Policy Gradient Stock GAN for Realistic Discrete Order Data Generation in Financial Markets

Masanori HIRANO  
School of Engineering  
The University of Tokyo  
Tokyo, Japan  
research@mhirano.jp

Hiroki SAKAJI  
School of Engineering  
The University of Tokyo  
Tokyo, Japan  
sakaji@sys.t.u-tokyo.ac.jp

Kiyoshi IZUMI  
School of Engineering  
The University of Tokyo  
Tokyo, Japan  
izumi@sys.t.u-tokyo.ac.jp

Abstract—This study proposes a new generative adversarial network (GAN) for generating realistic orders in financial markets. In some previous works, GANs for financial markets generated fake orders in continuous spaces because of GAN architectures’ learning limitations. However, in reality, the orders are discrete. For example, the orders have minimum order price unit, or order types. Thus, we change the generation method to place the generated fake orders into discrete spaces in this study. Because this change disabled the ordinary GAN learning algorithm, this study employed a policy gradient, frequently used in reinforcement learning, for the learning algorithm. In our experiments, we processed significant volumes of order data over half a year and evaluated our model. Through our experiments, we show that our proposed model outperforms previous models in generated order distribution. As an additional benefit of introducing the policy gradient, the entropy of the generated order distribution is increased, which improves the evaluation of the generated orders. In the future, higher performance GANs, better evaluation methods, or the applications of our GANs can be addressed.

Index Terms—Generative adversarial networks (GAN), Financial markets, Policy gradient, Order generation

I. INTRODUCTION

In financial markets, the realized order time series is a very tiny part of possible state spaces. In other words, there is only one path that has been realized among the various paths of market movements. Although the pattern of one order is limited, the state space could be huge when the orders are compiled. In addition, due to the lack of stationarity of financial markets, all possible states could not be realized in the past data, or data for some possible states could not be obtained enough.

Obtaining much more data would give us various advantages. For example, we could improve trading strategy performance based on much more data. Backtesting with more data could improve the accuracy of the estimation of risk levels. Moreover, a more accurate portfolio evaluation using various augmented time-series price paths would be available.

Although there are significant volumes of data in the financial market, it is still not enough, and many studies have been conducted to address the insufficiency of data. To this end, the prominent approach is data augmentation via GANs.

This study focuses on GANs for making realistic order time series in financial markets, especially in stock markets.

Although some research focuses on GANs for stock markets, the generated fake data remain unrealistic. Stock-GAN (S-GAN) [1] and Market GAN [2] are some previous works for stock markets. To train GANs, gradient connection between the generator and the critic is required. This requirement causes the generator outputs (the generated fake data) to be unrealistic. In the case of S-GAN, the values generated for buy/sell, types, prices, and volumes of orders, are continuous. It is easy for humans to identify such fake data because the actual orders are not continuous. While Market GAN generates discrete values and calculates the class probabilities, the generated fake data remain unrealistic because humans can easily distinguish between probabilistic class data and real data. The most important problem of current GANs for financial markets is that the generated fake data are placed in continuous space. For example, you can notice that the following generated orders are fake:

- A buy order has a probability of 0.6. (Occured in S-GAN and Market GAN)
- An order whose price is $100.0123 in the market, whose minimum order price unit (price tick size) is $0.01. (Occured in S-GAN)

In our opinion, also generated orders should be order-system-acceptable. Please imagine when you are trying to make a new order in a real market. It is possible that you make a strategy to make a new order with probability in your mind. However, when you submit your order via the ordering system, you have to decide your order is buy or sell.

In addition, there is also a major problem with treating buy/sell and order types as continuous values, and joining their state spaces next to each other. If buy/sell was expressed as [0, 1], buy and sell would be continuously joined in terms of their 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.

Thus, this discreteness of financial market orders should be incorporated into the design of the GANs, especially in the generators.

The most important part of solving the problem is the necessity of gradient connection between the generator and...
critic of GANs for training. Suppose the gradient connection between the generator and critic is not required. In that case, the generator can make more realistic orders without continuous unrealistic values or make sampled fake orders from class probability estimation.

For solving the current issue, this study proposes Policy Gradient Stock GAN (PGSGAN), a new GAN learning method for stock markets using policy gradient. Policy gradient is a learning algorithm that is frequently used in reinforcement learning. By incorporating the relationship between the generator and critic of GANs into the concept of reinforcement learning, we make policy gradient available in GANs for stock markets. This introduction enables GANs to remove the gradient connectivity between their generators and critics, and make more realistic discrete orders.

In our experiments, we employed more than 65 million order data over half a year from 10 stocks and built and evaluated our GAN model. Consequently, we successfully design the generator output of the GAN with more realistic market trading rules and improve the generation performance. Because this technology enables to augment of more realistic data, it is expected to improve the learnability of prediction tasks and other tasks via machine learning methods.

Although PGSGAN is designed according to the rules of the Tokyo Stock Exchange (TSE), it can also be applied to other markets with some small changes.

II. RELATED WORK

As mentioned above, S-GAN proposed in [1] was a GAN for stock markets, designed to generate realistic order time series and also generate best prices without real order book data by continuous double auction (CDA) network. Naritomi et al. [2] showed a basic GAN model for stock markets and that its generated data are beneficial for predicting the future price movement. Moreover, there exist studies which attempted to use GAN architecture directly for future price predictions [3], [4]. As observed from those studies, currently, GANs’ approaches are more practical.

The technology related to GANs has been improving, in the following respects. Initially, Goodfellow et al. [5] proposed the original GAN. Then, Mirza et al. [6] proposed the conditional GAN, whose idea was also used in this study, using conditional inputs. Radford et al. [7] proposed a deep convolutional GAN (DCGAN), which was also used in this study as a comparison model. Moreover, Yu et al. [8] proposed SeqGAN for sequence generation, such as text or music, using policy gradient. Although it could seems similar to our study, our GAN is not generating sequences. As an extension of GANs, adversarial feature learning [9] for embedding images into vectors and anomaly detections based on GAN [10]–[12] were proposed. Wasserstein GAN (WGAN) [13] was suggested based on the discussion of the learning stability of GANs [14]. Our study is based on WGAN. Gradient penalty [15] and spectral normalization [16] were the tools proposed for stabilizing the WGAN. In this study, thus, we employed spectral normalization.

III. MODELS

A. Policy Gradient Stock GAN (PGSGAN)

PGSGAN uses historical data and current conditions to generate the next order in financial markets. It is based on GAN [5] and policy gradient theorem [17]–[19]. More technically, we used Wasserstein GAN (WGAN) [13] as a basis, REINFORCE [20] with a baseline as a policy gradient algorithm, and convolutional neural network (CNN) as a part of our neural network. We also used batch normalization [21], layer normalization [22], and spectral normalization [16] for stabilizing the learning.

1) PGSGAN Architecture: Figure 1 indicates the outline of our PGSGAN.

The generators accept conditional data (historical data of markets) and random seeds for a generation. In our experiments, the conditional data comprises the last 20 order time series and the current best sell and buy prices. The order time series contains information, such as buy/sell, new/cancel, market order (or not), price (ticks from the best price), volume (scaled with dividing by the minimum volume unit), and best prices before the order. The fake seed has 128 dimensions and is randomly generated.

Then, the generator makes a policy for generating the fake next order. In our experiment, the generator used 14 convolutional layers and 5 linear layers to make 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)

After calculating the probabilistic policy, the generator makes a fake next order by weighted sampling, according to the policy.

However, the critic only maps the inputs into the scalar under the 1-Lipschitz constraint. This is the same as the basic WGAN. The critic accepts two inputs: conditional data (the historical data, same as the generator), and either the generated fake next order or the real next order. In our experiments, the real next orders were also converted into the above-explained range. This implies that only the price and volume of the real

\(^1\)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.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
2) **PGSGAN Learning Mechanism**: Because of the sampling process, the gradient connection between the generator and the critic is lost. Traditional GANs, especially all GANs for stock markets, rely on the gradient connection between their generators and critic for training their generator. However, our PGSGAN completely abandoned the connection and disabled the traditional learning theory for the generator, because of the sampling process for generating a fake order, based on a generated policy.

Thus, as a new learning theory for the generator, we employed the policy gradient widely used in reinforcement learning.

In the following, we use the 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
- \( C(x) \): 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 \( G(z) \).
- \( \theta_G, \theta_C \): params in the critic and generator, respectively.
- \( L_C, L_G \): loss function for the critic and generator.
- \( \| f \|_{L_1} \): 1-Lipschitz constraint for any function \( f \).
- \( \rho_{G(z)}(\tilde{x}) \): probability for the sampled fake order \( \tilde{x} \) according to the generated policy \( G(z) \).
- \( \text{NLL}_{G(z)}(\tilde{x}) \): negative log-likelihood for the sampled fake order \( \tilde{x} \) according to the generated policy \( G(z) \).

At first, PGSGAN will solve the following minimax game:

\[
\min_G \max_{||C||_{L_1} \leq 1} L_{GAN}(G, C),
\]

where

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

This form is similar to the original form of WGAN. However, the sampling term \( \mathbb{E}_{x \sim \mathbb{P}_r}[C(x)] \) has been added. This change is substantial for generator learning.

For the critic, the objective function is:

\[
\max_{||C||_{L_1} \leq 1} \left\{ \mathbb{E}_{x \sim \mathbb{P}_r}[C(x)] - \mathbb{E}_{x \sim G(z)}[C(\tilde{x})] \right\} \tag{2}
\]

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

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

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.

\(^2\) Correctly, it also accepts conditional data, but it is ignored in this notation for simplicity.
3) Additional Note and Actual Implementation for PGS-GAN: In PGS-GAN, we employ spectral normalization 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.

Figure 3 shows the details of our generator. We employ CNN as a basic foundation of our model. In the conditional layer, for the latter processing part of historical data, we employ average pooling. It is because this part processes data in the direction of time sequence. In very high-frequency trading, the sequence of some orders is not significant. Thus, we employ average pooling to buffer these orders. After the concatenate of processed historical data and the current best price, we employ a dilated convolution [23] and circled convolution, because the concatenated two inputs should be mixed equivalently. As the outputs, we employ logits for the convenience of calculation. As mentioned above, the loss function for the generator is calculated as equation 10. Thus, for compatibility of the negative log-likelihood (NLL), the logits are best for less computational error. Thus, 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 [16] is applied.

Basic architectures of the critic are almost the same as the generator, except for the final layers and each dimension.

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

We also implement the PGS-GAN with Hinge loss. Originally, Hinge loss was used in WGAN in Geometric GAN [24]. Hinge loss is defined as

\[
\begin{align*}
\max(x + 1, 0) & \quad \text{(critic training for fake data)} \\
\max(1 - x, 0) & \quad \text{(critic training for real data)} \\
x & \quad \text{(generator training)}
\end{align*}
\]

and insert into the last of the critic layers.

The others are the same as PGS-GAN.

C. Comparative Models

1) Stock GAN (S-GAN): Stock GAN (S-GAN) was proposed by [1], and based on WGAN-GP [15]. S-GAN has LSTM for processing conditional data (historical data) and CNN for processing the LSTM output, and either seed for a generation or the next order (fake/real) for a critic. In this study, we replicate this model as a comparative one. However, to evaluate fairly, we modify some of its architecture:

- Deletion of continuous double auction (CDA) network: This network was originally employed for updating best prices after the new order. However, in this study, we assume a situation that can use all market data. Thus, the best price estimation by CDA network is not required.
- Deletion of time signal: The original S-GAN accepts the time signal, which aims to identify when, in one day, the order was placed among the divided 24 classes as one of the inputs. However, the TSE, which we target in this study, has only 2.5-hours sessions (2 sessions per day) and does not have 24/7 markets. Thus, we decide to delete this.
- Price processed in this study changed from absolute value to relative scaled price: In the original study, it was generated for a very limited period of time (assumed to be about one day); thus, the importance of relativizing the price level was not too high. However, in this study, we target very long periods over half a year. Therefore, we also change the price inputs to ticks from the best price.
- Deletion of time since previous order: On a tick-time scale, the order arrival interval should be modeled separately because some orders are published simultaneously, and their sequence and interval may not be significant. Thus, to simplify the problem we address in this study, we ignore this interval prediction.

The other architectures remain the same.

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.

2) DCGAN: As another comparative model, we employ a well-known DCGAN. This model generates the order with continuous values similar to S-GAN. The architecture is based on CNN. Moreover, we also converted generated output to discrete values as the same as S-GAN.

IV. Experiments

In our experiments, we randomly selected 10 stocks under the criteria. The basic premise is that we target only Tokyo Stock Exchange (TSE) in this study. Thus, all stock candidates are listed on the TSE.

The first criterion is that the stocks must be included in Nikkei 225. Nikkei 225 3 is one of the major indices in TSE. The stocks included in Nikkei 225 are selected in terms of their liquidity and sector balance, indicating that they have enough liquidity and are traded frequently. In this study, we aim to generate realistic tick-scale orders. Thus, liquidity is required. The stocks included in this index are renewed when a stock is unlisted on TSE, and also periodically renewed every October.

The second criterion is that the stocks must not be included in TOPIX 100. TOPIX 100 is also one of the major indices in TSE and selected by Japan Exchange Group 4. TOPIX index series have some categories and some components. TOPIX

3https://indexes.nikkei.co.jp/en/nkave/index/profile?idx=nk225
4https://www.jpx.co.jp/english/markets/indices/topix/
100 includes the TOP-100 stocks whose total market value and liquidity are very high. It is very special for our study because the stocks included in TOPSX 100 are treated as special stocks in terms of the trading rule. These stocks have a smaller minimum order price unit (price tick size). Thus, we decide to ignore those included in TOPIX 100 due to the technical issue.

The third criterion is that the stocks are included in TOPIX 225, but not included TOPSX 100, stably in 2018 – 2020 period. It is because acceptance or deletion by indices have a significant impact on trading volume.

The last criterion is that the stocks have the same price tick size through the data periods. In TSE, the price tick size changes according to the price range. It changes at the price tick size through the data periods. In TSE, the price tick size is 0.01 yen, but not included TOPIX 100, stably in 2018 – 2020 period. It is because acceptance or deletion by indices have a significant impact on trading volume.

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

The total volumes of tick data we processed in this study are more than 65 million. Compared with the previous major work, such as Stock-GAN (S-GAN) [1], the data size we employed in this study is huge. In Stock-GAN (S-GAN) [1], there are two selected stocks in the experiment – Alphabet Inc. (GOOG) and Patriot National Bancorp Inc. (PNBK). In the study, GOOG had 230,000 orders, whereas PNBK had only 20,000 orders. S-GAN was tested for only almost one day. However, in this study, longer periods (9 months) and bigger data are tested. To enable this huge data processing, we built a golang-based backend data preprocessing server and a PyTorch-based python program for deep learning with gRPC over 10G ethernet between them.

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. For PGGAN, we calculate and inspect the negative log-likelihood (NLL) for real order and the entropy of the generated policy. The NLL is $\text{NLL}(G(x)) = −\ln \{1/(2 \times 2 	imes 2 \times 40 \times 40)\} \approx 9.45$. On the contrast, the entropy is defined as

$$H(G(z)) := \sum_{x \in \mathbb{X}} -p_G(z|x) \log_2 p_G(z|x),$$

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 PGGAN. Theoretically, the by-chance entropy is

$$\sum_{x \in \mathbb{X}} \frac{1}{2 \times 2 \times 2 \times 2 \times 40 \times 40} \log_2 \frac{1}{2 \times 2 \times 2 \times 2 \times 40 \times 40} = \log_2 (2 \times 2 \times 2 \times 2 \times 40 \times 40) \approx 13.64.$$
for all classes ($2 \times 2 \times 2 \times 40 \times 40$). Kullback–Leibler divergence is defined as:

$$D_{KL}(P,Q) = \sum_{x \in \mathcal{X}} P(x) \log_2 \left( \frac{P(x)}{Q(x)} \right),$$

(15)

where $\mathcal{X}$ 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 $P(x) \log_2 \left( \frac{P(x)}{Q(x)} \right)$ is calculated as 0 when $P(x) = Q(x) = 0$, KLD would be infinity if $\exists x, P(x) \neq 0, Q(x) = 0$ 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 discrete 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 $10^{-3}$, and the Adam optimizer. Moreover, the balance of learning chance of the generator and critic (two time-scale update rule [26]) 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.

V. RESULTS

Tables I and II show all the results of KLD and MSE, respectively, between fake and real distributions. Each row shows a randomly selected ticker (listed company). The best performances for each ticker are written in bold. S-GAN and DCGAN have no finite KLD by the reason mentioned above.

In terms of KLD, the performances of PGSGAN and PGSGAN-HL depend on tickers. However, our proposed models outperform others. All models successfully have MSE measures. According to the results of MSE, our PGSGAN-HL shows the best performances in all the selected tickers. Moreover, PGSGAN also outperforms S-GAN and DCGAN in all tickers.

As shown in previous studies, the S-GAN outperformed the DCGAN. However, compared with our proposed model, the performances of S-GAN are very limited.

To further inspect details of the distribution comparison, as shown in figure 4, we also make detailed figures of the generated orders’ distributions. Here, we only show one example from 5901 JP.

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.

VI. DISCUSSION

Our models surpass that of previous studies. PGSGAN-HL and PGSGAN outperforming the previous works implies that our implementation of policy gradient is beneficial for GANs for stock markets.

Under the given rules of the financial markets, mapping to a discrete space is more reasonable than mapping to a continuous space. Of course, the price or volumes can be placed on the continuous space. However, possible order space is completely discrete, even in price and volume.

For that implementation of discrete space, we employ the policy gradient, to fill in the disconnection of the gradient between the generator and the critic. As shown in the theoretical discussion, we have also succeeded in incorporating the policy gradient into the GAN in experiments.

As explained, generated policy entropy can be used to monitor the current learning status. To validate this by experiments, we plot each metric during the learning process of PGSGAN in figure 5. Obviously, the losses are not beneficial for monitoring the learning status because the generator and the critic are adversarial, and their losses are merely relative. This resembles other usual GANs, which usually employ the outer task to monitor the status. For an example of image generations, the inception score [27] and Fréchet Inception Distance [26] are introduced to evaluate the current learning level. However, setting these outer tasks for evaluating the learning status is complicated, especially in financial markets. According to the result in figure 5, the plot of the entropy is very similar to the MSE and KLD results. Thus, our experiments show that the entropy of generated policy is also beneficial.

Whereas the generated distribution is similar to the real distribution, this is a necessary but not a sufficient condition for a good generator. As we showed and explained in the results, our PGSGAN has a better fake distribution similar to the real distribution than other models. However, there is a
where $x$ is the real order, respectively.

From lower left to lower right, each box shows the entropy of generated orders, we employ NLL ($NLL_{G(z)}(x)$ where $x$ is the real order) as the metric. If the NLL is almost the same as the by-chance level, the generator would make only meaningless fake orders. By contrast, if the NLL is 0, the generated would make only the real orders and fail to make various likely fake orders. Moreover, if the NLL for one situation is always the same, not depending on the seeds, it would have a high possibility of mode collapse of generated policy. Thus, we calculate the mean of the standard deviation of NLL on each generating situation by changing seeds 100 times. This evaluation is only possible for PGSGAN/PGSGAN-HL. Thus, we find that PGSGAN/PGSGAN-HL shows the appropriate level of a variety of generations. The standard deviation of NLL for each generation by changing seeds is roughly 2.5–3.6 on average. Because the mean of NLL is roughly 5–6.5, the deviation of NLL is moderate enough. Moreover, the NLL is also moderate level. This result indicates that our proposed model has a sufficient variety of generations and mode collapses do not occur.

In summary, our model fulfills the necessary and sufficient conditions for good generation at the least level. Although there is a possibility of making a better model, we can say that our model fulfills the requirements of a good generator and performs better than the previous ones.

Lastly, the reason PGSGAN-HL showed better performances in many experiments than PGSGAN is the existence of Hinge loss. Different from PGSGAN-HL, the gradient of the generator of PGSGAN will be almost 0 in the middle of learning. As described earlier in equation 10, the gradient of the generator depends on and is proportional to the output of the critic. If the generated fake data is more perfect than the critic’s discrimination, the critic’s output will be 0 (See

![Fig. 5. 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 ($NLL_{G(z)}(x)$ 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.]

Table I: All the Results of KLD

| Ticker | PGSGAN (KLD) | PGSGAN-HL (KLD) | S-GAN (KLD) | DCGAN (KLD) |
|--------|--------------|-----------------|-------------|-------------|
| 5901 JP | 0.208467 ± 0.000101 | 0.168823 ± 0.000128 | ∞ ± NaN | ∞ ± NaN |
| 5333 JP | 0.257479 ± 0.000121 | 0.203760 ± 0.000121 | ∞ ± NaN | ∞ ± NaN |
| 8355 JP | 0.136612 ± 0.000071 | 0.139555 ± 0.000061 | ∞ ± NaN | ∞ ± NaN |
| 5631 JP | 0.215376 ± 0.000064 | 0.209126 ± 0.000077 | ∞ ± NaN | ∞ ± NaN |
| 9532 JP | 0.198947 ± 0.000053 | 0.190806 ± 0.000078 | ∞ ± NaN | ∞ ± NaN |
| 7012 JP | 0.150013 ± 0.000067 | 0.138801 ± 0.000096 | ∞ ± NaN | ∞ ± NaN |
| 2501 JP | 0.243355 ± 0.000108 | 0.216769 ± 0.000173 | ∞ ± NaN | ∞ ± NaN |
| 4005 JP | 0.136136 ± 0.000053 | 0.146412 ± 0.000071 | ∞ ± NaN | ∞ ± NaN |
| 7752 JP | 0.164498 ± 0.000036 | 0.123295 ± 0.000046 | ∞ ± NaN | ∞ ± NaN |
| 7911 JP | 0.228341 ± 0.000083 | 0.203080 ± 0.000107 | ∞ ± NaN | ∞ ± NaN |

Table II: All the Results of MSE

| Ticker | PGSGAN (MSE) | PGSGAN-HL (MSE) | S-GAN (MSE) | DCGAN (MSE) |
|--------|--------------|-----------------|-------------|-------------|
| 5901 JP | 0.000179 ± 0.000003 | 0.000386 ± 0.000003 | 0.001979 ± 0.000003 | 0.149898 ± 0.000003 |
| 5333 JP | 0.000264 ± 0.000002 | 0.000945 ± 0.000005 | 0.004611 ± 0.000011 | 0.143241 ± 0.000007 |
| 8355 JP | 0.000565 ± 0.000002 | 0.000392 ± 0.000001 | 0.012347 ± 0.000018 | 0.053950 ± 0.000016 |
| 5631 JP | 0.000499 ± 0.000002 | 0.000432 ± 0.000003 | 0.027290 ± 0.000035 | 0.107452 ± 0.000026 |
| 9532 JP | 0.000300 ± 0.000000 | 0.000276 ± 0.000003 | 0.030383 ± 0.000029 | 0.145841 ± 0.000011 |
| 7012 JP | 0.000332 ± 0.000003 | 0.000257 ± 0.000003 | 0.020980 ± 0.000042 | 0.145100 ± 0.000003 |
| 2501 JP | 0.000428 ± 0.000002 | 0.000391 ± 0.000003 | 0.014947 ± 0.000039 | 0.146345 ± 0.000012 |
| 4005 JP | 0.000512 ± 0.000004 | 0.000439 ± 0.000003 | 0.014975 ± 0.000047 | 0.109629 ± 0.000009 |
| 7752 JP | 0.000666 ± 0.000003 | 0.000691 ± 0.000004 | 0.009458 ± 0.000015 | 0.141400 ± 0.000005 |
| 7911 JP | 0.000580 ± 0.000003 | 0.000473 ± 0.000004 | 0.019275 ± 0.000029 | 0.107608 ± 0.000012 |
This causes the vanishing of the gradient in the generator and the explicit end of the learning of PGSGAN. By introducing Hinge loss, this can be avoided. Moreover, the learning can be continued for a longer time than PGSGAN.

As future work, higher performance GANs, evaluation methods better than MSE/KLD, or application of our GANs, should 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.

VII. CONCLUSION

We proposed a new GAN for generating realistic orders in financial markets. GANs in some previous works generated fake orders in continuous spaces because of GAN architectures’ learning limitations. However, the real orders are discrete. For example, the price and volumes have minimum units. Moreover, order types, such as sell/buy, are also not continuous; it is inappropriate to join their state spaces continuously. Thus, in this study, we changed the generated fake orders to discrete orders. Because this change disabled the ordinary GAN learning algorithm, this study newly employed policy gradient for the learning algorithm. Policy gradient is frequently used in reinforcement learning. In this study, we made it possible to use policy gradients by incorporating the relationship between the generator and the critic into the reinforcement learning framework. In our model, the generator makes a policy; then, according to the policy, randomly sampled fake orders are processed by the critic. Our experiments tested our models, policy gradient stock GAN (PGSGAN) and policy gradient stock GAN with Hinge loss (PGSGAN-HL), in terms of next order generations. The data we used in this study were the order data from TSE and contained more than 65 million order data. Then, we compared the generated fake orders’ distribution and the real order distribution in terms of their MSE and KLD. As a result, we demonstrated that our proposed model outperforms the previous ones. In addition, as a side benefit of introducing the policy gradient, we found that the entropy of the generated policy can be used to check the learning status of the GAN. Moreover, the combination of our model and Hinge loss (PGSGAN-HL) seems to be beneficial for better learning by avoiding the gradient vanishing. As future work, higher performance GANs, evaluation methods better than MSE/KLD, or application of our GANs, should be addressed.

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] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

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

[10] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidl-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.

[11] 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.

[12] 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.

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

[14] M. Arjovsky, L. Bottou, “Towards Principled Methods for Training Generative Adversarial Networks,” 2017, https://arxiv.org/abs/1701.04862.

[15] 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.

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

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

[18] 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. 843–846, 1983.

[19] 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.

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

[21] 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.

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

[23] F. Yu and V. Kolm, “Multi-scale context aggregation by dilated convolutions,” 2015, https://arxiv.org/abs/1511.07122.

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

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

[26] 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.

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