University of Tennessee, Knoxville

RECSEM REU 2024 Final Paper

Time Series Generative Modeling For Algorithmic Trading: Strategic Optimization Of GAN For Intraday Trading In The Stock Market

Lenox Baloglou
Wong Hoi Fai (Gordon)

Supervised by
Dr. Kwai Wong

August 2, 2024

1 Abstract

Day trading (also referred to as intraday trading) within the stock market refers to the process of buying and selling assets within a single trading day, and holding nothing overnight. This is to avoid any social and political factors unrelated to the actual stock market state affecting the prices too much. Trading only during open hours allows for a more predictable environment for analysis. As the data associated with this process is significantly more volatile than that associated to inter-day trading, understanding patterns is crucial to the success of the trader.

Building off of previous work that generates artificial data using TimeGAN, we will tailor the current code to align with the most important aspects of intraday trading. To test our results, we will use the TSGBench model that is currently available and fit the code with an extra parameter to quantify the speed, precision, and accuracy of our newly generated day trading data. We will then create an interface to combine the two models. Our ultimate goal is to be able to forecast stock market features to inform better day trading decisions, which need to be made at a moment's notice.

2 Background studies

2.1 Day Trading in the US

In the New York Stock Exchange, day trading refers to the business of buying and selling all assets in the stock market within open hours. This is between 9:30 am ET and 4:00 pm ET. Intraday trading offers many benefits including potential for high earnings and returns, immediate feedback, and no associated risk that comes from holding assets overnight. However, some of the downfalls include a highly volatile market, fast paced decision making, and a stressful environment.

The main feature of day trading is that decisions are made by the minute or even by the second. The prices are changing at the same rate and thus the chance of loss is high. Intraday trading is a zero-sum game, each trader's loss is another trader's gain. Overall, day trading differs greatly from standard trading and thus different features are more important to consider when observing and generating data. [1]

2.2 Time Series Synthesizer

Time Series Data is a sequence of data collected over a period of time, at regular intervals, and ordered in sequential time steps. There are 3 major components to the time series data that is required for analysis or predictions: the trend component, the seasonal component, and the residual component.

The trend component shows the long term movement of the data, whether it is climbing, declining, or maintaining at a steady level. The seasonal component shows repeating patterns at regular intervals. The residual component shows the unpredictable noise and irregularities. It is important to distinguish between the three to create realistic synthesized data. One of

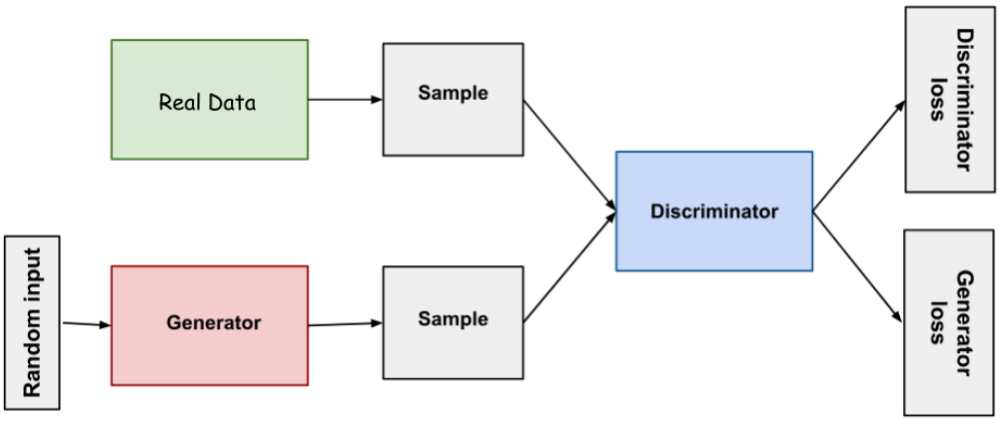

Figure 1: General GAN setup [2]

the most important tasks is to separate the residual component from the rest as it is irregular and disturbing the pattern coming from the other two components. [3] [4]

2.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) consist of pairs of generators and discriminators. The generator attempts to produce data that closely resembles raw data to fool the discriminator, while the discriminator tries to differentiate the samples the generator produces and samples of raw data. The competition between the two parts is a zero-sum game, where when one side wins the other loses.

As one component gets feedback from the other, the generator gets better at creating fake data that mimics the real data, and the discriminator gets better at separating fake data from the real. As the discriminator scores more harshly, the generator has to follow with more realistic data generation. Once the training process is complete, the generator can be used as a reliable time series synthesizer as it withstood the harsh scoring from the discriminator.

2.4 TimeGAN

The term TimeGAN stands for Time-series Generative Adversarial Networks [5]. This refers to the process of generating synthetic data for an application where the data is changing with respect to time. This process can be applied to the stock market to generate sample data and potentially forecast future prices. For some financial applications, raw data cannot be distributed to the public due to safety and privacy concerns, making the existence of a generative tool prudent.

The GAN model is based off of two types of features or variables: static features, which do not change with time, and temporal features, which do change with time. An example of a static feature is the name of a company selling stocks and an example of a temporal feature is the price of the stock. In or-

der to generate any artificial data, we must start with some quantities and train them to resemble real data. The starting information for the TimeGAN is a set of tuples with different combinations of static and temporal features that index along until the data resembles reality.

Figure 1 shows the basic format that each Generative Adversarial Network (GAN) takes. The inputs come in two forms: real images and random input or noise. The random vectors and inputs are then run through the generator which turns them into a sample fit to be fed into the discriminator. The sample from the real images also goes through the discriminator at this point. The discriminator then scores the data and this score improves the generator and itself.

2.5 Other forms of GAN

Other types of GANs were also relevant as we looked into optimization for day trading purposes. Score-based generative models, also known as SGMs or diffusion models, operate similarly to TimeGANs. However, each generation is based off of the 'score' of the previous one. SGMs have three main components: the encoder, decoder, and conditional score network, which allows for faster and more accurate improvements between generations. [6]

Common Source Coordinated GAN (COSCI-GAN) runs the tuples through multiple different channels of generators to accommodate multiple static features and their differences. We have chosen not to use COSCI-GAN for our project, due to its focus on classification as opposed to forecasting [7]. Each model has its own advantages and downfalls, depending upon its application. [8]

2.6 Why TimeGAN

For the ultimate goal of this project, we chose to work with TimeGAN as opposed to the other types of GANs available, such as COSCI-GAN or diffusion models.

The choice was made because TimeGAN has a focus on representation and temporal features, which are prominent in the day trading world. Temporal features are the most volatile aspect in intraday trading, so choosing a GAN that accentuates this was critical.

The COSCI-GAN model specializes in classification [7], which is not as important when working with data that is moving at such a high speed. Diffusion models were also a strong contender, but the models are mainly used for image generation, not large sets of time series data. Thus, after considering these three types of GANs, we decided to proceed with the available TimeGAN model. [10]

The included Figure 2 shows how the TimeGAN model uses the general GAN components of a generator and a discriminator to produce high quality artificial data with temporal features using the input of real sequences and random vectors. The additional steps shown here present the reconstruction

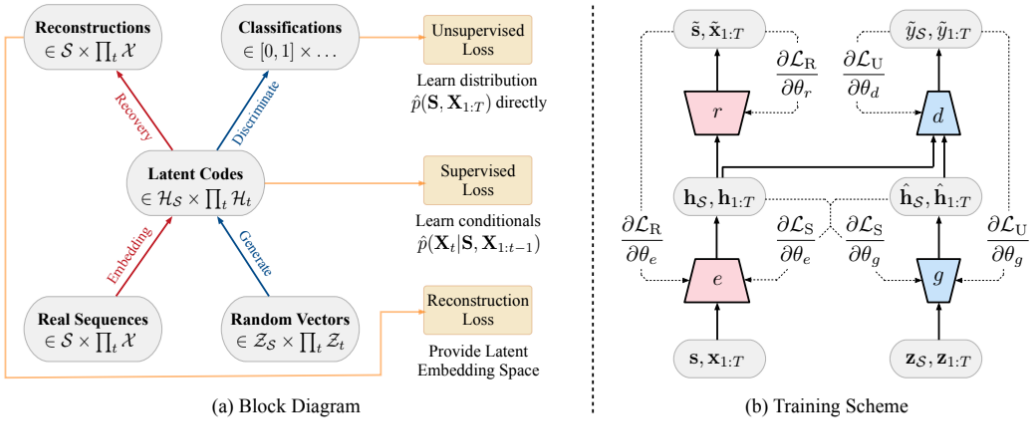

Figure 1: (a) Block diagram of component functions and objectives. (b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients.

Figure 2: TimeGAN Mechanism [5]

and classification components of TimeGAN specifically. All of these systems working together produce certain categories of loss which then feed into the next generation to improve the quality.

2.7 TSGBench

Since the data we are generating will never be exactly what the raw data is, it is imperative that we have the ability to evaluate and compare how accurate our results are in order to determine their usefulness. There is an existing evaluation model, Time Series Generation Benchmark (TSGBench), that scores the generated data in 12 different categories. The overarching sections are: model-based measures, feature-based measures, training efficiency, visualization, challenges in discriminative score and predictive score, and distance-based measures. [9]

Figure 3 shows how different methods of generating time-series data compare when graded using the evaluation measures present in TSGBench. The different colored lines represent the different methods for generating data, such as TimeGAN and COSCI-GAN. Each point on the line represents a data point. On the left circle, the different methods are evaluated using different features of the TSGBench, for example ED is Euclidean distance and DS is discriminative score. A dot placed closer to the inside of the circle signifies a better score. On the right circle, different applications of generating time-series data are being compared, for example we have the stock market, energy, and air quality.

2.8 Backtesting

Backtesting is another way to quantify how well our generated data works and how accurate our forecast is. This process involves applying our current model to historical situations in order to see how well our results fit the actual outcome. This provides a very

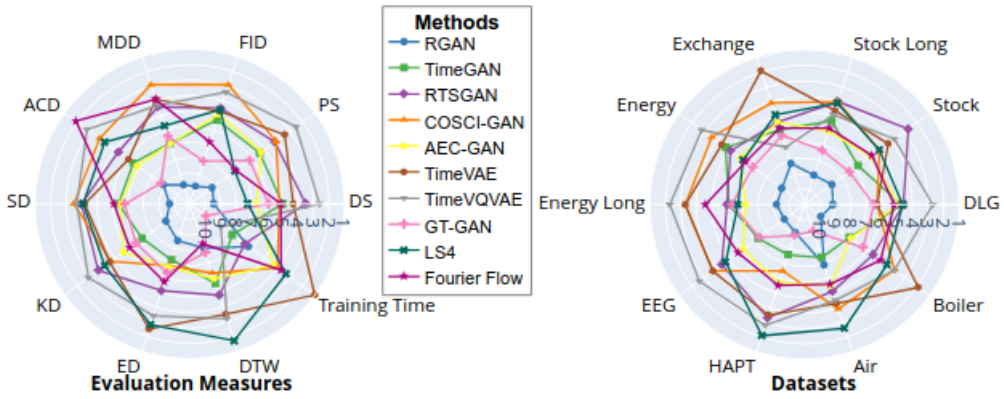

Figure 1: Method ranking across ten evaluation measures and ten datasets.

Figure 3: TSGBench Comparisons [9]

clear recommendation on how to improve our model and what its weaknesses are.

2.9 Project Goals

Our project has an emphasis on adapting the current TimeGAN model for intraday trading, which has different rules and principles when compared with inter-day trading. Trading within the same business day is much more volatile than trading over the course of days, weeks, months, or years. Most of the stock data available pertains to five crucial features based on a specified time frame: volume, highest price, lowest price, opening price, and closing price.

3 Methodology

3.1 TimeGAN adjustments

The main focus of our project is to optimize and train the current TimeGAN model to better generate data specific for day trading. Normal trading practices look for long term trend predictions in regards to industry performance, investor sentiment, and general economic environment. Day trading is less reliant on external factors and more dependent on the statistical analysis of the data.

Daily predictions are used for long-term stock management. The data is updated once everyday, so there is a lot of time to train the model. Using the New York Stock Exchange (NYSE) as an example, the Regular Trading Hours start at 9:30 am and end at 4:00 pm. There are 17.5 non-operational hours in between for the model to be trained on the latest data. On the contrary, intraday day trading predictions have to be done in a short time frame.

3.1.1 Optimization Methods

As the time allowance for training is much smaller, it is paramount

to update the model training method to adapt to such changes.

The difference lies in the time intervals, day trading moves on a much faster pace than inter-day trading, and thus the data is recorded every one minute as opposed to every 24 hours. This requires for some adjustments to be made to the current code to account for this quantity of data and the speed at which it is coming in.

When considering the business of intraday trading, generations are helpful, but the ability to forecast into the future informs better split second decisions. Our main goal is to use a training algorithm to be able to generate predictions a few minutes into the future. This will have significant impacts on the choices made when day trading, as everything is minute to minute. However, it is crucial to quantify the margin of error associated with our forecasts to determine how reliable and accurate they are. This is tremendously important when dealing with the financial market as these small sums of money add up quickly and have serious consequences. [11]

3.2 Additions to TSGBench

To give a numerical value to this error, we are working with the TSGBench code. We have appended the current model to include three additional functions for additional evaluation criteria. The first function simply calculates the time required for the new set of data to generate. This function records the time when the generation begins and records again when the generation ends, then subtracts them to give us the total time necessary for the data to generate.

We need our generations to be efficient and quick, as to keep up with the constant price changes throughout the day. A generation that is meant to forecast the next three minutes is useless if it takes four minutes to generate. Hence, we will use this function to quantify how much improvement our TimeGAN code needs regarding speed. [9]

The next function we implemented is to quantify the accuracy of our generated dataset. In order to give a value to this, we utilized the concept of Euclidean distance. Our function converts the generated data and the raw data into two separate arrays for easier comparison and then inputs both into the Euclidean distance formula to return a numerical value we call the similarity score. A lower similarity score means that the generated data is very close to the raw data we are working with, and that our generation is a good representation.

Precision is another concept that can be used to evaluate how trustworthy our data generations are. The goal of this function is to show how far apart a generated data set is from the previous one. This shows how consistent our generations are, and if our reliable results are able to be replicated. For this function, we again implemented the concept of Euclidean distance, but this time the two arrays are the i^{th} generated data set and the $i\text{-}1^{th}$ generated data set.

When implementing the precision and accuracy functions, we chose to measure in terms of Euclidean distance. This decision was made due to the simplicity of the data we are working with. The data can be easily converted into arrays, making it two-dimensional. Other measures or norms could have been utilized, but in the spirit of conserving time and space, Euclidean distance seemed a fair and reliable way to quantify these two important concepts.

3.3 Issues and Solutions

When working on this project, we encountered a few issues that were necessary to overcome to proceed with our work. The TimeGAN code that we are using from GitHub is written using TensorFlow 1.x. However, Python versions above 3.7 no longer support TensorFlow 1.x. We are utilizing Python version 3.10, three versions above the last version to accommodate TensorFlow 1.x. Hence, we conducted a soft migration to convert all of the GitHub code from using TensorFlow 1.x to TensorFlow 2.x. This was completed purely for compatibility reasons, and we do not use any new features to TensorFlow 2.x.

After this migration was completed, we decided to complete a hard migration from TensorFlow to PyTorch. The decision was made due to PyTorch being more flexible. It is more suitable for our research purposes and more user friendly.

Our next issue appeared in the creation of our new version of the TimeGAN code. When running the code we found on GitHub, we were getting the new dataset back in a form that was not what we anticipated. The output was in the form of batched time-series data, meaning it had three dimensions instead of the two that we anticipated. This was even more concerning because this result was unable to be input into TSGBench, so we were not able to score or grade the data that we were generating.

In order to combat this issue, we thought of three solutions: we could convert the generated TimeGAN data from a three dimensional array into a two dimensional array so that it could be input into the TSGBench, our next option was to adjust the TSGBench to allow for three dimensional inputs, and our final option was to index through the three dimensional generated data and put it into TSGBench batch by batch.

We opted to go with the third option, and thus wrote a function to append to TSGBench to insert each batch into the evaluation software array by array. This resulted in running the TSGBench code numerous times, depending on the dimensions of the data output by TimeGAN. This is a drawback to our design, as running TSGBench multiple times takes longer than running once, but combining the two programs served to provide a more user friendly experience and simplifies the process of generating and evaluating data.

Computing resources were an issue when completing this project, as well. We were working with two computers

1	2	3	4	5	6	7	8	9	10	11	12
2	3	4	5	6	7	8	9	10	11	12	13
3	4	5	6	7	8	9	10	11	12	13	14
					...						
2999	3000	3001	3002	3003	3004	3005	3006	3007	3008	3009	3010
3000	3001	3002	3003	3004	3005	3006	3007	3008	3009	3010	3011

Figure 4: Representation of the output from TimeGAN

with older CPU architecture, lacking the AVX command. Without this feature, we could not assign the GPU to model training. To combat this issue we had to utilize a shared computer using the SSH (Secure Shell) protocol. This is a method using to remotely connect and use a computer over a shared network.

The computer we used to SSH had more GPU resources and could better run and accommodate our program. However, we were sharing the SSH computer with four other groups of students working on similar projects that also required a lot of computing power. As a result, our access to this machine was limited greatly. Any project is enhanced by endless computing resources, so this was a drawback that we simply had to accommodate by scheduling our time on the machine and sacrificing some accuracy.

4 Results

4.1 TimeGAN Migration

4.1.1 Selection of PyTorch

PyTorch is a machine learning library known for its flexibility and modality. For easier manipulation of the model, we developed a PyTorch-based implementation of TimeGAN.

To take advantage of the modular nature of PyTorch, we made each of the components in TimeGAN a class. This provides us a separation of concerns, where the class definition specifies how the forward and backward propagation will operate. In contrast, the code outside the class does not have to mention how the propagation works. This modality makes the code easier to manage and debug as researchers can quickly locate the error.

After the three training stages, the weights and bias of each of the components are saved into ".pth" files. They hold the trained values of the model that can be loaded into the program state next time the program is ran. This allows the model to be trained

during the closed market hours and be used when the market is open for minute-to-minute data.

The other advantage of having the weights and bias saved down as a file is to be used for saving checkpoints. The model takes a long time to train. During training, there might be computer crashes, power outages, accidental cancellations and more unforeseeable accidents. The model weights, bias, and progress can be saved into files during training as a checkpoint, for the program to continue with the existing progress without having to start over from scratch.

4.1.2 Code Implementation

The 5 components are first initialized from their respective classes to be used in the upcoming functions.

In the proposed model, optimizers are used to refine the weights and bias. Optimizers are used to minimize the objective function. To implement it into the code, we used the "torch.optim .Adam()" optimizer function. Adam is an acronym for Adaptive Moment Estimation, and is an extension of Stochastic Gradient Descent (SGD). It is used to update the parameters of the model including the weights and bias.

There are three stages of training for TimeGAN, embedding network training, supervised loss only training, and joint training.

During embedding network training, only parameters of embedder and recovery are updated. In this process, the input is passed into the embedder and the result of the embedder is passed as input of recovery. The parameters are updated with the loss function between the result of recovery and the input of embedder. This stage of the training aims to strengthen the embedder's ability to create meaningful representation of the data in the latent space and the ability of the recovery to recover time series data from their latent representation.

During supervised loss only training, only the parameters of generator and supervisor are updated. In this process, the input is passed into the embedder and the result of the embedder is passed as input of supervisor. The parameters are updated with the loss function between the result of embedder and the result of the supervisor. This stage of the training aims to strengthen the supervisor's ability to generate the next sequence from the previous sequence.

During joint training, parameters of all five of the components are updated. However, not all of them are updated together. This training stage is split into two smaller sections. The first section of training is training the parameters of the Embedder, Generator, Supervisor, and the Recovery. The second section of training is training the discriminator. In each iteration of the joint training, the program does two training iteration of the first section and then one training iteration of the second section.

After all training has been completed, the model weights and bias are saved in ".pth" files. This allows the model to retain its knowledge in the next run without needing extra training. When

the program is ran with the option of not training, it will use the stored weights and bias. In this approach, the components are in evaluation mode, disabling automatic differentiation to save memory and improve performance of the code. It gains this advantage by not having to calculate the gradients in the forward pass, reducing the computational load and saving computing power. The program is then able to generate realistic synthetic data from a given set of time series data.

4.2 Combination of TimeGAN and TSGBench

Our efforts resulted in a document of Python code combining our new optimized version of TimeGAN, a transitional interface between the TimeGAN output and the TSGBench input, and our updated TSGBench code. This makes for a smooth and intuitive user experience, as data can be generated and scored within the same terminal. As discussed in Section 3.3, this was completed by indexing through the data grid column by column and inserting each individual column into TSGBench, one at a time.

Figure 4 shows how the data is returned from TimeGAN, each number has five associated features (high price, low price, volume, opening price, and closing price). The first row, numbers 1-12, represents the first 12 units of time series data. The next row, numbers 2-13, represents the addition of one more data point. This pattern continues until we reach the n^{th} row. In this case, the n is the number of data points in the input data minus the amount of columns. The amount of columns is up to the discretion of the user.

If the user is more concerned about generating data and wants to reduce time spent evaluating data, they can reduce the batch size. This will return the same quality of data, but with less information about how well it performs. However, grading your data is important since even small margins of error can greatly impact a trader's stock market successes.

5 Conclusion and Future Work

Within the 10 weeks we were given to complete this project, we accomplished our 3 main goals: to update and migrate the starting TimeGAN model, enhance the current TSGBench code, and connect the two programs into one smooth interface. The TimeGAN was successfully migrated to PyTorch, three new functions were added to TSGBench, and an interface to combine the two codes into one Python document was created.

If we were to devote more time to this project, the new steps would be to create forecasts into the future, develop the LLM model more thoroughly, and complete more back testing using previous data. To expand upon our current work, we would test our TimeGAN with other companies intraday data, such as Apple or Microsoft, if we were able to access it. This would expand the reach of our work and make it more useful to traders.

6 Acknowledgments

We would like to thank Dr. Kwai Wong for his mentorship, guidance, and assistance with this project. We would also like to thank the National Science Foundation and the University of Tennessee, Knoxville for funding and providing computational resources that promoted the completion of our project. Our coworkers and peers, are also owed an acknowledgment for their assistance and advice as we completed this project.

References

[1] Don Charles. *A Practical Introduction to Day Trading*. Cambridge Scholars Publishing, 2018.

[2] Gan structure. Available at: `https://developers.google.com/machine-learning/gan/gan_structure`, 2024. Accessed: July 17, 2024.

[3] Alexander Nikitin, Letizia Iannucci, and Samuel Kaski. Tsgm: A flexible framework for generative modeling of synthetic time series. 05 2023.

[4] Giuseppe Masi, Matteo Prata, Michele Conti, Novella Bartolini, and Svitlana Vyetrenko. On correlated stock market time series generation. pages 524–532, 11 2023.

[5] Jinsung Yoon, Daniel Jarrett, and Mihaela Schaar. Time-series generative adversarial networks. 12 2019.

[6] Haksoo Lim, Minjung Kim, Sewon Park, and Noseong Park. Regular time-series generation using sgm. 01 2023.

[7] Eoin Brophy, Zhengwei Wang, Qi She, and Tomas Ward. Generative adversarial networks in time series: A systematic literature review. *ACM Computing Surveys*, 55, 08 2022.

[8] Ali Seyfi, Jean-Francois Rajotte, and Raymond T. Ng. Generating multivariate time series with common source coordinated gan (cosci-gan). *36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 12 2022.

[9] Yihao Ang, Qiang Huang, Yifan Bao, Anthony K. H. Tung, and Zhiyong Huang. Tsgbench: Time series generation benchmark. 17(3), 2023.

[10] Yangming Li. Ts-diffusion: Generating highly complex time series with diffusion models. 11 2023.

[11] Junwen Yao, Jonas Mueller, and Jane-Ling Wang. Deep learning for functional data analysis with adaptive basis layers. 06 2021.