

XI'AN JIAOTONG-LIVERPOOL
UNIVERSITY

西 交 利 物 浦 大 学

COURSEWORK
SUBMISSION COVER
SHEET

Name	Zheng	Qichang
Student Number	1822029	
Programme	Economics and Finance	
Module Title	Final Year Project	
Module Code	BUS303	
Assignment Title	TimeGAN in Stock Prediction	
Submission Deadline	2 nd May 2022	
Module Leader	Chengyu Yang	

By uploading or submitting this coursework submission cover sheet, I certify the following:

- I have read and understood the definitions of collusion, copying, plagiarism, and dishonest use of data as outlined in the Academic Integrity Policy of Xi'an Jiaotong-Liverpool University.
- This work is my own, original work produced specifically for this assignment. It does not misrepresent the work of another person or institution as my own. Additionally, it is a submission that has not been previously published, or submitted to another module
- This work is not the product of unauthorized

collaboration between myself and others.

- This work is free of embellished or fabricated data.

I understand collusion, plagiarism, dishonest use of data, and submission of procured work are serious academic misconducts. By uploading or submitting this cover sheet, I acknowledge that I am subject to disciplinary action if I am found to have committed such acts.

Signature

Date

.....

For Academic Office use:	Date Received	Working Days Late	Penalty ¹

¹ Please refer to clause 43 (i), (ii) and (iii) of the Code of Practice on Assessment for the standard system of penalties for the late submission of assessment work based on **working days**.

TimeGAN in Stock Prediction

By

Qichang Zheng

1822029

Supervisor: Prof. Jia Zhai

**A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Bachelor of Science**

to

**International Business School Suzhou
Xi'an Jiaotong-Liverpool University**

29th April 2022

Abstract

Quantitative investing is popular among investors and analysts because of its ability to generate excess returns. Machine learning models are one of the quantitative investment models and get good prediction results. GAN (Generative Adversarial Networks) is a generative deep learning model that has achieved brilliant success in image processing. However, because stock data is time-series data with time-dependent nature, GAN-based stock prediction models are slow to be developed. In 2019, Yoon proposed a GAN model for time series, TimeGAN, which can handle the temporal correlations of time series data well (Yoon et al., 2019). In this paper, we want to apply TimeGAN to stock prediction, optimize this model and evaluate its prediction performance. The results of this paper may provide quantitative analysts and investors with a new model for quantitative investment while pointing out how to optimize this model as a future research direction.

摘要

量化投资在投资者和分析师中很受欢迎，因为它能够产生超额收益。机器学习模型是量化投资模型之一，并获得良好的预测结果。GAN 是一种生成式深度学习模型，在图像处理方面取得了辉煌的成就。然而，由于股票数据是具有时间相关性的时间序列数据，基于 GAN 的股票预测模型发展缓慢。2019 年，Yoon 等人提出了一个用于时间序列的 GAN 模型--TimeGAN，它可以很好地处理时间序列数据的时间相关性(Yoon et al., 2019)。在本文中，我们希望将 TimeGAN 应用于股票预测，优化该模型并评估其预测性能。本文的结果可能为量化分析师和投资者提供一个新的量化投资模型，同时指出如何优化这个模型作为未来的研究方向。

Table of Contents

1	Introduction	7
2	Literature Review.....	10
3	Data	18
3.1	Data Access	18
3.2	Data Storage	19
3.3	Data Visualization	20
3.4	Data Normalization	21
4	Methodology	22
4.1	Model Overview	22
4.2	Two Objectives.....	23
4.3	Model Structure	24
4.3.1	Embedding and Recovery Functions	24
4.3.2	Sequence Generator and Discriminator	25
4.3.3	Jointly Learning to Encode, Generate, and Iterate	26
4.4	Model hyperparameters	27
4.5	Model Optimization	29
4.6	Model saving and breakpoint training	30
4.7	Comprehension of Hyperparameters	31
5	Results and conclusions	34
5.1	Model Evaluation	34
5.1.1	Visualization	34
5.1.2	Discriminative Score	35
5.1.3	Predictive Score	36
5.2	Training Procedure and Findings	36
5.3	Results Depiction	37
5.4	Potential problem	40
5.5	Subsequent optimization	41
6	Acknowledgements.....	43
7	Appendix	44
8	Reference.....	46

List of Figures

Figure 1	Tushare's interface	19
Figure 2	get_data(symbol).....	20
Figure 3	view(symbol).....	21
Figure 4	Example of t-SNE	35
Figure 5	Example of PCA	35
Figure 6	Result part 1	38
Figure 7	Result part 2	38
Figure 8	Result part 3	39
Figure 9	Result part 4	39

1 Introduction

Since the birth of the financial market, investors and analysts have been conducting the stock price prediction task to gain economic profits from the accurate forecast of stock price. Therefore, according to their predictions, many of them take active investment strategies to gain economic returns. There are many successes of these strategies, such as the direct British investment in overseas real estate (McAllister, 1999), foreign investments in Vietnam (Meyer and Nguyen, 2005), the contrarian investments by Dutch pension funds, life insurance, and non-life insurers (De Haan and Kakes, 2011), children-oriented investments in Europe (Van Lancker, 2013), the investment strategies in advanced manufacturing technologies (AMT) in the Turkish automobile industry (Bülbül et al., 2013), the private sector's involvement in public-private partnerships in underdeveloped nations (Wang et al., 2019), the Russian market's active investments (Teplova et al., 2020), carbon-neutral investments in BRICS (Ji et al., 2021).

Numerous investors are in favor of a subset of active investing known as quantitative investing, which mixes financial data with mathematics and computer technology, such as machine learning algorithms. Additionally, 56 of 95 recent research on technical trading systems have shown good outcomes (Park and Irwin, 2007). Because of the advancement of computer technology, it has become an excellent tool for analysts and investment counseling in China, where traditional time series prediction technology has numerous flaws when it comes to nonlinearity prediction (Peng and Tang, 2019).

Numerous quantitative investments make use of machine learning algorithms, which have been shown to be quite effective in the nonlinear stock market (Zhang, 2020). The primary reason is that machine learning approaches offer significant benefits, particularly in nonlinear forecasting, owing to the use of NN (Neural Networks), as shown in several studies on the application of the machine learning to different nonlinear issues (Bohn et al., 2013, Duriez et al., 2017, Huré et al., 2019, Jiménez et al., 2020, Kanas and Yannopoulos, 2001, Kavadi et al., 2020, Long et al.,

2016, Meuleman and Scherer, 2013, Peng and Tang, 2019, Qi, 1999, Qian et al., 2020, Ryo and Rillig, 2017, Said et al., 2020, Tang et al., 2020, Wang et al., 2015, Wu and Christofides, 2019, Wu et al., 2019b, Wu et al., 2019a).

Several machine learning algorithms have been applied to quantitative investments. LSTM (Long Short Term Memory) is a sort of RNN (Recurrent Neural Networks) that excels at managing sequence data, particularly when dealing with long-term dependencies, and is therefore ideally suited for performing prediction jobs on stock time-series data (Yu et al., 2019). GRU (Gated Recurrent Unit) networks outperform standard RNNs in sequence learning tasks and avoid the disappearing and expansion of gradients seen in classic RNNs while learning long-term dependencies (Shen et al., 2018). GAN is one of the most successful generative models, especially in generating realistic high-resolution images (Goodfellow et al., 2020). GAN enables the deep representations to be learned without extensively annotated data (Creswell et al., 2018). Since GAN is specialized in generating realistic results, therefore it could have the penitential power to generate realistic future conditions and prices (Tang et al., 2019).

Goodfellow et al. proposed generative adversarial networks, a strong class of generative models, in 2014 (Goodfellow et al., 2014). The fundamental premise of GANs is inspired by the two-player zero-sum game, in which the total gains of both players are zero, and each player's gain or loss of utility is perfectly balanced by another player's loss or gain of utility. GANs often consists of a generator and a discriminator that learn concurrently. The generator attempts to replicate the distribution of existing data samples by generating new data samples. Often, the discriminator is a binary classifier that reliably distinguishes genuine data from created ones. Both the generator and discriminator may be structured in the manner of today's most popular deep neural networks. The optimization procedure for GANs is a minimax game with the objective of reaching Nash equilibrium, at which point the generator is judged to have captured the distribution of actual samples.

GANs' achievements in image generation have been well recognized. However, its performance in stream data generation has not been widely researched. This paper

aims to explore GAN's applications and performance in generating time series data. The results of this paper may benefit the analysts and investors in quantitative investment who are willing to gain profit from the prediction of stock price.

2 Literature Review

Stock market analysts are grouped into two types based on how they analyze shares: (1) fundamentalists, who focus only on market fundamentals; and (2) chartists, who use visual analysis tools. Fundamental analysis is based on an examination of the economy, industry, and society as a whole with the goal of assessing the worth of a particular company's stock.

Fundamental analysis checks a firm's income and dividends, takes interest rate forecasts into consideration, and assesses the risk connected with the company. It employs statistical, mathematical, and financial algorithms to analyze the company's official periodic financial statements in order to determine the share price as accurately as possible (Rusu and Rusu, 2003).

The technical analysis is entirely based on the examination of the stock market's internal data, with all economic, financial, political, and psychological elements embodied in a single element: the share quotation. Technical analysts examine the short-term movements in the share price, beginning with a review of the quotes' history over a period of at least six months, and assuming that previous behavior will continue into the future. Technical analysis provides insight into the stock market's potential future development (Rusu and Rusu, 2003).

This paper will focus on technical analysis.

The dynamism of economic processes cannot be disregarded. The time series model considers a variable's previous behavior and utilizes this knowledge to forecast its future behavior. It assumes that there is no knowledge about the causal link affecting the forecasted variable.

In general, a time series model is chosen when: (1) we have little knowledge about the elements affecting the variable's behavior, (2) we have a large quantity of data, or (3) the primary objective is short-term prediction.

The time series analysis procedure begins with the construction of a data model whose attributes are comparable to those of the examined process's generation. If we assume that the model's attributes for the investigated process will persist into

the future, the model may be utilized for prediction.

Extrapolation using time series is not the same as simple extrapolation. The distinction is that time series analysis assumes that the series whose behavior must be forecasted was formed by a stochastic (random) process whose structure can be identified and characterized. In other words, a time series model elucidates the (random) process that created the time series. The description is not in terms of cause-effect relationships, as with the regression model, but rather in terms of the way the event is integrated into the process.

Time series models may be deterministic or stochastic in nature. Deterministic models are those that make no reference to the source or random fluctuation of the series. A stochastic model of a time series has more information than a deterministic model, enabling more accurate forecasting.

The conventional techniques of time series analysis presuppose the existence of four factors in the series: the trend, the cyclic component, the seasonal component, and random fluctuations. The first three components are predictable and systematic, but the fourth is a residual component that imparts stochasticity to the examined events.

For time series forecasting, a variety of forecasting approaches are available. Box and Jenkins introduced autoregressive integrated moving average (ARIMA) models for time series analysis and forecasting (Box et al., 2015). Some research has been undertaken to anticipate stock market returns using ARIMA models (Al-Shiab, 2006, Ojo and Olatayo, 2009, Adebisi et al., 2014, Mondal et al., 2014). Numerous research showed that ARIMA models generated suboptimal predictions for financial time series data (Adebisi et al., 2014, Zhang, 2003, Khandelwal et al., 2015). To account for nonlinearities caused by economic regime shifts, several researchers employed Markov regime-switching models and threshold autoregressive (TAR) models that assumed nonlinear stationary processes to forecast stock prices (Hamilton, 1989, Tong, 1990). Tsay developed a simple yet generally applicable approach for the construction of threshold autoregressive models, as well as a test for threshold nonlinearity (Tsay, 1989). Gooijer investigated regime switching in a

moving average (MA) model and used validation criteria for the selection of self-exciting threshold autoregressive (SETAR) models (Gooijer, 1998). Several empirical research indicated that as compared to linear models, SETAR delivered better outcomes (Clements and Smith, 1999, Boero and Marrocu, 2002, Boero and Marrocu, 2004, Firat, 2017).

In the late 1980s, a family of artificial intelligence (AI) models for forecasting was presented, including feedforward, backpropagation, and recurrent neural network models. Artificial neural networks (ANNs) are data-driven, nonlinear, and self-adaptive, with very few apriori assumptions. This enhances the utility and attractiveness of ANNs for predicting financial time series. Among ANN models for predicting stock market returns, the feedforward neural network with a single hidden layer has become the most popular (Zhang, 2003). Numerous studies have shown that these models provide better accurate predictions than naive and linear models (Ghiassi et al., 2005, Mostafa, 2010, Qiu et al., 2016, Aras and Kocakoç, 2016).

GAN is one kind of AI model, and this paper will focus on the GAN model.

Under the aegis of artificial intelligence, the concept of GANs fulfills the research and application needs of a wide variety of domains, providing a boost to the growth of associated fields. GANs have emerged as a popular area of study in artificial intelligence. Yann LeCun refers to adversarial networks as “the trendiest thing in machine learning in the past twenty years” in a recent presentation on unsupervised learning. Nowadays, researchers studying GANs are particularly interested in the image and vision fields. It is currently feasible to produce photorealistic photographs of objects such as birds and faces using GANs, as well as interior and outdoor settings, to translate images from one domain to another, to generate high-definition images from low-definition photos, and so on (Goodfellow, 2016). Additionally, GANs have been used in the study of other branches of artificial intelligence, such as voice and language processing (Li et al., 2017, Yu et al., 2017), virus detection (Hu and Tan, 2017), and chess game programming (Chidambaram and Qi, 2017).

Since the proposal of GAN, many variants of GAN have been proposed due to

its feature and splendid performance.

Chen et al. introduce InfoGAN, an information-theoretic extension to the Generative Adversarial Network capable of learning disentangled representations entirely unsupervised (Chen et al., 2016). To be more specific, InfoGAN effectively disentangles writing styles from digit shapes in the MNIST dataset, a stance from lighting in 3D generated pictures, and background digits from the core digit in the SVHN dataset. Additionally, it identifies visual concepts on the CelebA face dataset, such as hairstyles, the presence/absence of spectacles, and emotions. Experiments demonstrate that InfoGAN may learn interpretable representations at a rate comparable to that of current supervised approaches.

By proving that GANs gain significantly from scaling, Brock et al. present the BigGAN, a model for training Generative Adversarial Networks at the biggest size ever tried, and investigate the instabilities associated with such a scale (Brock et al., 2018). They demonstrate that by adding orthogonal regularization to the generator, it becomes amenable to a simple “truncation approach,” which offers exact control over the trade-off between sample quality and variety by reducing the generator’s input variance. These modifications provide models that set a new benchmark for class-conditional image synthesis.

Karras et al. propose *StyleGAN*, which could generate realistic faces of different styles (Karras et al., 2019). Motivated by the research on style transmission (Huang and Belongie, 2017), they redesigned the generator architecture to reveal fresh control mechanisms for the picture synthesis process. The generator begins with a learned constant input and alters the picture’s “style” at each convolution layer depending on the latent coding, thereby directly changing the strength of image characteristics at various scales. Through an automatically learned, unsupervised separation of high-level characteristics (e.g., identity and posture when trained with human faces) and stochastic variance in the resulting images, the unique architecture enables intuitive, scale-specific control of the synthesis (e.g., freckles, hair).

If both the generator and discriminator are conditioned on some additional

information y , such as class labels or input from other modalities, generative adversarial networks may be expanded to a conditional model. Conditioning may be accomplished by feeding y into both the discriminator and generator as an extra input layer. Therefore, Mirza and Osindero offer the conditional version of generative adversarial networks, which can be created by simply providing both the generator and discriminator the data, y , that they desire to condition on (Mirza and Osindero, 2014).

To perform well in time series tasks, a generative model needs to retain temporal dynamics, which means that the latter sequences should maintain the former relationships between variables throughout time. However, existing techniques for applying generative adversarial networks (GANs) to time-series data do not sufficiently account for time series data's intrinsic temporal correlations. Simultaneously, supervised models for sequence prediction are fundamentally deterministic, allowing for more exact control over network dynamics. Yoon et al. provide a novel method for synthesizing realistic time-series data that joins together the flexibility of unsupervised learning and the control power provided by supervised training (Yoon et al., 2019). They motivate the network to account for the variation of the input data during sampling by using a jointly optimal learning embedding space for supervised and adversarial purposes. They demonstrate that the framework they offer outperforms state-of-the-art benchmarks consistently and significantly on both qualitative and quantitative measures of similarity and predictive ability.

Since the development of TimeGAN, there have been some applications of it.

Large datasets are required for any machine learning activity requiring high performance, accuracy, and generalization, such as prediction or anomaly detection. However, it is not unusual for datasets to be tiny or unbalanced since data collection may be challenging, time-consuming, and costly. These problems are prevalent in the work of gathering car sensor time-series data, particularly when the vehicle exhibits anomalous behavior, and may impede the automotive industry's progress. Nord train and evaluate a TimeGAN to generate multivariate time-series data that are

comparable to sensor readings from the air pressures in the braking system of automobiles that exhibit anomalous behavior (Nord, 2021).

Medical data is seldom made publicly accessible owing to the significant costs and hazards associated with de-identification. Due to the sensitive nature of such data, access to it is strictly restricted. Synthetic medical data that retain the usefulness of actual data while maintaining privacy may be a perfect replacement for improving research. Medical data is longitudinal in nature, with a single patient experiencing several temporal events that are impacted by static factors such as age, gender, and comorbidities, among others, making it difficult to extend current time-series generative models to create medical data. TimeGAN models time-series distributions directly as a combination of static and temporal factors. It generates realistic time series by minimizing both adversarial and supervised losses concurrently. Dash et al. discovered that TimeGAN is the only time-series generative model that solves the challenge of modeling static and temporal variables concurrently, and they utilize it as a benchmark for their methods (Dash et al., 2020).

The availability of high-quality cell utilization data (CUD) is critical for accurate lithium-ion (Li-ion) battery modeling. Additionally, the model should account for nonlinear and complicated system dynamics, such as a variety of aging processes and dynamic operational characteristics. Outliers' robust, realistic synthetic CUD generation is critical for expediting the development of domain-specific technologies. By maximizing both adversarial and supervised goals while retaining the temporal correlation dynamics of multivariate sequences, time-series generative adversarial networks (TimeGAN) have been the state-of-the-art for latent space sequential data modeling. The original TimeGAN formulation uses a binary cross-entropy loss function, which results in difficulties with diminishing gradient stability throughout the training phase (Mao et al., 2017). Without taking into account the impact of outliers, the least-squares formulation overcomes this difficulty (Belagiannis et al., 2015). Chatteraj et al. explore robust loss functions for the TimeGAN architecture to generate realistic Li-ion CUD (Chatteraj et al., 2021).

Preserving temporal dynamics is critical for obtaining high-fidelity time series

data. This implies that created sequences preserve the original data's connection between variables across time. While novel forms of GANs have been utilized to produce time-series data, they, like prior GAN implementations, require a significant amount of training time. A novel federated architecture (FeGAN) is presented that combines supervised and unsupervised training to create realistic time-series data (Belagiannis et al., 2015). The framework is built on the foundations of TimeGAN and FeGAN research. TimeGAN helps the network to imitate the structure of the training data by embedding it in a learning space. Plesner proposes this framework (FeTGAN) to generate synthetic time-series data of a comparable quality to the original TimeGAN without requiring nodes in the network to provide local data qualitatively or quantitatively (Plesner, 2021). Additionally, while utilizing a single machine, there is an estimated eleven percent increase in Floating Point Operations per second and up to a thirty percent increase when using several machines.

Accurate forecasting of photovoltaic (PV) power generation is critical for developing cost-effective and dependable power dispatch plans. At the moment, PV power prediction technology is in its infancy; in particular, for forecasting instances requiring a long forecasting period and when the weather changes abruptly, forecasting accuracy is insufficient to fulfill the demand for power grid dispatching. Lia et al. present a multi-step forward forecasting model for photovoltaic (PV) power that incorporates time-series generative adversarial networks (TimeGAN), DTW-based K-medoids clustering techniques, and a mixed neural network model composed of a CNN and a GRU (Lia et al.). Due to the unequal distribution of PV power data for various weather situations and an inadequate number of training samples, it is recommended to utilize TimeGAN to augment and expand historical PV power data.

TimeGAN is similar to another GAN-based model in that it only learns from a subset of the original time series's smaller sequences. When confronted with data augmentation for time series with various seasonal patterns, such as those present in mobile telecommunication network data, this tendency has serious consequences. Dimyati evaluated the effectiveness of the TimeGAN model using Dynamic Time

Warping (DTW) and various types of RNN as its architecture to generate synthetic mobile telecommunication network data that can be used to improve the forecasting performance of statistical and deep learning models when compared to baseline models trained exclusively on the original data (Dimyati, 2021). The findings of the experiment reveal that DTW aids TimeGAN in keeping its multiple seasonal features.

Although there are many applications of TimeGAN that are proved to be effective, its performance in the financial market has been poorly explored. Whether it could still maintain its advantage in generating realistic data in the financial market is not clear. Additionally, even its application in the financial market is not well developed. More specifically, no paper has adjusted and applied the model for financial market usage, let alone evaluation. Therefore, the aim of this paper is to add custom code and adapt the model so that it can be easily used for stock prediction and to evaluate the performance of different TimeGANs

3 Data

3.1 Data Access

To establish a strong link with industrial applications, various quantization platforms basically incorporated the database's API interface, considerably improving data collection and timeliness, resulting in a well-managed data value. Bloomberg, Reuters, and other notable worldwide news companies are widely known examples. Additionally, Wind and Tushare, Choice, and Oriental Fortune are also available. Among these initiatives is the creation of a data API.

Both macroeconomic and fundamental analyses are crucial in finance. Among them are the fundamentals that individuals may easily grasp in practice. Fundamental data includes financial status, profitability, market share, operation and management system, and talent mix. Financial analysts are often tasked with determining the investment worth of a publicly traded firm with just basic data. Naturally, assessing unlisted enterprises involves just the gathering of basic financial data or news patterns, both of which might have substantial implications. Finance, too, offers a multitude of natural language processing applications.

Tushare's common interface is available at <http://tushare.org>. The majority of the data is gathered by crawlers from other websites; the data is kept on its own server. The pro interface must be registered and utilized in accordance with the assigned score level. Its primary development objective for the future is a professional interface. It is required to introduce relevant data packets and train using libraries throughout the environment setup phase.

The data in this article comes from Tushare's pro interface (<https://www.tushare.pro/>) open and free public dataset for stock market research in China, which has the advantages of abundant data, ease of use, and implementation. Using the API to get basic market data on stocks is really convenient. Figure 1 illustrates Tushare's interface functionality for providing fundamental data about publicly traded firms.

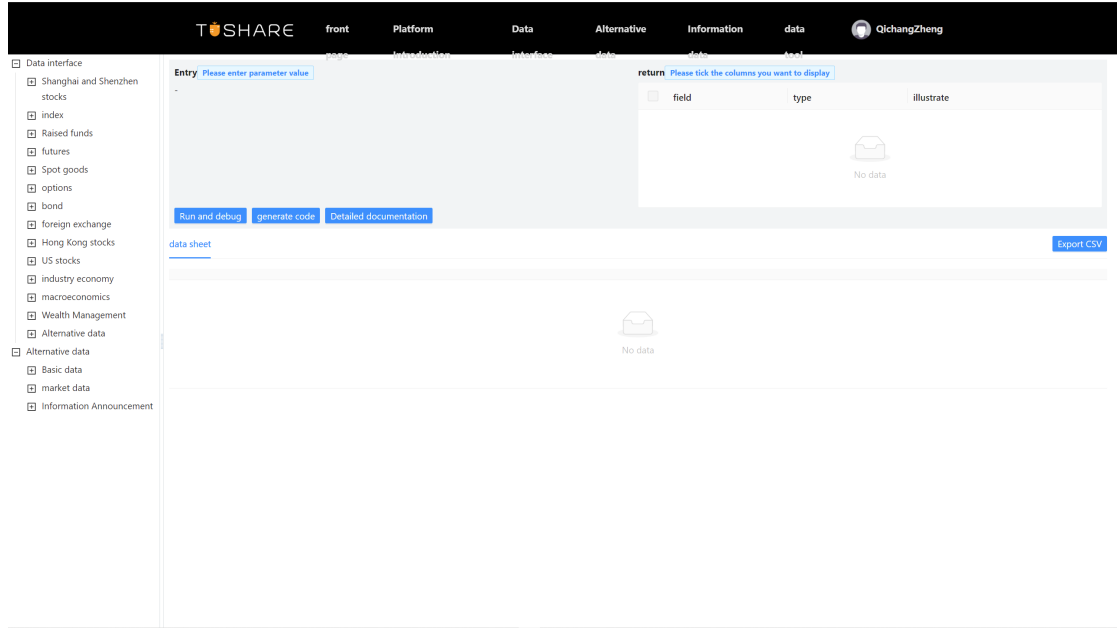


Figure 1 Tushare's interface

Stock price sequences are continuous-valued yet aperiodic; moreover, characteristics are associated. This article utilizes daily historical data for the ‘Guizhou Maotai’ stock from 2010 until 2022, including open, close, high, low, and volume. This paper uses historical data from the last ten years but does not include the most recent data. On the one hand, if we train with the most recent data, we would need to reacquire data for each training and use different data for each training, which is time-consuming and laborious. On the other hand, we believe that the past ten years of data are sufficient to reflect the characteristics of a stock, and adding the latest data does not significantly improve the performance of the model.

3.2 Data Storage

After acquiring the new data, we store them on Google Drive so that instead of reacquiring the same data from Tushare in the future, we can directly read the data already saved in the cloud. In other words, when we run the function ‘get_data(symbol)’, Python will firstly check whether there is readily available data in Google Drive. If there is, then simply read it. If not, then obtain the data through Tushare API and save it on Google Drive. After that, the program will tell you the

location of the data so that you can check it manually. Besides saving time, this operation can reduce the number of visits to the Tushare interface because each token has an upper limit on the number of visits to the interface per day, and access will be disabled when the upper limit is reached. Therefore, this operation is especially useful when debugging code. As is shown in the following figure.

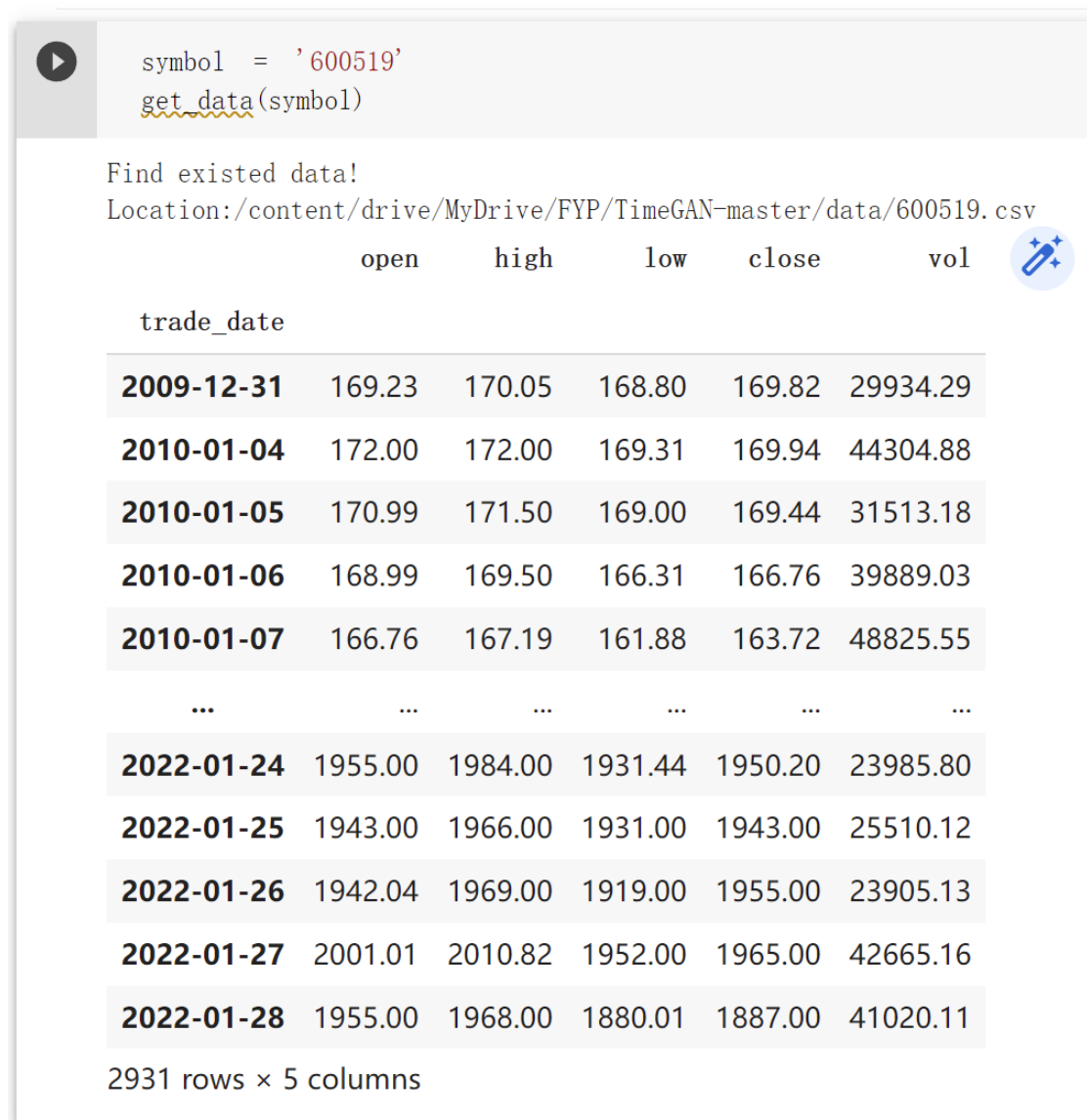


Figure 2 get_data(symbol)

3.3 Data Visualization

After obtaining the stock data, we visualize this data in Python for a quick overview through the library 'matplotlib'. We plot 5 line graphs with time as the horizontal

axis and then the open, close, high, low, and volume as the vertical axis. For ease of viewing, this paper brings these 5 line charts together.

We have implemented a very convenient function - 'view(symbol)'. Here, just enter the stock code and Python will automatically fetch and store the stock data, and then plot all the images at once. As is shown in the following figure

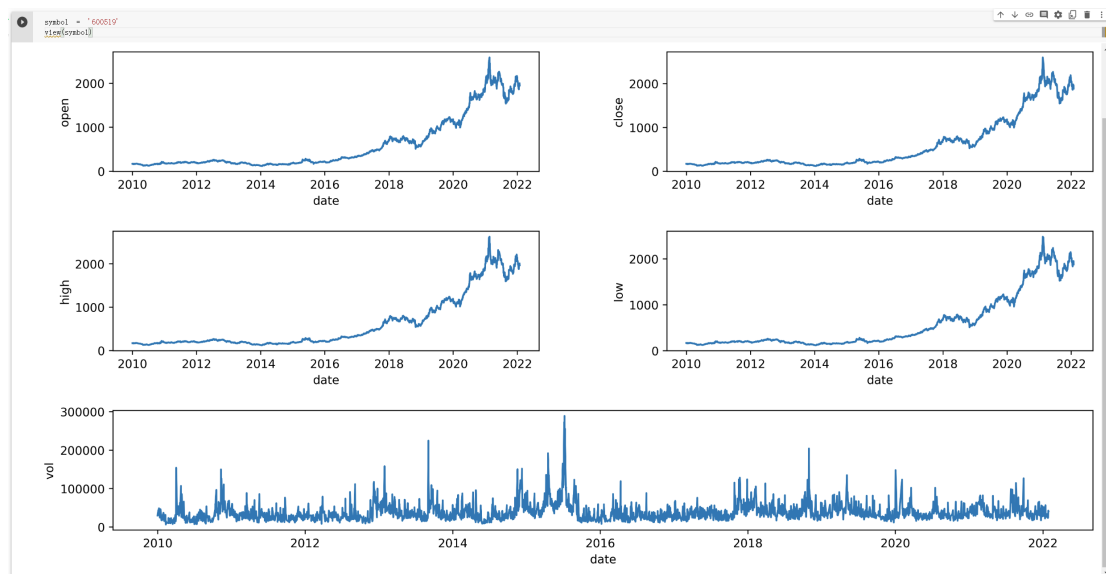


Figure 3 view(symbol)

3.4 Data Normalization

Data normalization is one of the pre-processing procedures where the data is either scaled or altered to create an equal contribution of each characteristic. The effectiveness of machine learning algorithms relies upon the quality of the data to develop a generalized prediction model of the classification issue. Consequently, the relevance of data normalization for enhancing data quality and the performance of machine learning algorithms has been highlighted in much research (Singh and Singh, 2020).

Normalization is the process of converting data to a given range, such as 0 to 1 or -1 to 1. When there are significant disparities in the ranges of distinct characteristics, normalization is necessary. This scaling strategy is advantageous when the data collection is devoid of outliers. The theoretical underpinnings of

normalization are readily apparent in Figure (1). The Normalization method used in our model is Mix Max Normalization (MMN). If it is required to cast the data to the range 0,1, then:

$$x' = \frac{x - \min}{\max - \min} \quad (1)$$

where \min and \max represent the feature's minimal and maximum values, respectively. x' denotes the normalized data and x denotes the denormalized data

If normalization is used, the denormalization procedure should be followed. More specifically, we use normalized data as input and get normalized output. Then we need to denormalize the output so to the original scale. For instance, the following equation may be used to denormalize data from the range 0 to 1:

$$x = [x' * (\max - \min)] + \min \quad (2)$$

where, \min and \max are the same values used previously in the normalization process.

Similarly, to normalize the data to the range of -1 and +1, then denormalize it:

$$x' = 2 * \left(\frac{x - \min}{\max - \min} \right) - 1 \quad (3)$$

$$x = \left[\left(\frac{x' + 1}{2} \right) (\max - \min) \right] + \min \quad (4)$$

4 Methodology

4.1 Model Overview

This paper will firstly describe the basic TimeGAN according to its proposer's paper and introduce our modifications and customized functions (Yoon et al., 2019).

Consider a generic data collection in which each instance is composed of two elements: static characteristics (that are consistent over time, such as race) and temporal features (that changes with time going, e.g., vital signs). Let \mathcal{S} signify a vector space of static characteristics, \mathcal{X} denote a vector space of temporal features, and $\mathbf{S} \in \mathcal{S}, \mathbf{X} \in \mathcal{X}$ denote random vectors that may be assigned with particular values denoted by \mathbf{s} and \mathbf{x} . Therefore, we consider representing our target data using tuples

of the type $(\mathbf{S}, \mathbf{X}_{1:T})$. Each sequence's length T is likewise a random variable, whose distribution is denoted as p by us for notational simplicity. Allowing for each sample to be indexed by $n \in \{1, \dots, N\}$ the training data, we may designate the training dataset $\mathcal{D} = \{(\mathbf{s}_n, \mathbf{x}_{n,1:T_n})\}_{n=1}^N$. Subscripts n are removed in the future unless specifically necessary.

Our objective is to use the input training data \mathcal{D} to discover a density $\hat{p}(\mathbf{S}, \mathbf{X}_{1:T})$ that is the closest approximation to $p(\mathbf{S}, \mathbf{X}_{1:T})$. This is a far-reaching objective as under the standard GAN framework, it may be hard to be optimized, mainly because it depends on the structure of data, such as dimension, length, and distribution. As a result, we decompose the joint $p(\mathbf{S}, \mathbf{X}_{1:T}) = p(\mathbf{S}) \prod_t p(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})$ with an autoregressive method to focus exclusively on the conditionals, yielding the complementary — and simpler — the objective of learning a density $\hat{p}(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})$ that best approximates $p(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})$ at any time t .

4.2 Two Objectives

Notably, this decomposes the sequence-level objective (mapping the joint distribution) into a series of incremental objectives (matching the conditionals). The first is global,

$$\min_{\hat{p}} D(p(\mathbf{S}, \mathbf{X}_{1:T}) \parallel \hat{p}(\mathbf{S}, \mathbf{X}_{1:T})) \quad (5)$$

where D is a suitable measure of how close the two distributions are. The second is a local,

$$\min_{\hat{p}} D(p(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1}) \parallel \hat{p}(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})) \quad (6)$$

irrespective of the t . Under an ideal discriminator in our GAN structure, the Jensen-Shannon divergence is employed in the former. The Kullback-Leibler divergence is applied in the latter when employing the real data as supervision under maximum-likelihood (ML) training. Notably, the former requires the presence of an ideal adversary (which we may not have), whereas the latter requires only the presence of real time series (which we really have). Thus, our objective will combine the GAN

(as shown in Expression 5) and the ML objectives (as shown in Expression 6). As will be demonstrated, this naturally results in a training part that entails the introduction of a supervised loss to support the learning of adversarial components.

4.3 Model Structure

TimeGAN is a four-component neural network that consists of an embedding part, a recovery part, a sequence generator, and the corresponding sequence discriminator. The important point is that the autoencoder (first two parts) needs to be trained concurrently with the last two parts which can be viewed as adversarial components. This operation enables TimeGAN to encode features into latent space, produce representations in latent space, decode them from latent space, and iterate through time simultaneously. The embedding function constructs the latent space, then the GAN works based on the representations inside the latent space as mentioned, and supervised loss is used to synchronize the latent variation of actual and generated data. Each will be discussed in detail.

4.3.1 Embedding and Recovery Functions

The mappings between feature and latent space are generated by the embedding and recovery procedure, which provides lower-dimensional representations for the GAN to understand the underlying temporal variation of the data better. Let $\mathcal{H}_S, \mathcal{H}_X$ be the representation in latent space corresponding to the \mathcal{S}, \mathcal{X} in feature space, respectively. Then, using autoencoder $e: \mathcal{S} \times \prod_t \mathcal{X} \rightarrow \mathcal{H}_S \times \prod_t \mathcal{H}_X$, static and temporal features are assigned to their latent codes $\mathbf{h}_S, \mathbf{h}_{1:T} = e(\mathbf{s}, \mathbf{x}_{1:T})$. We implement e in this paper using a recurrent network,

$$\mathbf{h}_S = e_S(\mathbf{s}), \mathbf{h}_t = e_X(\mathbf{h}_S, \mathbf{h}_{t-1}, \mathbf{x}_t) \quad (7)$$

where $e_S: \mathcal{S} \rightarrow \mathcal{H}_S$ denotes a static embedding network and $e_X: \mathcal{H}_S \times \mathcal{H}_X \times \mathcal{X} \rightarrow \mathcal{H}_X$ denotes a recurrent embedding network that is responsible for capturing temporal features. After that, to restore from the low-dimensional data, the recovery function $r: \mathcal{H}_S \times \prod_t \mathcal{H}_X \rightarrow \mathcal{S} \times \prod_t \mathcal{X}$ restores static and temporal representations

in latent space to their representations in feature space $\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T} = r(\mathbf{h}_s, \mathbf{h}_{1:T})$ in a reverse direction. At each step, we implement r using a feedforward network.

$$\tilde{\mathbf{s}} = r_s(\mathbf{h}_s), \tilde{\mathbf{x}}_t = r_x(\mathbf{h}_t) \quad (8)$$

where $r_s: \mathcal{H}_s \rightarrow \mathcal{S}$ is a recovery function dealing with static embeddings and $r_x: \mathcal{H}_x \rightarrow \mathcal{X}$ is a recovery network for temporal ones. Notably, except for the requirement that the autoencoder should be autoregressive and follow causal ordering (i.e., the output(s) at any stage can only be determined by previous information), it could take any form of structure. For example, the former can be implemented using temporal convolutions, while the latter can be implemented using a decoder with an attention layer.

4.3.2 Sequence Generator and Discriminator

Rather than directly generating synthetic output in feature space, the generator firstly generates output in latent space. Let $\mathcal{Z}_s, \mathcal{Z}_x$ represent vector spaces that define known distributions, and random vectors are sampled from them as input for generating into $\mathcal{H}_s, \mathcal{H}_x$. The generating function $g: \mathcal{Z}_s \times \prod_{\mathcal{Z}_x} \rightarrow \mathcal{H}_s \times \prod_t \mathcal{H}_x$ then converts a tuple of static and temporal random vectors to synthetic latent codes $\hat{\mathbf{h}}_s, \hat{\mathbf{h}}_{1:T} = g(\mathbf{z}_s, \mathbf{z}_{1:T})$. g is implemented using a recurrent network,

$$\hat{\mathbf{h}}_s = g_s(\mathbf{z}_s), \hat{\mathbf{h}}_t = g_x(\hat{\mathbf{h}}_s, \hat{\mathbf{h}}_{t-1}, \mathbf{z}_t) \quad (9)$$

where $g_s: \mathcal{Z}_s \rightarrow \mathcal{H}_s$ is a static feature generator network and $g_x: \mathcal{H}_s \times \mathcal{H}_x \times \mathcal{Z}_x \rightarrow \mathcal{H}_x$ is a temporal feature recurrent generator network. The random vector \mathbf{z}_s may be drawn from any distribution, and \mathbf{z}_t follows a stochastic process; in this case, the Gaussian distribution is used for \mathbf{z}_s and the Wiener process is used for \mathbf{z}_t . In the end, the discriminator also works with the representations in latent space. The discrimination function $d: \mathcal{H}_s \times \prod_t \mathcal{H}_x \rightarrow [0,1] \times \prod_t [0,1]$ uses the static and temporal codes as input and return classifications $\tilde{y}_s, \tilde{y}_{1:T} = d(\tilde{\mathbf{h}}_s, \tilde{\mathbf{h}}_{1:T})$. The $\tilde{\mathbf{h}}_*$ notation denotes either real (\mathbf{h}_*) or synthetic ($\hat{\mathbf{h}}_*$) embeddings; similarly, the \tilde{y}_* notation denotes classifications of either real (y_*) or synthetic (\hat{y}_*) data. Here we

implement d via a bidirectional recurrent network with a feedforward output layer,

$$\tilde{y}_S = d_S(\tilde{\mathbf{h}}_S)\tilde{y}_t = d_X(\tilde{\mathbf{u}}_t, \tilde{\mathbf{u}}_t) \quad (10)$$

where $\tilde{\mathbf{u}}_t = \tilde{c}_X(\tilde{\mathbf{h}}_S, \tilde{\mathbf{h}}_t, \tilde{\mathbf{u}}_{t-1})$ and $\tilde{\mathbf{u}}_t = \tilde{c}_X(\tilde{\mathbf{h}}_S, \tilde{\mathbf{h}}_t, \tilde{\mathbf{u}}_{t+1})$ represent the forward and backward hidden state sequences, \tilde{c}_X, \tilde{c}_X stands for the recurrent functions, and output layer classification functions are denoted by d_S, d_X . Likewise, other than the autoregressive requirement on the generator, the architecture does not suffer from any other restrictions; for the sake of simplicity, we will use a conventional recurrent formulation.

4.3.3 Jointly Learning to Encode, Generate, and Iterate

The embedding and recovery algorithms should enable reliable reconstructions $\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T}$ of the real data $\mathbf{s}, \mathbf{x}_{1:T}$ from their latent representations $\mathbf{h}_S, \mathbf{h}_{1:T}$. As a result, our primary goal function is rebuilding loss.

$$\mathcal{L}_R = \mathbb{E}_{\mathbf{s}, \mathbf{x}_{1:T} \sim p} \left[\|\mathbf{s} - \tilde{\mathbf{s}}\|_2 + \sum_t \|\mathbf{x}_t - \tilde{\mathbf{x}}_t\|_2 \right] \quad (11)$$

During training using TimeGAN, two distinct sorts of inputs are fed to the generator. To begin, in the pure open-loop mode, the autoregressive generator accepts synthetic embeddings $\hat{\mathbf{h}}_S, \hat{\mathbf{h}}_{1:t-1}$ (i.e., its own prior outputs) in order to construct the following synthetic vector \mathbf{h}_t . The unsupervised loss is then used to construct gradients. This is expected—that is, to maximize (the goal of discriminator) or minimize (the goal of the generator) the probability of delivering accurate classifications $\hat{y}_S, \hat{y}_{1:T}$ for both the training data $\mathbf{h}_S, \mathbf{h}_{1:T}$ and the generator’s synthetic output $\hat{\mathbf{h}}_S, \hat{\mathbf{h}}_{1:T}$,

$$\mathcal{L}_U = \mathbb{E}_{\mathbf{s}, \mathbf{x}_{1:T} \sim p} \left[\log y_S + \sum_t \log y_t \right] + \mathbb{E}_{\mathbf{s}, \mathbf{x}_{1:T} \sim \hat{p}} \left[\log(1 - \hat{y}_S) + \sum_t \log(1 - \hat{y}_t) \right] \quad (12)$$

Relying merely on the binary adversarial feedback from generator is not sufficient to motivate the generator to account for the data’s time-dependent conditional distributions. To improve the model efficiency, we incorporate a loss to subsequent supervised learning. Alternatively, we train in closed-loop mode, in

which the sequences of latent representations of the real data $\mathbf{h}_{1:t-1}$ (i.e., generated by the autoencoder) are fed to the generator in order to construct the next latent representation. Gradients may now be estimated on the basis of a loss that represents the distance between the original distribution $p(\mathbf{H}_t | \mathbf{H}_S, \mathbf{H}_{1:t-1})$ and generated distribution $\hat{p}(\mathbf{H}_t | \mathbf{H}_S, \mathbf{H}_{1:t-1})$. The supervised loss, which is well-known, is obtained by applying maximum likelihood,

$$\mathcal{L}_S = \mathbb{E}_{\mathbf{s}, \mathbf{x}_{1:T} \sim p} \left[\sum_t \|\mathbf{h}_t - g_{\mathcal{X}}(\mathbf{h}_S, \mathbf{h}_{t-1}, \mathbf{z}_t)\|_2 \right] \quad (13)$$

where $g_{\mathcal{X}}(\mathbf{h}_S, \mathbf{h}_{t-1}, \mathbf{z}_t)$ is a close approximation to $\mathbb{E}_{\mathbf{z}_t \sim \mathcal{N}}[\hat{p}(\mathbf{H}_t | \mathbf{H}_S, \mathbf{H}_{1:t-1}, \mathbf{z}_t)]$.

using a single sample \mathbf{z}_t —as stochastic gradient descent requires. In summary, at each step of a training iteration, we compare the real next-step latent representation (as generated by the autoencoder from raw data) to the synthetic next-step latent vector (as generated by the generator that relies on the real historical time series data of representations in latent space). While \mathcal{L}_U urges the generator to generate realistic sequences (as determined by an imperfect opponent), \mathcal{L}_S assures that equivalent stepwise transitions are produced (evaluated by real data).

4.4 Model hyperparameters

Before training the model, we need to set the hyperparameters to customize the model according to our needs, or we can use the default hyperparameters.

- | | |
|---|-------------------|
| ● Symbol: stock code | Default: ‘600519’ |
| ● module: gru, lstm, or lstmLN | Default: ‘gru’ |
| ● hidden_dim: hidden dimensions | Default: 24 |
| ● seq_len: length of each sequence | Default: 24 |
| ● num_layer: number of layers | Default: 3 |
| ● iteration: number of training iterations | Default: 2000 |
| ● batch_size: the number of samples in each batch | Default: 256 |
| ● random: whether to use randomized input data | Default: True |

Time series prediction may well be broadly defined as the process of extracting relevant information from previous data and then predicting future values. Long-range dependencies inherent in time series are often a barrier for conventional algorithms. However, Long Short-Term Memory (LSTM) methods, a subset of deep learning schemes, promise to successfully overcome the issue. Moreover, LSTMs are used to solve the issue of disappearing gradients (Hochreiter and Schmidhuber, 1997). When the time step is large, the gradient becomes either too tiny or too big, resulting in a vanishing gradient issue. This issue happens when the optimizer back propagates, causing the algorithm to execute even if the weights nearly never change (Azari, 2019). LSTMs are a kind of RNN that is particularly well suited to learning long-term dependencies where a critical component that improves the capacity of LSTMs to simulate long-term dependencies is a component termed a memory block (Hochreiter and Schmidhuber, 1997). The memory cell is responsible for remembering the neural network's temporal state, while the gates produced by multiplicative units govern the pattern of information flow. These gates are divided into input gates, output gates, and forget gates based on their practical use. Input gates regulate the amount of new information that flows into the memory cell, forget gates regulate the amount of memory cell information that remains in the current memory cell via recurrent connection, and output gates regulate the amount of information that is used to compute the memory block's output activation and then flows into the rest of the neural network (Hua et al., 2019). In other words, these gates determine whether or not data may flow through based on its priority. Additionally, the gates allow the network to learn which items to keep, which items to forget, which items to remember, which items to pay attention to, and which items to output. The cell state and concealed state are employed in collecting data for processing in the following state. As a result, the vanishing gradient may be safeguarded. (Yamak et al., 2019).

Like the LSTM, the Gated Recurrent Unit is a Recurrent Neural Network. It does, however, have a simpler structure than LSTM. It is devoid of an output gate but contains an update z and a reset r . These gates are vectors that specify which data

should be sent to the output and the reset gate specifies how the new input is to be combined with the prior memory (Yamak et al., 2019). The Update function determines how much of the previous memory to retain (Gulli and Pal, 2017).

4.5 Model Optimization

The mechanics of our method of training are shown in the following part. Let $\theta_e, \theta_r, \theta_g$, and θ_d be the networks' parameters of embedding, recovery, generator, and discriminator parts, respectively. Both the reconstruction and supervised losses are used to train the first two components,

$$\min_{\theta_e, \theta_r} (\lambda \mathcal{L}_S + \mathcal{L}_R) \quad (14)$$

where $\lambda \geq 0$ works as a weight hyperparameter of the two losses to achieve a balance. Notably, \mathcal{L}_S is used so that the embedding function has little motivation to just lower the dimensions of the latent space; it is also actively conditioned to aid the generator in deriving temporal correlations among the input. Following that, the generator and discriminator networks are trained antagonistically in the following manner,

$$\min_{\theta_g} \left(\eta \mathcal{L}_S + \max_{\theta_d} \mathcal{L}_U \right) \quad (15)$$

where $\eta \geq 0$ also works as a weight hyperparameter like λ . In other words, besides the unsupervised minimax task, the generator minimizes the supervised loss. TimeGAN is concurrently trained to disentangle raw data into latent representations, generate synthetic representations in latent space, and iterate by merging the goals in this way across time.

In reality, Yoon et al. see that TimeGAN is insensitive to λ and η through their experiments where they set $\lambda = 1$ and $\eta = 10$ (Yoon et al., 2019). Notably, although GANs are not well-known for their simplicity of training, they declare to detect no extra problems in TimeGAN. The autoencoder assists in enhancing adversarial learning, which works with lower-dimensional representation in latent space. Similarly, the guided loss constrains the generator's stepwise dynamics. For each of these reasons, Yoon et al. do not anticipate that TimeGAN will be more difficult to

train than GAN, and typical strategies for enhancing GAN training will remain relevant (Yoon et al., 2019).

Adaptive Moment Estimation (Adam) is used in our model for optimization. Adam is a widely popular deep neural network training technique that is used in a wide variety of machine learning frameworks. Kingma and Ba describe Adam, as a method for optimizing stochastic objective functions using first-order gradients and adaptive estimates of lower-order moments (Kingma and Ba, 2014). The method combines the advantages of two recently popular optimization techniques: AdaGrad’s ability to handle sparse gradients and RMSProp’s ability to handle nonstationary objectives (Kingma and Ba, 2014). The approach is simple to develop, computationally efficient, requires minimal memory, is invariant to gradient diagonal rescaling, and is ideally suited for issues with a large amount of data and/or parameters. Additionally, the method is applicable to nonstationary objectives and problems involving extremely noisy and/or sparse gradients. Adam performs well in practice and compares well to other stochastic optimization approaches, as shown by empirical data (Kingma and Ba, 2014). Adam is durable and well-suited to a broad variety of non-convex optimization problems in the area of machine learning, as Kingma and Ba discovered (Kingma and Ba, 2014).

4.6 Model saving and breakpoint training

Since GAN is a huge model containing an extremely large number of parameters, it usually requires a large amount of training to achieve better results. Therefore, we write a function to iteratively train this model named ‘train(parameters, number)’. After setting the number of iterations in the hyperparameters, running this function, the program will train the three parts of the model in turn ‘iterations’ times and then loop ‘number’ times.

As mentioned above, GAN models usually require extensive training, which can take a long time to train, often tens or hundreds of hours. However, programs often cannot run that long continuously. For example, sometimes the program

crashes due to insufficient RAM, or Google Colab, although it can provide a powerful GPU, has a limit on how long the program can run continuously. Therefore, this paper implements a breakpoint training feature. When training a model, the program will first look for whether there is an already created model based on the set hyperparameters. If there is, it reads this model directly and also gets and outputs the number of times this model has been iteratively trained. If not, then it goes to create a new model based on the hyperparameter settings and starts training. At the end of each training cycle, we save the trained model to the corresponding location for future training or direct reading of the model for prediction. In fact, these two operations are based on the same principle. When making predictions, we read the model as we have trained, except that we force the number of iterations to 0 and then input the data in time order so that the output of the model is the prediction result.

GAN models are customized, which means that for each stock, for each set of hyperparameters, we will have a unique GAN model. Therefore, when creating and saving a model, we create a folder for each stock to save the model belonging to that stock, and name it with the stock code. Within this folder, we then create different folders with the names of the hyperparameter sets to save the corresponding models. Moreover, since the models trained with or without disrupted input data have a large difference, we subdivide them again according to whether the input data is disrupted or not. The models trained by chaotic input data are stored in the 'Checkpoint_random' folder summary, and the models trained by sequential input data are stored in the 'Checkpointing_static' folder. For example, Guizhou Maotai's stock code is 600519, and then we use the default hyperparameter set, then the model storage location is '. /checkpoint_random/600519/gru_24_24_3', where 24, 24,3 represent the three hyperparameters: 'hidden_dim', 'seq_len', and 'num_layer' respectively.

4.7 Comprehension of Hyperparameters

Here this paper will briefly elaborate on the financial implications of these

hyperparameters intuitively, which is only a conjecture based on the principles of the model and for which there is no research that provides reliable evidence.

First of all, 'hidden_dim' is supposed to specify the dimension in which the data is disentangled, and the larger this dimension is, the more factors are split out of the data. For example, when `hidden_dim = 24`, a piece of data ends up being disentangled into 24 factors. For example, when the data is a picture of a human face, the factors that are split out might be eye size, eyebrow size, nose size, mouth size, pupil color, lip color, skin color, etc. Financial data is similarly disentangled, and the factors obtained from the disentanglement may help the machine to better understand the features of the data and their composition, so that it can better make predictions or perform other tasks. However, this parameter is not as large as it could be, because too many factors may confuse the machine and allow it to assign an excessive weight to unimportant factors, which is obviously not reasonable.

Next, let's look at the next hyperparameter, 'seq_len'. This hyperparameter is typically found in models that process stream data, usually time-series data. In general, this hyperparameter specifies the data of the machine when making predictions, i.e., the data that the machine can use as a reference. For example, in our model, when `seq_len=24`, the model predicts the data of a certain trading day using the data of the last 24 trading days, which makes sense in finance because financial data is time-dependent where the later data will be affected by the earlier data, which is what many good time series models take advantage of, such as AR, ARMA, ARIMA. Of course, the hyperparameter is not as large as it should be, too many inputs can also confuse the machine and make it not know which one is more important.

Finally, let's understand the hyperparameter 'num_layer'. This hyperparameter specifies the number of layers that the neural network stacks. Usually, the more layers this number is, the more abstract data the neural network can understand. For example, when `num_layer=1`, the factor of eye shape obtained from the face by deconvolution may only represent the overall curvature of the eye; when `num_layer=3`, this factor may represent Almond Eyes, Round Eyes, Monolid Eyes,

Protruding Eyes, Downturned Eyes, Upturned Eyes, Close Set Eyes, Wide Set Eyes, Deep-Set Eyes, Hooded Eyes, etc. Although more layers can help the model to understand the model more deeply, it also means more errors, and the accumulation of errors at each layer may lead to poor final results.

In a word, for these hyperparameters, too small is easy to cause the model not to understand the data, too large is easy to confuse the model, resulting in the wrong weight assignment and error, and will also lead to a significant increase in the difficulty of model training, consuming a lot of time without convergence.

5 Results and conclusions

5.1 Model Evaluation

The results of this model are sequences of predicted data. The author of this model also provides several evaluation methods.

To determine the quality of produced data, this paper adheres to three standards: (1) *diversity*—samples should be dispersed evenly throughout the genuine data set; (2) *fidelity*—the indistinguishability between samples and real data set; and (3) *usefulness*—samples should have the same performance as the real data set when employed for the same prediction objectives (i.e., train-on-synthetic and test-on-real).

5.1.1 Visualization

On both the real and generated datasets, we perform t-SNE and PCA analysis (flattening the temporal dimension).

The Stochastic Neighbor Embedding (SNE) algorithm begins by translating the high-dimensional Euclidean distances between datapoints to conditional probabilities that measure similarity. t-SNE is capable of capturing a significant portion of the local structure in high-dimensional data while also exposing global structures such as the existence of clusters at several scales. t-SNE emphasizes modeling different datapoints with high pairwise distances and modeling comparable datapoints with short pairwise distances. Van der Maaten and Hinton’s trials on a range of data sets demonstrate that t-SNE outperforms currently available state-of-the-art approaches for displaying a variety of real-world data sets (Van der Maaten and Hinton, 2008).

PCA is a way of identifying patterns of data, and expressing the data in such a way as to highlight their similarities and differences. Since patterns in data can be hard to find in data of high dimension, where the luxury of graphical representation is not available, PCA is a powerful tool for analyzing data (Smith, 2002).

This graphic depicts the distance between the distribution of produced samples and the original distribution in two-dimensional graphs, providing an evaluation of

diversity qualitatively. The two images generated by this operation describe the distance using different criteria, and the smaller this gap is, the better the prediction of our model is. Here is the example of t-SNE and PCA, respectively:

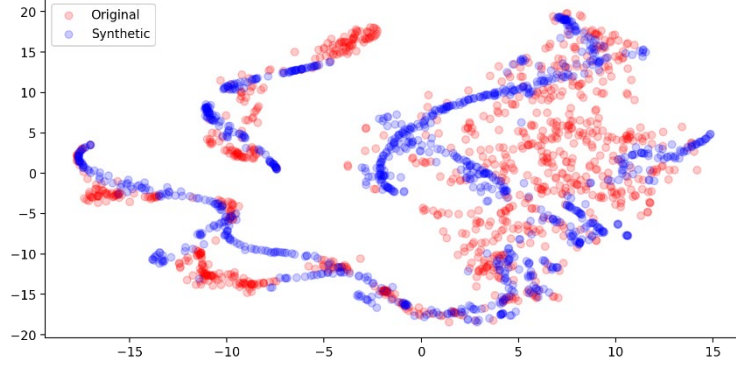


Figure 4 Example of t-SNE

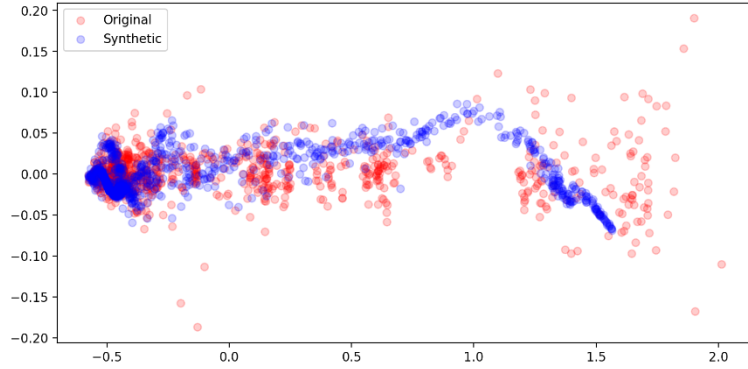


Figure 5 Example of PCA

5.1.2 Discriminative Score

This article will train an RNN model (by optimizing a two-layer LSTM) for conditional time-series classification to distinguish between original and generated time-series datasets in order to get a quantitative measure of similarity. To begin, each authentic sequence is classified as genuine, whereas each manufactured sequence is designated as not authentic. Then, using a conventional supervised task, an off-the-shelf (RNN) classifier is developed and trained to tell sequences from the two datasets from real and fake. Following that, the classification error on the held-out test set is reported, providing a measure of *fidelity* quantitatively. The score is the

accuracy of the discriminator. The lower the score, the better my generator fools the discriminator. It means that the discriminator simply cannot distinguish between my synthetic data and real data.

5.1.3 Predictive Score

To be useful, generated data must perform as well as real data in predictive tasks. We anticipate TimeGAN to be particularly adept at capturing conditional distributions throughout time. As a result, this paper trains another RNN model with two-layer LSTM for conditional sequence prediction which uses the synthetic dataset to predict next-step temporal vectors over each input sequence. The trained model is then evaluated against the original dataset. The mean absolute error (MAE) is used to quantify performance; for event-based data, we calculate the MAE as $|1 - \text{the estimated probability that the event occurred}|$. This provides a quantitative evaluation of *usefulness*. The score is the difference between the data generated by my generator and the new model. The lower the score, the better the prediction.

5.2 Training Procedure and Findings

When training the model, this paper improves the training effect by disrupting the order of the input data, and we find that the training effect is significantly better than using the original order of input. However, when performing prediction tasks after the model is trained, we want the prediction results to be time-ordered rather than random. Therefore, this paper does not randomize the input data when performing the prediction task. That is, when predicting, the program ignores the setting of the hyperparameter 'random' and forces the use of time-ordered data, or, in other words, forces 'random' = False.

We write the function code for prediction: `prediction(parameters, ori_data)`. This function allows the program to read the corresponding model based on the hyperparameter settings, and then generate a prediction based on the input data. Sometimes, the input data is the previous history of the stock, i.e.,

`'get_data(symbol)'`. More often, the input data is the stock's most recent data because investors and analysts are more interested in tomorrow's forecast results. For this reason, we write the code to get the most recent data of the stock: `get_latest_data(parameters)`. Using it as input, we can get the model's forecast for the next trading day.

We find that repeatedly cycling through the three parts of this model, with several thousand iterations each time, has better results than cycling only once and iterating tens of thousands of times, probably because the three parts of the model depend on each other and need to be jointly improved, and the deficiency of any one side will limit the effect of the other side, similar to the 'barrel effect'.

When training the model, this paper will try different combinations of hyperparameters to achieve better predictions, and we judge whether to complete the training by observing the change in the loss. When the loss no longer decreases significantly and fluctuates at a certain position, we believe that the model has been trained to a better state. This paper then evaluates the model and selects high-quality models.

5.3 Results Depiction

As mentioned earlier, this paper customizes different models by changing the hyperparameters 'module', 'seq_len', 'hidden_dim', and 'num_layer'. Due to the limitation of computing power, this paper cannot train a large number of models in a short time. Therefore, this paper restricts the choice of 'module' to 'gru' and 'lstm', 'seq_len' and 'hidden_dim' to 6, 12, 24, and 48, and let them change simultaneously, and restricts 'num_layer' to 3 and 6. In this way, we get $2 * 4 * 2 = 16$ models. This paper trains these models separately for evaluation. The results of each model evaluation are two images, t-SNE and PCA, and two scores, the discriminative score and the predictive score, respectively. To facilitate the evaluation and comparison of the results of different models, we put these four evaluation results together as a set of evaluation results, which means that we will get 12 sets of evaluation results. The

following figure shows the evaluation of the prediction results of different models trained with the stock data of ‘Guizhou Maotai’.

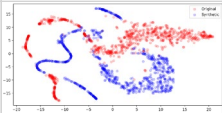
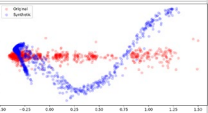
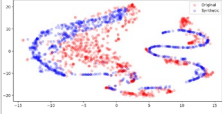
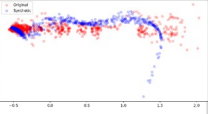
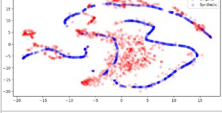
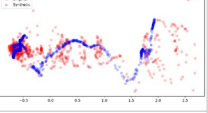
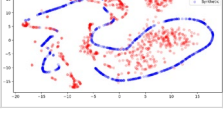
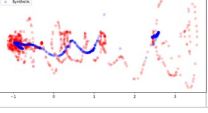
	t-SNE	PCA	Discriminative Score	Predictive Score
gru_6_6_3			0.2345	0.057
gru_12_12_3			0.142	0.0555
gru_24_24_3			0.0741	0.0561
gru_48_48_3			0.2196	0.0548

Figure 6 Result part 1

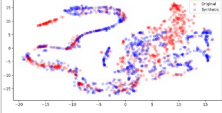
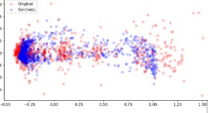
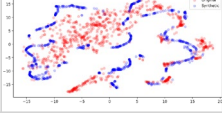
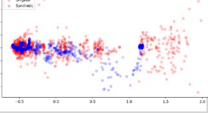
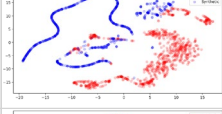
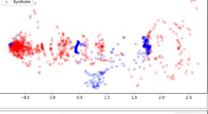
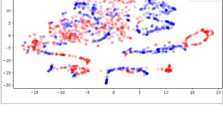
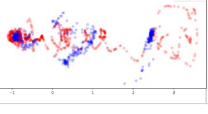
gru_6_6_6			0.1075	0.0534
gru_12_12_6			0.0253	0.0539
gru_24_24_6			0.3302	0.0577
gru_48_48_6			0.1887	0.0535

Figure 7 Result part 2

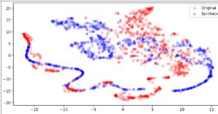
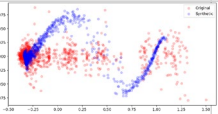
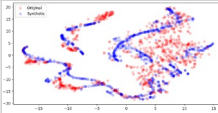
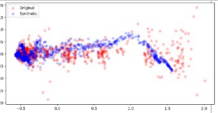
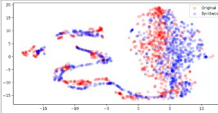
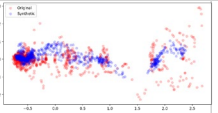
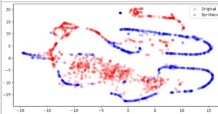
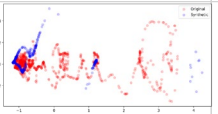
	t-SNE	PCA	Discriminative Score	Predictive Score
lstm_6_6_3			0.1152	0.0587
lstm_12_12_3			0.0704	0.0526
lstm_24_24_3			0.007	0.0528
lstm_48_48_3			0.2697	0.0653

Figure 8 Result part 3

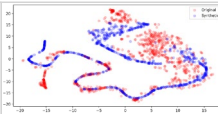
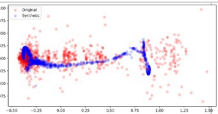
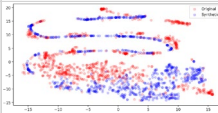
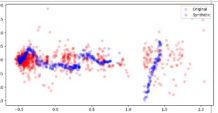
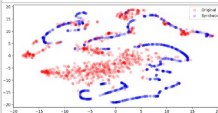
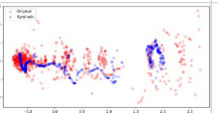
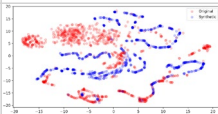
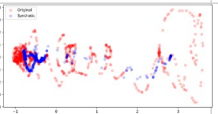
lstm_6_6_6			0.1186	0.0611
lstm_12_12_6			0.1062	0.0538
lstm_24_24_6			0.0907	0.053
lstm_48_48_6			0.2433	0.0545

Figure 9 Result part 4

Note that we find that the discrimination scores of the model fluctuated greatly, so we paid more attention to the other three metrics when evaluating the model. We see that the lstm models perform better than the grus. Although larger and more complex deep learning models tend to imply better results, in our experiments, blindly increasing a certain hyperparameter only gives worse results because overly complex models are usually difficult to train and converge, which leads to poorer results in one part of our model and thus affects the overall results as mentioned earlier.

By looking through the results more carefully, we find that the model becomes less effective when ‘hidden_dim’ and ‘seq_len’ change to the larger one, 48. Moreover, when these two hyperparameters are small, namely 6, increasing ‘num_layers’ would get better models. The reason may be, as this paper mentioned before, that larger hyperparameters could enhance the model’s comprehension of data, while the overly complex parts of the model may be difficult to train and converge, thus affecting the effectiveness of the overall model.

The output of this model is multiple time series segments with the length of the segments equal to the hyperparameter ‘seq_len’. This is because the model generates the next predicted data based on the previous real data in each segment. It is these time series segments that this paper uses in the evaluation of the model. What we usually need is the last data of the last segment, which is the data of the latter time point of the input time-series data. For example, if we use daily stock data in this paper, then the last data of the model output result is the data of the next trading day predicted by the model. This can be another method of evaluation, but in this paper we focus on the 3 evaluation methods mentioned above. This paper suggests that subsequent research should pay attention to the design of the discriminator and not overpower it, limit its performance, or weaken it appropriately as training proceeds.

5.4 Potential problem

We found that the discriminative score in model evaluation not only fluctuates greatly, but also when the model undergoes a lot of training, usually after more than 100,000 iterations, this metric increases significantly, even reaching 0.4 or more, and the performance of the model does not improve significantly according to the other three evaluation metrics, which may be due to the fact that during the process of model training discriminator becomes too powerful, causing the generator to increasingly come to deceive the discriminator. This problem was also found by other researchers in the GAN model. For example, Lin et al. found that the discriminator is generally excessively powerful relative to the generator, and the

generator may acquire restricted gradients from the discriminator and therefore is difficult to converge (Lin et al., 2021). Therefore, this paper

5.5 Subsequent optimization

First, as mentioned earlier, due to time and computing power constraints, this paper has only trained and evaluated 16 models as a reference, and what we have selected are probably not the best model, so this paper suggests that it is necessary to customize more models for evaluation to select the best model. In addition, we found that when evaluating model performance, there was variation in the results for the same model repeatedly, especially the discriminative score, which may have led to the fact that this paper does not evaluate the model accurately enough to select the best model. Therefore, this paper believes that subsequent evaluations should improve the stability of the evaluation results, such as averaging multiple evaluations or even using the mean and variance of multiple evaluations as a new metric.

Other methods have been tried to improve the model performance. For example, this paper adjusts the calculation of the model loss function. Because this model has four parts and multiple loss functions for each part. Therefore, by adjusting the formula of the loss function, we can change the importance of a certain part in the model training and optimize the whole model by improving the performance of a certain part of the model. Of course, this can also lead to a part of the model not being fully optimized, thus pulling down the performance of the whole model. As mentioned earlier, the generator of the GAN model is the concentration, because ultimately what we need is to generate data with the generator of GAN, and the discriminator of GAN is only used to help the generator to train. So we need to focus on training the generator.

In fact, by carefully reading the source code, this paper finds that the authors share the same view, and we find that they manually scale the loss of the generator directly by a factor of 100, and the other three parts of the model also use this loss directly or indirectly. Therefore we can view it as focusing on the generator part

when training the model account for the generator part when training other parts. Based on this, we modified the loss function formula of the model, mainly by linear scaling up and down. However, we did not find that the performance of the model underwent a significant improvement as a result, but this might be due to our too few experiments.

Another approach is to adjust the loss function settings of the model. The commonly used loss functions are ReLU (Rectified Linear Unit), Leaky ReLU, Concatenated ReLU, Parametric ReLU, Sigmoid, Tanh (Hyperbolic Tangent), ELU (Exponential Linear Unit), Scaled ELU, GELU (Gaussian Error Linear Units). da S Gomes et al. conducted an experiment to compare the performance of different activation functions in a time series model and they showed that in bigger networks, the models with logsig, tanh, and rootsig activation functions performed better (da S Gomes et al., 2011).

In summary, this paper believes that future optimization for this model could consider adjusting the formulation of the model loss function, including linear and nonlinear scaling up and down, as well as considering different sigmoid-like and tanh loss functions, or using appropriate loss functions based on our unique autoencoder.

Word count: 9557

6 Acknowledgements

We would appreciate reviewers for their important comments. The source code of TimeGAN is supported by Yoon et al. from <https://github.com/jsyoon0823/timeGAN>.

Much enhancement has been done due to supervisor Prof. Jia Zhai's advice.

7 Appendix

Source code and other materials can be accessed at:

<https://drive.google.com/drive/folders/1ef43ShMpUWhAS1SNBb7vMzKHYbUVU-o4?usp=sharing>.

Here we look at the integration of an autoencoder in the GAN framework. An autoencoder structure is composed of two components: an encoder that compresses data x into latent variable z ; and a decoder that reconstructs encoded data to its original form. This structure is suitable for stabilizing GAN since it learns the posterior distribution $p(z | x)$ in order to reconstruct data x , which mitigates mode collapse caused by GAN's inability to map data x to z . Additionally, an autoencoder may facilitate abstract operations by learning a latent representation of a complicated, high-dimensional data space using an encoder $X | Z$, where X and Z signify the data space and the latent space, respectively. Learning a latent representation may enable sophisticated data space modifications through interpolation or conditional concatenation in the latent space (Alqahtani et al., 2021).

A pair of data and related characteristics, such as the data's class label, are required for supervised algorithms, and the properties are often utilized as an extra input vector. By adding an information vector c to the generator and discriminator, conditional GAN (CGAN) adds an extra requirement such as a class label to oversee the data production process (Mirza and Osindero, 2014). The generator accepts not just a latent vector z but also an extra information vector c , and the discriminator accepts samples and the additional information vector c in order to discern between genuine and bogus samples given c . Thus, CGAN is able to regulate the number of digits produced, which is not achievable with normal GAN. Additionally, plug-and-play generative networks (PPGNs) are a sort of generative model that generates data in response to a specified condition (Nguyen et al., 2017).

In contrast to the supervised approaches outlined above, unsupervised methods make no use of labeled data. As a result, they need a separate technique to decouple significant characteristics from the latent space. To capture the prominent semantic

aspects of actual samples, InfoGAN decomposes an input noise vector into a conventional incompressible latent vector z and another latent variable c . Then, InfoGAN optimizes the mutual information between c and a produced sample $G(z; c)$ in order for c to capture certain salient properties of actual data. In other words, the generator concatenates the inputs $(z; c)$ and optimizes the mutual information, $I(c; G(z; c))$, between a given latent code c and the produced samples $G(z; c)$, in order to develop meaningful feature representations. However, calculating mutual information $I(c; G(z; c))$ requires estimating the posterior probability $p(c | x)$ explicitly, which is difficult. Thus, InfoGAN employs a variational method, maximizing a lower limit in lieu of a goal value $I(c; G(z; c))$ (Alqahtani et al., 2021).

8 Reference

- ADEBIYI, A. A., ADEWUMI, A. O. & AYO, C. K. 2014. Comparison of ARIMA and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014.
- AL - SHIAB, M. 2006. The predictability of the Amman stock exchange using the univariate autoregressive integrated moving average (ARIMA) model. *Journal of Economic and Administrative Sciences*.
- ALQAHTANI, H., KAVAKLI-THORNE, M. & KUMAR, G. 2021. Applications of generative adversarial networks (gans): An updated review. *Archives of Computational Methods in Engineering*, 28, 525-552.
- ARAS, S. & KOCAKOÇ, İ. D. 2016. A new model selection strategy in time series forecasting with artificial neural networks: IHTS. *Neurocomputing*, 174, 974-987.
- AZARI, A. 2019. Bitcoin price prediction: An ARIMA approach. *arXiv preprint arXiv:1904.05315*.
- BELAGIANNIS, V., RUPPRECHT, C., CARNEIRO, G. & NAVAB, N. Robust optimization for deep regression. Proceedings of the IEEE international conference on computer vision, 2015. 2830-2838.
- BOERO, G. & MARROCU, E. 2002. The performance of non - linear exchange rate models: a forecasting comparison. *Journal of Forecasting*, 21, 513-542.
- BOERO, G. & MARROCU, E. 2004. The performance of SETAR models: a regime conditional evaluation of point, interval and density forecasts. *International Journal of Forecasting*, 20, 305-320.
- BOHN, B., GARCKE, J., IZA-TERAN, R., PAPROTNY, A., PEHERSTORFER, B., SCHEPSMEIER, U. & THOLE, C.-A. 2013. Analysis of car crash simulation data with nonlinear machine learning methods. *Procedia Computer Science*, 18, 621-630.
- BOX, G. E., JENKINS, G. M., REINSEL, G. C. & LJUNG, G. M. 2015. *Time series analysis: forecasting and control*, John Wiley & Sons.
- BROCK, A., DONAHUE, J. & SIMONYAN, K. 2018. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*.
- BülBül, H., ÖMÜRBEK, N., PAKSOY, T. & BEKTAŞ, T. 2013. An empirical investigation of advanced manufacturing technology investment patterns: Evidence from a developing country. *Journal of Engineering and Technology Management*, 30, 136-156.
- CHATTORAJ, S., PRATIHAR, S., PRATIHAR, S. & KONIK, H. Improving Stability of Adversarial Li-ion Cell Usage Data Generation using Generative Latent Space Modelling. ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021. IEEE, 8047-8051.
- CHEN, X., DUAN, Y., HOUTHOOFT, R., SCHULMAN, J., SUTSKEVER, I. & ABBEEL, P. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29.
- CHIDAMBARAM, M. & QI, Y. 2017. Style transfer generative adversarial networks: Learning to play chess differently. *arXiv preprint arXiv:1702.06762*.
- CLEMENTS, M. P. & SMITH, J. 1999. A Monte Carlo study of the forecasting performance of empirical SETAR models. *Journal of Applied Econometrics*, 14, 123-141.
- CRESWELL, A., WHITE, T., DUMOULIN, V., ARULKUMARAN, K., SENGUPTA, B. & BHARATH, A. A. 2018. Generative adversarial networks: An overview. *IEEE Signal*

- Processing Magazine*, 35, 53-65.
- DA S GOMES, G. S., LUDERMIR, T. B. & LIMA, L. M. 2011. Comparison of new activation functions in neural network for forecasting financial time series. *Neural Computing and Applications*, 20, 417-439.
- DASH, S., YALE, A., GUYON, I. & BENNETT, K. P. Medical time-series data generation using generative adversarial networks. *International Conference on Artificial Intelligence in Medicine*, 2020. Springer, 382-391.
- DE HAAN, L. & KAKES, J. 2011. Momentum or contrarian investment strategies: evidence from Dutch institutional investors. *Journal of Banking & Finance*, 35, 2245-2251.
- DIMYATI, H. 2021. Time-series Generative Adversarial Networks for Telecommunications Data Augmentation.
- DURIEZ, T., BRUNTON, S. L. & NOACK, B. R. 2017. *Machine learning control-taming nonlinear dynamics and turbulence*, Springer.
- FIRAT, E. 2017. SETAR (self-exciting threshold autoregressive) non-linear currency Modelling in EUR/USD, EUR/TRY and USD/TRY parities. *Mathematics and Statistics*, 5, 33-55.
- GHIASSI, M., SAIDANE, H. & ZIMBRA, D. 2005. A dynamic artificial neural network model for forecasting time series events. *International Journal of Forecasting*, 21, 341-362.
- GOODFELLOW, I. 2016. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*.
- GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A. & BENGIO, Y. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.
- GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A. & BENGIO, Y. 2020. Generative adversarial networks. *Communications of the ACM*, 63, 139-144.
- GOOIJER, J. D. 1998. On threshold moving - average models. *Journal of Time Series Analysis*, 19, 1-18.
- GULLI, A. & PAL, S. 2017. *Deep learning with Keras*, Packt Publishing Ltd.
- HAMILTON, J. D. 1989. A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica: Journal of the econometric society*, 357-384.
- HOCHREITER, S. & SCHMIDHUBER, J. 1997. Long short-term memory. *Neural computation*, 9, 1735-1780.
- HU, W. & TAN, Y. 2017. Generating adversarial malware examples for black-box attacks based on GAN. *arXiv preprint arXiv:1702.05983*.
- HUA, Y., ZHAO, Z., LI, R., CHEN, X., LIU, Z. & ZHANG, H. 2019. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 57, 114-119.
- HUANG, X. & BELONGIE, S. Arbitrary style transfer in real-time with adaptive instance normalization. *Proceedings of the IEEE international conference on computer vision*, 2017. 1501-1510.
- HURÉ, C., PHAM, H. & WARIN, X. 2019. Some machine learning schemes for high-dimensional nonlinear PDEs. *arXiv preprint arXiv:1902.01599*, 2.
- JI, X., ZHANG, Y., MIRZA, N., UMAR, M. & RIZVI, S. K. A. 2021. The impact of carbon neutrality on the investment performance: Evidence from the equity mutual funds in BRICS. *Journal of Environmental Management*, 297, 113228.

- JIMÉNEZ, A. A., ZHANG, L., MUÑOZ, C. Q. G. & MÁRQUEZ, F. P. G. 2020. Maintenance management based on Machine Learning and nonlinear features in wind turbines. *Renewable Energy*, 146, 316-328.
- KANAS, A. & YANNOPOULOS, A. 2001. Comparing linear and nonlinear forecasts for stock returns. *International Review of Economics & Finance*, 10, 383-398.
- KARRAS, T., LAINE, S. & AILA, T. A style-based generator architecture for generative adversarial networks. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019. 4401-4410.
- KAVADI, D. P., PATAN, R., RAMACHANDRAN, M. & GANDOMI, A. H. 2020. Partial derivative nonlinear global pandemic machine learning prediction of covid 19. *Chaos, Solitons & Fractals*, 139, 110056.
- KHANDELWAL, I., ADHIKARI, R. & VERMA, G. 2015. Time series forecasting using hybrid ARIMA and ANN models based on DWT decomposition. *Procedia Computer Science*, 48, 173-179.
- KINGMA, D. P. & BA, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- LI, J., MONROE, W., SHI, T., JEAN, S., RITTER, A. & JURAFSKY, D. 2017. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*.
- LIA, Q., ZHANG, X., MAB, T., LIUD, D., WANG, H. & HUA, W. Multi-step ahead photovoltaic power forecasting model based on TimeGAN, Soft DTW-based K-medoids clustering, and a CNN-GRU hybrid neural network.
- LIN, Y., WANG, Y., LI, Y., GAO, Y., WANG, Z. & KHAN, L. Attention-based spatial guidance for image-to-image translation. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021. 816-825.
- LONG, A. W., PHILLIPS, C. L., JANKOWSKI, E. & FERGUSON, A. L. 2016. Nonlinear machine learning and design of reconfigurable digital colloids. *Soft Matter*, 12, 7119-7135.
- MAO, X., LI, Q., XIE, H., LAU, R. Y., WANG, Z. & PAUL SMOLLEY, S. Least squares generative adversarial networks. Proceedings of the IEEE international conference on computer vision, 2017. 2794-2802.
- MCALLISTER, P. 1999. Globalization, integration and commercial property. Evidence from the UK. *Journal of Property Investment & Finance*.
- MEULEMAN, B. & SCHERER, K. R. 2013. Nonlinear appraisal modeling: An application of machine learning to the study of emotion production. *IEEE Transactions on Affective Computing*, 4, 398-411.
- MEYER, K. E. & NGUYEN, H. V. 2005. Foreign investment strategies and sub - national institutions in emerging markets: Evidence from Vietnam. *Journal of management studies*, 42, 63-93.
- MIRZA, M. & OSINDERO, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.
- MONDAL, P., SHIT, L. & GOSWAMI, S. 2014. Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*, 4, 13.
- MOSTAFA, M. M. 2010. Forecasting stock exchange movements using neural networks: empirical evidence from Kuwait. *Expert systems with applications*, 37, 6302-6309.
- NGUYEN, A., CLUNE, J., BENGIO, Y., DOSOVITSKIY, A. & YOSINSKI, J. Plug & play

- generative networks: Conditional iterative generation of images in latent space. Proceedings of the IEEE conference on computer vision and pattern recognition, 2017. 4467-4477.
- NORD, S. 2021. Multivariate Time Series Data Generation using Generative Adversarial Networks: Generating Realistic Sensor Time Series Data of Vehicles with an Abnormal Behaviour using TimeGAN.
- OJO, J. & OLATAYO, T. 2009. On the estimation and performance of subset autoregressive integrated moving average models. *European Journal of Scientific Research*, 28, 287-293.
- PARK, C. H. & IRWIN, S. H. 2007. What do we know about the profitability of technical analysis? *Journal of Economic surveys*, 21, 786-826.
- PENG, J. & TANG, Q. Application of NARX Dynamic Neural Network in Quantitative Investment Forecasting System. International Symposium on Intelligence Computation and Applications, 2019. Springer, 628-635.
- PLESNER, M. 2021. FeTGAN: Federated Time-Series Generative Adversarial Network.
- QI, M. 1999. Nonlinear predictability of stock returns using financial and economic variables. *Journal of Business & Economic Statistics*, 17, 419-429.
- QIAN, E., KRAMER, B., PEHERSTORFER, B. & WILLCOX, K. 2020. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406, 132401.
- QIU, M., SONG, Y. & AKAGI, F. 2016. Application of artificial neural network for the prediction of stock market returns: The case of the Japanese stock market. *Chaos, Solitons & Fractals*, 85, 1-7.
- RUSU, V. & RUSU, C. 2003. Forecasting methods and stock market analysis. *Creative Math*, 12, 103-110.
- RYO, M. & RILLIG, M. C. 2017. Statistically reinforced machine learning for nonlinear patterns and variable interactions. *Ecosphere*, 8, e01976.
- SAID, M., ABDELLAFOU, K. B. & TAOUALI, O. 2020. Machine learning technique for data-driven fault detection of nonlinear processes. *Journal of Intelligent Manufacturing*, 31, 865-884.
- SHEN, G., TAN, Q., ZHANG, H., ZENG, P. & XU, J. 2018. Deep learning with gated recurrent unit networks for financial sequence predictions. *Procedia computer science*, 131, 895-903.
- SINGH, D. & SINGH, B. 2020. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97, 105524.
- SMITH, L. I. 2002. A tutorial on principal components analysis.
- TANG, C., ZHU, W. & YU, X. 2019. Deep hierarchical strategy model for multi-source driven quantitative investment. *IEEE Access*, 7, 79331-79336.
- TANG, Y., KURTHS, J., LIN, W., OTT, E. & KOCAREV, L. 2020. Introduction to Focus Issue: When machine learning meets complex systems: Networks, chaos, and nonlinear dynamics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30, 063151.
- TEPLOVA, T., SOKOLOVA, T., FASANO, A. & RODINA, V. 2020. Determinants of return rates of Russian equity and bond mutual funds: Active investment strategies and commissions. *Voprosy Ekonomiki*, 40-60.
- TONG, H. 1990. *Non-linear time series: a dynamical system approach*, Oxford university press.
- TSAY, R. S. 1989. Testing and modeling threshold autoregressive processes. *Journal of the American statistical association*, 84, 231-240.
- VAN DER MAATEN, L. & HINTON, G. 2008. Visualizing data using t-SNE. *Journal of machine*

learning research, 9.

- VAN LANCKER, W. 2013. Putting the child-centred investment strategy to the test: evidence for the EU27. *European Journal of Social Security*, 15, 4-27.
- WANG, D., ZHANG, M., LI, Z., CUI, Y., LIU, J., YANG, Y. & WANG, H. Nonlinear decision boundary created by a machine learning-based classifier to mitigate nonlinear phase noise. 2015 European Conference on Optical Communication (ECOC), 2015. IEEE, 1-3.
- WANG, H., LIU, Y., XIONG, W. & SONG, J. 2019. The moderating role of governance environment on the relationship between risk allocation and private investment in PPP markets: Evidence from developing countries. *International Journal of Project Management*, 37, 117-130.
- WU, Z. & CHRISTOFIDES, P. D. 2019. Economic machine-learning-based predictive control of nonlinear systems. *Mathematics*, 7, 494.
- WU, Z., TRAN, A., RINCON, D. & CHRISTOFIDES, P. D. 2019a. Machine - learning - based predictive control of nonlinear processes. Part II: Computational implementation. *AIChE Journal*, 65, e16734.
- WU, Z., TRAN, A., RINCON, D. & CHRISTOFIDES, P. D. 2019b. Machine learning - based predictive control of nonlinear processes. Part I: Theory. *AIChE Journal*, 65, e16729.
- YAMAK, P. T., YUJIAN, L. & GADOSEY, P. K. A comparison between arima, lstm, and gru for time series forecasting. Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence, 2019. 49-55.
- YOON, J., JARRETT, D. & VAN DER SCHAAR, M. 2019. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 32.
- YU, L., ZHANG, W., WANG, J. & YU, Y. Seqgan: Sequence generative adversarial nets with policy gradient. Proceedings of the AAAI conference on artificial intelligence, 2017.
- YU, Y., SI, X., HU, C. & ZHANG, J. 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, 31, 1235-1270.
- ZHANG, G. P. 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50, 159-175.
- ZHANG, K. Research on Quantitative Investment Based on Machine Learning. 2020 2nd International Conference on Economic Management and Cultural Industry (ICEMCI2020), 2020. Atlantis Press, 245-249.