# Towards realistic markets

Tengyingzi Ma
17-703-240

September 15, 2022

## 1 Introduction

Generative adversarial networks (GANs) has been widely used to generate synthetic images since it was proposed. Recently, it was also been adapted to simulate trading environment. In the paper [CPC+21], a conditional Wasserstein GANs approach is proposed to generate synthetic market using historical data. Unfortunately the code is not publicly available. In this project, our goal is first to reproduce the result in the paper, then work on improvement.

## 2 Background

In this section we briefly introduce the data used during the experiments, and GANs.

### 2.1 Observed Data

The data used in this project is level-1 AAPL from https://lobsterdata.com/info/DataSamples.php. It contains the evolution of the Apple limited order book between 09:30:00 and 10:30:00 on 21 June 2012. The data file consists of two separate files, the message file and the orderbook file. Where the Message file contains six columns: Time, Type, Order ID, Size, Price, Direction. And the Orderbook file has four columns: Best ask price, Best ask volume, Best bid price, Best bid volume.

In the preprocessing step, we applied the same process as in the paper [CPC+21]. The time series data used in the training has 10 features: Price, Size, Direction, Time, Best bid price, Best bid volume, Best ask price, Best ask volume, Mid price, Time period. I would like to mention here in particular that the Time feature is calculated using the message file and is the interarrival time between two orders. Whereas the Time period feature is calculated using the orderbook file, which is the interarrival time between two orders with different bid/ask price. The mid price feature is the mean of the best ask price and the best bid price.

In order to train the network more steadily, all data were normalized between $-1$ and 1 using min-max scaler. Where for the features Size, Time, Best bid volume and Best ask volume, they were additionally treated with box-cox transformation to reduce anomalies and normally distribute the data. Each training sample contains 50 time steps. As for the target sample, it is the order in the next time step compared to the training sample. Finally, we have a total of 57400 samples, which are divided into a training set containing 40000 samples, a validation set containing 10000 samples, and a test set containing 7400 samples.

### 2.2 Generative Adversarial Networks (GANs)

GANs was first proposed in [GPAM+14]. The idea is that two networks, a generator and a discriminator, compete with each other while improving simultaneously. Compare to the normal

GANs that uses only noise as input, CGANs uses historical data as part of the input to the generator to specify the type of the data you want to generate. For time series data, ultimately we want the generated data to have a similar distribution to the target data. To compare two probability distributions, we have several methods, such as relative entropy, also known as KL-divergence, and the Wasserstein metrics, as well as the total variation metrics. Wasserstein GANs, which was proposed in [ACB17], uses wasserstein distance between the generated data and the real data as loss function, providing more stable training compared to relative entropy. In this project, we use conditional Wasserstein GANs with a gradient penalty to model the market behavior by generating new orders as a function of the current market state.

# 3 Models

In this project, we implemented two different WGANs architectures from two papers ([CPC+21],[LWL+20]). More specifically, both WGANs use the same critic, but different generators. In both models, we mainly use Long Short-Term Memory(LSTM) layer and (transposed) convolutional layer. LSTM is working particularly well with time series data, capturing the long-term dependencies in the historical data and producing a compact encoding. The (transposed) convolutional layer, on the other hand, is good at capturing patterns that are invariant to certain transformations.

## 3.1 Generator

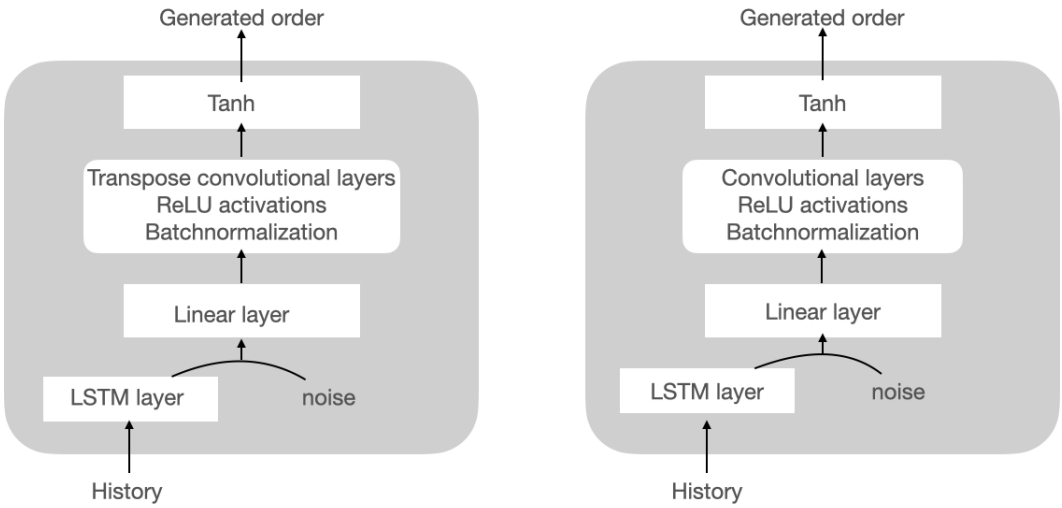


Figure 1: Two generators' architectures

As shown in Figure 1, these are the architectures of the two different generators. On the left, it is the CGAN structure proposed in [CPC+21]. And the stock-GAN proposed in [LWL+20] on the right of the figure. It looks like they have very similar structures, but not in terms of implementation. Both generators take the historical data as input $\boldsymbol{y} = (y_1, ..., y_{10}) \in \mathbb{R}^{50\times 10}$, and generate next order $\boldsymbol{x} = (x_1, ..., x_4) \in \mathbb{R}^{1\times 4}$.

In the CGAN architecture, the output $\boldsymbol{x}$ is smaller in width compared to the input $\boldsymbol{y}$, and it also contains multiple transpose convolutional layers, which are mainly used to upsample the input. These two conditions require that the layers in the generator before the transpose convolutional layers lose most of the information, and only then can we get the desired output shape. So in the implementation, we first apply an LSTM layer with input size $\mathbf{H}_{in} = 10$

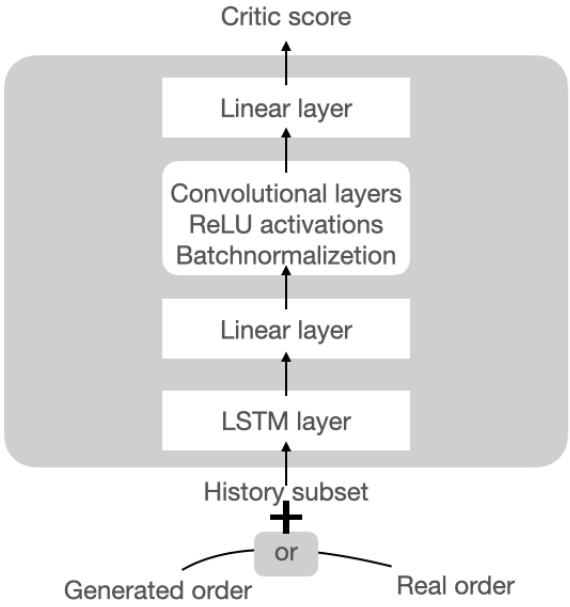


Figure 2: Critic's architecture

and output size $\mathbf{H}_{out} = 1$, directly on the input data $\boldsymbol{y}$. After this step, our input becomes $\tilde{\boldsymbol{y}} \in \mathbb{R}^{50\times 1}$. Then we concatenate $\tilde{\boldsymbol{y}}$ with random noise $\boldsymbol{z} \in \mathbb{R}^{50\times 1} \sim \mathcal{N}(0,1)$, and feed it into a linear layer with input feature $\mathbf{H}_{in} = 2$ and output feature $\mathbf{H}_{out} = 2$. Afterwards, we apply three layers of transpose convolutional layers followed by ReLU activations and Batch normalizations. Finally, the last transpose convolutional layer is applied, followed by Tanh activation, to obtain the output order.

As for the generator of stock-GAN, there are more options for the shape of each layer, since convolutional layers is used to subsample the input. In our implementation, similar to the CGAN described above, an LSTM layer is first applied directly on the input data $\boldsymbol{y}$. Unlike the first layer of CGAN, this time the LSTM layer has an input size $\mathbf{H}_{in} = 10$, and an output size $\mathbf{H}_{out} = 10$. It retains most of the information. The output of the LSTM layer now has the shape of $\tilde{\boldsymbol{y}} \in \mathbb{R}^{50\times 10}$. We then concatenate it with a random noise having same shape $\boldsymbol{z} \in \mathbb{R}^{50\times 10}$, and feed it into a linear layer with input feature $\mathbf{H}_{in} = 20$ and output feature $\mathbf{H}_{out} = 16$. The following structure is three layers of convolutional layers followed by ReLU activations and Batch normalizations, and a final convolutional layer followed by Tanh activation.

## 3.2 Critic

The two models share the same critic. The input of the critic is the first four columns of the historical data (Price, Size, Direction, Time) $\boldsymbol{y}$ concatenated to the generated order $\boldsymbol{x}$ or the real order $\boldsymbol{x}^*$. It thus takes the shape $\boldsymbol{y}' \in \mathbb{R}^{51\times 4}$. The critic has a similar structure to the generators, shown in Figure 2. The first layer is an LSTM layer with an input size of $\mathbf{H}_{in} = 4$ and an output of size $\mathbf{H}_{out} = 20$. The second layer is a linear layer with input features $\mathbf{H}_{in} = 20$ and output features $\mathbf{H}_{out} = 16$. After that, it has four convolutional layers followed by ReLU activations and Batch normalization. Finally, it outputs the critic score through a linear layer.

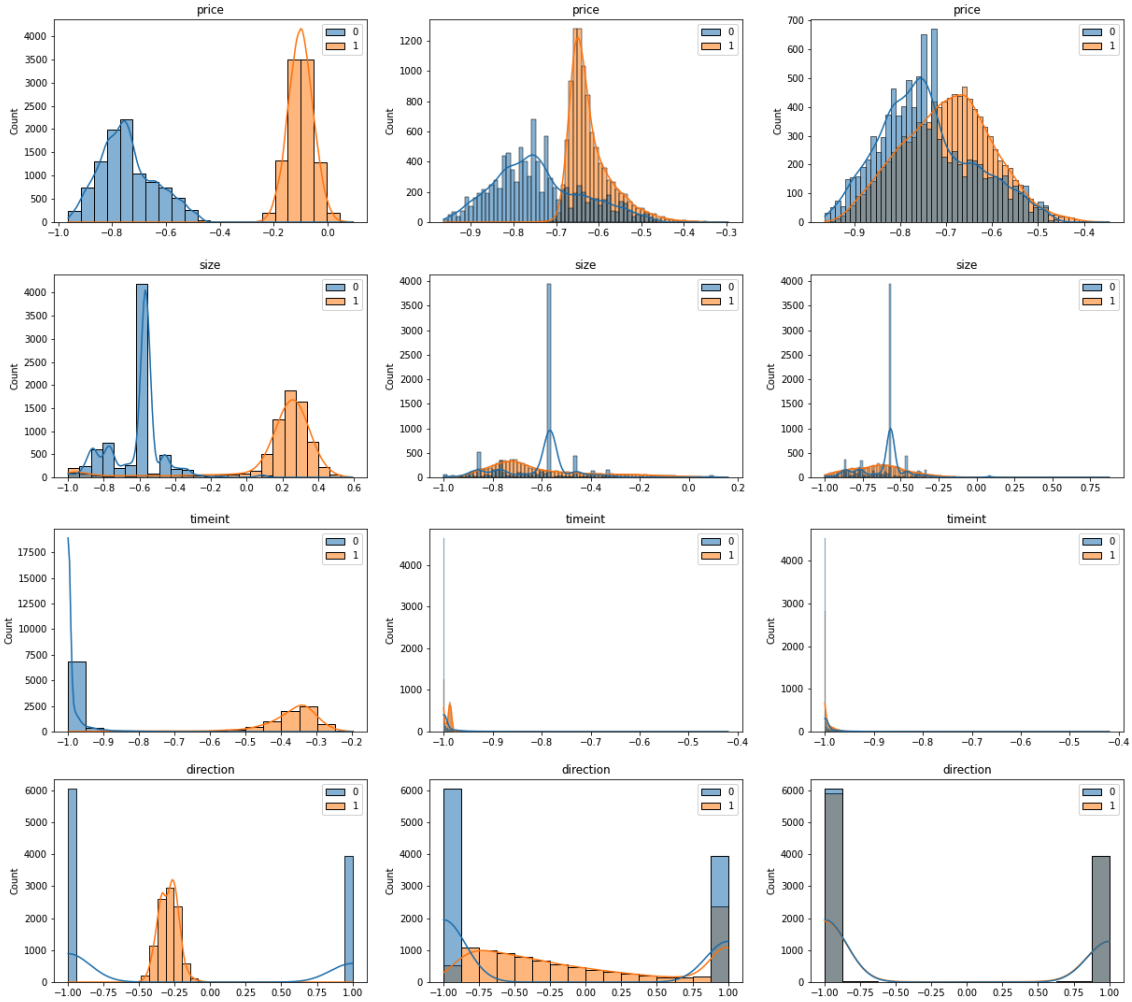


Figure 3: Results from CGAN: "0" is real order,"1" is generated order. From left to right, they are results after 10,60,120 epochs.

We use wasserstein distance with gradient penalty as the critic's loss function:

$$\min_G \max_D \mathop{\mathbb{E}}_{x\sim p_d}[D(x \mid y)] - \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}[D(G(z \mid y))] + \lambda \mathop{\mathbb{E}}_{z\sim\mathcal{N}(0,1)}\left[(\|\nabla D(G(z \mid y))\|_2 - 1)^2\right] \tag{1}$$

where the weight $\lambda$ is set to 10 as in the paper. Compared to traditional WGAN, adding gradient penalty allows easier optimisation and convergence by enforcing a soft Lipschitz constraint.

# 4 Experiments

During the experiments, the different models were performed directly on real data. The training model consists of five parameters: batch size, epochs, learning rate of the generator, learning rate of the critic and critic iterations. Batch size, epochs, learning rate are all common machine learning parameters. Critic iterations, on the other hand, is often used in GANs. The idea is

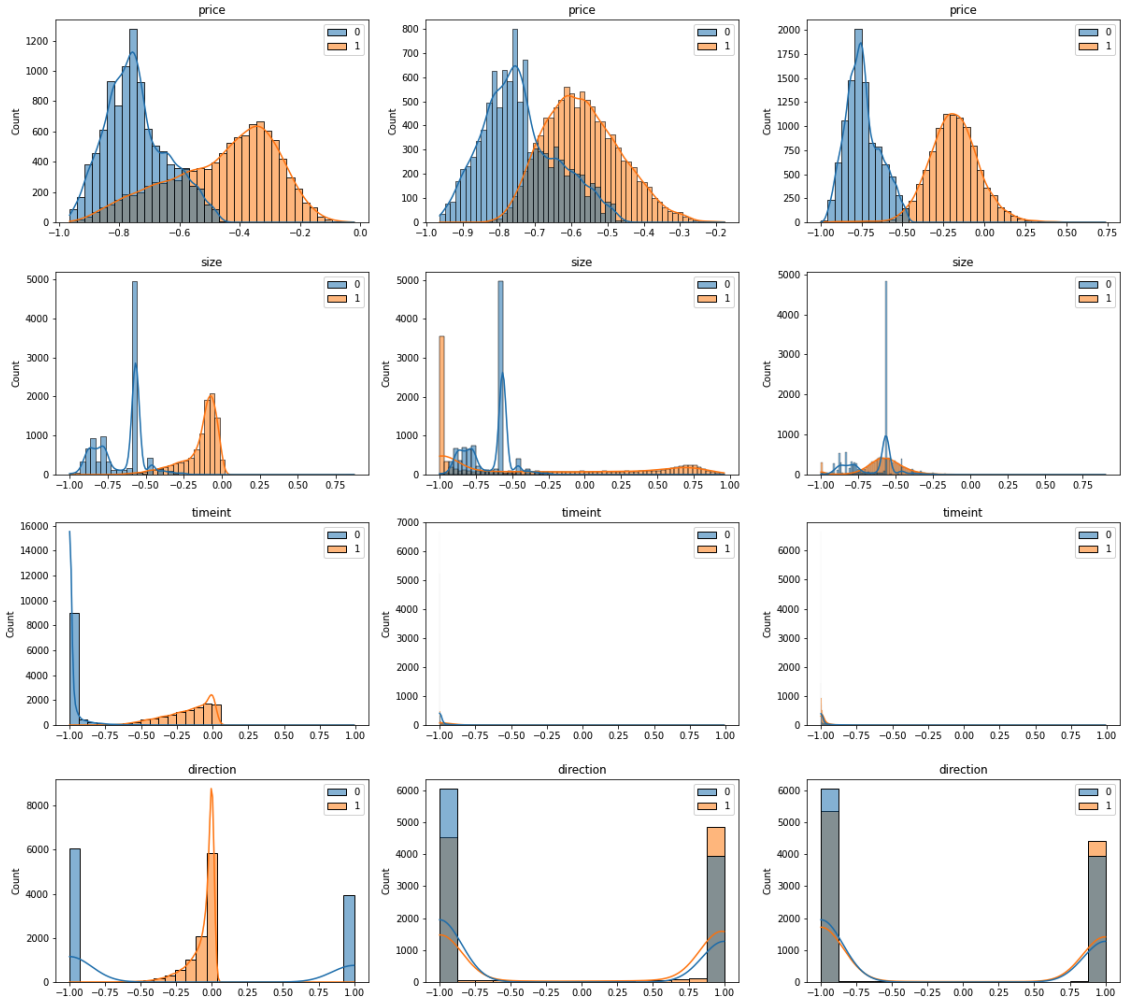


Figure 4: Results from stock-GAN: "0" is real order,"1" is generated order. From left to right, they are results after 10,50,100 epochs.

that in each batch training, we can train the critic multiple times, while training the generator only once, so that the critic can be trained optimally. However, this did not work well in our case. We first tried to set the critic iterations = 5, which means that in each batch, the critic is trained five times and the generator once. The result was the losses for both the critic and the generator got stuck, indicating that the critic became too powerful in a short period of time, and the generator could not improve any further. Then in later training, we only set critic iterations = 1, and the results were better. This phenomenon also tells us that the goal is not to train two very powerful networks to generate new data and to criticise the performance. Instead, the ideal situation would be for both networks to grow simultaneously, competing with each other and learning from each other. Based on this principle, we can adjust other parameters during the training, for example, if the critic still get too powerful in a short time, we can lower the learning rate of the critic and vice versa.

For CGAN we got our best results (shown in Figure 3) using parameters: Batch size = 100, Epochs = 120, Learning rate of the generator = 0.0001, Learning rate of the critic = 0.0001, and critic iterations = 1. In Figure 3, "0" is real order, and "1" is generated order. From top to bottom, it shows the corresponding distributions from price, size, time, and direction

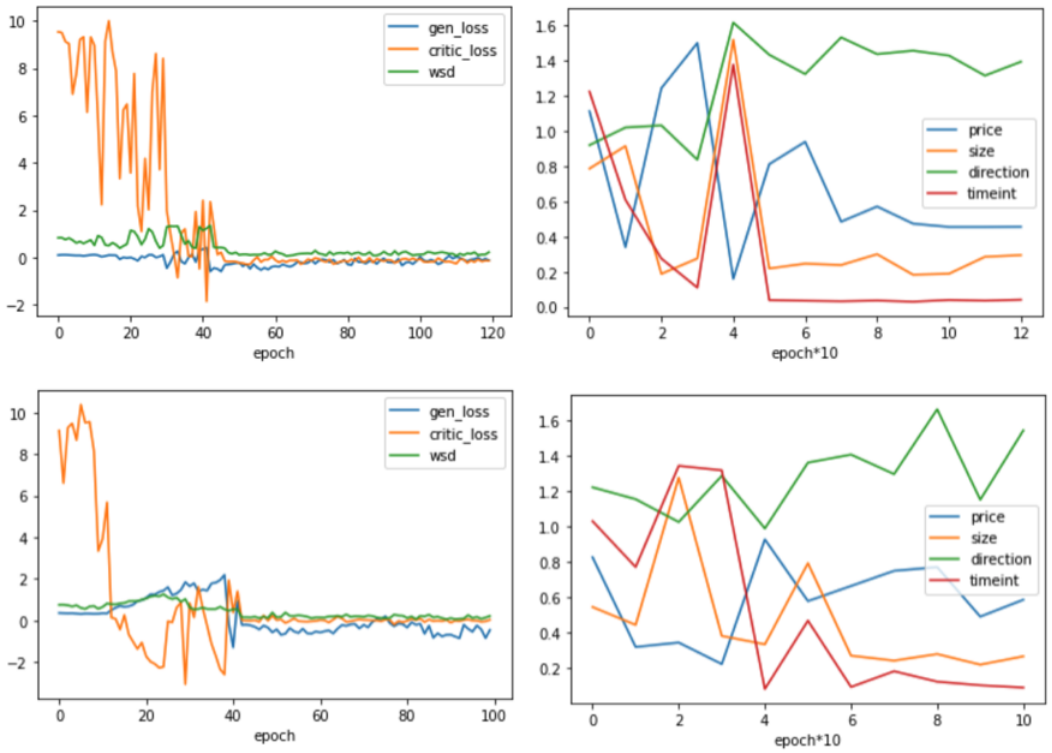


Figure 5: Training losses and sliced-wasserstein distances: the top line shows for CGAN, and the bottom for stock-GAN. On the left, they are plots of generator loss, critic loss and sliced-wasserstein distances after each epoch for training set. The right column shows the sliced-wasserstein distances for four values separately on validation set.

accordingly. From left to right, it shows the training results from the beginning to the end, the left column shows results after 10 epochs, middle column are results after 60 epochs, and the right column show the results after 120 epochs. We can see for all four distributions, the generated data are converging in the direction of the real data. After 120 epochs, they are all very close to the true distribution, especially for the direction, which is almost the same.

For stock-GAN, we also use the same parameters as in CGAN training mentioned above, except here epochs = 100. Because we observed this model is less stable, and more training epochs does not improve much the results. The results are shown in Figure 4. Similarly, from top to bottom, they are distributions of price, size, time and direction. From left to right, they are results after $10, 50, 120$ iterations. And "0" represents real order, whereas "1" represents generated order. We can see that in this model, the distributions converge faster than CGAN at the beginning. After 50 iterations, the distributions of the generated order already resemble the shape of the real orders. However, after another 50 iterations of training, the results did not get much better, especially for the price, where the distributions are getting further and further away. So in the end, it did not give better result than CGAN.

To compare the performance of each model, looking at the results plots is not the only way to do so. In this project, we use the sliced-wasserstein distance to measure how close the generated distribution is to the real one. The Monte-Carlo approximation of the p-Sliced Wasserstein distancehe is defined as:

$$\mathcal{SWD}_p(\mu,\nu) = \mathop{\mathbb{E}}_{\theta\sim\mathcal{U}(\mathbb{S}^{d-1})} \left(\mathcal{W}_p^p\left(\theta_{\#}\mu, \theta_{\#}\nu\right)\right)^{\frac{1}{p}} \tag{2}$$

where $\theta_{\#}\mu$ stands for the pushforwards of the projection $X \in \mathbb{R}^d \mapsto \langle \theta, X \rangle$.

Figure 5 shows the training losses and the sliced-wasserstein distance for training set and validation set. The top line are the training for CGAN, and the bottom line is for stock-GAN. The left column shows the loss for both generator and critic, as well as the sliced-wasserstein distance after each epoch. The right column are the sliced-wassertein distance for each value (price, size, direction, time), each 10 epochs on validation set. We can see for both models, the losses change dramatically during the first 40 epochs. After 40 epochs, the training get stable, where both losses having similar value. As for the slice-wasserstein distance, it also become stable after 40 epochs. The sliced-wasserstein distance for different values even show better details of the training. We can see the distance of different values got stable at different time. For example, for GCAN shown on the top right in figure 5, time first get stable, followed by size and price. For direction, because it only has value $-1$ or 1, so a small discrepancy can cause a large difference in sliced-wasserstein distance. From this plot, we can also see that the stock-GAN(bottom) is less stable than the CGAN(top).

# 5 Conclusion and Outlook

During our experiments, we have successfully reproduced the results in the paper [CPC+21], and we have also compared its performance with another model presented in paper [LWL+20]. In addition, we introduced the sliced-wasserstein distance as a useful measure to assess the performance of the model.

We also noticed some points we can do in the future for improvements. The first one is to train more generator than the critic. This could allow them to converge faster than they do now. The second one is to apply other architectures to improve the generated orders' variance. We observed a lack of variance in the generated orders compared to real orders in our results. Some techniques for generating high-resolution images ([KALL17]) may be helpful.

# References


[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. 01 2017.

[CPC+21] Andrea Coletta, Matteo Prata, Michele Conti, Emanuele Mercanti, Novella Bartolini, Aymeric Moulin, Svitlana Vyetrenko, and Tucker Balch. *Towards Realistic Market Simulations: a Generative Adversarial Networks Approach*. 10 2021.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. 10 2017.

[LWL+20] Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael Wellman. Generating realistic stock market order streams. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:727–734, 04 2020.