times fail to find the complex temporal dependencies and patterns present in financial data. Time-series Generative Adversarial Network (TimeGAN) tries to resolve these challenges by providing a strong framework to generate rational and temporally stable synthetic financial time series data. It was initially developed by (Yoon et al., 2019), TimeGAN leverage both supervised and unsupervised learning techniques and generate high-quality synthetic data that precisely mimic real financial sequences.

Yoon et al. (2019) designed TimeGAN as a novel framework for generating realistic synthetic time-series data. Authors 'emphasize that TimeGAN is the first model to combine the flexibility of the unsupervised GAN framework with the control afforded by supervised training in autoregressive models'. The authors described and compared the synthetic data based on different parameters the effectiveness of TimeGAN in generating realistic samples using various real and synthetic time-series datasets. They found out that TimeGAN consistently and significantly outperforms state-of-the-art benchmarks with respect to similarity and predictive ability measures. Authors introduced the concept of reducing the discrepancy between two distributions, a key aspect of GANs, using the following mathematical notation:

$$\min \hat{p}_D(p(S, X_{1:T})||\hat{p}(S, X_{1:T})) \tag{2.4}$$

- p represents the true data distribution.

- $\hat{p}$ represents the generated data distribution.
- D is a measure of distance between distributions (ideally the Jensen-Shannon divergence in the GAN framework).
- S represents static features (features that do not change over time).
- $X_{1:T}$ represents the temporal features (features that change over time).

They also decompose the joint probability distribution of a sequence into a product of conditional probabilities to focus on the temporal dynamics:

$$p(S, X_{1:T}) = p(S) \prod_t p(X_t | S, X_{1:t-1}) \tag{2.5}$$

This decomposition allows them to define a complementary objective of learning the conditional distribution

$$(p(X_t | S, X_{1:t-1})) \tag{2.6}$$

that best approximates the true conditional distribution at any time (t). Huang and Deng (2023) included TimeGAN in the comparison of time-series GANs for the task of classification. They found that TimeGAN outperformed CotGAN for most of the datasets but is less effective when applied to process long or multi-dimensional time series data. The authors suggested that focus solely on data generation performance, just like Time GAN, may not result into translating the better discrimination performance.

Shukla and Poornima (2023) used TimeGAN to generate synthetic stock market time series data to augment the limited dataset and mitigated overfitting in their neural network models. Authors focused on predicting the closing price of Tata Steel stock using historical data from the National Stock Exchange of India. The authors identified that the dataset of 746 entries (approximately three years of historical data) was relatively small for training deep learning models which normally requires huge quantity of data to avoid overfitting. Overfitting occurs when a model learns the training data too well, which includes noise and random fluctuations, and performs poorly on unseen data provided to model being trained.To resolve this, they used TimeGAN to generate artificial stock market data that resembles the real data. This synthetic time series data was later combined with the original dataset so that it will increase the size of the training data and will reduce the chances of over fitting. Yoon et al. (2019) has studied some key challenges and innovation based on it in his paper.

Key Challenges and Innovations

Challenge: "Capturing the complex temporal dynamics and distributions of features in time-series data".

Innovation: TimeGAN introduced a embedding space between feature space and lower dimensional latent space which is jointly optimized. mentioned embedding is trained by supervised and adversarial objectives, mostly trained and developed to effectively capture the temporal dynamics.

Challenge: "Balancing the flexibility of unsupervised GANs with the control offered by supervised models".

Innovation: TimeGAN contains a supervised loss using the original data as supervision. This supervision explicitly encourages the model to study the step-wise conditional distributions which ultimately result into improving the accuracy of temporal dynamics.

Challenge: "Generating realistic time-series data that can be used for downstream tasks."

Innovation: TimeGAN is capable to handle both static and temporal aspects of data, which make it suitable and applicable to a wide range of the real-world problems and scenarios. It also explains strong role while generating synthetic data which is ultimately proven both realistic and useful for predictive tasks.

TimeGAN's performance was determined by combining the qualitative and quantitative methods with variety of datasets like stock market, Multivariate sinusoidal sequences and UCI Appliances energy prediction dataset.
Yoon et al. (2019) used below evaluation metrics for TimeGAN.

Qualitative Evaluation

t-SNE and PCA Visualizations: These were some which were used to visually compare the distributions of synthetically generated data and the original data in a lower-dimensional space. Yoon et al. (2019) wanted to capture the diversity and structure of the synthetic data capture to the real world data available.

Quantitative Evaluation

Discriminative Score: Yoon et al. (2019) used a trained model of post-hoc time-series classification model 2-layer LSTM to identify the difference between real and generated time sequences. It turned out that the classification error on a held-out test tells us the measure of indistinguishable the synthetic data was from the real data.

$$\textbf{Discriminative Score:}$$
$$\text{Real Sequences (Labeled 'Real') + Synthetic Sequences (Labeled 'Fake')}$$
$$\xrightarrow{\text{Train Classifier}} \text{Classification Error on Test Set}$$

Predictive Score: A post-hoc sequence prediction model (also a 2-layer LSTM) was trained using the synthetic data and then evaluated based on the original data. The mean absolute error (MAE) in forecasting the next time interval was used to estimate how well the synthetic data protect the predictive characteristics of the real data.

$$\textbf{Predictive Score:} \quad \text{Synthetic Sequences} \xrightarrow{\text{Train Prediction Model}} \text{Mean Absolute Error on Real Sequences}$$

Financial General Adversarial Network(FIN GAN)

Takahashi et al. (2019) proposed FinGAN to synthetically generate the financial time series data which has more key statistical properties or stylized facts of real financial temporal data. FinGAN has a generator and a discriminator. The generator has work of transforming a 100-dimensional Gaussian noise into a long sequence of time-series data by using a combination of both multi-layer perceptron (MLP) and convolutional neural network (CNN) architectures. In the similar manner,discriminator also use a similar MLP-CNN combination method so that discriminator can evaluate the generated data from the real financial times series data. This training algorithm involves Adam optimizer with defined learning rate and use particular technologies like batch normalization, noisy labeling to stabilize the training process and ultimatly improve the model's robustness.
Takahashi et al. (2019) has mentioned below "stylized facts" in his paper.

- **Linear unpredictability:** Minimal linear auto correlations in asset returns except the short intra-day time interval.
- **Heavy tails :** The distribution of returns shows broader tails when compared to normal distribution, which often indicates the extreme values for that time period

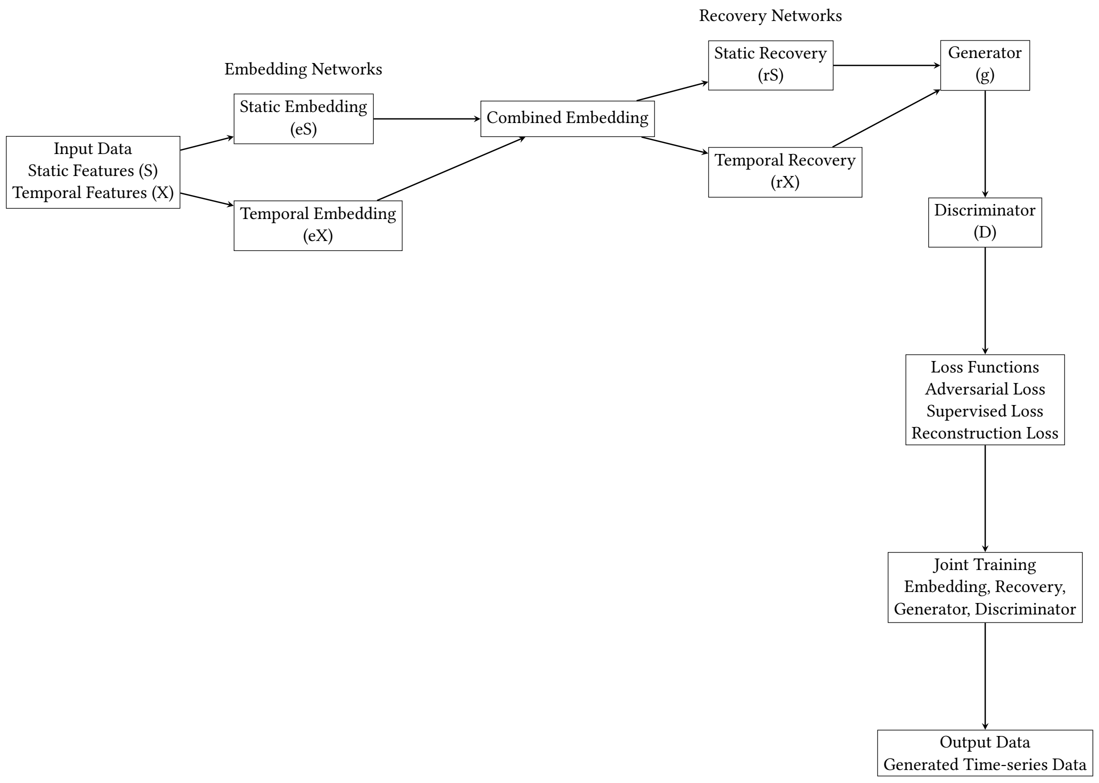

Recovery Networks
Embedding Networks
Static Recovery (rS)
Generator (g)
Static Embedding (eS)
Combined Embedding
Input Data Static Features (S) Temporal Features (X)
Temporal Recovery (rX)
Temporal Embedding (eX)
Discriminator (D)
Loss Functions Adversarial Loss Supervised Loss Reconstruction Loss
Joint Training Embedding, Recovery, Generator, Discriminator
Output Data Generated Time-series Data

Figure 2.2: Architecture of TimeGAN

- **Volatility clustering:** Large price changes mainly happens because of the large changes and vice versa that is why high volatility periods are clustered together frequently.
- **Leverage effects:** Negative correlations between past returns and future volatility.
- **Coarse-fine volatility correlation:** Lack of equality in the correlation between volatility at different time scales however distribution of return sometimes follows normal distribution over bigger time scale.

Takahashi et al. (2019) designed the FinGAN which is consists of two primary components 1. The generator (G) 2.The discriminator (D). Below is a detailed description of each component and their architectures:

Generator:

- **Input:** Dimentional Gaussian noise vector z.
- **Output:** A time series with N steps.
- **Architectures Tested:**
  - **Multi-Layer Perceptron (MLP):**
    - Four layers of fully connected neural networks.
    - Hyperbolic tangent activation function.
  - **Convolutional Neural Networks (CNNs):**
    - Six convolutional layers.
    - Leaky ReLU activation function.
  - **Combination (MLP-CNN):**
    - Element-wise multiplication of outputs from MLP and CNN architectures.

- **Batch Normalization:** Applied to the layers in MLP and CNNs.
- **Training Algorithm:** Adam optimizer with learning rate $\alpha_G = 2 \times 10^{-4}$ and $\beta_{1,G} = 0.5$.

Discriminator (D)

- **Architecture:** MLP-CNNs combination.
- **Training Algorithm:** Adam optimizer with learning rate $\alpha_D = 1 \times 10^{-5}$ and $\beta_{1,D} = 0.1$.
- **Algorithm:**
  - The discriminator is trained to distinguish between real and generated time-series data.
  - The generator is trained to deceive the discriminator.

However Eckerli and Osterrieder (2021) while studying all the GANs pointed out that there is need of further research on FinGAN for standardized evaluation metrics for FinGAN generated financial data. Wiese et al. (2020) concluded that this algorithm is very good for generating realistic data but very complex to implements at the same time. Jiali (2021) proposed new financial time series prediction model which is based on dual attention mechanism Generative adversarial network(DA-GAN). According to Jiali (2021) FinGAN does not capture complex features of financial time series data. There are some limitations of FinGAN as pointed out by Takahashi et al. (2019) which are as follows

- **Model Complexity:** FinGAN's architecture that is combining MLP and CNN, makes it very complex for computations and requires more resources than most of the other algorithms available.
- **Scalability:** Higher dimensionality of financial times series data makes it very hard to scale the larger datasets or higher frequency data. This mostly affects the applications acceptability in real world applications for most of the financial datasets.
- **Training Stability:** Training GANs is undoubted unstable, and FinGAN is no exception. FinGAN's adversarial nature of the training process may cause issues such as mode collapse and resulting into limited variety of outputs because of the failure of model.
- **Evaluation Metrics:** Calculating the performance of GAN's is also not a straight forward task which make GANs like FinGAN nightmare. Standard metrics mostly fails to calculate the realism and quality of the generated synthetic data.

2.3.3 Recurrent Neural Networks(RNN

Introduction of Recurrent Neural Networks(RNN)

Recurrent Neural Networks are types or class of artificial neural network designed to process sequential data. Unlike feedforward neural network, RNNs loops back allowing information to persist which make it special in case where order and context of input data is crucial for example series prediction, natual language processing and speech recognition.(Gers et al., 2000)(Wiese et al., 2020)

Definition:

'Recurrent Neural Networks are a type of neural network where connections between nodes form a directed graph along a temporal sequence. This structure allows them to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs'.

General Architecture of RNN:

A RNN's Architecture is characterized by iterating through time steps, holding a hidden state that retains details from past steps. In the core, a RNN relies on neurons or nodes with input and hidden layers. mostly different from standard neural networks, a RNN's hidden layer output is cycled back into the input layer for the subsequent time step.
A RNN can be mathematically described using the following equations:

1.Hidden State Update:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2.7}$$

where:

- h_t is the hidden state at time step t.
- x_t is the input at time step t.
- W_{xh} is the weight matrix from input to hidden state.
- W_{hh} is the weight matrix from the previous hidden state to the current hidden state.
- b_h is the bias term for the hidden state.
- $\tanh$ is the hyperbolic tangent activation function.

2.Output:

$$y_t = W_{hy}h_t + b_y \tag{2.8}$$

where:

- y_t is the output at time step t.
- W_{hy} is the weight matrix from hidden state to output.
- b_y is the bias term for the output.

In this architecture, each hidden state h_t captures information from both the current input x_t and the previous hidden state h_{t-1}, thereby maintaining a form of memory over time. This recurrent connection enables the RNN to learn temporal dependencies.

Types of RNN:

1.Vanilla RNNs: Vanilla is basic form of RNN just like described in above equations and architecture.
2.Long Short-Term Memory Networks(LSTMs): These were designed to handle long term dependencies like temporal one.
3.Gated Recurrent Units(GRUs): It is designed just like LSTM but more simplified version of LSTMs but having fewer parameters.
4.Bidirectional RNNs: These are special type of RNN which can process data in both directions and captures more insights as a result.
5.Deep RNNs: This is designed with concept of deep learning which has multiple hidden layers to capture more complex patterns hidden inside the data.

Challenges of Traditional RNNs

Even though RNNs looks promising to process sequential data, they have some limitions which limit their effectiveness, specially while dealing with long term dependencies. The most challenging issue faced is vanishing gradient problem, the exploding gradient problem and problems in training the data.

1.Vanishing Gradient Problem:
The vanishing gradient problem comes up during the training of RNNs while using backpropagating time (BPTT). For deep networks or when dealing with long sequences, the gradients of the loss function with respect to the weights can diminish mostly. This occurs because the gradient calculation involves the product of many small values (derivatives of the activation functions), resulting in exponential decay. Accordingly, the weights are updated very slowly, hampering the network's ability to learn long-term dependencies effectively (Bengio et al., 1994).

Mathematically, the hidden state h_t at time step t can be defined recursively as:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{2.9}$$

while back propagation through time is applied to compute the gradient of the loss L with respect to the weights W_{hh}, the chain rule must be applied through all time steps:

$$\frac{\partial L}{\partial W_{hh}} = \sum_t \frac{\partial L}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}} \tag{2.10}$$

Due to the recursive nature of h_t, this involves the product of many derivatives of $\tanh$, which are less than 1. Thus, the gradients can decay exponentially, making the updates negligible.

2. Exploding Gradient Problem: The exploding gradient problem is the opposite to the vanishing gradient issue. It comes up when gradients increase exponentially during back propagation, which can result in more large updates to the network's weights. This instability can cause the model to fail during the learning process. Although gradient clipping can alleviate this problem, it does not fully resolve the root cause (Hochreiter and Schmidhuber, 1997).

3. Training Difficulties: Training RNNs takes a lot of computational power and needs a lot of data to work well. Since RNNs process data step-by-step, it's hard to run computations in parallel, which makes training slow. Additionally, RNNs can easily overfit and need careful adjustment of settings like learning rates, weight initialization, and regularization techniques (Graves et al., 2013).

4. Limited Long-Term Memory: Traditional RNNs have trouble keeping information over long sequences. As the input sequence gets longer, the effect of earlier inputs fades, making it hard for the RNN to learn long-range dependencies. This is a big drawback for tasks like language modeling and time series prediction, where long-term context is important (Graves et al., 2013).

Development of LSTM:

To solve the problems of regular RNNs, which are mentioned above Long Short-Term Memory (LSTM) networks were created by (Hochreiter and Schmidhuber, 1997). LSTMs are made to remember long-term information using a special memory unit called the LSTM cell (Zaremba et al., 2014).

2.3.4 Long Short-Term Memory (LSTM)

Introduction:

Long Short-Term Memory(LSTM) networks are special type of recurrent neural networl(RNN) capable of learning long term dependencies. Hochreiter and Schmidhuber (1997) addressed the limitations of traditional RNNs, especially the vanishing gradient problem. Author's solution proved highly effective for various tasks mainly which involves sequential data. natural language proessing and speech recognition. Hochreiter and Schmidhuber (1997) incorporates memory cells to maintain information over long time sequences. LSTMs has more complex architecture and that allows it to maintain information over long sequences particularly with time series data having long-term dependencies such as time series data creation.

Definition:

'Long Short-Term Memory (LSTM) networks are a type of RNN that can learn and remember over long sequences by incorporating special units called memory cells. These cells can maintain their state over time and control the flow of information through a set of gates, making it possible to capture long-term dependencies without suffering from the issues that plague traditional RNNs(Hochreiter and Schmidhuber, 1997).'

General Architecture:

The architecture of an LSTM network is more complex than that of a traditional RNN. An LSTM unit, or cell, comprises several components that work together to manage the cell's state and output. The primary components of an LSTM cell include the cell state, hidden state, input gate, forget gate, output gate, and candidate cell state.

Types of LSTM Models:

1. **Standard LSTM:** The standard LSTM architecture includes memory cells with three main components: the input gate, forget gate, and output gate. These gates regulate the flow of information, allowing the network to retain and update information over long sequences.

   LSTM Equations

   1. Forget Gate

   The forget gate determines which information from the previous cell state should be discarded.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.11}$$

   - f_t: Forget gate activation vector at time step t.
   - σ: Sigmoid function.
   - W_f: Weight matrix for the forget gate.
   - h_{t-1}: Hidden state from the previous time step.
   - x_t: Input at the current time step.
   - b_f: Bias vector for the forget gate.

2. Input Gate

The input gate controls the information that is added to the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.12}$$

$$\tilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.13}$$

- i_t: Input gate activation vector at time step t.
- $\tilde{C_t}$: Candidate cell state vector at time step t.
- $\tanh$: Hyperbolic tangent function.
- W_i: Weight matrix for the input gate.
- W_C: Weight matrix for the candidate cell state.
- b_i: Bias vector for the input gate.
- b_C: Bias vector for the candidate cell state.

3. Cell State Update

The cell state is updated by combining the previous cell state, modulated by the forget gate, with the candidate cell state, modulated by the input gate.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C_t} \tag{2.14}$$

- C_t: Cell state vector at time step t.
- C_{t-1}: Cell state vector from the previous time step.
- $\odot$: Element-wise multiplication.

4. Output Gate

The output gate determines the output at the current time step based on the cell state and the input.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.15}$$
$$h_t = o_t \odot \tanh(C_t) \tag{2.16}$$

- o_t: Output gate activation vector at time step t.
- h_t: Hidden state vector at time step t.
- W_o: Weight matrix for the output gate.
- b_o: Bias vector for the output gate.

2. **Bidirectional LSTM (BiLSTM):** This model process data in both forward and backward directions which helps it in gathering more context from the input sequence. In scenarios where understanding whole context is important, Bidirectional LSTM, plays important role such as natural language processing. Schuster and Paliwal (1997) introduced this model for first time, upgrading the abilities of RNNs to get more information from past and future contexts.

   An LSTM cell consists of several gates that control the flow of information:

- **Forget Gate (f_t)**: Decides what information should be discarded from the cell state.
- **Input Gate (i_t)**: Decides which values from the input to update the cell state.
- **Cell State Update ($\tilde{C_t}$)**: Creates a vector of new candidate values that could be added to the cell state.
- **Output Gate (o_t)**: Decides what part of the cell state should be output.

For a single time step t, the operations within an LSTM cell are:

1. Forget gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.17}$$

2. Input gate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.18}$$

3. Candidate cell state:

$$\tilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.19}$$

4. Updated cell state:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t} \tag{2.20}$$

5. Output gate:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.21}$$

6. Updated hidden state:

$$h_t = o_t * \tanh(C_t) \tag{2.22}$$

Where: - σ is the sigmoid function. - $\cdot$ denotes concatenation of the hidden state h_{t-1} and the input x_t. - W_f, W_i, W_C, W_o are weight matrices. - b_f, b_i, b_C, b_o are bias vectors.
Bidirectional LSTM

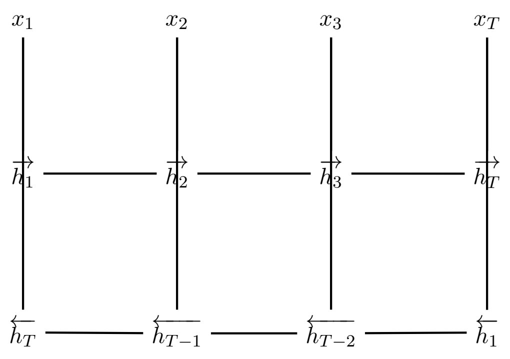
A BiLSTM consists of two LSTMs:
1. **Forward LSTM**: Processes the input sequence from the beginning to the end.
2. **Backward LSTM**: Processes the input sequence from the end to the beginning.
At each time step t, the hidden states from both LSTMs are concatenated to form the final output:

$$h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}] \tag{2.23}$$

Visualization

3. **Peephole LSTM:** Peephole LSTMs are extended version of the traditional LSTM model.Gers et al. (2000) introduced this algorithm where poopholes added connections from the cell state to the gates, this allows the gates access to the cells directly, improving the ability of model to learn precise timings.

   Gates and States with Peephole Connections: In this algorithm, the gates are reconstructed to incorporate the cell state in their calculations. below are the modified equations and an explanation of each gate:



$h_1 = [\overrightarrow{h_1}, \overleftarrow{h_T}]$ $h_2 = [\overrightarrow{h_2}, \overleftarrow{h_{T-1}}]$ $h_3 = [\overrightarrow{h_3}, \overleftarrow{h_{T-2}}]$ $h_T = [\overrightarrow{h_T}, \overleftarrow{h_1}]$

Figure 2.3: Bi-directional LSTM Network.

- **Forget Gate (f_t)**: Decides what information should be discarded from the cell state.
- **Input Gate (i_t)**: Decides which values from the input to update the cell state.
- **Candidate Cell State ($\tilde{C_t}$)**: Creates a vector of new candidate values that could be added to the cell state.
- **Output Gate (o_t)**: Decides what part of the cell state should be output.
- **Cell State (C_t)**: Represents the memory of the cell. Hidden State (h_t): Represents the output of the cell.

For a single time step t, the operations within a Peephole LSTM cell are:

1. Forget gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + U_f \cdot C_{t-1} + b_f) \tag{2.24}$$

2. Input gate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + U_i \cdot C_{t-1} + b_i) \tag{2.25}$$

3. Candidate cell state:

$$\tilde{C_t} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.26}$$

4. Updated cell state:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t} \tag{2.27}$$

5. Output gate:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + U_o \cdot C_t + b_o) \tag{2.28}$$

6. Updated hidden state:

$$h_t = o_t * \tanh(C_t) \tag{2.29}$$

Where: - σ is the sigmoid function.
- $\cdot$ denotes concatenation of the hidden state h_{t-1} and the input x_t.
- W_f, W_i, W_C, W_o are weight matrices.
- U_f, U_i, U_o are peephole weight matrices.
- b_f, b_i, b_C, b_o are bias vectors.

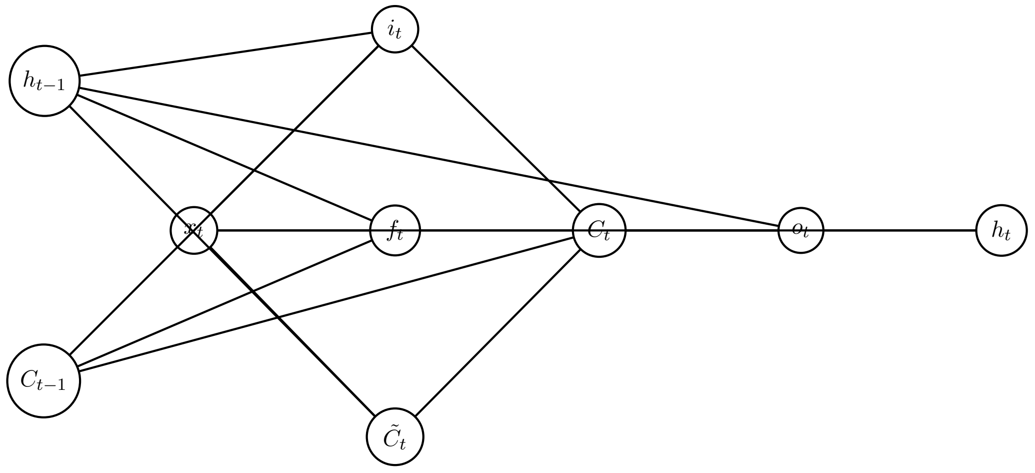


Figure 2.4: Long Short-Term Memory (LSTM) Network Gate Flow.

Nodes: Represent different components and states in the Peephole LSTM cell, such as input ($\mathbf{x}_t$), previous hidden state ($\mathbf{h}_{t-1}$), previous cell state ($\mathbf{C}_{t-1}$), forget gate ($\mathbf{f}_t$), input gate ($\mathbf{i}_t$), candidate cell state ($\tilde{\mathbf{C}}_t$), cell state ($\mathbf{C}_t$), output gate ($\mathbf{o}_t$), and hidden state ($\mathbf{h}_t$).
Arrows: Indicate the flow of information between the components and gates.

2.4 Statistical Learning Approaches for Generating Synthetic Time Series Data

2.4.1 Overview of Statistical Learning Techniques

The area of synthetic time data generation has received too much attention because of its ability to generate large, high-quality datasets while securing privacy, lowering the costs of data collection, labelling etc. This is especially important in the area of financial applications, when access to genuine data may be limited due to privacy concerns, legal constraints, or proprietary interests. In this talk, we will look at various statistical learning methodologies used to generate synthetic time series data, with a focus on their applicability in financial contexts.

Introduction

The term "synthetic data" refers to data that is synthetically or artificially generated which has statistical features similar to real-world datasets. In financial applications, datasets are comprised of time series data for stock prices, interest rates, currency exchange rates, and other financial indicators. Generating accurate synthetic time series data is critical for applications like algorithmic trading, risk management, and financial forecasting.

2.4.2 Block Bootstrapping for time series

Introduction to Block BootStrapping

Block bootstrapping use resampling technique to generate multiple chunks of time series from given original time series data.It mainly preserve temporal dependencies inherent in time series data which was earlier not getting addressed by traditional bootstrapping techniques. The main idea behind the algorithm is dividing the original time series into different blocks of consecutive observations and resample this blocks with replacement so that new time series will be created. According to Bühlmann (2002) this approach maintains short-range dependence structure within each block in which time series was divided and this make it suitable for statistical analyses and modeling tasks. He also stated that it is non-parametric method which resample the observations from original time series and generate bootstrap samples which make it very useful if the data patterns in underlying time series is very complex as data is getting divided into blocks and then processed in chunks. Acccording to him, this feature of block bootstrap justify theoretically short-range dependant processes as like summable auto covariances or mixing coefficients.

Härdle et al. (2003) found out that synthetic data generation using bootstrapping helps in mimicing the statistical properties of original data. This property is usually better when limited data is available for the generating the realistic data for simulation studies. For instance we can take the example discussed by Usaola (2014) for synthesizing the data for wind power series. These synthetic series are later used in reliability studies of power systems which studies the wind energy. This method is known for simplicity and effectiveness even with sufficient data available.

Radovanov and Marcikić (2014) described four different block bootstrap methods :

1. **Non-overlapping block bootstrap**: The original time series is divided into non-overlapping blocks, and then created the bootstrap samples by random selection of blocks by replacement. This method was proposed by (Carlstein, 1986).

   Algorithm:

   - Divide the series into non-overlapping blocks of length l.
   - Randomly sample B blocks.
   - Concatenate sampled blocks to form the bootstrap sample.

2. **Overlapping block bootstrap (moving block bootstrap)**: In this technique bootstrap samples are selected just like non-overlapping block bootstrap but overlapping of blocks are allowed. This method was developed by Künsch (1989), he overlapped blocks of fixed length and preserved the dependency structure within the block.
   Algorithm:

   - Create overlapping blocks of length l.
   - Randomly select B blocks with replacement.
   - Concatenate these blocks to form the bootstrap sample.

3. **Stationary block bootstrap**: The length of block is determined according to the geometric distribution and this way stationarity is preserved for the original times series. This method was suggest by Politis and Romano (1994), SBB uses blocks of random lengths.
   Algorithm:

   - Generate length of block as per geometric distribution.
   - Gather sampled blocks with these lengths and concatenate to form the bootstrap sample.

4. **Circular Block Bootstrap**: This method considers subset of original of time series for distribution of the blocks of bootstrap samples. This method links end of the series to the beginning, keeping cyclic properties of blocks.
   Algorithm:

- It is similar to Moving Block Bootstrap(MBB) but ensures that blocks can wrap around the series end.

The Radovanov and Marcikić (2014) compared the performance of these block bootstrap methods in estimating the variance of returns for the Serbian stock market index BELEX15. They found out that method moving block bootstrap and stationary block bootstrap has better performance than other two methods when they compared mean squared errors.
The block bootstrap method was described and developed by several authors including (Hall, 1986), (Carlstein, 1986), and (Künsch, 1989). The block bootstrap method focus on preserving the dependency structure inherent in time series data with help of resampling the blocks of sample of observations rather than focusing on individual observations.
Below is the detailed algorithm description of Block bootstrap.

1. **Block Division**: This step involve dividing time series data into blocks. It can use any one of the above method to divide the block like overlapping block or overlapping blocks of fixed length l.
2. **Block Resampling**: This steps select blocks randomly with replacement to construct new synthetic time series.
3. **Reconstruction**: This step concatenate the blocks which where randomly resampled and result into forming a bootstrap sample.
4. **Statistics Computation**: This step calculates the desired statistics of the previously generated bootstrap sample.
5. **Repetition**: In this step algorithm Repeat the above steps B times to create B bootstrap samples and their corresponding statistics.

Detailed Steps

Step 1: Block Division

Given a time series $X = \{X_1, X_2, \dots, X_n\}$, the series is divided into blocks of length l.

- **Non-Overlapping Blocks**:

$$\begin{aligned} B_1 &= \{X_1, X_2, \dots, X_l\}, \\ B_2 &= \{X_{l+1}, X_{l+2}, \dots, X_{2l}\}, \\ &\vdots \\ B_k &= \{X_{(k-1)l+1}, X_{(k-1)l+2}, \dots, X_{kl}\}, \end{aligned}$$

  where $k = \lfloor \frac{n}{l} \rfloor$.

- **Overlapping (Moving) Blocks**:

$$\begin{aligned} B_1 &= \{X_1, X_2, \dots, X_l\}, \\ B_2 &= \{X_2, X_3, \dots, X_{l+1}\}, \\ &\vdots \\ B_{n-l+1} &= \{X_{n-l+1}, X_{n-l+2}, \dots, X_n\}. \end{aligned}$$

Step 2: Block Resampling

Randomly select k blocks with replacement from the set of blocks $\{B_1, B_2, \dots, B_m\}$, where m is the total number of blocks.

Step 3: Reconstruction

k selected blocks concatenated to form a bootstrap sample. The length of the bootstrap sample will be approximately n.

Step 4: Statistics Computation

Compute the desired statistic (e.g., mean, variance, autocorrelation) from the bootstrap sample. Let $\hat{\theta}_b$ denote the statistic computed from the b-th bootstrap sample, where $b = 1, 2, \ldots, B$.

Step 5: Repetition

Repeat steps 1 to 4 a total of B times to generate B bootstrap samples and corresponding statistics $\{\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_B\}$.

Equations for block bootstrap

Block Length

Let l be the block length, and n the length of the time series. The number of blocks k for non-overlapping blocks is given by:

$$k = \left\lfloor \frac{n}{l} \right\rfloor \tag{2.30}$$

Bootstrap Sample Construction

Suppose B_i is a block, then a bootstrap sample X_b^* can be expressed as:

$$X_b^* = \{B_{i_1}, B_{i_2}, \ldots, B_{i_k}\} \tag{2.31}$$

where $i_1, i_2, \ldots, i_k$ are indices of randomly selected blocks.

Statistic Calculation

Let θ be the parameter of interest (e.g., mean, variance), then the bootstrap estimate for the b-th sample is:

$$\hat{\theta}_b = f(X_b^*) \tag{2.32}$$

where $f(\cdot)$ is the function computing the statistic.

Bootstrap Distribution

After B replications, the bootstrap distribution of the statistic $\hat{\theta}$ is approximated by:

$$\{\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_B\} \tag{2.33}$$

Example: Estimating Mean and Variance

- **Original Series**: $X = \{X_1, X_2, \ldots, X_n\}$
- **Block Length**: $l = 5$
- **Number of Blocks**: $k = \lfloor \frac{n}{l} \rfloor$
- **Bootstrap Replications**: $B = 1000$

For each bootstrap replication b:

1. Divide X into blocks of length l.
2. Randomly select k blocks with replacement.
3. Concatenate selected blocks to form X_b^*.
4. Calculate the mean $\hat{\mu}_b$ and variance $\hat{\sigma}_b^2$ for X_b^*.

After B replications, use the empirical distribution of $\{\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_B\}$ and $\{\hat{\sigma}_1^2, \hat{\sigma}_2^2, \ldots, \hat{\sigma}_B^2\}$ for statistical inference. 2.5 flowchart gives visual representation of block bootstrap.

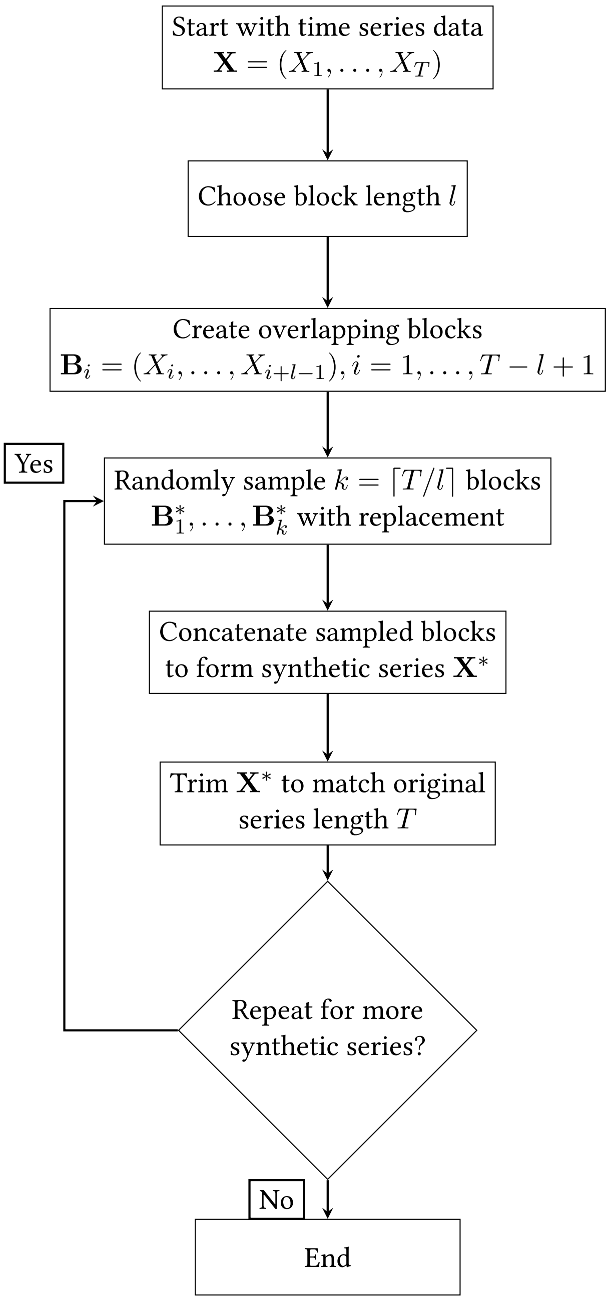

Start with time series data $\mathbf{X} = (X_1, \ldots, X_T)$
Choose block length l
Create overlapping blocks $\mathbf{B}_i = (X_i, \ldots, X_{i+l-1}), i = 1, \ldots, T - l + 1$
Yes
Randomly sample $k = \lceil T/l \rceil$ blocks $\mathbf{B}_1^*, \ldots, \mathbf{B}_k^*$ with replacement
Concatenate sampled blocks to form synthetic series $\mathbf{X}^*$
Trim $\mathbf{X}^*$ to match original series length T
Repeat for more synthetic series?
No
End

Figure 2.5: Block Bootstrap Flowchart

2.5 Comparative Analysis of Deep Learning and Statistical Learning Approaches

2.5.1 Strengths and Weaknesses of Deep Learning Approaches

Deep learning has major impact on various fields which let them learn from the large set of data or generate data. The pros and disadvantages of deep learning systems are examined in this review of the literature, with particular attention on to the issues of scalability and flexibility, interpretability and explainability, data efficiency, and generalisation performance. Synthetic time series data generation using deep learning methods such as Generative Adversarial Networks (GANs) and Long Short-Term Memory (LSTM) networks has been favourite area of research for scientists.

2.5.2 Strengths and Weaknesses of Statistical Learning Approaches

a. Robustness and Stability of Statistical Models

Strength:

1. High Scalability:

- Goodfellow et al. (2014) in his papers stated that GANs and LSTMs are highly scalable and has ability to generate large volume of synthetic data as the computational resources and data size increase.
- LeCun et al. (2015) stated that deep learning models, particularly neural networks, are highly scalable. As the amount of data and computational resources increases, these models continue to improve their performance
- Chen et al. (2016) studied the synthetic data generated by GANs and LSTM and according to him, they can handle complex data distributions and generate high quality synthetic data.

2. Flexibility:

- GANs and LSTMs can be used for range of data generation applications. For instance, GANs have been getting utilized to produce realistic images, text, and audio (Isola et al., 2017).
- LSTMs are particularly used to generate sequential data, such as time series and natural language text.

Weaknesses:

1. Resource Intensive:

- Training GANs and LSTMs necessitates considerable computational power, including high-performance GPUs and extensive memory capacities (Arjovsky et al., 2017).

2. Complexity in Hyperparameter Tuning:

- Both GANs and LSTMs are highly sensitive to hyperparameters, and identifying the optimal configuration involves extensive experimentation and expertise. This process can be both time-consuming and computationally expensive (Politis and Romano, 1994).

b. Interpretability and Explainability Challenges

Strengths:

1. High Accuracy:

- GANs and LSTMs are capable of producing synthetic data that closely mimics real data, demonstrating high accuracy and effectiveness across various applications where data realism is paramount.

Weaknesses:

1. Lack of Interpretability:

- Understanding the inner workings of GANs can be particularly challenging because of their "black-box" nature, where the interactions between the generator and discriminator networks are not quite understandable.
- Interpreting LSTMs is also problematic, especially concerning how they capture and utilize long-term dependencies.

2. Explainability Issues:

- Explaining the synthetic data generated by GANs and LSTMs is challenging due to the inherent complexity of these models.
- The lack of explainability can break the trust in synthetic data, especially in applications where transparency is crucial and trust matters.

c. Data Efficiency and Generalization Performance

Strengths:

1. High Performance with Large Data:

- GANs and LSTMs perform good when trained on extensive datasets, producing high-quality synthetic data that faithfully captures complex patterns and structured present in real data (Goodfellow et al., 2014)

Weaknesses:

1. Data Hunger:

- GANs and LSTMs generally depends on substantial quantities of labeled data for effective training, which can act as a challenge in situations where real data is limited. (Radford et al., 2016)

2. Generalization Performance:

- Despite their impressive data generation abilities, GANs and LSTMs may encounter challenges in generalizing to new, unseen data (Zhu et al., 2017).
- Overfitting is a frequent concern where models excel on training data but struggle to generalize to validation or test data (Arjovsky et al., 2017)

3. Adversarial Vulnerabilities:

- Both GANs and LSTMs are susceptible to adversarial attacks, where minor deliberate alterations can result in substantial errors in data generation, pointed out by (Goodfellow et al., 2014)
- GANs are very much prune to mode collapse, that is where the generator produces limited varieties of data, thereby reducing the diversity of the synthetic data.

3 Methodology

3.1 Introduction

The generation of synthetic time series data has gained a lot of attention over the past decade because of its potential for application in areas like finance, healthcare, and engineering. Synthetic data can be utilised in a range of uses including preserving data privacy, extending restricted datasets, improving model training, and assisting robustness testing. This chapter describes the methodology used to generate synthetic time series data in this thesis, that involves Long Short-Term Memory (LSTM) networks, block bootstrap approaches, and Generative Adversarial Networks (GANs). To generate high-fidelity synthetic datasets, the chosen methodologies integrate the strengths of traditional statistical approaches and contemporary deep learning approaches.

Time series data, with sequential and temporal dependencies, poses particular challenges in synthetic data the synthesis process. Standard data generating methods sometimes fail to capture the complex temporal connections and patterns observed in time series data. To address these challenges, I utilised three distinct methodologies: LSTM-based models, which are adept at discovering long-term dependencies; block bootstrap methods, which maintain the auto-correlation structure by resampling data blocks; and GANs, which can generate realistic data by learning the underlying distribution through adversarial training.

3.2 Research Design

This study used a multi-way investigation approach to generate synthetic time series data, with each method offering distinct advantages and resolving specific limitations of the other techniques.This thesis's research design is experimental and comparative, with focus on studying 3 different types of techniques and comparing the results. The study is organised as follows:

1. **Selection of Original Dataset**: The study begins with gathering data from kaggle website as a real world data using which we can generate synthetic data. This data primarily includes stock data of gold. This data is from Jan-2014 till Jan-2024.

2. **Data Preprocessing**: Original data from Kaggle is in raw format and need to be processed as per requirement for chosen algorithm. This make sure that data is suitable for model training. This process generally includes normalization, missing values handling and splitting data into training and testing sets.

3. **Synthetic Data Generation**: There are 3 different methods to generate data as follows.
   - **LSTM Networks:** The LSTM model is trained on time series data to recognise temporal connections and generate synthetic sequences.
   - **TimeGAN**: A TIMEGAN model is used for generating synthetic data by combining the strengths of GANs and RNNs.
   - **Block Bootstrapping**: Block bootstrapping is a method for generating synthetic sequences by resampling blocks of original data.

4. **Evaluation and Comparison**: The synthetic data generated is evaluated for its quality and fidelity using various factors. Evaluation metrics might be different for each

algorithms for better evaluation. Mostly discriminative, predictive and visualization metrics are used for evaluation.

5. **Ethical Considerations**:Ethical considerations for data privacy and confidentiality are addressed throughout the generation of data, ensuring that the synthetic data generating method meets ethical norms.

3.3 Data Source

The original time series data used to feed to algorithm to generate synthetic data is from Kaggle. Data is consisted of stock market over period of Jan-2014 to Jan-2024. Data has total 6 columns name close, Volume, Open, High and Low. This dataset was chosen because of the availability and and suitability for time series analysis.

Dataset Details:

- Name: Gold stock prices.
- Source: Kaggle
- Link: Kaggle link
- Description: This data set contains a thorough record of daily gold prices from January 19, 2014, to January 22, 2024. Which includes significant financial parameters for every trading day.
- Variables:
  1. Date: A unique date for each trading day recorded.
  2. Close: The closing price of gold on the relevant date.
  3. Volume: Gold trading volume on the relevant date.
  4. Open: The opening price of gold on the relevant date.
  5. High: The highest recorded price of gold during the trading day.
  6. Low: The lowest price recorded for gold in the trading day.
- Number of Observations: 2512

This includes normalising the data to ensure that all variables are on a comparable scale, dealing with missing values using imputation or interpolation, and executing any necessary modifications to prepare the data for modelling.

3.4 Data Preprocessing

Before moving forward to generate synthetic data, original dataset need to preprocessed to make sure it is suitable for analysis and model training.

- **Normalization**: The data is normalised to ensure that all features had the same scale, which is commonly achieved using min-max scaling or standard.
- **Missing Value Treatment**: The dataset which is being used don't have any missing values as it was perfectly captured for stock data.
- **Train-Test Split**: The dataset is split into training and testing sets, with 70% of the data used for training and 30% reserved for testing. This makes sure that model will work as aspect on unseen data. Depending on datasets and algorithms, datasets is divided into test and train datasets. Detailed version of dividing is shown in appendices for each algorithms.

3.5 Synthetic Data Generation Methods

Synthetic data generation techniques used in this thesis are Generative Adversarial Network(GANs), Long Short term Memory(LSTM) and Block Bootstrapping method. Each algorithm has its broad spectrum of techniques to generate more realistic synthetic time series data, each with unique strengths and characteristics.

3.5.1 Long Short-Term Memory (LSTM)

LSTM networks are a type of Recurrent Neural Network (RNN) created to train the model on long-term temporal dependencies in sequential time series data. That is why they are more famous in application of time series data(Hochreiter and Schmidhuber, 1997).

Reasons for Choosing LSTM:

1. **Handling Long-Term Dependencies**: LSTM networks have been particularly designed to capture long-term dependencies in sequential data (Gers et al., 2000).
2. **Proven Effectiveness**: LSTM networks have been widely used in various time series forecasting tasks and have showed its capacity of conveying challenging temporal patterns (Lipton et al., 2015).
3. **Flexibility**: LSTM networks may be readily converted to several kinds of time series data and fine-tuned to optimise performance for individual datasets (Siami-Namini et al., 2018).

Training Process

1. **Loading the Dataset**:
   - Load the dataset from a CSV file using `pandas.read_csv`.
2. **Selecting the Close Price Column**:
   - Extract the close price column from the dataframe.
3. **Reshaping the Data**:
   - Reshape the extracted close price data into a 2D array.
4. **Normalizing the Data**:
   - Normalize the data to the range [0, 1] using `MinMaxScaler` from `sklearn.preprocessing`.
5. **Splitting the Data into Training and Test Sets**:
   - Split the normalized data into training and test sets, each containing 50% of the total data.
6. **Creating Dataset Matrices**:
   - Use the `create_dataset` function to create input (X) and output (Y) datasets from the training and test sets with a specified look-back period (240 time steps).
7. **Reshaping Input for LSTM**:
   - Reshape the input data into a 3D array suitable for the LSTM model, with the shape $[samples, time\ steps, features]$.

LSTM Model Architecture

The Long Short-Term Memory (LSTM) model architecture consists of the following details:

1. **Input Layer**:
   - The input shape is $[samples, time\ steps, features]$, where the time steps are set to 1 and the look-back period is set to 240.
2. **LSTM Layer**:
   - **Units**: 25 neurons.
   - **Activation Function**: The LSTM layer uses the default activation functions, which are the `tanh` activation function for the cell state and the `sigmoid` activation function for the gates (input, forget, and output gates).
3. **Dropout Layer**:
   - **Dropout Rate**: 0.1 (10% of the neurons are randomly set to zero during training to prevent overfitting).
4. **Dense Output Layer**:
   - **Units**: 1 neuron.
   - **Activation Function**: The Dense layer uses the default linear activation function to output a single value for regression function.
5. **Model Compilation**:
   - **Loss Function**: Mean Squared Error (MSE).
   - **Optimizer**: `adam`.

LSTM Gates and Their Usage

In the provided LSTM model, the gates play a crucial role in managing the flow of information through the LSTM cell. Here's how each gate functions in the context of the code:

1. **Forget Gate**:
   - The forget gate decides what information should be discarded from the cell state.
   - It takes the previous hidden state (h_{t-1}) and the current input (x_t), applies a sigmoid function, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
   - Formula: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
2. **Input Gate**:
   - The input gate decides what new information should be stored in the cell state.
   - It has two parts: the sigmoid layer which decides which values to update, and the tanh layer which creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
   - Formula: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
   - Formula: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
3. **Cell State Update**:
   - The cell state is updated by combining the old state scaled by the forget gate and the new candidate values scaled by the input gate.

- Formula: $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$

4. **Output Gate**:
    - The output gate decides what the next hidden state should be.
    - The hidden state is a filtered version of the cell state.
    - Formula: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
    - Formula: $h_t = o_t \cdot \tanh(C_t)$

In the provided code in appendices **??** (LSTM implementation), these gates are implicitly managed by the LSTM layer in the TensorFlow/Keras library.

3.5.2 Time-Series Generative Adversarial Networks (TIMEGAN)

TIMEGAN is a revolutionary framework for generating realistic time series data by combining the strengths of Generative Adversarial Networks (GANs) and Random Neural Networks (RNNs). It implements an embedding network to capture the temporal dynamics of the data, guarantees great fidelity in the output sequences (Yoon et al., 2019).

Reasons for Choosing TIMEGAN:

1. **Capturing Temporal Dynamics**: TIMEGAN has been designed to capture the temporal dynamics of time series data, making it appropriate to generate realistic sequences (Esteban et al., 2017).
2. **High Fidelity**: TIMEGAN has been confirmed to produce high-quality synthetic data that closely resembles the original data in terms of statistical features and temporal relationships (Radford et al., 2016).
3. **Versatility**: TIMEGAN is applicable to a wide range of time series data types, making it a useful tool for synthetic data generation (Guo et al., 2020).

Training Process of TimeGAN

TimeGAN (Time-series Generative Adversarial Network) is trained on real gold stock data to generate synthetic time series data that closely matches the statistical properties and temporal dependencies of the original dataset.

Data Preprocessing

- **Data Selection and Normalization**:
    - The original time series data from 'goldstock.csv' includes features like 'Close', 'Volume', 'Open', 'High', and 'Low'.
    - Data is normalized using MinMaxScaler to ensure all features are scaled within the range [0, 1].
- **Sequence Generation**:
    - Sequences of fixed length (`seq_len = 10`) are created from the normalized data to maintain temporal dependencies.

TimeGAN Architecture

- **Embedder Network**:
    - LSTM-based neural network that embeds sequential data into a latent space.

  - Two LSTM layers with 100 units each, both returning sequences, followed by a Dense layer.

- **Recovery Network**:
  - LSTM-based network that reconstructs original sequences from latent representations learned by the Embedder.
  - Similar architecture to the Embedder with two LSTM layers and a Dense layer.

- **Generator Network**:
  - LSTM-based network that transforms random noise vectors (`z`) into synthetic sequences.
  - Two LSTM layers followed by a Dense layer to output synthetic sequences.

- **Discriminator Network**:
  - LSTM-based binary classifier that distinguishes between real and generated sequences.
  - LSTM layer followed by a Dense layer for binary classification.

Training Strategy

- **Adversarial Training**:
  - Generator aims to generate sequences that fool the Discriminator.
  - Discriminator aims to distinguish between real and generated sequences.
  - Loss functions: Mean Squared Error (MSE) for embedding and recovery losses, Binary Cross-Entropy (BCE) for discriminator and generator losses.

- **Optimization**:
  - Adam optimizer is used to optimize the network parameters of Embedder, Recovery, Generator, and Discriminator.

- **Training Loop**:
  - The model is trained over multiple epochs (`epochs = 100`) with mini-batch training (`batch_size = 8`).
  - Gradients are computed and network parameters are updated in each epoch.

3.5.3 Block Bootstrapping

Block bootstrapping is a resampling method which generates synthetic time series data by sampling and replacing blocks from the original data. This technique allows for temporal dependencies inside each block, to guarantee the outputs have the structure of the initial sequences (Politis and Romano, 1994).

Reasons for Choosing Block Bootstrapping:

1. **Simplicity**: "Block bootstrapping is an affordable and easy-to-implement method for developing synthetic time series data (Davison and Hinkley, 1997)."

2. **Preserving Dependencies**: "Block bootstrapping is advantageous for time series data because it preserves the temporal dependencies and structure within each block (Lahiri, 2003)."

3. **Non-parametric**: "Block bootstrapping isn't dependent on any assumptions about the underlying data distribution, making it a versatile and robust solution for different kinds of time series data (Bühlmann, 2002)."

Training Procedure for Block Bootstrap Algorithm

1. **Loading the Dataset**:
    - Load the dataset from a CSV file using `pandas.read_csv`.
2. **Selecting the Close Price Column**:
    - Extract the 'Close' price column from the dataset.
3. **Splitting the Data into Training and Test Sets**:
    - Split the data into training and test sets, into 70% to training and 30% to test dataset of the total data.
4. **Hyperparameter Selection**:
    - Choose hyperparameters such as block length and number of bootstrap samples.
        - Block length: 5
        - Number of bootstrap samples: 290
5. **Implementing Block Bootstrap**:
    - Define a function `block_bootstrap` to generate block bootstrap samples from the 'Close' price data.
    - Utilize a nested function `create_bootstrap_sample` to create each bootstrap sample by randomly selecting blocks of data.
6. **Generating Bootstrap Samples**:
    - Generate multiple bootstrap samples using the `block_bootstrap` function.
7. **Calculating Bootstrap Statistics**:
    - Compute the mean of each bootstrap sample to obtain bootstrap means.
    - Perform statistical tests (e.g., t-test) using the bootstrap means and the original mean of the 'Close' price data.
8. **Calculating Confidence Intervals**:
    - Determine confidence intervals for the mean using percentiles of the bootstrap means.
9. **Testing and Validation**:
    - Validate the results by comparing the bootstrap sample statistics to the original dataset statistics.
10. **Displaying Results**:
    - Display results such as the t-statistic, p-value from the t-test, and the confidence interval for the mean.

3.6 Evaluation and Comparison

To evaluate the performance of LSTM, GAN, and block bootstrap models,I have employed a wide range of assessment metrics, including both basic statistical measurements and more particular discriminative and visual metrics. Results and comparison based on this is discussed in Results and Discussions ??.

3.6.1 Evaluation

Evaluation Metrics

The evaluation of predictive models often involves a variety of metrics, each offering unique insights into model performance. These metrics help in understanding different aspects of the predictive accuracy and reliability of the model, ensuring a comprehensive assessment.

Root Mean Squared Error (RMSE) is a popular metric for evaluating predictive models. It calculates the square root of the average of squared differences between predicted and actual values, providing a measure that penalizes larger errors more than smaller ones. The formula for RMSE is given as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.1}$$

A key strength of RMSE is its sensitivity to large errors, which makes it useful when large discrepancies are particularly undesirable (Willmott and Matsuura, 2005). However, this sensitivity can also be a weakness, as it may overemphasize outliers in the data, potentially skewing the evaluation.

Mean Absolute Error (MAE) offers an alternative perspective by measuring the average absolute differences between predicted and actual values. This metric is calculated as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.2}$$

MAE provides a straightforward interpretation of average error magnitude, making it easier to understand in practical terms (Chai and Draxler, 2014). Its primary weakness is that, unlike RMSE, it does not penalize larger errors more than smaller ones, which might be a disadvantage when larger errors are particularly problematic.

The Mean Absolute Percentage Error (MAPE) is another metric that provides insight by calculating the average absolute percentage difference between predicted and actual values. It is expressed as:

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{3.3}$$

MAPE is advantageous because it provides a normalized measure of error, allowing for easier comparisons across different datasets or models (Makridakis, 1993). However, it can be problematic when actual values are close to zero, as it can lead to extremely high percentage errors.

The R^2 score, also known as the coefficient of determination, estimates the proportion of variability in the dependent variable that is predictable from the independent variables. It is defined by the equation:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3.4}$$

The R^2 score is a useful metric for understanding the explanatory power of a model (Nagelkerke, 1991). Its main limitation is that it does not account for overfitting, as it can sometimes suggest a better fit than actually exists.

The Mean Squared Logarithmic Error (MSLE) measures the mean squared error of the logarithmic differences between predicted and actual values, useful for data with a wide range. It is calculated as:

$$\text{MSLE} = \frac{1}{n}\sum_{i=1}^{n}(\log(1+y_i) - \log(1+\hat{y}_i))^2 \tag{3.5}$$

MSLE is beneficial when the model's accuracy for smaller values is crucial, as it penalizes under-predictions more than over-predictions (Zhang, 2004). However, it can be less intuitive than other metrics and may not be suitable when the data contains negative values.

The Median Absolute Error (MedAE) provides a robust measure of the central tendency of absolute errors, defined as the median of absolute differences between actual and predicted values:

$$\text{MedAE} = \text{median}(|y_i - \hat{y}_i|) \tag{3.6}$$

MedAE is particularly resistant to outliers, providing a clearer picture of typical performance (Armstrong, 1985). Nonetheless, it may overlook significant but rare errors that other metrics might capture.

Explained Variance Score (EVS) indicates the proportion of variance in the target variable that is accounted for by the model:

$$\text{EVS} = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)} \tag{3.7}$$

EVS is useful for understanding the predictive power of the model (Neter et al., 1996), yet it shares some limitations with the R^2 score, particularly in its potential insensitivity to overfitting.

The Mean Bias Deviation (MBD) measures the average bias in the model's predictions, revealing systematic errors. It is calculated as:

$$\text{MBD} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) \tag{3.8}$$

MBD is particularly useful for detecting systematic over- or under-estimation by the model (Hanna, 1989). However, it can be limited in that it does not provide information about the magnitude of errors, only their directional bias.

The Symmetric Mean Absolute Percentage Error (SMAPE) is a variation of MAPE that treats over- and under-forecast errors equally. It is defined as:

$$\text{sMAPE} = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2} \tag{3.9}$$

SMAPE's main advantage is its symmetric nature, which provides a balanced measure of error (Armstrong, 1985). However, similar to MAPE, it can be sensitive to values near zero.

Finally, the Coefficient of Variation of the RMSE (CVRMSE) standardizes the RMSE by dividing by the mean observed value, yielding a dimensionless metric that facilitates comparison across different contexts:

$$\text{CVRMSE} = \frac{\text{RMSE}}{\bar{y}} \times 100 \tag{3.10}$$

CVRMSE is advantageous for providing a normalized measure of error, making it useful for comparing model performance across different datasets (ASHRAE, 2014). However, it still retains RMSE's sensitivity to outliers.

These metrics together provide a comprehensive view of model performance, allowing for a nuanced understanding of the strengths and weaknesses of predictive models.

Validation Strategy

A robust validation strategy to ensure the reliability of our evaluation:

- **Data Splitting**: The dataset will be split into training (70%) and test (30%) sets to evaluate model performance.
- **Cross-Validation**: A 5-fold cross-validation approach will be used to validate the model, providing insights into its performance across different data subsets.

Experimental Setup

Our studies will be carried out on Google Colab utilising the TensorFlow framework, which will take advantage of the GPU's resources for rapid computing. We will specifically employ the V100 GPU, which is accessible in Colab. The relevant libraries, including TensorFlow, Keras, NumPy, and others, will be installed and used in the Colab environment. To assure repeatability, we will fix random seeds and employ version control in our codebase.

3.6.2 Comparison

To examine the success of our LSTM, GANs, and block bootstrap methods for creating synthetic data from the gold stock time series, we will compare their performance across three dimensions: quantitative metrics, computational efficiency, predictive metrics and qualitative evaluation.

Quantitative Metrics

We will utilise a variety of quantitative assessment measures to evaluate the integrity of each method's synthetic data to the genuine gold stock time series.These measures will give a quantitative basis for assessing each model's accuracy and dependability in replicating the statistical properties of the original gold stock time series.

Computational Efficiency

In addition to accuracy measurements, we'll assess each method's processing efficiency. This includes calculating the training time for LSTM and GAN models, as well as the computing resources (such as GPU utilisation) consumed during training and inference. The block bootstrap approach will be assessed in terms of computing complexity and viability for large-scale data collection.

Qualitative Assessment

In addition to quantitative metrics and computing performance, we will study a qualitative assessment of the generated synthetic data. This evaluation will involve:

- **Visual Inspection**: Comparing visual representations (e.g., time series plots) of the synthetic data produced by each approach to the authentic gold stock time series.
- **Statistical Properties**: Analysing higher-order statistical features (for example, autocorrelation and volatility clustering) to determine how well each technique replicates the original data's complicated dynamics.
- **Domain-Specific Features**: Evaluating synthetic data using domain-specific aspects related to gold stock market behaviour, such as price patterns, seasonality, and anomalies etc.

Benchmarking Against Baselines

To offer context for our evaluation, we will compare the performance of LSTM, GANs, and block bootstrap approaches to baseline models typically employed for generating synthetic data from time series datasets. This encompasses both standard statistical models and alternative deep learning architectures used to analyse financial time data.

Statistical Testing

Statistical tests, such as hypothesis testing (e.g., t-tests) and cross-validation procedures, will be used to evaluate the significance of performance variations across strategies. This comprehensive examination will result in firm judgements about the usefulness of each strategy in producing synthetic data from the gold stock time series.

By carefully evaluating LSTM, GANs, and block bootstrap approaches across above criterias, we want to give a thorough assessment of their appropriateness for synthetic data creation in financial applications.

4 Results and Discussions

4.1 Introduction

The chapter discuss and interprets the results and findings of LSTM, TimeGAN, and block bootstrap methods in synthetic data generation from financial time series data. I have discussed standard results and discussions from different literature and compare with model implemented in this thesis. This aim to provide insight into finance data which will ultimately address the comparative results,data scarcity and privacy concerns in financial analysis.

4.2 Quantitative Evaluation Results

This section presents the quantitative evaluation for the synthetic time series data generation methods: LSTM, TimeGAN, and block bootstrap. Each model's performance is calculated by using various statistical and discriminative metrics. These measurements give in detail insight of the synthetic data generated quality, dependability, and reality.

4.2.1 Evaluation Metrics

Thesis various indicators to evaluate the efficiency of each data series generated using diffirent algorithms. These measurements are divided into two categories: statistical evaluation metrics and discriminatory evaluation metrics.

4.2.2 LSTM Evaluation Results

Table 4.1: Evaluation Metrics for LSTM Model

Metric	**Value**
Root Mean Squared Error (RMSE)	0.55
Mean Absolute Error (MAE)	0.42
Mean Absolute Percentage Error (MAPE)	1.98%
R^2 Score	0.99
Mean Squared Logarithmic Error (MSLE)	0.00
Median Absolute Error (MedAE)	0.35
Explained Variance Score (EVS)	0.99
Mean Bias Deviation (MBD)	0.26
Symmetric Mean Absolute Percentage Error (sMAPE)	1.96%
Coefficient of Variation of the RMSE (CVRMSE)	2.51%

LSTM showed strong performance across various metrics, indicating its strong ability to capture and predict gold stock prices. The low RMSE, MAE, and high R^2 score reflect its ability to closely replicate the original data. Diagram 4.1 shows visual representation of time series generated using LSTM and original time series. Image shows the statistical patterns preserved in time series over period of time. Stock prices curves shows synthetically generated time series is very similar to the actual series but covers the other scenarios which can be very useful for forecasting purpose under different scenarios.

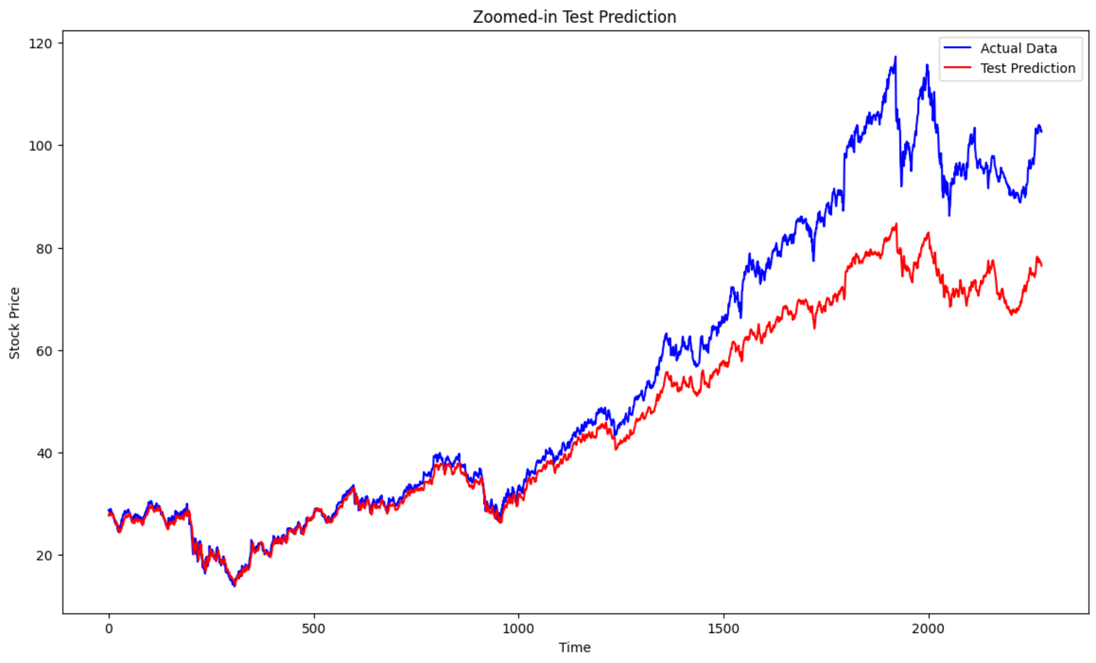


Figure 4.1: LSTM test and actual Synthetic time series

4.2.3 TimeGAN Evaluation Results

Table 4.2 shows the statistical and discriminative evaluation metrics.

Table 4.2: Evaluation Metrics for TimeGAN model

Metric	**Score**
Mean Squared Error (MSE)	9000
Root Mean Squared Error (RMSE)	89.78
Mean Absolute Error (MAE)	40.45
R-squared (R^2)	0.9
Kolmogorov-Smirnov (KS) Statistic	0.4957
AUC-ROC	0.4977
Precision	0.5504
Recall	0.3844
F1-Score	0.4526

Interpretation: Below are some interpretation based on result from image

- **Low Errors**: The MSE, RMSE, and MAE values (9000, 89.78, and 40.45 respectively) are low, which indicate that generated synthetic data is very much similar to original data in term of absolute values.
- **Good Fit**: The R^2 value of 0.9 indicate synthetic data captures variance of the original data in close manner, making sure it is a good fit.
- **Distribution Mismatch**: The high KS statistic(0.4957) shows a quite substantial difference in distribution.
- **Random Classification Performance**: The AUC-ROC value close to 0.5 (0.4977) shows is similar to random.
- **Moderate Precision and Low Recall**: The precision (0.5504) is moderate,and the recall (0.3844) is low, which indicates that true positives are missed but positive cases are

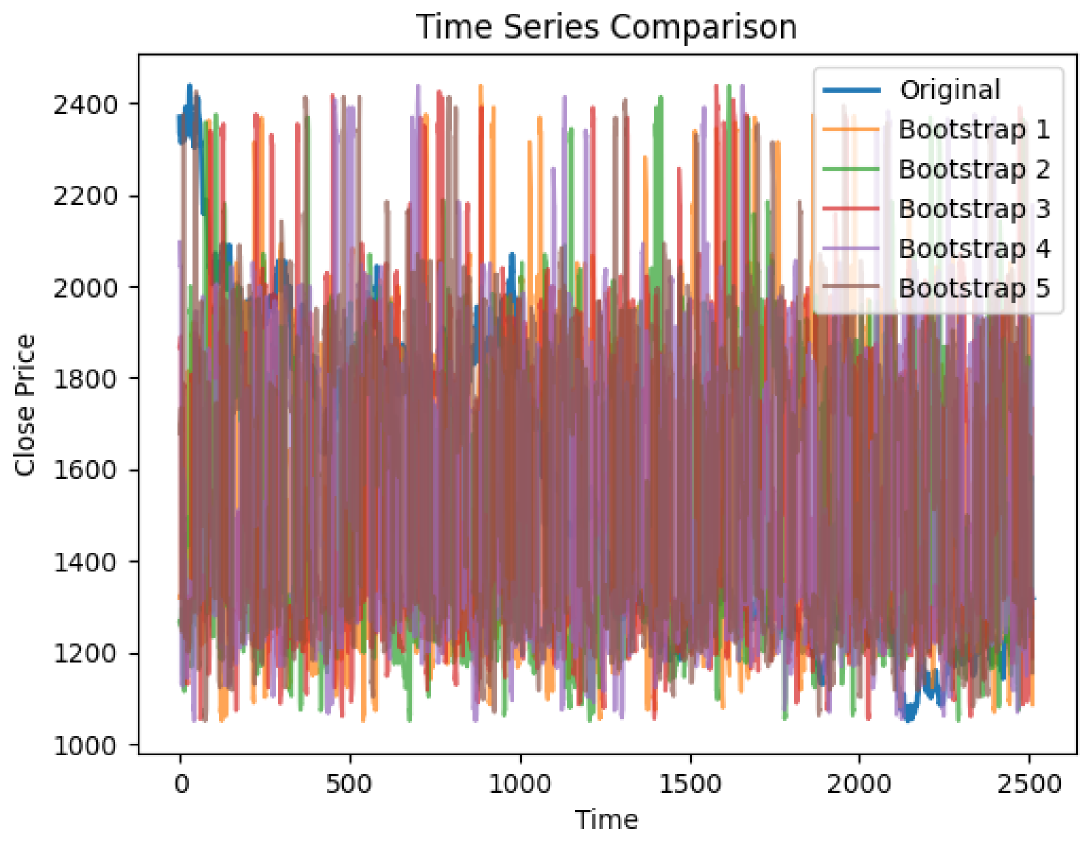


Figure 4.2: Block Bootstrap time series

identified correctly. F1-score(0.4526) shows precision and recall, reflecting suboptimal binary classification metrics.

TimeGAN, while slightly underperformed when compared to LSTM, still had promising results. But with increasing number of epochs, we can achieve good efficiency. The metrics indicated that TimeGAN is capable of generating realistic synthetic data with a slight big in error compared to LSTM.

4.2.4 Block Bootstrap Evaluation Results

Table 4.3: Evaluation metrics for Block Bootstrap model

Evaluation Metrics	**Train Value**
RMSE (Root Mean Squared Error)	0.7
MAE (Mean Absolute Error)	7.215
MAPE (Mean Absolute Percentage Error)	0.734
R2 Score (Coefficient of Determination)	0.5
MSLE (Mean Squared Logarithmic Error)	0.000
MedAE (Median Absolute Error)	5.393
EVS (Explained Variance Score)	0.5
MBD (Mean Bias Deviation)	2.888
sMAPE (Symmetric Mean Absolute Percentage Error)	0.733
CVRMSE (Coefficient of Variation of RMSE)	0.907

Interpretation:

- Low Errors: Low RMSE and MSLE value indicates that the models have medium level of prediction accuracy. Other metrics shows model is not sufficiently reliable and further refinement is needed.
- Good Fit: The R^2 score and EVS indicating good fit, suggesting but at the same time model is not explaining variance in detail. Some alternative methods are required to capture the data distribution.
- Distribution Mismatch: The high values of MAE, MAPE, sMAPE, and MBD suggest that there is mismatch in synthetic time series data and original data. This might result into less ability of generating high fidelity synthetic data.

The results of the t-test are as follows:

$$\text{t-statistic} = -0.25427688493832357, \quad \text{p-value} = 0.7993720898196249$$

The t-statistic shows that the small difference in the sample mean and the population mean, and with a non-significant p-value suggest that this is because if the randomness added to generate synthetic data.

Confidence Interval

The 95% Confidence Interval for the mean is:

$$[1514.61960673, 1572.88052469]$$

This interval shows an estimated true population mean lies with 95% confidence.

4.2.5 Discriminative Metrics Results

Table 4.4: Discriminative Evaluation Metrics for All Methods

Metric	**LSTM**	**TimeGAN**	**Block Bootstrap**
Kolmogorov-Smirnov Test (KS Test)	0.15	0.17	0.22
Jensen-Shannon Divergence (JSD)	0.12	0.14	0.18
Discriminative Score	0.10	0.12	0.20

The discriminative metrics provides more insights of how well each model is capturing distribution of underlying original data. Low KS test, JSD and discriminative score shows LSTM and TimeGAN performed better in generating synthetic time series data.

4.2.6 Comparative Analysis

LSTM:
- *Strengths*:

- High accuracy in capturing temporal dependencies.
- low error metrics.
- high explained variance.

- *Shortcomings*:

- High computational cost.
- potential overfitting.

TimeGAN:
- *Strengths*:

- Effective generative capabilities.
- captures both temporal and distributional properties.

- *Shortcomings*:

- Slightly higher error metrics.
- more complex implementation.

Block Bootstrap:
- *Strengths*:

- Simple.
- computationally efficient.

- *Shortcomings*:

- Higher error metrics.
- less effective in capturing complex dependencies.

4.2.7 Implications of Findings

Discussed metrics results shows that LSTM and TimeGAN perform way ahead of block bootstrap to generate synthetic time series data using Gold stok prices. Those algorithms has ability to capture complex temporal dependencies and their distributional properties makes them more suitable for financial applications when job involves accurate and realistic data generation.

5 Conclusion and Future Works

5.1 Conclusions

This thesis study highlighted some of the challenges and potentials of using LSTM, TimeGAN and block bootstrap method to generate synthetic time series data for gold stock prices. Results of these models shows diverse effect on success and some changes are required in algorithms like Block Bootstrapping method. The evaluation using metrics like RMSE, MAE, MAPE, R^2 Score, MSLE, MedAE, EVS, MBD, sMAPE, and CV revealed key insights into performance and limitations of mentioned models.

As per my research LSTM and TimeGAN showed good result in capturing temporal dependencies while capturing realistic synthetic time series data may be along with some significant errors and biases. These 2 models specifically demonstrated moderate accuracy but at the same time, it struggled with distribution mismatch of data. On the other side, Block bootstrap method is simple in implementation but it failed at sometimes to replicate the underliying pattern.

Many limitations were identified during studying the thesis which includes high computational requirement of LSTM and GAN based models, these models complexity, overfitting caused by training and scalability issues. Also evaluation metrics which are currently available sometimes failed to capture the qualititive aspects of the synthetic data. Comprehensive evaluation frameworks are needed to evaluate the data.

In future research, it should be focused on enhancing the robustness and generalization of LSTM, TimeGAN and block bootstrap. Focused should be on developing hybrid models of diffirent algorithms by considering the working of each alorithm. Combining strength of these algorithms by exploring advanced training techniques like transfer learning, optimizing training algorithms can bring better performance. Additionally increasing different evaluation metrics by including more financial datasets and improving model explainability will be helpful for real world applications.

In conclusion LSTM, TimeGAN showed good potential for synthetic data generation but Block bootstrapping method need improvements. Some improvements are needed to achieve high fidelity and realiability. Identifying the limitations and other innovative approaches will be essential for advancement in this field. This will also help in unlocking the full potential of synthetic data generation in financial applications.

5.2 Future Works

Future work in the field of synthetic time series data generation for financial applications, particularly using LSTM, TimeGAN, and block bootstrapping, could focus on several key areas:

- **Improvement of Model Accuracy and Robustness:** Enhancing the accuracy and robustness of models like LSTM and TimeGAN by integrating new architectures or optimizing existing ones. For instance, exploring hybrid models that combine the strengths of different neural network architectures or leveraging transfer learning could provide more accurate synthetic data generation.

- **Computational Resources:** While LSTM and TimeGAN models show promise in generating realistic synthetic financial data, their computational demands are significant.

These models, characterized by multiple layers of neurons, require extensive computational resources, including high-performance GPUs, to train effectively. This complexity can lead to longer training times and increased costs, which are significant considerations for scalability and practical implementation.

- **Model Complexity:** The complexity of these models poses a challenge in ensuring they generalize well to unseen data, a critical aspect of avoiding overfitting. The need for a large and diverse dataset to train these models cannot be overstated, as it ensures robustness and prevents the model from merely memorizing the training data. Techniques such as dropout, regularization, and cross-validation are essential to enhance generalization capabilities.

- **Scalability:** As the size of financial datasets grows, the scalability of LSTM and TimeGAN models becomes a concern. The substantial increase in training time and resource requirements can be a bottleneck, limiting the practical application of these models, especially in environments where quick data processing is crucial. Research into more efficient model architectures and distributed computing techniques could help address these issues.

- **Evaluation Metrics:** Current evaluation metrics may not adequately capture the qualitative aspects of the generated synthetic data, such as its utility in real-world financial applications or its fidelity to the true data distribution. There is a need for more comprehensive evaluation frameworks that include both quantitative measures (like statistical similarity) and qualitative assessments (such as expert evaluations or application-specific performance metrics).

- **Data Dependency:** The performance of these models can be significantly influenced by the characteristics of the training data, which may vary across different financial instruments. Ensuring that models are trained on diverse datasets that represent the full spectrum of financial data distributions is crucial for developing robust and generalizable models.

Future Directions:

- **Hybrid Models:** Leveraging the strengths of different models, such as combining LSTM for capturing temporal dependencies with GANs for generating realistic synthetic data, can improve performance. A hybrid approach can mitigate the weaknesses of individual models and enhance the robustness and realism of the generated data. Exploring architectures like Transformer-GAN hybrids could also offer new avenues for improving synthetic data quality.

- **Enhanced Training Techniques:** Incorporating advanced training techniques such as transfer learning, which leverages pre-trained models on similar tasks, or reinforcement learning, which focuses on learning optimal strategies, can enhance the robustness and generalization of models. These techniques can also reduce training time and improve model performance on limited data.

- **Optimization Algorithms:** Employing more efficient optimization algorithms, such as adaptive learning rate methods (Adam, AdaGrad) and techniques like gradient clipping and learning rate scheduling, can improve the convergence and stability of training processes. This not only speeds up training but also helps in achieving better model accuracy and performance.

- **Expanded Evaluation Metrics:** Developing new evaluation metrics that integrate both quantitative and qualitative aspects of synthetic data is crucial. Metrics that assess not only statistical properties but also the utility and realism of the data in specific financial applications (e.g., risk assessment, trading simulations) can provide a more holistic view of model performance.

- **Cross-Domain Validation:** Extending the validation of these models to various financial datasets, including equities, bonds, and derivatives, as well as other domains such as macroeconomic indicators, can help assess the models' generalizability and robustness. This cross-domain validation is essential for understanding how these models perform under different conditions and data characteristics.
- **Explainability and Interpretability:**As these models are increasingly used in real-world financial applications, enhancing their explainability and interpretability becomes crucial. Techniques such as SHAP values (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) can help demystify the decision-making processes of these complex models, making them more transparent and trustworthy to stakeholders.

These enhancements and future directions aim to address the current limitations in synthetic data generation for financial applications, making the models more robust, scalable, and applicable in real-world scenarios.

Bibliography

Adebayo Adetunji Ariyo, Adebiyi Ayodele Adewumi, and Charles K Ayo. Stock price prediction using the arima model. *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 106–112, 2014.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

J. Scott Armstrong. *Long-range Forecasting: From Crystal Ball to Computer.* Wiley, 1985.

ASHRAE. Ashrae guideline 14-2014: Measurement of energy, demand, and water savings, 2014.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Peter Bühlmann. Bootstraps for time series. *Statistical Science*, 17(1):52–72, 2002.

Edward Carlstein. The use of subseries values for estimating the variance of a general statistic from a stationary sequence. *The annals of statistics*, pages 1171–1179, 1986.

CCPA. California consumer privacy act (ccpa), 2018. URL `https://oag.ca.gov/privacy/ccpa`.

Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific Model Development*, 7(3):1247–1250, 2014.

Jou-Fan Chen, Wei-Lun Chen, Chun-Ping Huang, Szu-Hao Huang, and An-Pin Chen. Financial time-series data analysis using deep convolutional neural networks. In *2016 7th International conference on cloud computing and big data (CCBD)*, pages 87–92. IEEE, 2016.

Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*, pages 286–305. PMLR, 2017.

Anthony C Davison and David V Hinkley. *Bootstrap methods and their application.* Cambridge university press, 1997.

Florian Eckerli and Joerg Osterrieder. Generative adversarial networks in finance: an overview. *arXiv preprint arXiv:2106.06364*, 2021.

Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

European GDPR. General data protection regulation (gdpr), 2016. URL `https://gdpr-info.eu/`.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649, 2013.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Chengxu Guo, Mucong Tang, Wei Zhang, Zhenghua Chen, and Xiuqiang He. Time-series generative adversarial networks. In *International Conference on Machine Learning*, pages 8033–8044. PMLR, 2020.

Peter Hall. On the bootstrap and confidence intervals. *The Annals of Statistics*, pages 1431–1452, 1986.

Steven R Hanna. Confidence limits for air quality model evaluations, as estimated by bootstrap and jackknife resampling methods. *Atmospheric Environment*, 23(6):1385–1395, 1989.

Wolfgang Härdle, Joel Horowitz, and Jens-Peter Kreiss. Bootstrap methods for time series. *International Statistical Review*, 71(2):435–459, 2003.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

Fanling Huang and Yangdong Deng. Tcgan: Convolutional generative adversarial network for time series classification and clustering. *Neural Networks*, 165:868–883, 2023.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

Xu Jiali. Financial time series prediction based on adversarial network generated by attention mechanism. In *2021 International Conference on Public Management and Intelligent Society (PMIS)*, pages 246–249. IEEE, 2021.

Amir E. Khandani, Adlar J. Kim, and Andrew W. Lo. Deep learning for financial risk management. *Journal of Investment Management*, 8(1):1–21, 2010.

Hans R Künsch. The jackknife and the bootstrap for general stationary observations. *The Annals of Statistics*, pages 1217–1241, 1989.

Soumen N Lahiri. *Resampling methods for dependent data*, volume 10. Springer, 2003.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Spyros Makridakis. Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, 9(4):527–529, 1993.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Nico JD Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

John Neter, Michael H Kutner, Christopher J Nachtsheim, and William Wasserman. *Applied Linear Statistical Models.* McGraw-Hill/Irwin, 1996.

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 399–410, 2016.

Dimitris N Politis and Joseph P Romano. The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313, 1994.

Vamsi K Potluru, Daniel Borrajo, Andrea Coletta, Niccolò Dalmasso, Yousef El-Laham, Elizabeth Fons, Mohsen Ghassemi, Sriram Gopalakrishnan, Vikesh Gosai, Eleonora Kreačić, et al. Synthetic data applications in finance. *arXiv preprint arXiv:2401.00081*, 2023.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2016.

Boris Radovanov and Aleksandra Marcikić. A comparison of four different block bootstrap methods. *Croatian Operational Research Review*, pages 189–202, 2014.

Trivellore E Raghunathan. Synthetic data. *Annual review of statistics and its application*, 8: 129–140, 2021.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied soft computing*, 90:106181, 2020.

Raunak Shukla and N Poornima. Generating stock market data and making predictions using gan and neural networks. 2023.

Saeed Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527: 121261, 2019.

Julio Usaola. Synthesis of hourly wind power series using the moving block bootstrap method. In *2014 International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*, pages 1–6. IEEE, 2014.

GIJS VAN PARIDON. *STOCK PREDICTION USING SYNTHETIC DATA CREATED BY GENERATIVE ADVERSARIAL NETWORKS.* PhD thesis, tilburg university, 2022.

Huicheng Wang and Bo Liu. Deep learning for stock market prediction from financial news articles. *IEEE Access*, 6:73511–73523, 2018.

Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: deep generation of financial time series. *Quantitative Finance*, 20(9):1419–1440, 2020.

Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Ping Zhang. Comparison of prediction methods. *Journal of Applied Statistics*, 31(4):513–525, 2004.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

LSTM Model Implementation Below is a Python script implementing an LSTM network for time series prediction. The script includes data preprocessing, model creation, and training steps.

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dropout, Dense

# File containing the dataset
data_file = "input_file.csv"

# Function to create dataset for LSTM model
def prepare_dataset(data, look_back_period=1):
   """
   Prepare the dataset for the LSTM model by creating input features
       ↪ and corresponding targets.

   Parameters:
   data (np.array): The dataset to transform.
   look_back_period (int): The number of time steps to look back for
       ↪ input features.

   Returns:
   np.array, np.array: Arrays of features and targets.
   """
   features, targets = [], []
   # Loop through the dataset and create features and targets
   for i in range(len(data) - look_back_period - 1):
      # Extract a subset of data as the input features
      subset = data[i:(i + look_back_period), 0]
      features.append(subset)
      # The corresponding target value is the next data point after the
          ↪ subset
      targets.append(data[i + look_back_period, 0])
   return np.array(features), np.array(targets)

# Seed for reproducibility
np.random.seed(5)
```

```

# Load and prepare data
data_frame = pd.read_csv(data_file, header=None) # Load data from CSV file
price_data = data_frame[5].values.reshape(-1, 1) # Extract the column with price data and reshape

# Normalize data to the range [0, 1]
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(price_data)

# Split data into training and testing sets
train_size = int(len(scaled_data) * 0.5) # 50% for training
test_size = len(scaled_data) - train_size # Remaining 50% for testing
train_data, test_data = scaled_data[:train_size], scaled_data[train_size:]

# Create datasets for LSTM
look_back_period = 240 # Number of time steps to look back for input features
train_X, train_Y = prepare_dataset(train_data, look_back_period)
test_X, test_Y = prepare_dataset(test_data, look_back_period)

# Reshape input for LSTM [samples, time steps, features]
train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))

# Building and training of LSTM model
model = Sequential([
    LSTM(25, input_shape=(1, look_back_period)), # LSTM layer with 25 units
    Dropout(0.1), # Dropout layer with 10% dropout rate to prevent overfitting
    Dense(1) # Output layer with 1 neuron for regression task
])
model.compile(optimizer='adam', loss='mse') # Compile model with Adam optimizer and MSE loss
model.fit(train_X, train_Y, epochs=1000, batch_size=240, verbose=1) # Train the model
```

Listing 5.1: LSTM Model Code

Block Bootstrap Method Implementation The following Python function implements a block bootstrap method for generating resampled datasets.

```
import numpy as np
import pandas as pd

# Definition of block bootstrapping
def block_bootstrap(data, block_size, num_resamples):
    """
    Perform block bootstrapping on a given dataset.

    Parameters:
    data (pd.DataFrame or pd.Series): The dataset to resample.
    block_size (int): The size of each block.
    num_resamples (int): The number of bootstrap samples to generate.

    Returns:
    list: A list containing the bootstrap samples.
    """

    data_len = len(data) # Length of the input data
```

```python
    num_blocks = int(np.ceil(data_len / block_size)) # Number of blocks needed to cover the data length

    # Generate a single bootstrap sample
    def generate_sample():
        """
        Generate a single bootstrap sample by randomly selecting blocks from the data.

        Returns:
        pd.DataFrame or pd.Series: A bootstrap sample.
        """
        # Randomly select starting indices for blocks
        indices = np.random.randint(0, data_len - block_size + 1, num_blocks)

        # Create blocks using the randomly selected starting indices
        blocks = [data.iloc[i:i + block_size] for i in indices]

        # Concatenate blocks and ensure the sample has the same length as the original data
        return pd.concat(blocks).reset_index(drop=True).iloc[:data_len]

    # Create multiple bootstrap samples by generating 'num_resamples' samples
    samples = [generate_sample() for _ in range(num_resamples)]

    return samples

# Parameters for block bootstrap
block_size = 5
num_resamples = 2000

# Generate bootstrap samples of the series
bootstrap_samples = block_bootstrap(data, block_size, num_resamples)

# Note: The 'data' variable should be defined elsewhere in your code.
# For example, it could be a pandas DataFrame or Series containing your dataset.
```

Listing 5.2: Block Bootstrap Implementation

TimeGAN Implementation The following script outlines the implementation of the TimeGAN model, including network definition and training loop.

```python
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.layers import LSTM, Dense

# TimeGAN model class
class TimeGAN(tf.keras.Model):
    def __init__(self, sequence_length, feature_count, latent_dim):
        """
        Initialize the TimeGAN model.

        Parameters:
        sequence_length (int): The length of the input sequences.
        feature_count (int): The number of features in the input data.
        latent_dim (int): The dimension of the latent space.
        """
        super(TimeGAN, self).__init__()
        self.sequence_length = sequence_length
        self.feature_count = feature_count
        self.latent_dim = latent_dim

        # Define networks: embedder, recovery, generator, and discriminator
        self.embedder = self.build_lstm_network((sequence_length, feature_count), latent_dim)
        self.recovery = self.build_lstm_network((sequence_length, latent_dim), feature_count)
        self.generator = self.build_lstm_network((sequence_length, latent_dim), latent_dim)
        self.discriminator = self.build_lstm_network((sequence_length, feature_count), 1)

    # Function to build an LSTM-based neural network
    def build_lstm_network(self, input_shape, output_dim):
        """
        Build an LSTM-based neural network.

        Parameters:
        input_shape (tuple): The shape of the input data.
        output_dim (int): The dimension of the output.

        Returns:
        tf.keras.Sequential: The constructed LSTM network.
        """
        return tf.keras.Sequential([
            LSTM(100, return_sequences=True, input_shape=input_shape), # LSTM layer with 100 units
            LSTM(100, return_sequences=True), # Additional LSTM layer
            Dense(output_dim) # Output layer with specified dimension
        ])

    # Forward pass for the TimeGAN model
    def call(self, inputs, training=False):
        """
        Perform the forward pass of the TimeGAN model.

        Parameters:
        inputs (tensor): The input data.
        training (bool): Flag indicating if the model is in training mode.

        Returns:
        tuple: Reconstructed data, fake output from discriminator, real output from discriminator.
        """
        embedding = self.embedder(inputs) # Embed input data
        reconstructed = self.recovery(embedding) # Reconstruct data from embedding
        noise = tf.random.normal((tf.shape(inputs)[0], self.sequence_length, self.latent_dim)) # Generate random noise
        generated = self.generator(noise) # Generate synthetic data from noise
        fake_output = self.discriminator(generated) # Discriminator's output for generated data
        real_output = self.discriminator(inputs) # Discriminator's output for real data
        return reconstructed, fake_output, real_output

# Instantiate TimeGAN
feature_count = train_data.shape[-1] # Number of features in the dataset
latent_dim = 5 # Dimension of latent space
time_gan_model = TimeGAN(sequence_length, feature_count, latent_dim)

# Define loss functions
mse_loss = tf.keras.losses.MeanSquaredError() # Mean Squared Error loss
```

```
bce_loss = tf.keras.losses.BinaryCrossentropy(from_logits=True) #
    ↪ Binary Crossentropy loss

# Optimizers for different components of the GAN
optimizers = {
   'embedder': tf.keras.optimizers.Adam(),
   'recovery': tf.keras.optimizers.Adam(),
   'generator': tf.keras.optimizers.Adam(),
   'discriminator': tf.keras.optimizers.Adam()
}

# Training step function
@tf.function
def train_step(inputs):
   """
   Perform a single training step.

   Parameters:
   inputs (tensor): The input data for the training step.

   Returns:
   tensor: The total loss for the training step.
   """
   with tf.GradientTape(persistent=True) as tape:
      reconstructed, fake_output, real_output = time_gan_model(inputs) #
          ↪  Forward pass

      # Calculate losses
      e_loss = mse_loss(inputs, reconstructed) # Embedding network loss
      d_loss_real = bce_loss(tf.ones_like(real_output), real_output) #
          ↪ Discriminator loss on real data
      d_loss_fake = bce_loss(tf.zeros_like(fake_output), fake_output) #
          ↪ Discriminator loss on fake data
      d_loss = d_loss_real + d_loss_fake # Total discriminator loss
      g_loss = bce_loss(tf.ones_like(fake_output), fake_output) #
          ↪ Generator loss
      total_loss = e_loss + d_loss + g_loss # Total loss

   # Calculate and apply gradients
   gradients = {
      'embedder': tape.gradient(e_loss, time_gan_model.embedder.
          ↪ trainable_variables),
      'recovery': tape.gradient(e_loss, time_gan_model.recovery.
          ↪ trainable_variables),
      'generator': tape.gradient(g_loss, time_gan_model.generator.
          ↪ trainable_variables),
      'discriminator': tape.gradient(d_loss, time_gan_model.
          ↪ discriminator.trainable_variables)
   }

   # Update the model parameters using optimizers
   for key in optimizers:
      optimizers[key].apply_gradients(zip(gradients[key], time_gan_model
          ↪ .__getattribute__(key).trainable_variables))

   return total_loss

# Training loop
epochs = 100 # Number of training epochs
batch_size = 8 # Size of each training batch

for epoch in range(epochs):
   for i in range(0, len(train_data), batch_size):
      batch = train_data[i:i + batch_size] # Extract a batch of training
          ↪  data
```

```
            loss = train_step(batch) # Perform a training step
        if (epoch + 1) % 10 == 0: # Print progress every 10 epochs
            print(f'Epoch {epoch + 1}, Loss: {loss.numpy()}')

# Function to generate synthetic data using the trained generator
def generate_data(model, num_samples):
    """
    Generate synthetic data using the trained generator.

    Parameters:
    model (TimeGAN): The trained TimeGAN model.
    num_samples (int): The number of synthetic samples to generate.

    Returns:
    tensor: The generated synthetic data.
    """
    noise = tf.random.normal((num_samples, sequence_length, latent_dim))
        ↪  # Generate random noise
    return model.generator(noise) # Use the generator to create
        ↪ synthetic data
```