Ryan Shea

March 1, 2023

Enhancing Stock Market Modeling by Using GANs for Synthetic Data Creation

Overview

The utilization of deep learning models in finance has witnessed a surge in popularity in recent times. They have been used in many different fields such as risk management, asset pricing, algorithmic trading strategies, and more. The advantage of processing massive datasets quickly and efficiently is one of the main reasons for their popularity in this field, whereas vast amounts of data are created every day. One area in which they are overlooked is in creating synthetic time-series stock data. There have been many uses of synthetic data in the fields of fraud detection and the like, but there is not a solid framework for time series applications. This is a large problem as this data can be important for creating and testing other models, especially when real data is limited. One important aspect for this is risk management, particularly when looking at the unpredictable market crashes – scenarios investors fear the most. Due to the scarcity of data pertaining to crashes, many risk management models tend to overfit to the last major crash. A generative model that could generate decades – or even centuries – worth of quality returns on any stock or portfolio enables the risk management model to have a more robust understanding of the asset's Conditional Value-at-Risk (CVaR) through the use of a Monte Carlo (MC) simulation. Given these reasons, the objective of this project is to demonstrate the possibility of creating synthetic stock data in a simplified scenario. To achieve this goal, a deep learning model will be trained on 5-day log return samples and will then synthesize artificial data that mirrors its structure.

Data Collection

The data needed for this project was obtained using python's "yfinance" package. The Adjusted Close price of all stocks in the S&P 500 from Jan 1, 2019 to Jan 31, 2022 were used and converted to log returns (in the formula $ln(S_t/S_{t-1})$). Next, around 50,000 5-day samples were obtained by iterating through each ticker, and selecting 5 consecutive days below. The pseudocode can be found below:

```
1. initialize an empty list
2. for i = 1, 2, 3, ..., 100:
3.     for each ticker in the S&P 500:
4.         randomly select 5 consecutive days from the returns data
5.         append the 5-day returns to the list
6. return the list
```

This allows for a large amount of data samples with many diverse structures to be sampled, ensuring a robust and comprehensive evaluation of the underlying structure of stock price movements, in which traditional methods such as Geometric Brownian Motions (GBMs) can not capture.

GBMs have been the main choice for synthetic data generation in the finance space, so they will be used as a baseline. The accuracy of a neutral machine learning model – a LightGBM Classifier – will attempt to differentiate first between the real data and GBMs, and then will be retrained with the GAN outputs and a sample of real returns. This will allow a direct comparison in the accuracy between the industry standard and the GAN's output. The GBMs were created by sampling a random mean and standard deviation of the real returns and then were fed into the GBM formula 50,000 times. After this, there are now roughly 50,000 random samples of real log returns and 50,000 samples of GBM paths. The baseline accuracy in differentiating between log returns and GBM paths is roughly 0.779. This implies that the model is able to somewhat understand the difference between real and fake data, but there is room for improvement. “Perfect” fake data would completely mirror real data’s structure, and the theoretically optimal accuracy would be 0.5 as the model would be simply guessing each time and is unable to find any structures or patterns that would allow it to differentiate between the two.

Machine Learning Methods

The deep learning model used is a Generative Adversarial Network (GAN). GANs consist of two different neural networks: a generator, which learns patterns and structures of data and then generates fake data with similar structures, and a discriminator, which learns the difference between the real and fake data and classifies each sample accordingly. These two networks train adversarially where the generator attempts to “trick” the discriminator into believing that the fake data is real, and vice versa. GANs are popular in synthetic image creation but have not been applied widely in the time-series space.

The structure of both neural networks are standard multilayer perceptrons (MLPs). This structure was chosen with the goal in mind to keep things as simple as possible to begin analysis of time-series data synthesis. If the MLP model can outperform something more complicated, such as a Long-Short Term Memory (LSTM) recurrent neural network, there is no reason to increase complexity, so this is a good place to start. The GAN was constructed using the python library PyTorch. The Discriminator has a structure of an input layer (5 inputs correspond with each return on each day), 3 hidden layers with various activation functions and dropout layers (to avoid overfitting). The Generator also has an input layer and 3 hidden layers as well. There are various hyperparameters in the GAN itself: the train size, the epochs, the batch size, the learning rate, beta 1 and beta 2 for the Adam Optimizer, and the clip value – for the discriminator minimum and maximum. It currently has two methods: training and sample, which generates n samples. The loss function both the generator and discriminator use is binary crossentropy.

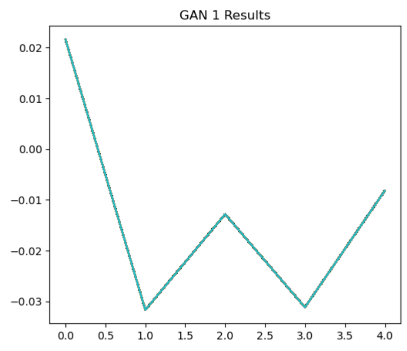
In order to get the best possible model, a random search of the hyperparameters was utilized. The algorithm works as follows:

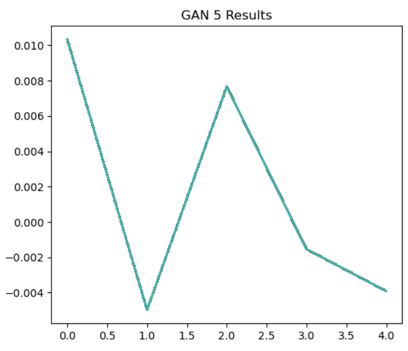
```
d = dict where keys are hyperparameters and values are
    lists of possible values

1. for i = 1, 2, 3, ..., number of models:
2.     initialize an empty dictionary
3.     for each key in d:
4.         randomly select a value from the list
           of possible values and store it in the
           dictionary
5.     train a model with the selected hyperparameters
6.     generate 1000 samples from the model
```

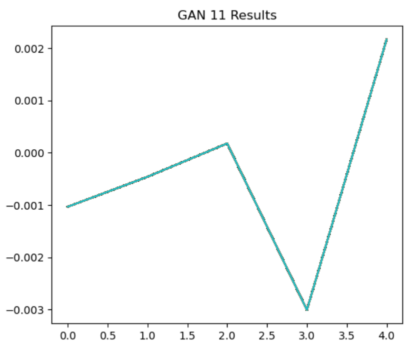
This allows a wide variety of models with various hyperparameters to see which do better. A total of 13 different GAN networks were trained using the method above.

Results

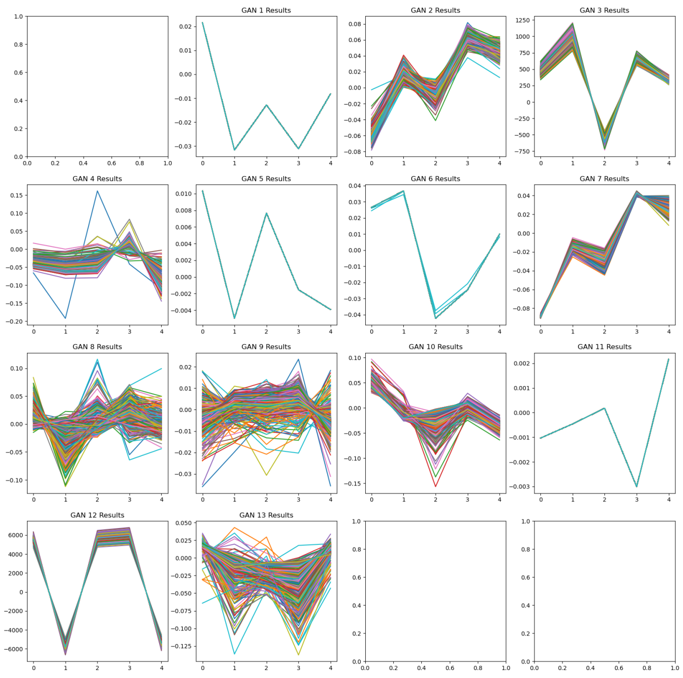
The results from this experiment were slightly unexpected. Many of the models seemed to have an overfitting problem, especially when the number of epochs was high. GANs 1, 5 and 11 specifically have this overfitting problem by looking at the lack of noise:





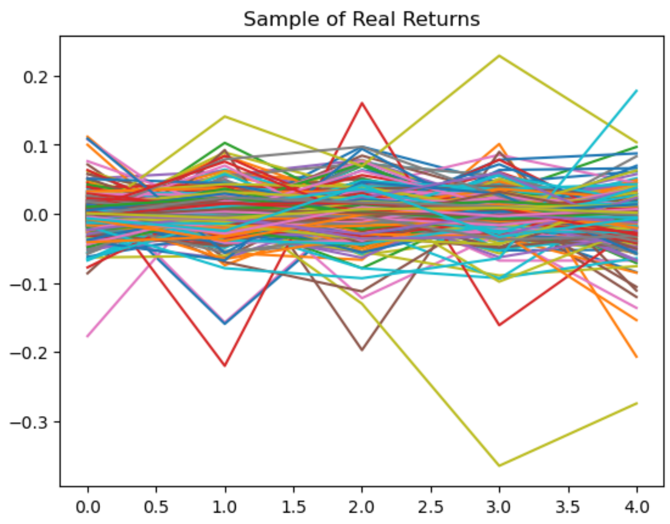


Keep in mind that there are 1,000 different paths shown above for each plot, but only one is visible: this implies massive overfitting for these models specifically. Despite regularization techniques such as dropout, these models seemed to learn the structures almost too well, and do not have any noise in their paths at all.



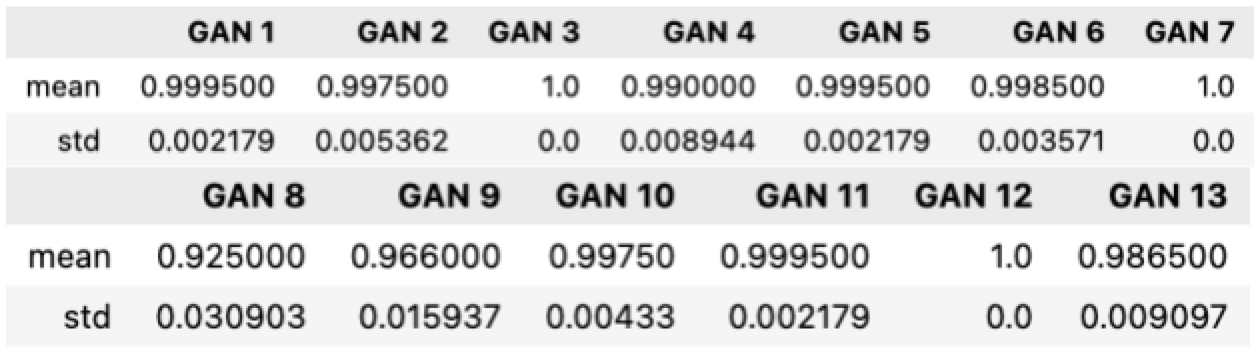
There is also a large difference in the scale: GAN 1 makes 2-3% jumps each day while the other 2 make significantly smaller jumps. This could be attributed to the various clip values between the models. They are being clamped differently depending on that specific hyperparameter. Other models tend to perform slightly better in the sense that some stochastic components are modeled and are not completely taken out. In general, however, it is easy enough to tell by looking alone that most of the models have seemed to overfit some sort of structure they found in the returns.

When compared to most real returns, it is easy to tell by just looking: the samples of real returns almost always stay around 0 with only a few paths breaking 0.2. This is in sharp contrast with the fake returns where it is



possible to view the relative structure in some, if not all, of the data.

Now it is time to see numerically how the LightGBM did as a discriminator on the GANs outputs.. Recall that the accuracy on GBMs was around 0.779. The mean and standard deviation of a 20-fold cross validation accuracy score was calculated for each GAN.



	GAN 1	GAN 2	GAN 3	GAN 4	GAN 5	GAN 6	GAN 7
mean	0.999500	0.997500	1.0	0.990000	0.999500	0.998500	1.0
std	0.002179	0.005362	0.0	0.008944	0.002179	0.003571	0.0

	GAN 8	GAN 9	GAN 10	GAN 11	GAN 12	GAN 13
mean	0.925000	0.966000	0.99750	0.999500	1.0	0.986500
std	0.030903	0.015937	0.00433	0.002179	0.0	0.009097

Each model performed significantly worse than the GBM baseline test, with all but GANs 8 and 9 having an accuracy of over 99%. This implies that GANs of this structure (MLP) are not valid for creating synthetic data.

Next Steps

There are many steps in order to continue research. First and foremost, thstructure must be recreated with a much larger emphasis on regularization as overfitting was a large issue across the board. This could be in the form of more aggressive dropout layers, adding an early stopping to the training process, or just simplifying the model by decreasing the units or number of hidden layers. All of these strategies may allow the model to keep some of the noise that it is picking up.

At the same time, the noise in the data is also introducing many local minimum on the gradient, leading to these suboptimal models. Another possible solution would be to eliminate some of this noise by picking only one stock as opposed to all the stocks in the S&P 500. The volatility of one stock is very different from the volatility of another, and these various changes between samples might have been too noisy in general for the model. While stock data is non-stationary, the price paths of one individual stock might be easier to approximate when compared to all of them.

It may also be helpful to pick a more complex model that might be more robust to the noise. For example, a convolutional neural network (CNN) might be a better choice as the pooling layers may help smooth out the noise. An LSTM is popular for time series predictions, so that could be a possible structure to use as well.

There is also a chance that these models are not getting enough data for synthesizing accurate price paths. More inputs such as the difference between (normalized) Open and Close prices and Volume for each trading day may give it a better estimate of the impact that volatility will have on the noise and structure of the returns. They may be features that the model will need to perform better.

Finally, a more efficient hyperparameter tuning method may be helpful as well. This could be in the form of Bayesian Optimization as opposed to a simple random search. This may more efficiently find optimal hyperparameters and create models that are able to effectively model the noise. Finally, just building more models with a wider range of hyperparameters may prove to be a solution in itself, one of them may eventually do better than traditional GBM methods. GAN 8 had an accuracy score around 0.92, so there is a chance that over time one of the models might get closer to the global solution. While the outcome of this project was slightly disappointing, there are many different directions to extend research and construct a model that is significantly improved from its current state.