International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2024, Vol. 8, No. 1, IJSCAI832

PGSGAN: Policy Gradient Stock GAN

Masanori Hirano [*], Hiroki Sakaji [†], Kiyoshi Izumi [*]

Abstract

We introduce a novel generative adversarial network (GAN) designed to generate realistic trading orders for financial markets. Past models of GANs for creating synthesized trading orders have always been focused on continuous spaces because their architecture has constraints coming from the learning algorithm. Contrary to this, actual orders are placed on discontinuous space, which comes from the actual trading rules such as a minimum unit price for orders. In this study, therefore, we adopt a different approach supporting the actual trading rules and placing generated orders to discontinuous state space. The modification led to the inability to apply the standard GAN's learning algorithm, requiring us to use a policy gradient, a solution commonly used in the realm of reinforcement learning, as our learning strategy. Our experiments handle massive amounts of order data over half a year. Our experimental results indicated that the proposed model surpassed previous models in terms of the distribution pattern of the generated orders. A noteworthy advantage of integrating the policy gradient into our model is that it provides the ability to monitor the GAN's learning progress by analyzing the entropy of the generated policy.

Keywords: Generative adversarial networks (GAN), Financial markets, Policy gradient, Order generation

1 Introduction

In financial markets, the realized sequence of orders is a small subsection within a vast range of potential states. To put it differently, the variety of possible market trajectories comprises one actual path and countless unrealized possibilities. Even though the state space of a single order may be confined, the aggregation of these orders might expand the state space extensively. Furthermore, the inherent non-stationarity of financial markets implies that all possible states might not be realized in historical data, leading to an inadequate amount of data for several potential states.

Accumulating a larger volume of data can provide several benefits. It could enhance the effectiveness of trading strategies derived from such data. Performing a backtest with this data increases the precision of risk-level estimations. Also, a correct assessment of portfolios can be achieved by using a variety of simulated time-series price paths.

[*] The University of Tokyo, Tokyo, Japan
[†] Hokkaido University, Hokkaido, Japan

Despite the availability of extensive financial market data, it is often inadequate, leading to many research efforts focused on overcoming this data scarcity. In this regard, a leading solution is augmenting the available dataset with GANs.

To this end, the prominent approaches are artificial market simulations and data augmentation via GANs.

Artificial market simulations aim to simulate virtual markets under hypothetical situations and examine those situations that have not occurred in the actual financial market. Events such as financial crises are rare, but occured variedly. However, data about such events are insufficient. Additionally, the effects of external factors, such as new regulations, remain unknown. Therefore, by controlling the situations in artificial market simulations, we receive insights and benefits.

The other approach, GANs, aims to make realistic order time series to augment past data. This approach is more pragmatic than artificial market simulations. For predictions, deep learning approaches are gaining popularity; however, they require plenty of data. GANs can fill this need.

This study focuses on the latter approach, that is, GANs for making realistic order time series in financial markets, especially in stock markets.

Several studies have aimed to apply GANs within stock markets, but the generated synthetic data often lacks realism. Notable works in this area are Stock-GAN (S-GAN) [1], and Market GAN [2]. For GANs to function effectively, a gradient link between the generator and the critic is imperative. The necessity for this link can lead to data that seems invalid. With S-GAN, for instance, all order values such as buy/sell decisions, order types, prices, and volumes are continuous. Distincting such generated data is relatively easy because actual market orders are not continuous. Though Market GAN produces discrete values and determines the probability of the classes, the generated data remains implausible because humans can easily distinguish the generated data from the real data. The predominant issue with current financial market GANs is the placement of the generated fake data within a continuous space. It is crucial that generated data are acceptable to the order system in order to mimic real-world situations effectively.

Another problem is the treatment of buy/sell decisions and order types as continuous variables and adjoining their state spaces. For instance, if buy/sell was represented as a range from 0 to 1, the two would be bound together across this state space. For example, suppose the order is 100 shares at $200. In that case, when the best quote is $190, the meaning is completely different between sell and buy; the buy at $200 is effectively a take order; the sell at $200 is effectively a make order.

To address those issues, this research proposes Policy Gradient Stock GAN (PGSGAN), a new GAN learning method for stock markets using a policy gradient. This technique is commonly utilized in reinforcement learning. Through the integration of reinforcement learning principles into the GAN structure, the need for a gradient link between the generator and the critic can be eliminated, resulting in more realistic discrete generated orders.

In our studies, we processed over 65 million orders spanning more than six months from 10 different stocks and built our GAN model. This experiment led to the successful creation of a GAN output that aligns with authentic market trading rules, thereby improving the data generation performance. This model could be used to augment more realistic data, improving the machine learning prediction tasks' learnability.

While the PGSGAN was crafted on the rules of the Tokyo Stock Exchange (TSE), it can be easily adapted for other markets with minor modifications.

In the following sections, we will discuss the related work in Section 2. Section 3

explains the details of models in Section 3. Then, the experiments and the results are described in Section 4 and 5, respectively. Section 6 discusses the results. Finally, we will conclude our study in Section 7. This paper is an extended version of our previous paper [3].

2 Related Work

As mentioned above, there are two main types of data augmentation approaches in financial markets.

In the artificial market simulation context, Maeda *et al.* [4] attempted to make a model learning the better trading strategy via augmented data by artificial market. Returning to the basics of artificial market simulation, Edmonds *et al.* [5] argued that agent-based simulation is useful for social sciences. The importance of agent-based simulation, especially for the financial markets, was discussed in [6, 7]. Mizuta [8] demonstrated that a multi-agent simulation for the financial market could contribute to the implementation of rules and regulations in actual financial markets. Indeed, Mizuta *et al.* [9] tested the effect of price tick size, that is, the price unit for orders, which led to a discussion of tick size devaluation in the Tokyo Stock Exchange Market, based on the data from an artificial market simulation. Hirano *et al.* [10] assessed the effect of the regulation of the capital adequacy ratio (CAR) and observed the risk of market price shock and depression due to CAR regulation from the generated data under the hypothetical situation realized in their artificial market simulation. Although artificial market simulations do not seem to augment data, these can be called thus, because they generate data and use it for discussion.

The other approach employed in this study is GANs. As previously discussed, Li *et al.* pioneered the Stock-GAN (S-GAN)[1]. S-GAN was a groundbreaking advance in the use of GANs for stock market modeling. It aimed to generate order time series without the need for authentic order book data through the use of continuous double auction (CDA) networks. Furthermore, Naritomi *et al.* [2] demonstrated a GAN model's effectiveness for stock markets, showcasing its capability to produce data useful for future price movement forecasts. In the realm of GAN usage for future price predictions, some studies [11, 12] have delivered promising results. Collectively, these studies reflect the growing practicality and effectiveness of GAN-based approaches in financial markets.

The development of GAN technology has seen notable improvements over time. Goodfellow *et al.* [13] proposed the original GAN. Subsequently, Mirza *et al.* [14] proposed the concept of Conditional GAN, which leverages conditional inputs, and the design is also used in this study. Radford *et al.* [15] later introduced the Deep Convolutional GAN (DCGAN), another model we referenced for the comparative model of this study. As other learning architectures, the least-squares GAN [16], generalized f-GAN [17], Laplacian pyramid GAN [18], variational autoencoder GAN [19], image-to-image translation GAN (known as pix2pix) [20], self-attention GAN [12], cycle GAN [21] for image translations, style GAN [22] for style converts, and progressive growing GAN [23] for high-resolution images were proposed. In addition, Yu *et al.*[24] introduced SeqGAN as a method to generate sequences such as text or music using policy gradient techniques. Although their work may appear similar to ours, our GAN does not focus on sequence generation. Moreover, as an extension of GANs, Donahue *et al.* [25] proposed adversarial feature learning for embedding images into vectors, Schlegl *et al.* [26], Zenati *et al.* [27], and Deecke *et al.* [28] proposed anomaly detection based on GANs. Wasserstein GAN (WGAN) [29] was sug-

gested based on the discussion of the learning stability of GANs [30], which our study is based on. To stabilize the WGAN, various techniques have been proposed, such as gradient penalty [31] and spectral normalization [32]. In our study, we have chosen to use spectral normalization as a means to achieve this stabilization.

3 Models

3.1 Policy Gradient Stock GAN (PGSGAN)

The PGSGAN leverages historical data and current conditions to generate the next order in financial markets. It is built upon the GAN framework [13] and incorporates the policy gradient theorem [33, 34, 35].

More specifically, our approach is based on the Wasserstein GAN (WGAN) [29]. We employ the REINFORCE algorithm [36] with a baseline as the policy gradient algorithm. To enhance the performance of our neural network, we utilize a convolutional neural network (CNN) architecture.

In order to stabilize the learning process, several normalization techniques are employed. Batch normalization [37], layer normalization [38], and spectral normalization [32] are utilized in our model. These techniques play a crucial role in improving the stability and efficiency of the learning process.

3.1.1 PGSGAN Architecture

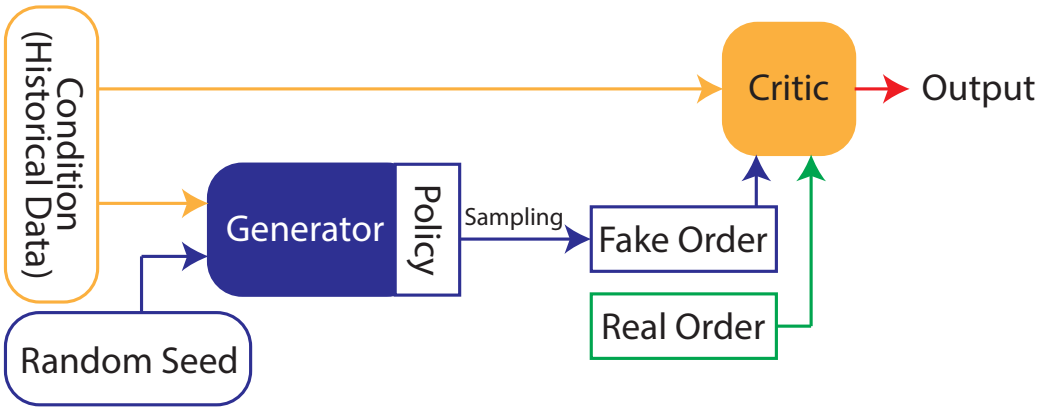

Figure 1: The outline of PGSGAN

Figure 1 indicates the outline of our PGSGAN.

The generators in our experiments have the capability to process conditional data, which consists of historical market data. They also require random seeds as input for generating fake order data. Specifically, the conditional data includes the previous 20 order time series as well as the current best sell and buy prices.

The order time series provides information such as whether it is a buy or sell order, if it is a new order or a cancellation, whether it is a market order, the price relative to the best price, the volume (scaled by dividing it with the minimum volume unit), and the best prices before the order is placed.

To initialize the generation process, a random seed is generated with 128 dimensions. This seed is used by the generator to create a policy that determines how the next fake order will be generated. In our experiments, the generator utilizes 14 convolutional layers followed by 5 linear layers (explains the details later) to generate the following policy:

- Sell or Buy – 2 classes (probability)
- New or Cancel – 2 classes (probability)
- is a market order? (Is MO) – 2 classes (probability)
- relative price (ticks from the best price) – 40 classes (probabilities for 0 – 39) [1]
- volume (scaled by dividing by minimum volume unit) – 40 classes (probabilities for 0 – 39)[1]

Once the generator has computed the probabilistic policy, it proceeds to generate a fake next order using weighted sampling based on this policy.

On the other hand, the critic has a different role and is designed to map inputs into a scalar value while satisfying the 1-Lipschitz constraint. This constraint is similar to the one used in the basic Wasserstein GAN (WGAN) framework.

The critic in our experiments takes two inputs: the conditional data (same as used by the generator), and either the generated fake next order or the real next order. To ensure consistency, real next orders are also transformed into the same range as the generated fake orders, as described earlier.

In this transformation process, only the price and volume of the real data are mapped into the range of 0-39 (integer values). However, if the real order is a market order, the price value is specifically set to 0.

3.1.2 PGSGAN Learning Mechanism

The sampling process in PGSGAN, which generates a fake order based on a generated policy, results in a loss of the gradient connection between the generator and the critic. The traditional GANs, including those designed for stock markets, heavily rely on the gradient connection between their generator and critic for training the generator effectively.

Given this lack of gradient connection, our PGSGAN diverges from the traditional learning theory typically used for generator training. Instead, we adopt a new learning approach for the generator using the policy gradient method, which is widely utilized in the field of reinforcement learning.

In the subsequent discussion, we will use the following notations:

- z: random variables (seed for generator. In our experiment, $z \in \mathbb{R}^{128}$.)
- $\mathbb{P}_z$: the distribution of random variables
- $\mathbb{P}_r$: the distribution of real data

[1] When the value is equal or more than 40, it was regarded as 40th class. Although these cases (40 ticks over or more than 4000 shares) could happen, the percentages are very limited in TSE. Moreover, these cases usually occurred by events outside markets themselves; thus, we ignore the detailed modeling of these cases in this study.

- $C(x)$: the critic as a function. The output is scalar. Here, x is a given input from the outside.[2]
- $G(z)$: the generator as a function. The output is a policy. Usually, the generator accepts random seeds.[2]
- $\tilde{x} \sim G(z)$: the sampled fake order $\tilde{x}$ follows the policy generated by $G(z)$.
- θ_C, θ_G: params in the critic and generator, respectively.
- L_C, L_G: loss function for the critic and generator.
- $||f||_{L\leq 1}$: 1-Lipschitz constraint for any function f.
- $p_{G(z)}(\tilde{x})$: probability for the sampled fake order $\tilde{x}$ according to the generated policy $G(z)$.
- $\mathrm{NLL}_{G(z)}(\tilde{x})$: negative log-likelihood for the sampled fake order $\tilde{x}$ according to the generated policy $G(z)$. $\mathrm{NLL}_{G(z)}(\tilde{x}) = -\ln\{p_{G(z)}(\tilde{x})\}$.

At first, PGSGAN will solve the following minimax game:

$$\min_G \max_{||C||_{L\leq 1}} L_{\mathrm{GAN}}(G,C), \tag{1}$$

where

$$L_{\mathrm{GAN}}(G,C) := \mathbb{E}_{x\sim\mathbb{P}_r}[C(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}\left[\mathbb{E}_{\tilde{x}\sim G(z)}[C(\tilde{x})]\right].$$

This form is similar to the original form of WGAN. However, the sampling term $\mathbb{E}_{\tilde{x}\sim G(z)}$ has been added. This change is substantial for generator learning.

For the critic, the objective function is

$$\max_{||C||_{L\leq 1}} \left\{\mathbb{E}_{x\sim\mathbb{P}_r}[C(x)] - \mathbb{E}_{\tilde{x}\sim G(z)}[C(\tilde{x})]\right\} \tag{2}$$

because the generator and its seeds do not matter to the critic. Thus, the loss function for the critic is:

$$L_C := \mathbb{E}_{\tilde{x}\sim G(z)}[C(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[C(x)]. \tag{3}$$

These are the same as WGAN because the generator does not matter for the critic; only the fake data affect the critic.

In contrast, the learning theory for the generator is complicated. The outline of the learning is shown in Figure 2.

The generator's objective function is:

$$\min_G \left\{\mathbb{E}_{x\sim\mathbb{P}_r}[C(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}\left[\mathbb{E}_{\tilde{x}\sim G(z)}[C(\tilde{x})]\right]\right\}. \tag{4}$$

Like the original WGAN, the first term of this equation is unchangeable for the generator. Thus, the objective function is re-written as:

$$\max_G \mathbb{E}_{z\sim\mathbb{P}_z}\left[\mathbb{E}_{\tilde{x}\sim G(z)}[C(\tilde{x})]\right]. \tag{5}$$

[2]Correctly, it also accepts conditional data, but it is ignored in this notation for simplicity.

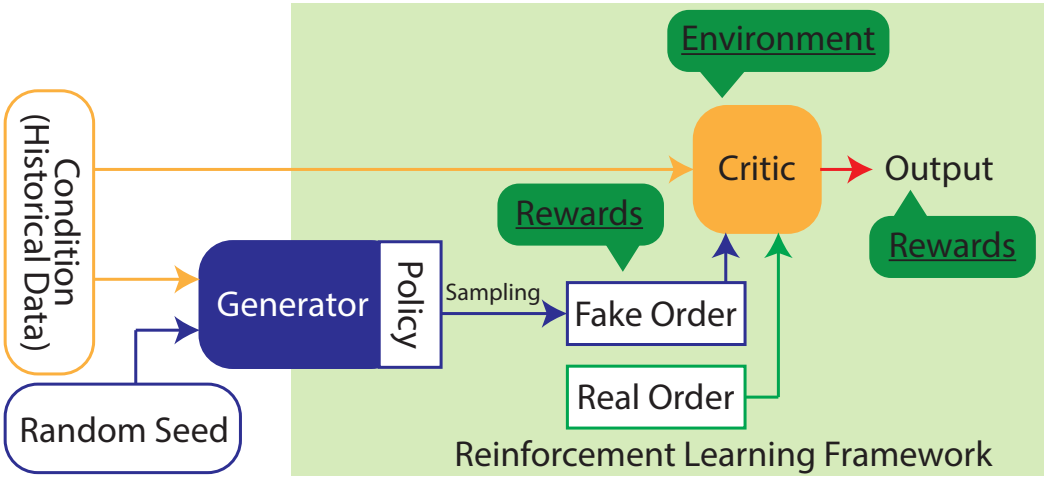

Figure 2: Outline of a generator's learning in terms of a reinforcement learning (RL) framework

This objective function cannot be converted into a backpropagation of a neural network because of the lack of gradient connection for the generator. Thus, here, we employ REINFORCE, one of the policy gradient methods, as a learning algorithm from reinforcement learning.

As Figure 2 shows, the form of the generator can be thought of as reinforcement learning, in which the generator is an actor pursuing higher rewards and generating a policy for action. Then, according to the generated policy, an action is taken: making fake orders. Through the unknown environment, then, the action makes a reward: the output from the critic. Finally, according to the rewards, the actor, that is, the generator, is updated.

According to REINFORCE, the parameter is updated as:

$$\theta \leftarrow \theta + \alpha \cdot G \nabla_\theta \ln \pi_\theta(a|s), \tag{6}$$

where θ is model parameter, α is the learning rate, G is the return (usually the sum of discounted future rewards; however, in this study, just upcoming reward itself caused by action a), $\pi_\theta(a|s)$ is a probability of action a under the state s according to the current policy π_θ. By introducing baseline, equation 6 is changed to

$$\theta \leftarrow \theta + \alpha \cdot (G - B) \nabla_\theta \ln \pi_\theta(a|s), \tag{7}$$

where B is the baseline. (In this study, we employ mean of G in one batch.)

By applying REINFORCE for PGSGAN, the parameter update of the generator is:

$$\theta_G \leftarrow \theta_G + \alpha (C(\tilde{x}) - B) \nabla_{\theta_G} \ln p_{G(z)}(\tilde{x}), \tag{8}$$

where $\tilde{x} \sim G(z)$, $z \sim \mathbb{P}_z$, and B is the mean of $C(\tilde{x})$ among a learning batch. Thus, the loss function for the generator is defined as:

$$L_G := -(C(\tilde{x}) - B) \ln p_{G(z)}(\tilde{x}) \tag{9}$$

$$= (C(\tilde{x}) - B) \cdot \mathrm{NLL}_{G(z)}(\tilde{x}). \tag{10}$$

Therefore, the generator is enabled to learn.

3.1.3 Additional Notes and Actual Implementation for PGSGAN

In PGSGAN, we employ spectral normalization [32] in all layers, which is required to realize 1-Lipschitz constraint in critics. However, for learning stability, we use it also in the generator.

The layers processing the conditional data in both the generator and the critic have the same architecture; however, they are trained separately and not shared.

In our implementation (including the common architectures for conditional data, which has 44,750 parameters), the number of parameters in the critic and generator are 113,071 and 141,625, respectively. The detailed implementations are shown in Figures 3 and 4.

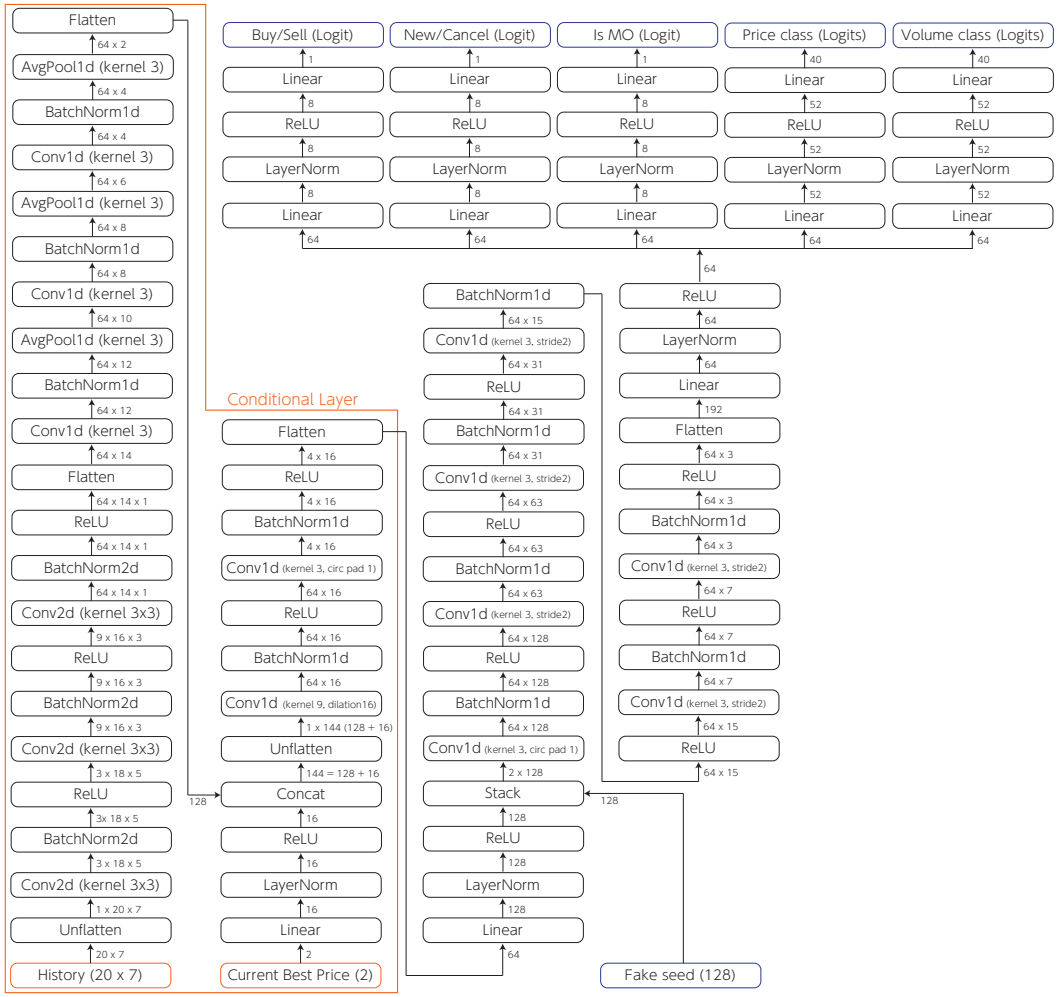

Figure 3: The detailed architectures of the generator

Figure 3 shows the details of our generator. In our model, we utilize Convolutional Neural Networks (CNNs) as the foundational architecture. The conditional layer, which is responsible for processing the historical data, incorporates average pooling. This is because in high-frequency trading, the sequence of certain orders may not be as significant. By employing average pooling, these orders can be buffered effectively.

After concatenating the processed historical data with the current best price, we apply dilated convolution and circled convolution. These techniques are utilized to ensure an

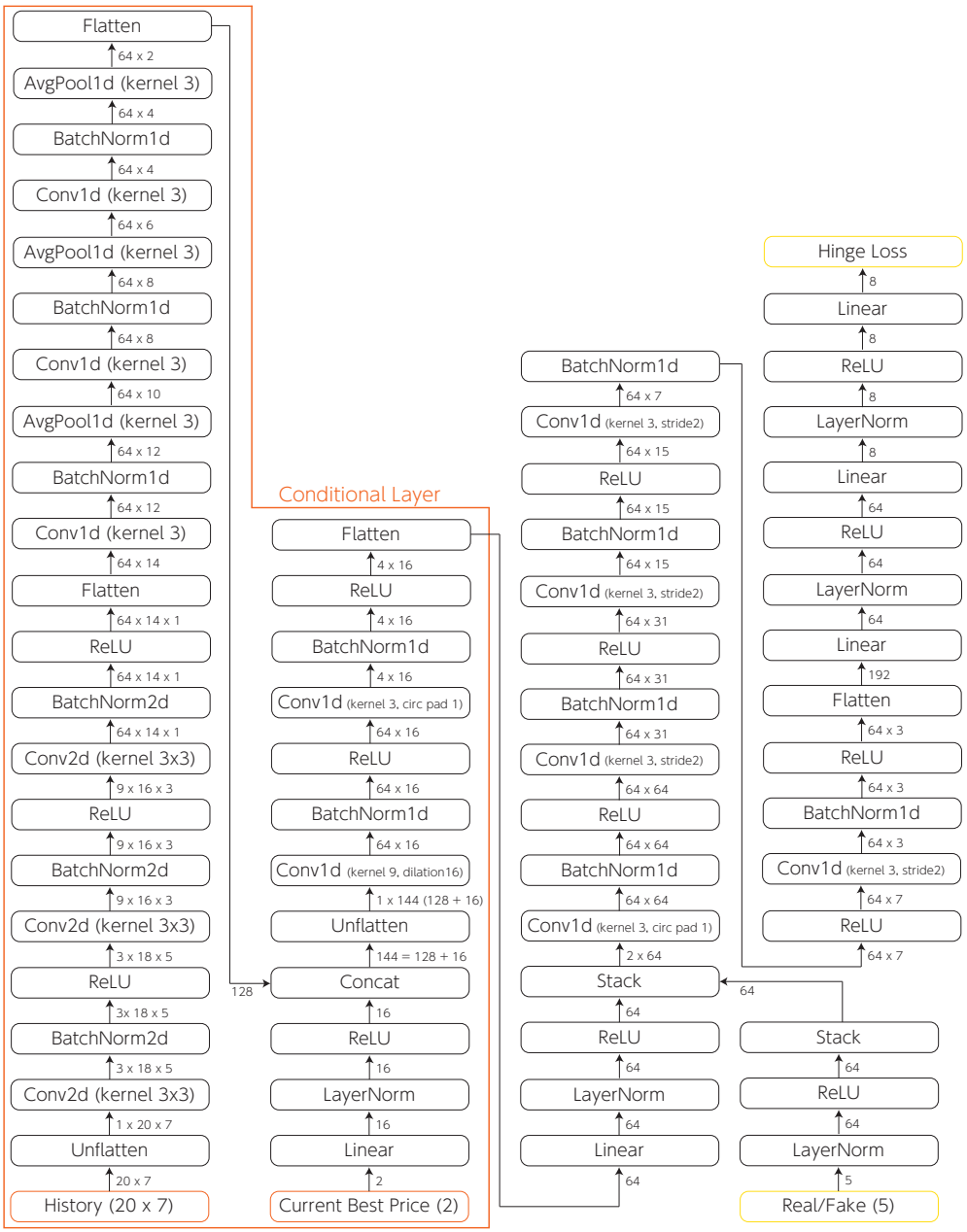

Figure 4: The detailed architectures of the critic

equal mixing of the two inputs. Dilated convolution, as introduced in [39], allows us to expand the receptive field of the convolutional layer, capturing broader context information. The circled convolution aids in mixing the concatenated inputs thoroughly.

For the output of our model, we employ logits. This choice is convenient for subsequent calculation and analysis. The loss function for the generator is calculated as

$$L_G := -(C(\tilde{x}) - B) \ln P_{G(z)}(\tilde{x}) \tag{11}$$
$$= (C(\tilde{x}) - B) \cdot \mathrm{NLL}_{G(z)}(\tilde{x}). \tag{12}$$

Thus, for compatibility of the negative log-likelihood (NLL), the logits are best for less computational error. For making actual policies, these logits are put into sigmoid or softmax.

As mentioned above, we also set sell/buy, new/cancel, and whether the order is market order (MO), as two classes of output. However, for the convenience of calculation, the outputs have one class. Thus, for making the policy, we convert them into two classes. Moreover, in all layers, spectral normalization [32] is applied.

Contrary, Figure 4 shows the details of the critic. Basic architectures of the critic are almost the same as the generator, except for the final layers and each dimension.

3.2 Policy Gradient Stock GAN with Hinge Loss (PGSGAN-HL)

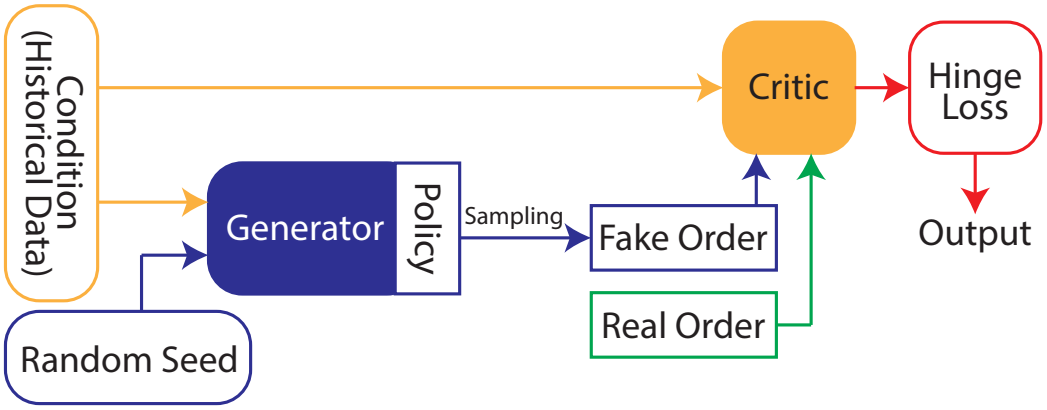

Figure 5: The outline of PGSGAN-HL

We also implement the PGSGAN with Hinge loss as shown in Figure 5. Originally, Hinge loss was used in WGAN in Geometric GAN [40].

Hinge loss is defined as

$$\begin{cases} \max(x+1,0) & \text{(critic training for fake data)} \\ \max(1-x,0) & \text{(critic training for real data)} \\ x & \text{(generator training)} \end{cases} \tag{13}$$

and insert into the last of the critic layers.

The others are the same as PGSGAN.

3.3 Comparative Models

3.3.1 Stock GAN (S-GAN)

Stock GAN (S-GAN) was proposed by [1], and based on WGAN-GP [31].

In our study, we aim to replicate S-GAN as a comparative baseline. However, to ensure a fair evaluation, we modify certain parts of its architecture. The modifications are as follows:

- Deletion of Continuous Double Auction (CDA) network: The original S-GAN model incorporated a CDA network to update the best prices after receiving a new order. However, since our study assumes access to comprehensive market data, there is no need for the CDA network's best price estimation. Hence, we remove this network from our model.

- Deletion of time signal: The original S-GAN model included a time signal input, which aimed to identify the specific time of the day when an order was placed. It divided the day into 24 classes for this purpose. However, the stock exchange we are focusing on, the Tokyo Stock Exchange (TSE), operates with only 2.5-hour sessions (two sessions per day). Consequently, the need for the time signal input is eliminated, and we remove it from our model.

- Change in price processing: In the original S-GAN model, absolute price values were used because it was designed for generating data over a relatively short period, assuming approximately one day. However, in our study, we target much longer periods, extending over half a year. To address this, we modify the price inputs to represent ticks from the best price, thereby normalizing them and accounting for relative price movements.

- Deletion of time since the previous order: In the tick-time scale, the arrival interval between orders should be modeled separately since some orders can be published simultaneously, and the sequence or interval might not be significant. However, to simplify the problem and focus on our specific objectives, we ignore the prediction of the order arrival interval.

While we make these modifications, the remaining parts of the S-GAN architecture remain unchanged.

However, unlike other models, we set the total learning epochs of S-GAN to half of that for others (About the learning epoch, we have explained the learning epoch in the experiments section). This is because S-GAN employed gradient penalty [31] for 1-Lipschitz constraint. Gradient penalty requires an additional backpropagation for calculating the penalty in the loss. Thus, it requires more computational resources than others.

Moreover, we converted generated output to discrete values for fair evaluation by just rounding in the evaluation phase because the generated output is continuous numbers.

3.3.2 DCGAN

In addition to replicating the S-GAN model, we also include another well-known comparative model called DCGAN.

The DCGAN is based on CNN architecture and is capable of generating orders with continuous values, similar to the S-GAN model. However, in order to ensure a fair comparison between the models, we convert the generated output of the DCGAN to discrete values, just like in the S-GAN.

The overall architecture of DCGAN consists of deep convolutional layers, with the generator network responsible for generating fake orders and the critic network used to distinguish between real and generated orders.

By including DCGAN as another comparative model, we can assess its performance alongside S-GAN and gain a better understanding of their respective strengths and weaknesses in generating fake orders.

4 Experiments

In our experiments, we randomly select 10 stocks for evaluation purposes under the specific criteria. These criteria focused on stocks listed on the Tokyo Stock Exchange (TSE), which is the primary target of our study.

The first criterion was that the stocks must be included in the Nikkei 225 index. The Nikkei 225 [3] is a major index in the TSE and consists of stocks that are carefully selected based on criteria such as liquidity and sector balance. These stocks are traded frequently and provide the necessary liquidity for generating realistic tick-scale orders. The composition of the Nikkei 225 index is periodically updated, and stocks may be replaced if they are delisted or during the annual renewal in October.

The second criterion was that the selected stocks should not be included in the TOPIX 100 index. The TOPIX 100 is another major index maintained by the Japan Exchange Group [4]. It comprises the top 100 stocks with high liquidity and market value in the TSE. These stocks have a special status in terms of trading rules, with a smaller minimum order price unit (price tick size). Due to the challenges and computational resources required to model these special stocks, we decided to exclude them from our study.

The third criterion was that the chosen stocks should be included in the TOPIX 225 index but not in the TOPIX 100 index consistently during the 2018-2020 period. This criterion is important as changes in index inclusion or exclusion can significantly impact trading volume.

The final criterion was that the selected stocks should have the same price tick size throughout the specified data periods. In the TSE, the price tick size changes based on the price range of the stock. It typically changes at price levels of $N \times 10^M$ ($N = 1,3,5$, and $M = 3,4,5,6,7$). Due to current technological limitations, our model cannot account for changes in price tick size. Thus, we opted for stocks that maintained a consistent price tick size throughout the data periods, ensuring compatibility with our model.

By adhering to these criteria, we aimed to select a diversified set of stocks that would enable us to evaluate the performance of our models effectively.

As a data period, we employed January – September in 2019, because we avoid the periodical updates of indices. Nikkei 225 is periodically renewed on the every first business day of August, and TOPIX 100 on every last business day of August.

According to these criteria, we obtain 81 stocks. Only 125 stocks are included in Nikkei 225 and not in TOPIX 100. Thus, candidates for random selection are more than half of the

[3] https://indexes.nikkei.co.jp/en/nkave/index/profile?idx=nk225

[4] https://www.jpx.co.jp/english/markets/indices/topix/

stock candidates. From these 81 stocks, we choose 10 at random, which is shown in Table 1. The data are split as train:valid:test = 8:1:1 in temporal sequence.

Table 1: Selected Stocks

Ticker	Name	Classification (by Bloomberg)	# of orders in data
5901 JP	Toyo Seikan Group Holdings, Ltd.	Containers & Packaging	6,569,563
5333 JP	NGK Insulators, Ltd.	Auto Parts	6,077,554
8355 JP	Shizuoka Bank, Ltd.	Banks	5,307,488
5631 JP	Japan Steel Works, Ltd.	Other Machinery & Equipment	6,787,814
9532 JP	Osaka Gas Co., Ltd.	Gas Utilities	7,914,464
7012 JP	Kawasaki Heavy Industries	Diversified Industrials	9,122,778
2501 JP	Sapporo Holdings, Ltd.	Alcoholic Beverages	4,852,475
4005 JP	Sumitomo Chemical Co., Ltd.	Basic & Diversified Chemicals	6,319,126
7752 JP	Richo Co. Ltd.	Consumer Electronics	6,942,513
7911 JP	Toppan Inc.	Printing Services	6,057,922

The scale of data processed in this study is substantial, with the total volumes of tick data exceeding 65 million, as shown in Table 1.

This surpasses the data size used in previous prominent works such as S-GAN [1]. In the S-GAN study, the experiment was conducted using only two selected stocks: Alphabet Inc. (GOOG) and Patriot National Bancorp Inc. (PNBK). GOOG had a dataset consisting of 230,000 orders, while PNBK had a much smaller dataset with only 20,000 orders. Moreover, the testing duration for S-GAN was limited to nearly one day.

In contrast, our study spans longer periods, specifically 9 months, and incorporates significantly larger datasets. To accommodate the processing of such vast amounts of data, we developed a backend data preprocessing server using Golang, while the deep learning tasks were handled by a PyTorch-based Python program. These two components communicate using gRPC over a 10G Ethernet connection, facilitating efficient data transfer and processing between them.

By utilizing a combination of optimized data infrastructure and efficient deep learning frameworks, we were able to handle the substantial data size and duration required for our study effectively.

The test task is the next order generation. The generation of a long time series is also a repetition of the prediction of the next order. For simplification, we narrow it down to the generation of the next order. This is partially reasonable because there has been no research on generating only long order time series, but it should be addressed in future work. For PGSGAN, we calculate and inspect the negative log-likelihood (NLL) for real order and the entropy of the generated policy. The NLL is $\mathrm{NLL}_{G(z)}(x)$ where x is the real order. This shows how the generator policy successfully fits the real order. Although a low NLL indicates a better fit with the real data, a complete fit is not beneficial as a generator. Thus, our experiments use the log-likelihood as one index for check learning status, but do not pursue the lowest NLL. Theoretically, the by-chance NLL is $-\ln\{1/(2\times 2\times 2\times 40\times 40)\}\approx 9.45$. On the contrast, the entropy is defined as

$$H(G(z)) := \sum_{x\in\mathbb{X}} -p_{G(z)}(x)\log_2 p_{G(z)}(x), \tag{14}$$

where $\mathbb{X}$ is all order classes. This entropy indicates how well the generator policy is learned. If the policy is learned well, the probability of each class of the generated policy will be well skewed. Therefore, this index is useful for checking the convergence of PGSGAN.

Theoretically, the by-chance entropy is

$$\sum_{x\in\mathbb{X}} -\frac{1}{2\times2\times2\times40\times40}\log_2\frac{1}{2\times2\times2\times40\times40} \tag{15}$$

$$=\log_2(2\times2\times2\times40\times40)\approx13.64. \tag{16}$$

where $\mathbb{X}$ is all order class.

Moreover, we also compare the distribution of generated and real orders, for all the orders. Here, we employ Kullback–Leibler divergence (KLD) [41] and Mean Square Error (MSE) for all classes ($2\times2\times2\times40\times40$). Kullback–Leibler divergence is defined as:

$$D_{KL}(P,Q)=\sum_{x\in\mathbb{X}}P(x)\log_2(P(x)/Q(x)), \tag{17}$$

where $\mathbb{X}$ is all order class, and $P(x)$ and $Q(x)$ indicate the probability of the real orders and the generated orders for class x, respectively. Even though $P(x)\log_2(P(x)/Q(x))$ is calculated as 0 when $P(x)=(Q(x)=)\ 0$, KLD would be infinity if $\exists x, P(x)\neq0, Q(x)=0$ due to some reasons, such as mode collapse of generator. Moreover, for DCGAN and S-GAN, because the generated output is continuous numbers, we round the output to translate into the discreate values. Because the generated output should be different based on random seeds, we evaluate the generator 100 times with different seeds in each situation in test data.

As experiments settings, we employ a batch size of 2048, 5000 epochs maximum, the learning rate (both the generator and critic) of 10^{-5}, and the Adam optimizer. Moreover, the balance of learning chance of the generator and critic (two time-scale update rule [42]) is set to $1:5$. In addition, to improve computational efficiency, models are saved for every 10 epochs, and only those models could be used for tests.

5 Results

The results of the Kullback-Leibler Divergence (KLD) and Mean Squared Error (MSE) between fake and real distributions are presented in Tables 2 and 3, respectively. Each row corresponds to a randomly selected ticker representing a listed company. The best performances for each ticker are indicated in bold. It is worth noting that both S-GAN and DCGAN exhibit no finite KLD due to a mentioned reason.

Regarding the KLD metric, the performance of PGSGAN and PGSGAN-HL varies across different tickers. However, our proposed models consistently outperform the others. All models achieve successful MSE measurements. Based on the MSE results, our PGSGAN-HL demonstrates superior performance across all selected tickers. Furthermore, PGSGAN also outperforms S-GAN and DCGAN for all tickers.

Previous studies have shown that S-GAN outperformed DCGAN. However, when compared to our proposed model, the performance of S-GAN is significantly limited.

To further inspect details of the distribution comparison, as shown in Figures 6 – 15, we also make detailed figures of the generated orders' distributions.

Table 2: All the results of KLD

Ticker	PGSGAN (KLD)	PGSGAN-HL (KLD)	S-GAN (KLD)	DCGAN (KLD)
5901 JP	0.208467 ± 0.000107	$\mathbf{0.168823 \pm 0.000128}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
5333 JP	0.257479 ± 0.000121	$\mathbf{0.203760 \pm 0.000121}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
8355 JP	$\mathbf{0.136612 \pm 0.000071}$	0.139955 ± 0.000061	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
5631 JP	0.215376 ± 0.000064	$\mathbf{0.209126 \pm 0.000077}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
9532 JP	0.198947 ± 0.000053	$\mathbf{0.190806 \pm 0.000078}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
7012 JP	0.150013 ± 0.000067	$\mathbf{0.138801 \pm 0.000096}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
2501 JP	0.243355 ± 0.000108	$\mathbf{0.216768 \pm 0.000173}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
4005 JP	$\mathbf{0.136136 \pm 0.000053}$	0.144642 ± 0.000071	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
7752 JP	0.164498 ± 0.000036	$\mathbf{0.123295 \pm 0.000046}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$
7911 JP	0.228341 ± 0.000083	$\mathbf{0.203080 \pm 0.000107}$	$\infty \pm \mathrm{NaN}$	$\infty \pm \mathrm{NaN}$

Table 3: All the results of MSE

Ticker	PGSGAN (MSE)	PGSGAN-HL (MSE)	S-GAN (MSE)	DCGAN (MSE)
5901 JP	0.000479 ± 0.000003	$\mathbf{0.000386 \pm 0.000003}$	0.019759 ± 0.000053	0.148968 ± 0.000005
5333 JP	0.002164 ± 0.000002	$\mathbf{0.000945 \pm 0.000005}$	0.004611 ± 0.000011	0.143241 ± 0.000007
8355 JP	0.000505 ± 0.000002	$\mathbf{0.000392 \pm 0.000001}$	0.012347 ± 0.000018	0.093950 ± 0.000016
5631 JP	0.000499 ± 0.000002	$\mathbf{0.000432 \pm 0.000003}$	0.027200 ± 0.000035	0.107452 ± 0.000026
9532 JP	0.000300 ± 0.000000	$\mathbf{0.000278 \pm 0.000003}$	0.030383 ± 0.000029	0.145481 ± 0.000011
7012 JP	0.000332 ± 0.000003	$\mathbf{0.000257 \pm 0.000003}$	0.020080 ± 0.000042	0.145100 ± 0.000003
2501 JP	0.000428 ± 0.000002	$\mathbf{0.000391 \pm 0.000003}$	0.014947 ± 0.000039	0.146345 ± 0.000012
4005 JP	0.000512 ± 0.000004	$\mathbf{0.000393 \pm 0.000003}$	0.014715 ± 0.000017	0.109629 ± 0.000009
7752 JP	0.000866 ± 0.000003	$\mathbf{0.000691 \pm 0.000004}$	0.009458 ± 0.000015	0.141400 ± 0.000005
7911 JP	0.000580 ± 0.000003	$\mathbf{0.000473 \pm 0.000004}$	0.019275 ± 0.000029	0.107608 ± 0.000012

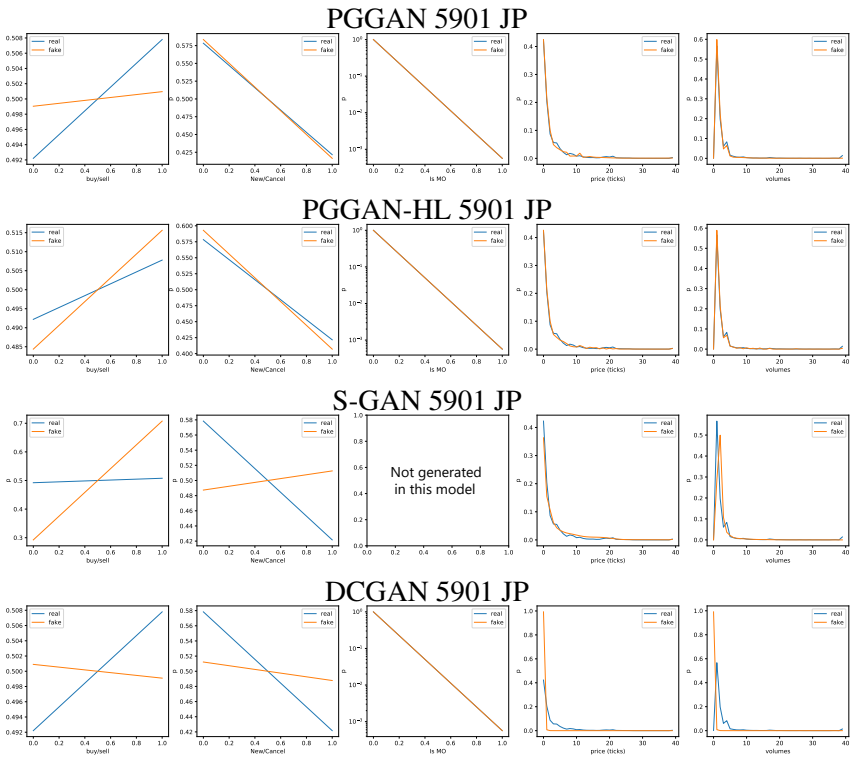

Figure 6: The results of real and fake data distribution from each models in 5901 JP. An example of the distribution of real (blue) and generated fake (orange) data. From left to right, each box shows the distribution of sell/buy, new/cancel, MO (or not), price (ticks from the opposite best price), and volume (divided by the minimum volume). In the leftmost box, 0.0 and 1.0 in the horizontal axis correspond to sell and buy, respectively. In the second left box, 0.0 and 1.0 correspond to new and cancel, respectively. In the middle box, 0.0 and 1.0 correspond to non-MO and MO, respectively. In the vertical direction, only the model is different; therefore, the real data (blue) is the same; only the vertical axis scale is different.

According to the figure, the failure in reproducing the volume and price distributions in DCGAN is notable.

Further, the failure in reproducing the tiny bumps in the price and volume distributions of S-GAN is interesting. Unlike PGSGAN/PGSGAN-HL, S-GAN has smoother distributions of price and volumes, which is the bigger difference from the real distribution than PGSGAN/PGSGAN-HL

6 Discussion

Our models not only outperform previous studies, but their superiority also suggests that implementing policy gradient in GANs for stock markets is beneficial. By adhering to the given rules of financial markets, it is more reasonable to map to a discrete space rather than a continuous one.

To handle the discrete nature of the problem, we utilize policy gradient to bridge the gradient gap between the generator and the critic. As supported by the theoretical discussion, we successfully incorporate the policy gradient into the GAN framework through experimental validation.

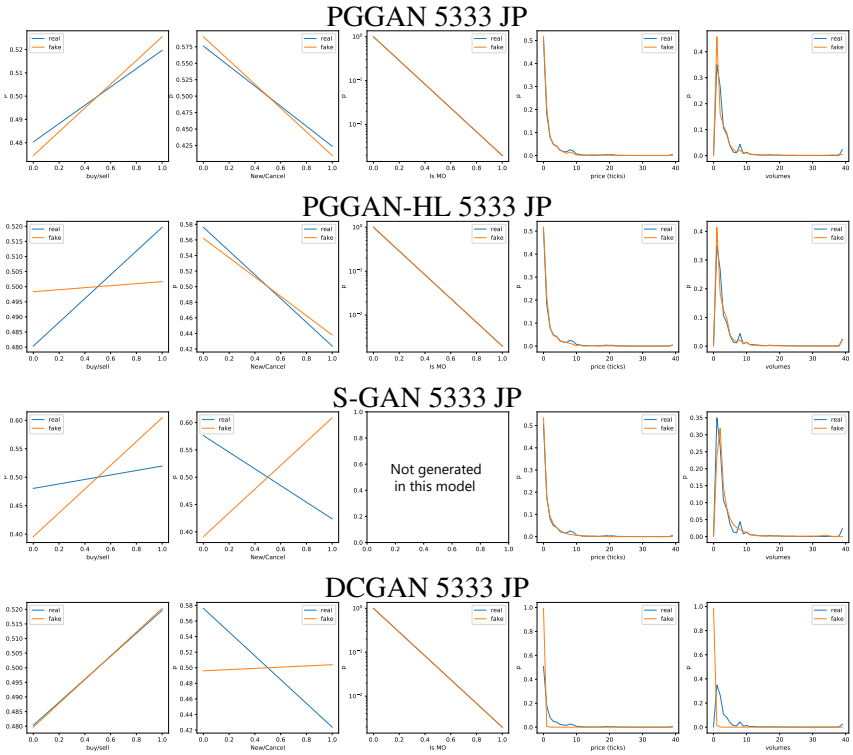

Figure 7: The results of real and fake data distribution from each models in 5333 JP.

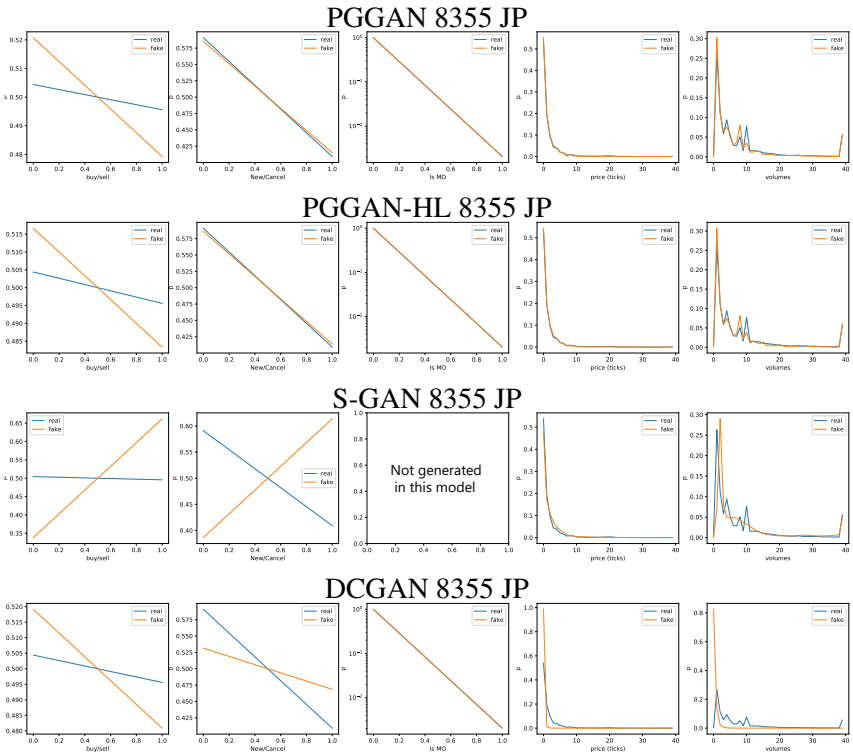

Figure 8: The results of real and fake data distribution from each models in 8355 JP.

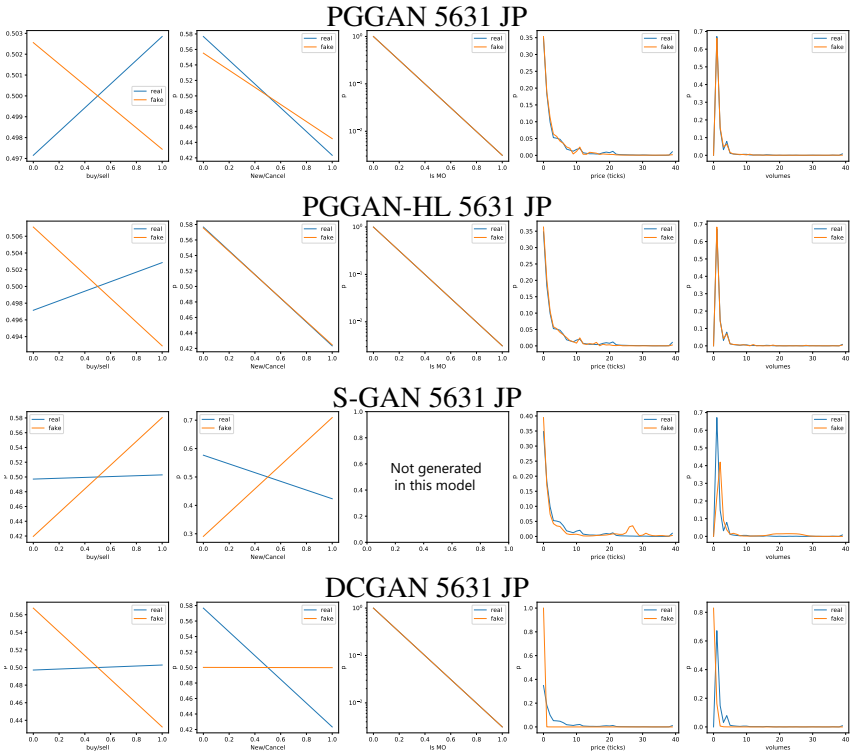

Figure 9: The results of real and fake data distribution from each models in 5631 JP.

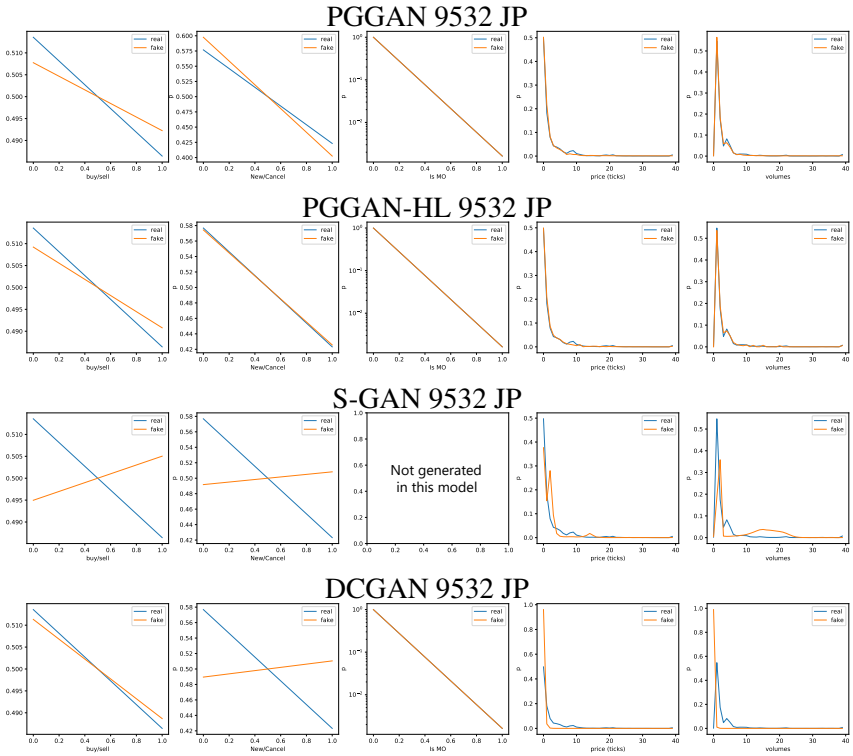

Figure 10: The results of real and fake data distribution from each models in 9532 JP.

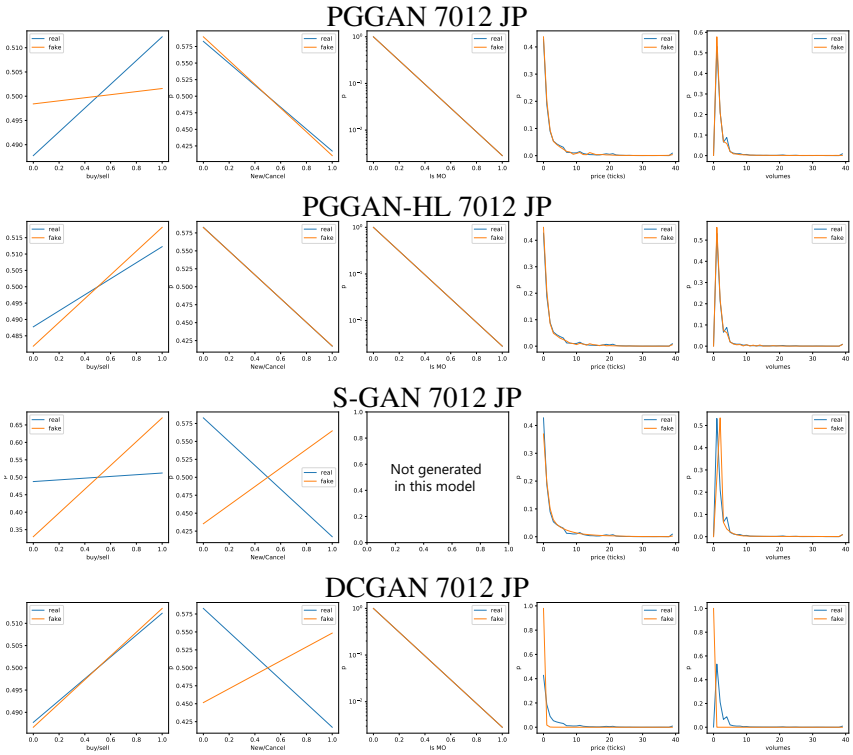

Figure 11: The results of real and fake data distribution from each models in 7012 JP.

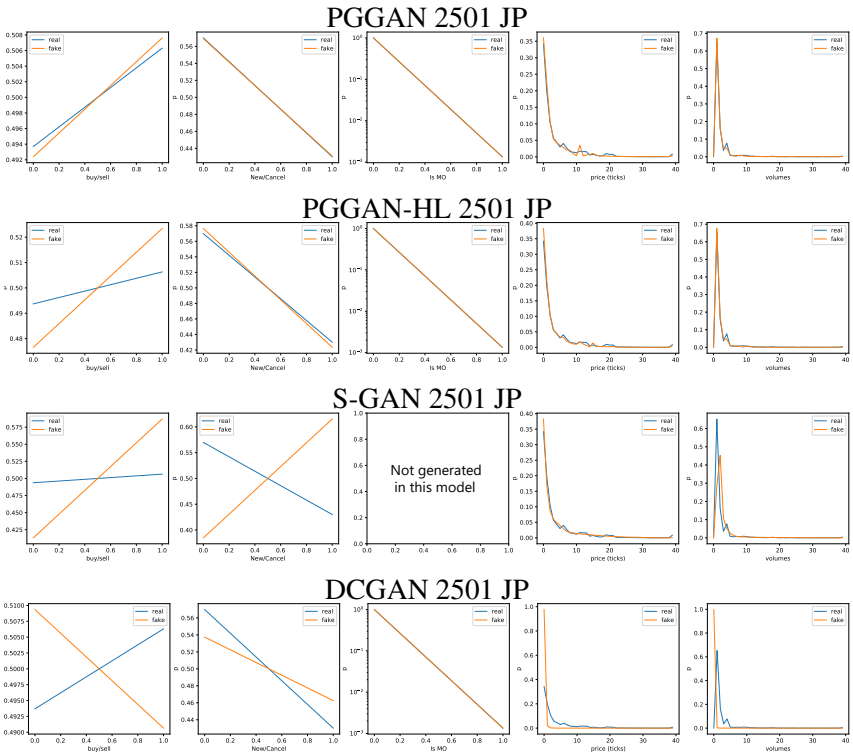

Figure 12: The results of real and fake data distribution from each models in 2501 JP.

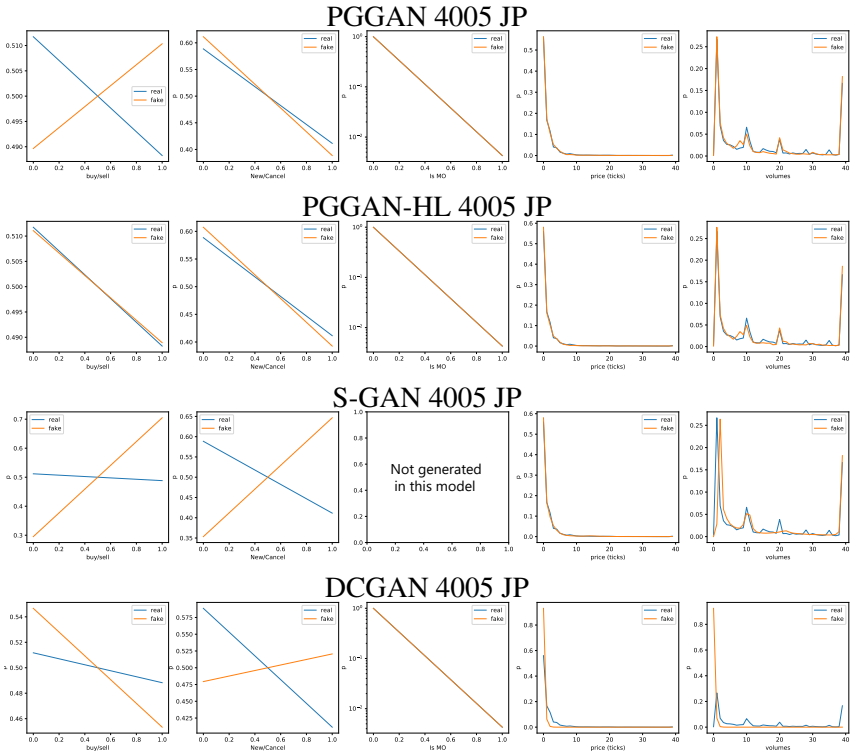

Figure 13: The results of real and fake data distribution from each models in 4005 JP.

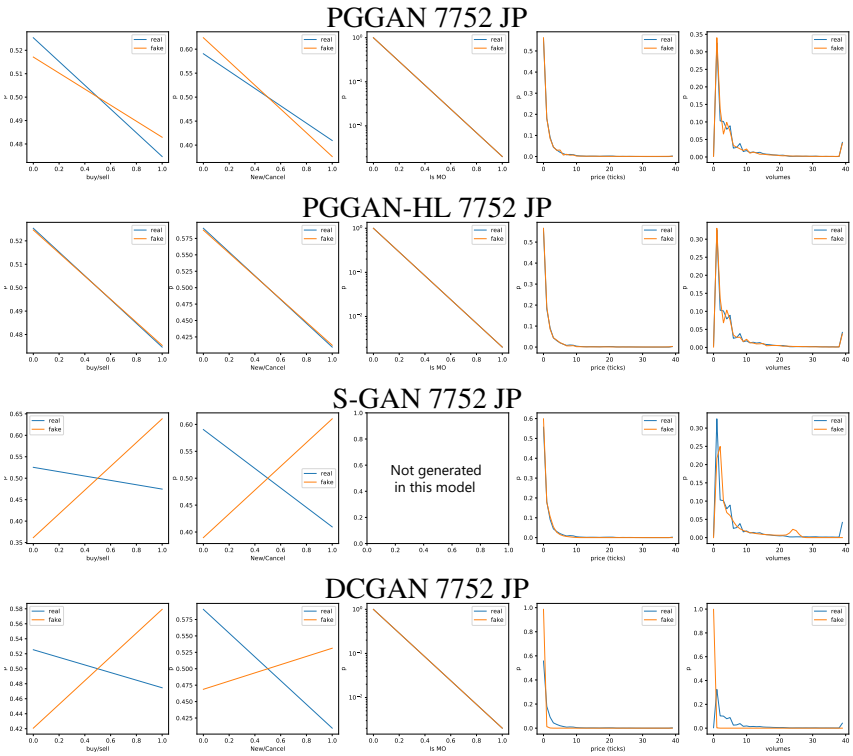

Figure 14: The results of real and fake data distribution from each models in 7752 JP.

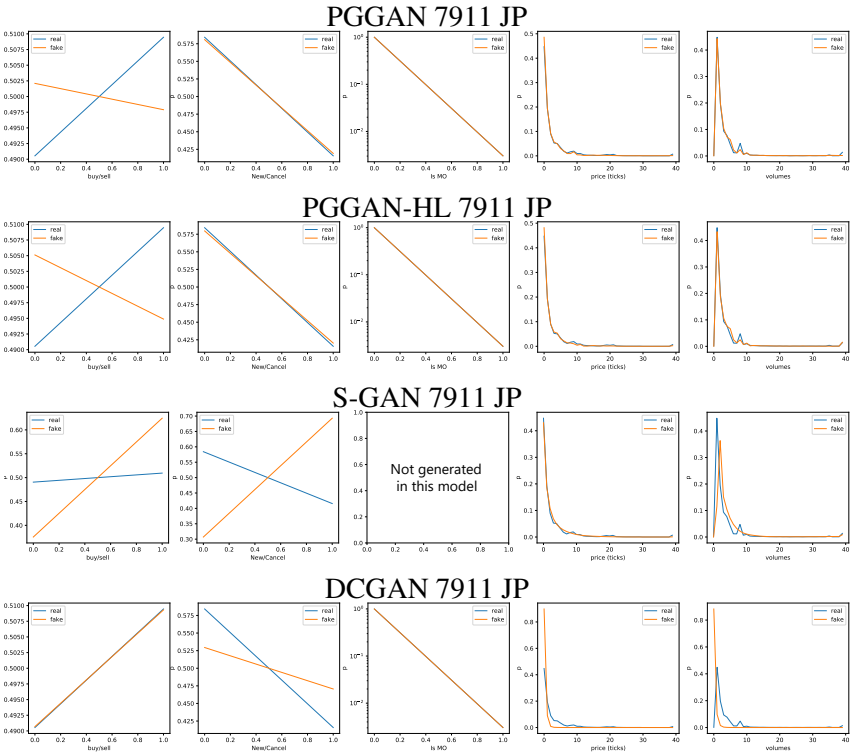

Figure 15: The results of real and fake data distribution from each models in 7911 JP.

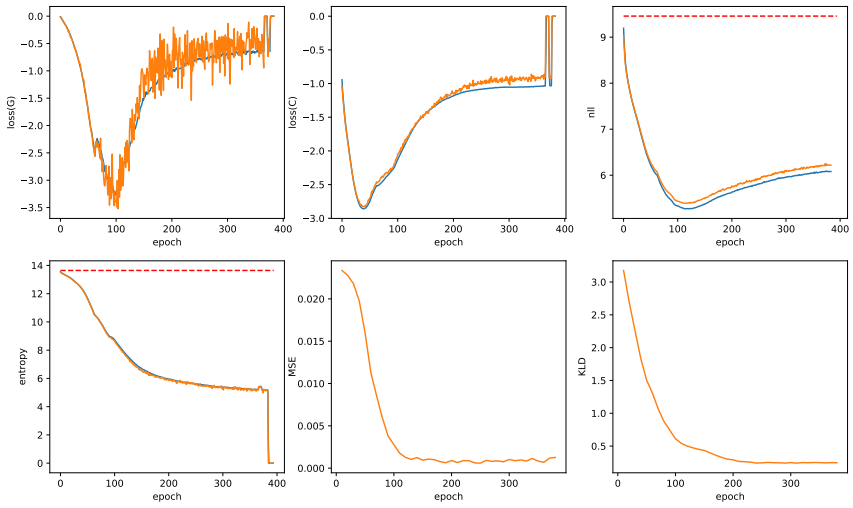

Figure 16: An example of the metrics in the learning process. (PGSGAN 5901 JP) From upper left to upper right, each box shows the loss of generator, loss of critic, and NLL ($\mathrm{NLL}_{G(z)}(x)$ where x is the real order), respectively. From lower left to lower right, each box shows the entropy of generated policy, the MSE between real and fake distributions, and the KLD between them, respectively. Horizontal axis represents epochs. Blue and orange lines correspond to train and valid results, respectively. The MSE and KLD are calculated only in the valid data. Moreover, a broken red line means the by-chance level explained previously.

To validate the importance of monitoring the learning status using generated policy entropy, we conducted experiments and plotted the metrics during the learning process of PGSGAN (as shown in Figure 16). It became evident that the loss values of the generator and critic were not suitable for monitoring the learning status because those losses have adversarial and relative nature. This is consistent with other GAN models, which often rely on external tasks to evaluate the learning progress.

In the case of image generation, metrics such as the Inception Score [43] and Fréchet Inception Distance [42] have been introduced to assess the learning status. However, defining such external tasks for evaluating the learning status can be challenging, especially in financial markets.

In our experiments shown in Figure 16, we observed that the entropy of the generated policy exhibited similar patterns to the MSE and KLD results, suggesting that the entropy can serve as a useful metric for monitoring the learning status.

While it is crucial for the generator to produce a distribution similar to the real distribution, this alone is not sufficient to determine the quality of the generator. As shown in our results, PGSGAN exhibits a better fake distribution that closely resembles the real distribution compared to other models. However, there is a risk of mode collapse, where the generator may either generate meaningless fake orders or mimic the real distribution without considering the underlying context.

To investigate the generator's capability to produce diverse and appropriate fake orders, we utilized the NLL metric ($\mathrm{NLL}_{G(z)}(x)$ where x represents the real order). By changing the fake seeds 100 times in the test, we assessed the generator's ability to generate various fake orders. If the NLL is similar to the by-chance level, the generator is likely to produce only meaningless fake orders. Conversely, if the NLL is consistently 0, the generator would fail to generate diverse likely fake orders and would solely produce the real orders. Moreover, if the NLL remains unchanged for different seeds, it suggests a high probability of mode collapse in the generated policy.

For PGSGAN/PGSGAN-HL, we computed the mean standard deviation of NLL for each generation scenario by changing seeds 100 times. The evaluation revealed that both PGSGAN and PGSGAN-HL exhibit an appropriate level of generation diversity, with an average standard deviation of NLL ranging from 2.5 to 3.6. This indicates that our proposed model generates a diverse range of orders without suffering from mode collapse.

In summary, our model satisfies both the necessary and sufficient conditions for good generation. While there is a possibility of further improving the model, it can be confidently stated that our model fulfills the requirements of a good generator and outperforms previous models.

The reason why PGSGAN-HL consistently outperforms PGSGAN in various experiments is the inclusion of Hinge loss. Unlike PGSGAN-HL, the gradient of the generator in PGSGAN approaches zero during the middle of the learning process. As described in Equation 10, the gradient of the generator depends on and is proportional to the output of the critic. If the generator produces fake data that is superior to the critic's discrimination ability, the critic's output becomes zero, resulting in the vanishing gradient problem and abrupt termination of learning for PGSGAN. The introduction of Hinge loss in PGSGAN-HL prevents this issue and allows for continued learning for a longer period compared to PGSGAN.

However, the shorter learning of PGSGAN due to its explicit end of learning may be beneficial when we use big data, although it is a trade-off for its performance. In the settings of our experiments, each model took the following learning time:

- PGSGAN: 3–5 days.
- PGSGAN-HL: About 20 days.
- S-GAN: About 3 months
- DCGAN: About 10 days.

These times are roughly calculated because they depend on the size of the data (stocks) and computational resources. These values are measured with high-end GPUs of NVIDIA Geforce RTX 20 series, such as 2080, 2070super, and 2080Ti. Whereas this is not accurate, it is beneficial for discussing the computational burden. According to these results, PGSGAN has a significantly short learning time. It suggests that PGSGAN is possibly beneficial when we employ significantly huge data for training, even if we sacrifice its performance a little bit.

As future work, higher performance GANs, evaluation methods better than MSE/KLD, or application of our GANs, can be pursued. We have only focused on the next orders' generation to simplify the problem. However, the challenge to make longer time series needs to be addressed.

7 Conclusion

This study proposes a new GAN framework specifically designed for generating realistic orders in financial markets. Unlike previous works that generated fake orders in continuous spaces, our approach recognizes that real orders in financial markets are inherently discrete. For example, price and volumes have minimum units, and order types such as sell/buy are categorically distinct. Consequently, it is inappropriate to treat these aspects as continuous variables and combine them seamlessly in state spaces.

To address this, we modified the generation of fake orders to be discrete, which required a departure from the traditional GAN learning algorithm. Instead, we leveraged policy gradient, a common technique in reinforcement learning, to train our models. By incorporating the relationship between the generator and critic into the reinforcement learning framework, we made policy gradients applicable to stock market GANs.

In our proposed models, the generator produces a policy, and based on that policy, randomly sampled fake orders are evaluated by the critic. We conducted experiments using order data from the Tokyo Stock Exchange, consisting of over 65 million order records. To assess the performance of our models, Policy Gradient Stock GAN (PGSGAN) and Policy Gradient Stock GAN with Hinge loss (PGSGAN-HL), we compared the distributions of the generated fake orders to those of the real orders using metrics such as Mean Squared Error (MSE) and Kullback-Leibler Divergence (KLD). The results demonstrate that our models outperform previous approaches.

In addition to superior performance, we observed a side benefit of introducing the policy gradient framework. The entropy of the generated policy can serve as an indicator to monitor the learning status of the GAN, providing insights into the diversity and quality of generated orders.

Furthermore, by incorporating Hinge loss in PGSGAN-HL, we mitigate the issue of gradient vanishing that may occur during learning. This combination proves to be beneficial, enabling more effective learning compared to PGSGAN alone.

As future work, it is worth exploring higher-performing GAN architectures, evaluation methods beyond MSE and KLD, and potential applications of our GAN models in different contexts within the field of financial markets.

Acknowledgment

We thank the Japan Exchange Group, Inc. for providing the data. This work was supported by JSPS KAKENHI Grant Number JP 21J20074 (Grant-in-Aid for JSPS Fellows).

References

[1] J. Li, X. Wang, Y. Lin, A. Sinha, and M. Wellman, "Generating Realistic Stock Market Order Streams," *AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, pp. 727–734, 2020.

[2] Y. Naritomi and T. Adachi, "Data Augmentation of High Frequency Financial Data Using Generative Adversarial Network," in *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. IEEE, 2020, pp. 641–648.

[3] M. HIRANO, H. SAKAJI, and K. IZUMI, "Policy Gradient Stock GAN for Realistic Discrete Order Data Generation in Financial Markets," in *14th IIAI International Congress on Advanced Applied Informatics*. IEEE, 2023, pp. 361–368.

[4] I. Maeda, D. deGraw, M. Kitano, H. Matsushima, H. Sakaji, K. Izumi, and A. Kato, "Deep reinforcement learning in agent based financial market simulation," *Journal of Risk and Financial Management*, vol. 13, no. 4, p. 71, 2020.

[5] S. M. Edmonds and Bruce, "Towards Good Social Science," *Journal of Artificial Societies and Social Simulation*, vol. 8, no. 4, 2005.

[6] J. D. Farmer and D. Foley, "The economy needs agent-based modelling," *Nature*, vol. 460, no. 7256, pp. 685–686, 2009.

[7] S. Battiston, J. D. Farmer, A. Flache, D. Garlaschelli, A. G. Haldane, H. Heesterbeek, C. Hommes, C. Jaeger, R. May, and M. Scheffer, "Complexity theory and financial regulation: Economic policy needs interdisciplinary network analysis and behavioral modeling," *Science*, vol. 351, no. 6275, pp. 818–819, 2016.

[8] T. Mizuta, "An Agent-based Model for Designing a Financial Market that Works Well," 2019, http://arxiv.org/abs/1906.06000.

[9] T. Mizuta, S. Kosugi, T. Kusumoto, W. Matsumoto, K. Izumi, I. Yagi, and S. Yoshimura, "Effects of Price Regulations and Dark Pools on Financial Market Stability: An Investigation by Multiagent Simulations," *Intelligent Systems in Accounting, Finance and Management*, vol. 23, no. 1-2, pp. 97–120, 2016.

[10] M. Hirano, K. Izumi, T. Shimada, H. Matsushima, and H. Sakaji, "Impact Analysis of Financial Regulation on Multi-Asset Markets Using Artificial Market Simulations," *Journal of Risk and Financial Management*, vol. 13, no. 4, p. 75, 2020.

[11] X. Zhou, Z. Pan, G. Hu, S. Tang, and C. Zhao, "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets," *Mathematical Problems in Engineering*, vol. 2018, 2018.

[12] K. Zhang, G. Zhong, J. Dong, S. Wang, and Y. Wang, "Stock Market Prediction Based on Generative Adversarial Network," vol. 147, pp. 400–406, 2019.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[14] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," *arXiv*, vol. 1411, no. 1784, pp. 1–7, 2014, http://arxiv.org/abs/1411.1784.

[15] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," 2015, http://arxiv.org/abs/1511.06434.

[16] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *IEEE international conference on computer vision*, 2017, pp. 2794–2802.

[17] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," in *30th International Conference on Neural Information Processing Systems*, 2016, pp. 271–279.

[18] E. Denton, S. Chintala, A. Szlam, and R. Fergus, "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 1486–1494, 2015.

[19] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *International conference on machine learning*, 2016, pp. 1558–1566.

[20] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[21] C. Chu, A. Zhmoginov, and M. Sandler, "CycleGAN, a Master of Steganography," 2017, http://arxiv.org/abs/1712.02950.

[22] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[23] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," in *6th International Conference on Learning Representations*, 2018.

[24] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[25] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial Feature Learning," 2016, https://arxiv.org/abs/1605.09782.

[26] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery," *Lecture Notes in Computer Science*, vol. 10265, pp. 146–147, 2017.

[27] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient GAN-Based Anomaly Detection," 2018, http://arxiv.org/abs/1802.06222.

[28] D. Li, D. Chen, J. Goh, and S.-k. Ng, "Anomaly detection with generative adversarial networks for multivariate time series," 2018, https://arxiv.org/abs/1809.04758.

[29] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017, https://arxiv.org/abs/1701.07875.

[30] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," 2017, http://arxiv.org/abs/1701.04862.

[31] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs Montreal Institute for Learning Algorithms," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5767–5777, 2017.

[32] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *6th International Conference on Learning Representations*, 2018.

[33] I. H. Witten, "An adaptive optimal controller for discrete-time markov environments," *Information and control*, vol. 34, no. 4, pp. 286–295, 1977.

[34] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983.

[35] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[36] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *The 32nd International Conference on Machine Learning*, vol. 1, 2015, pp. 448–456.

[38] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," 2016, http://arxiv.org/abs/1607.06450.

[39] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, https://arxiv.org/abs/1511.07122.

[40] J. H. Lim and J. C. Ye, "Geometric GAN," 2017, http://arxiv.org/abs/1705.02894.

[41] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[42] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[43] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," Tech. Rep., 2016.