A Brief Explanation and Numerical Test of Quant GANs

22nd January 2026

Contents

1 Introduction

Financial markets have become incredibly large and deeply rooted in Western society. As a consequence, it is important for markets to be priced correctly. Indeed, a young couple buying their first house, a farmer using futures contracts to lock in revenues, and a start-up seeking funding all depend on correctly priced financial markets. Starting on the cusp of the 20th century with Louis Bachelier and exploding in popularity by the success of the likes of Ed Thorp and James Simons, mathematical analysis of financial markets has become the norm.

Computers and vast amounts of data allow such investors to spot opportunities in the market and develop strategies that automatically exploit them.

Building a price model generally begins with determining a probability distribution for how the price moves. Note that this also applies to actuarial science, where such distributions play a crucial role in assessing risk. In these cases, it's important that the chosen distribution correctly matches what actually happens in the future or that the strategy is robust to any differences. One method to achieve both is by generating more data to work with. For some models, like Black-Scholes [2], this is easy as it contains a built-in generating distribution. However, in these cases the distribution has been fixed a priori, which is inflexible. At the other end of the spectrum one could bootstrap historical data, meaning put all past movements in a bag and sample from it. While there may be more advanced ways of bootstrapping, a downside is the core assumption that history repeats itself. The method we'll consider in this article places itself somewhat between these two, and originates in Wiese et al.'s QuantGAN paper [12]. It uses the Generative Adversarial Network (GAN) framework [4], which, at a high level, teaches a neural network to turn Gaussian noise into high-quality data. We'll explain in more detail how the method works in the next section.

In that paper, the authors test the performance of their method by comparing both the return distributions and correlations of the generated data with real data to check that it shows characteristic behaviors of stock market data (known as stylized facts). In this article we take a different approach to test the quality of the data generated: we train our own discriminator and compare its performance with other models, such as Black-Scholes. We used this implementation of QuantGANs, with minor modifications, to generate data to be evaluated. It turns out that our discriminator, a small LSTM model, was able to distinguish QuantGAN data better than it could GARCH(1,1), which in some sense implies the QuantGAN setup may be flawed, as explained in section 5.

Sections that are slightly more technical are marked with a * and can be skipped without loss of continuity.

2 How does the paper work?

2.1 Up-to-speed on Neural Networks

For our purposes, machine learning concerns itself with approximating a function we do not know, based on finitely many input-output pairs. Neural networks are the backbone of much of machine learning today and, in particular, are key to understanding the paper and this article. They have been hugely successful in identifying patterns in complex data, culminating in today's craze for large language models and, according to some, suggesting that artificial general intelligence is on the horizon. Essentially, 'neural network' is an umbrella term for a family of parametrized functions which, as the name suggests, is loosely based on the structure of biological neurons. Their most basic form is known as

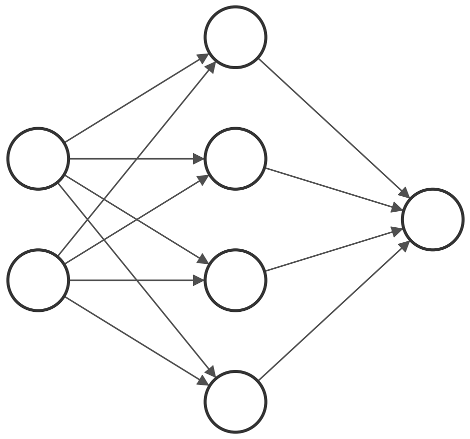

Figure 1: Schematic of an MLP, the most basic form of a neural network.

a multi-layer perceptron and is best described with a picture as in Figure 1. The input is fed into the leftmost neurons and information flows left to right through the network of neurons, with each performing a simple, parametrized calculation and passing the result on; the final output is the value in the rightmost neuron(s). Note that networks are built out of layers of neurons; the network in Figure 1 has an input layer, a hidden layer, and an output layer. A network is 'trained' in essentially the same way as a child: give it an input and correct its output. To correct the network and cause it to change its behavior, we not only calculate how its output could be improved; we go through each layer of computation that brought it to its output and update each step there too. As if we not only tell a child 'that's an apple, not a tree', but also go and 'tell' every one of their neurons 'you should have fired more' or 'you shouldn't have fired at all'. The algorithm to do this efficiently is known as backpropagation, since the error 'backpropagates' through the network, but we won't go into any more detail here. Neural networks can be constructed in many shapes, with additional computations done in between layers, but they generally share this same forward-and-back propagation structure.

2.2 GAN

The workhorse of the paper is the Generative Adversarial Network (GAN) framework [4]. This paper provides a process to train a neural network to replicate a target probability distribution for which we have only finitely many samples, for example, to generate pictures of (nonexistent) faces or in our case financial data. The idea is to train two networks: a 'generator' and a 'discriminator' (an art analogy would be training an art forger and an art authenticator). The generator takes as input random noise and its goal is to output samples from the target distribution (replicate a Van Gogh), while the discriminator is trained to be able to tell what samples are 'real', i.e. from the target distribution, and which are ‘fake’ (verify Van Gogh’s). The key point is that the fake samples used to train the discriminator are the output of the generator, so the discriminator learns exactly what ‘quirks’ give the generator away, and conversely the generator is trained to maximize the probability that the discriminator makes a mistake. This is where the term ‘adversarial’ comes from, and the goal for the end of training is that the generator can perfectly replicate the target distribution, leaving the discriminator guessing.

2.3 TCN*

The specific type of neural network the paper uses for the generator and discriminator is known as a Temporal Convolutional Network (TCN) (e.g. [1]). It differs from an MLP in two key ways, which make it naturally suited to sequential data. Firstly, a neuron is not connected to every neuron in the previous layer. Instead, for every layer l in the network, we attach two numbers K_l and D_l, known as the kernel size and dilation respectively. Then a neuron in layer l at time t is connected to neurons at time $t, t-D_l, ..., t-(K_l-1)D_l$ in layer $l-1$, see Figure 2. Secondly, each neuron represents a *vector* of information of size N_l. Thus, instead of a connection being represented by a scalar weight, each connection is a matrix of size $N_l \times N_{l-1}$. These matrices are furthermore shared across the layer, per input connection, meaning each layer is parametrized by K_l matrices of size $N_l \times N_{l-1}$ (or equivalently a third order tensor). Besides reducing parameter count, this has the effect of making the TCN more translation invariant, which is a natural quality to have in sequence modeling. The authors also include skip connections, which are essentially $K = D = 1$ connections between nonadjacent layers. This helps mitigate the problem of gradient vanishing because gradients can flow through the network faster via skip connections, and it may improve results of the generator by effectively allowing layers to learn adjustments to the identity map, instead of the full transformation [1]; equivalently, the layers work with differenced data.

2.4 Lambert W Transform*

Now the authors showed that when the input noise is Gaussian, the TCN generator (in fact any neural network) will output a distribution for which all moments exist. However, it is well known that financial data has heavy tails (and indeed the paper’s data set has an excess kurtosis of about 4) meaning no matter how much it trains, our generator would never be able to replicate this important quality of the real data distribution. One solution would be to use heavy-tailed input noise, however, the authors avoided this based on optimization stability concerns. Indeed, they were able to show that if the p-th moment of the input noise exists, then the p-th moment of the gradient of the generator with respect to its parameters exists, so that using a Gaussian would guarantee a desirable gradient. They do not prove the converse, but if it is true, then heavy-tailed input noise would cause heavy-tailed gradients, which can cause difficulties for optimization. Instead, the authors rely on the Lambert W probability transform

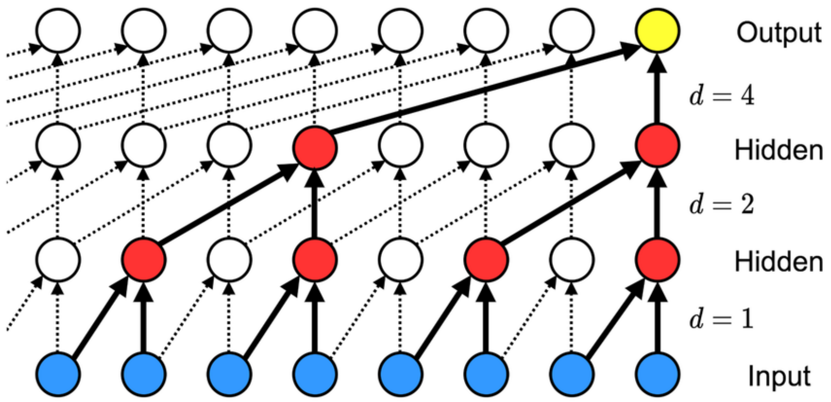

Figure 2: Schematic of a TCN, credits to [6]. Here d is the dilation of each layer and $K_l = 2$ for all layers.

[3]. This transform takes a random variable X to

$$Y = U \exp\left(\frac{\delta}{2}U^2\right)\sigma + \mu, \tag{1}$$

for $U = (X - \mu)/\sigma$, and it can be shown that for $\delta > 0$, Y has heavier tails than X. Moreover, this transform is bijective—its inverse is given by an evaluation of the usual Lambert W function—so if the *inverse* Lambert W transform of the data (determined by μ, σ, δ) has no excess kurtosis, then we can rest assured it can be modeled by the TCN generator. Now, the parameters to calculate a suitable inverse transform can be found by iteratively estimating μ and σ (as sample mean and standard deviation) and then minimizing the excess kurtosis as a function of δ. Indeed, note that μ and σ are the mean and standard deviation of the *unknown* random variable X (with the notation as above, Y is our heavy-tailed financial data and we want to find X), so they are calculated as a function of the data and δ. Conversely, we can only optimize for δ once we have fixed μ and σ.

3 Research Context

Before continuing to the evaluation of the Quant GAN method, we first put the paper into some context. The authors Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer were at the time affiliated with TU Kaiserslautern and the Fraunhofer ITWM (the industrial math institute of Kaiserslautern, Germany). The paper was published in the journal Quantitative Finance in April 2020.

The authors were not the first to use the GAN framework to generate financial data. They mention a master thesis from 2019 [7] which similarly used GANs to generate S&P500 (and VIX) data and trained prediction models on the artificial data. These models were subsequently tested against real data left out of the training set, however it contained no code nor many details on the networks used. Similarly, this 2019 paper [11] uses MLP and CNN based GANs to replicate many stylized facts of financial data without providing any more details of the models used. We furthermore found this 2017 paper [13] which used a GAN framework to predict stock prices and claimed to be the first to apply 'adversarial training to [the] stock market'. That said, it seems the QuantGAN authors were the first to provide a complete, systematic study of applying a GAN framework to unconditional financial data generation.

Since then there have been numerous papers on financial data generation, and the majority have been GAN-based according to a recent survey [8]. Other methods include variational autoencoders and diffusion models.

4 Method

We test the quality of the GAN-generated data by training our own discriminator between real and fake data. In theory, if the generator perfectly replicates the real data, no model should be able to reliably distinguish real from fake, so the higher the accuracy of a discriminator, the lower the quality of our data. The advantage of using machine learning to compare real to fake data is we introduce less inductive bias: a human might compare histograms, (partial) autocorrelation functions, etc. of both datasets; a model with sufficient capacity could also consider these methods, but it may converge faster onto other 'hidden' indicators that give fake data away. To determine a baseline, we additionally train and test the same discriminator under the same conditions on two simpler generators.

4.1 Baseline models

We choose two well-known models as baselines: Black-Scholes (BS) and GARCH(1,1). The former assumes the price follows geometric Brownian motion, which is a type of stochastic process that essentially boils down to assuming log returns are Gaussian. The latter is slightly more involved: it models the variance of returns as an ARMA(1,1) process, which itself is a simple time series process. This turns out to be a natural choice as it can be observed that for financial data, return volatility *clusters*, one interpretation for which is the market tends to swing between (prolonged) states of calm or of unrest, and GARCH models replicate this behavior. We refer the curious reader, for example, to [9] and [10] respectively for more information.

4.2 Discriminator model

We experimented with several architectures and decided to use the Long Short-Term Memory (LSTM) architecture [5]. From the outset, we opted against using the same model as the discriminator used in training, the TCN. The LSTM is a classical recurrent architecture for modeling time series, and was designed to capture long-term dependencies, which we expect financial data to have. Indeed, out of all architectures considered, LSTM used the fewest amount of trainable parameters to achieve a certain accuracy on the baseline. Additionally, unlike for instance the TCN, LSTM can handle different lengths of input sequences without (significantly) changing the capacity of the model, which is useful to compare performance across different sequence lengths, as we will do.

4.3 Implementation

As mentioned, we used this implementation of the paper, with minor modifications, to generate data which we then trained an LSTM on. For this we used a one layer PyTorch implementation of LSTM with `hidden_size` = 5 (the model had only 166 learnable parameters) and trained and validated the model on sequences of length 5, 20, and 60 corresponding to a week, month, and 3-month window of data (since the S&P500 data we have is daily, on weekdays). To be precise, sequence length refers to the number of data points the model uses to predict whether the data is real or not. The validation set was kept around 10% of the size of the training set. Our code can be found here. Additionally, for every training set we reinitialized and trained the model 10 times over which we average.

5 Results

After several attempts and hyperparameter tuning, we were able to get satisfactory QuantGAN data that matched the real distribution and n-differenced distributions for $n = 5, 20, 100$. It is saved to `gan_data.npy` in the same repo if the reader is curious.

5.1 Sequence length 5

Since the data is stochastic, we expect the model to decrease in accuracy as we decrease sequence length. Thus, for a week-long sequence length, we expect the model to have low accuracy on all datasets. This is indeed reflected in Figure 3, where we see that both the model training loss and validation accuracy plateau rapidly at high and low levels respectively. Interestingly, the model has almost exactly the same accuracy on GARCH as on GAN datasets and collectively slightly worse than it does on GBM. That said, we note that for all three the accuracy is greater than 50%, so it would be incorrect to call the data indistinguishable at this timeframe.

5.2 Sequence length 20

At this sequence length, shown in Figure 4, we see that the previous concurrence between GAN and GARCH has split, with, surprisingly, the model having

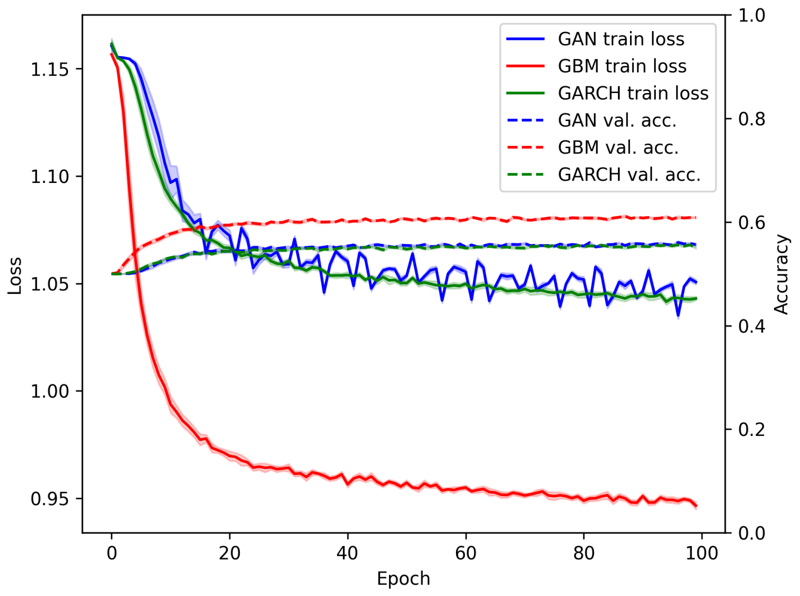

Figure 3: Sequence length 5

lower accuracy and higher training loss on GARCH data. Indeed, the model is now relatively comfortably able to distinguish real data from GAN data, at an accuracy upwards of 70%, and its GAN accuracy is closer to GBM than to GARCH.

5.3 Sequence length 60

The most noticeable difference seen in Figure 5, when we increase the sequence length to 3 months, is the large increase in variance of the loss and accuracy across iterations. This was caused by the fact that sometimes training would completely fail and the model would still be around 50% accuracy after 100 epochs. One explanation is that while LSTM is designed to mitigate vanishing gradients, its gradients still may explode. However, that is likely not the case since training loss still gradually decreased over time, showing that learning occurs. Another explanation could be that the model capacity is not high enough to have any meaningful benefit from the extra long-term dependencies, while its gradients are being calculated for the full sequence, which destabilizes training. Indeed, we see that both final accuracies and losses are somewhat near the length 20 results. This could be further numerically tested by evaluating the 20 sequence length trained model on sequence length 60 test data, but we leave that as future work. (Conversely, it would be interesting to consider the first $n-1$ outputs of a model trained on sequence length n. Ideally, the model slowly builds up conviction and the outputs increasingly converge to the final

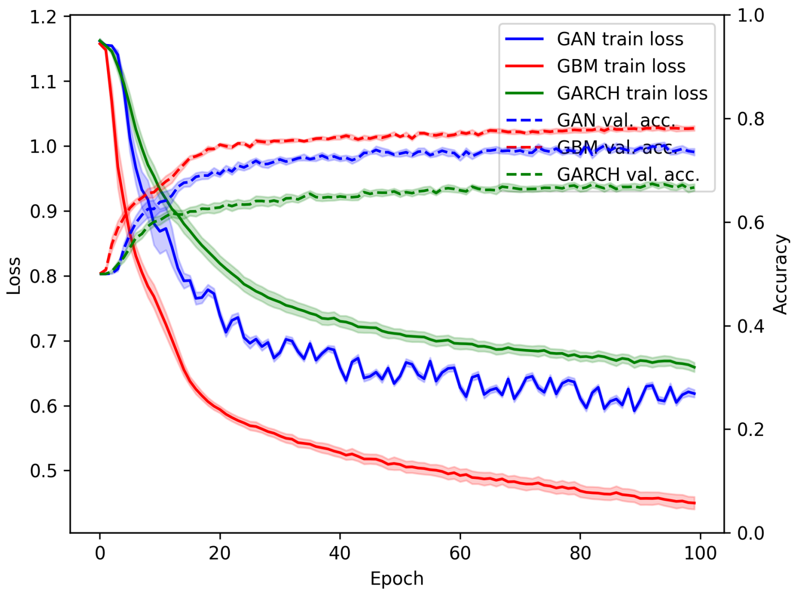

Figure 4: Sequence length 20

answer. In the worst, or least understandable case, the previous outputs seem like random noise.)

5.4 Discussion

We surprisingly conclude that GAN data is more detectable than GARCH data, despite the latter's simplicity. It should be noted this does not imply that GARCH data fits the real market data better than GAN, since the GARCH data may be particularly ill-suited to be distinguished by a small LSTM. That said, it does mean that GAN has identifiable quirks which are not present in the real or GARCH data, and we would argue that if the generated financial data can be easily distinguished from real data, it likely is not a suitable approximation. This is especially true if the generated data is to be used as training data for another machine learning setup. It also suggests that the generator might be better trained with an LSTM discriminator. Finally, it exemplifies an innate disadvantage of the GAN framework: the quality of the generator depends on the quality of the discriminator. And so, even though inspection of the generator (which was successfully trained) by comparing histograms etc. produces satisfying results, it turns out that the discriminator did not capture a simple characteristic that a small LSTM model could, and thus the resulting generator may be flawed.

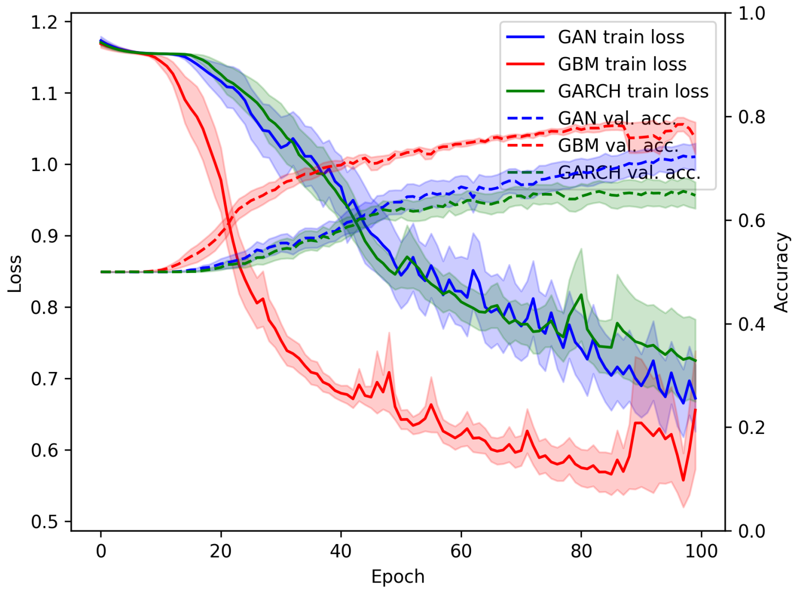

Figure 5: Sequence length 60

References

[1] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.* 2018. arXiv: `1803.01271 [cs.LG]`. URL: `https://arxiv.org/abs/1803.01271`.

[2] Fischer Black and Myron Scholes. "The Pricing of Options and Corporate Liabilities". In: *Journal of Political Economy* 81.3 (1973), pp. 637–654. ISSN: 00223808, 1537534X. URL: `http://www.jstor.org/stable/1831029` (visited on 01/20/2026).

[3] Georg M. Goerg. *The Lambert Way to Gaussianize heavy tailed data with the inverse of Tukey's h as a special case.* 2012. arXiv: `1010.2265 [math.ST]`. URL: `https://arxiv.org/abs/1010.2265`.

[4] Ian J. Goodfellow et al. *Generative Adversarial Networks.* 2014. arXiv: `1406.2661 [stat.ML]`. URL: `https://arxiv.org/abs/1406.2661`.

[5] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. eprint: `https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf`. URL: `https://doi.org/10.1162/neco.1997.9.8.1735`.

[6] Kookjin Lee, Jaideep Ray, and Cosmin Safta. "The predictive skill of convolutional neural networks models for disease forecasting". In: *PLOS ONE* 16 (July 2021), e0254319. DOI: `10.1371/journal.pone.0254319`.

[7] Fernando de Meer Pardo. "Enriching Financial Datasets with Generative Adversarial Networks". MA thesis. Delft University of Technology, 2019. URL: `https://repository.tudelft.nl/record/uuid:51d69925-fb7b-4e82-9ba6-f8295f96705c`.

[8] James Meldrum et al. *New Money: A Systematic Review of Synthetic Data Generation for Finance*. 2025. arXiv: `2510.26076 [cs.LG]`. URL: `https://arxiv.org/abs/2510.26076`.

[9] Bernt Øksendal. *Stochastic Differential Equations An Introduction with Applications*. Springer, 2003.

[10] Richard A. Davis Peter J. Brockwell. *Time Series: Theory and Methods*. Springer, 1991.

[11] Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. "Modeling financial time-series with generative adversarial networks". In: *Physica A: Statistical Mechanics and its Applications* 527 (2019), p. 121261. ISSN: 0378-4371. DOI: `https://doi.org/10.1016/j.physa.2019.121261`. URL: `https://www.sciencedirect.com/science/article/pii/S0378437119307277`.

[12] Magnus Wiese et al. "Quant GANs: deep generation of financial time series". In: *Quantitative Finance* 20 (Apr. 2020), pp. 1–22. DOI: `10.1080/14697688.2020.1730426`.

[13] Xingyu Zhou et al. "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets". In: *Mathematical Problems in Engineering* 2018.1 (2018), p. 4907423. DOI: `https://doi.org/10.1155/2018/4907423`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1155/2018/4907423`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/4907423`.