Generative Adversarial Network for Financial Time Series

Mirko Corletto
Technical University of Munich
Department of Electrical and Computer Engineering
Munich, Germany
mirko.corletto@tum.de

I. Introduction

Generative Adversarial Networks (GANs) have become the state of the art in image generation tasks. While image generation is certainly the most common use case for GANs, there are other data structure types where an unsupervised generation of new data can be very valuable. In this project, I will deal with the generation of new time series and especially with financial time series. Financial time series (such as indices or stocks) are highly non-stationary and strongly affected by noise. Additionally, there exists only a limited amount of data for financial time series. This makes it very hard for machine learning (ML) models to generalize well on the data and usually leads to overfitting. If a GAN would be able to generate new time series with a similar structure as the underlying financial time series, the synthetically generated time series could be used to train a ML predictor and thus reduce overfitting.

GANs can not only be used to generate random data with a specific structure, but they can also be used to generate data with specific properties (e.g changing a person's hairstyle in human face generation) [4]. For financial time series, this could be a specific trend for example. Financial time series behave quite differently in bullish and bearish market cycles. By generating new time series with a specific trend, the performance of a ML predictor could be evaluated in different market conditions.

II. Related Work

GANs were first introduced by Ian Goodfellow et.al. in 2014 [2]. In their original form, GANs are pretty hard to train. Therefore multiple improvements to the original GAN have been proposed over the last couple of years. Lucic et.al performed an empirical study, comparing the performance of multiple states of the art GANs [5]. The Wasserstein GAN (WGAN) uses the Wasserstein distance to calculate the loss, resulting in a much smoother convergence. Since calculating the Wasserstein distance in its original form is intractable, the authors of the original paper proposed to clip the weights between $[-0.01, 0.01]$ to enforcing a Lipschitz constraint [1]. While WGAN certainly improved the performance of GANs, clipping the weights between $[-0.01, 0.01]$ introduces new complications. Another way to assure Lipschitz continuity, which uses spectral normalization, was proposed by Miyato et.al [6].

When dealing with time series, the generator and discriminator should be able to capture temporal dependencies. Classical convolutional neural networks are not particularly suited for this kind of task. Mogren proposed to use LSTMs to model the generator and discriminator of a GAN for the generation of new music [7]. Wiese et.al. used Temporal Convolutional Networks (TCN) to model the generator and discriminator of their GAN. They generated new financial time series based on the daily returns of the S&P500 index [9].

III. Finding the Optimal Model

During the implementation of my project, I proceeded systematically. Starting from the basic structure of the GAN that was introduced during class, I first replaced the 2D CNNs of the generator and discriminator with 1D TCNs. Because the PyTorch module torch.nn does not support TCN blocks yet, I used the TCN implementation from locuslab[1]. The high non-stationarity of financial time series does not only complicate the generation of new samples but also makes it quite difficult to verify the quality of the newly generated samples. Therefore I decided to check the behavior of my time series GAN as well as tuning hyperparameters and implementing improvements based on the following simple experiment.

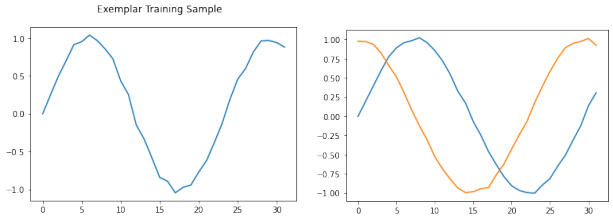
A. Experiment 1: Sinus Wave Time Series Generation

First of all, I created 1.000 sinus waves with uniform random frequency and random noise of the following structure:

$$y = \sin(\frac{x}{randu(4,16)} * \pi) + randn(32) * randu(0, 0.05) \quad (1)$$

In a second moment, I created additional cosine waves of the same structure in order to check whether my implementation works for multivariate time series. Figure 3 shows one exemplar training sample of the univariate sinus wave time series on the left side and one exemplar training sample of the multivariate sine/cosine time series on the right side. After I managed to generate new sine waves with my base-structure TCN time series GAN implementation, I proceeded with implementing the changes of the spectrally normalized GAN (SN-GAN) described in [6]. For this purpose, I exchanged the weight

[1] https://github.com/locuslab/TCN/blob/master/TCN/tcn.py



(a) univariate (b) multivariate

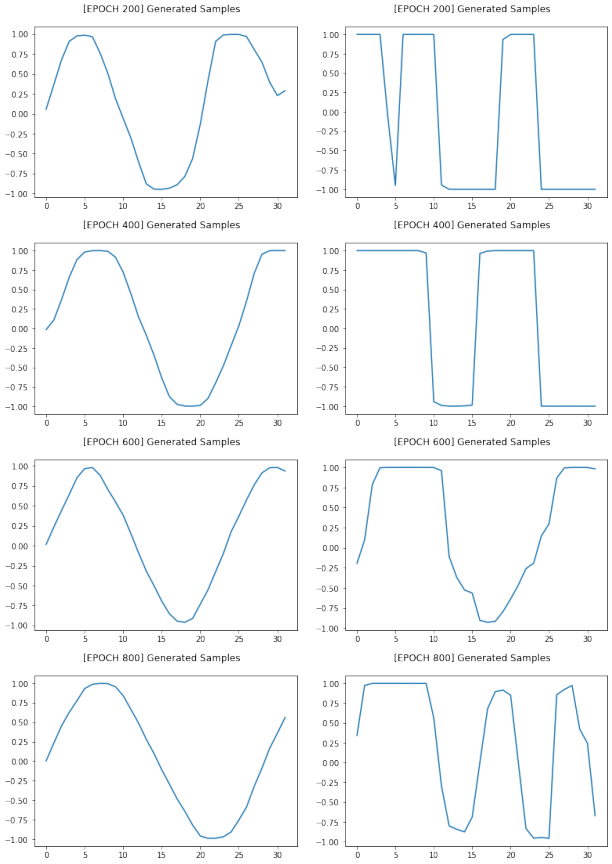
Fig. 1: Exemplar univariate and multivariate training sample of the first experiment.

normalization of the convolutional layers with the spectral normalization from pytorch[2]. Since using Wasserstein distance to calculate loss should result in smoother convergence, I chose to use Wasserstein distance instead of binary cross entropy. However, with the Wasserstein distance I was not able to create new sinus wave samples. I assume this has something to do with not respecting the Lipschitz continuity due to my TCN model structure. To ensure the Lipschitz continuity, I added an additional gradient penalty resulting in the WGAN-GP implementation as proposed in the paper of Gulrajani et.al. [3]. With an additional gradient penalty, I was able to generate new sinus wave samples. However, as Figure 2 shows, SN-GAN with binary cross-entropy loss had a smoother and faster convergence as WGAN-GP with spectral normalization.

After some further hyperparameter tweaks and model improvements (e.g. flipping positive and negative labels and smoothening the labels as proposed in the medium article "Keep Calm and train a GAN."[3]), I had my final model structure. Figure 1 compares the multivariate time series generation of the final TCN SN-GAN structure with the multivariate time series generation of the base TCN GAN I started from. With the final SN-GAN structure, I was able to generate new multivariate sine/cosine wave samples after approx. 300 epochs, whereas with the base TCN GAN it took about 3000 epochs to converge to a good result (see Figure 3).

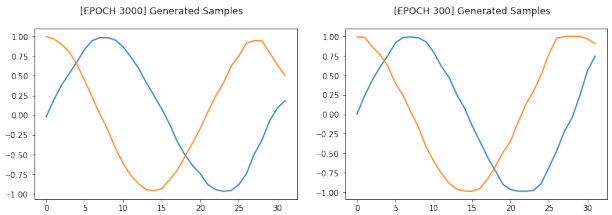
[2]https://pytorch.org/docs/stable/generated/torch.nn.utils.spectral_norm.html

[3]https://medium.com/@utk.is.here/keep-calm-and-train-a-gan-pitfalls-and-tips-on-training-generative-adversarial-networks-edd529764aa9



(a) SN-GAN with binary cross entropy loss. (b) WGAN-GP with spectral normalization.

Fig. 2: Results of the first experiment where I generated univariate sinus waves.



(a) Base TCN GAN (b) Final TCN SN-GAN

Fig. 3: Convergence comparison of the base TCN GAN and the final TCN SN-GAN.

IV. Final Model Structure

As pointed out in the proposal, the TCN model uses dilated causal 1D convolutions to model the sequence to sequence structure that is needed to capture long and short-term dependencies (see Figure 4).

Both, the generator and discriminator have 8 stacked temporal blocks that are composed of two 1D dilated convolutional layers. The 1D convolutions are performed with a kernel size of 8 and stride of 1. Starting with a padding of 7 in the first temporal block, the padding is doubled in each temporal block,

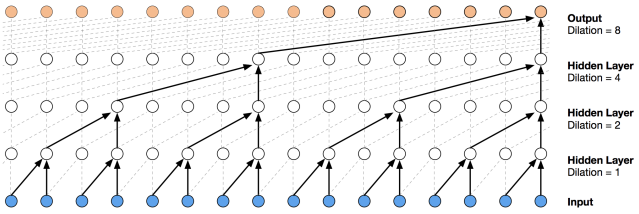


Fig. 4: Dilated causal 1D convolution structure (temporal block of the TCN model) [8].

reaching a padding of 896 in the last temporal block. The dilation is increased exponentially from 1 in the first temporal block to 128 in the last temporal block. The weights of all convolutional layers are normalized with spectral normalization. Additional regularization is applied for the generator weights by using dropout with a probability of 20%.

The generator takes as input a random noise vector of size 100 and returns new samples of shape [B,τ, i], where B is the batch size which was set to 64, τ the observation length which was set to 32 and i the number of features.

The discriminator takes as input time series of shape [B,τ, i], and returns values of shape [B, τ, 1] which are then used to calculate the binary cross-entropy loss for real and fake samples. To calculate the cross-entropy, real samples are labeled with 0.1 and fake samples with 0.9.

For both, the training of the generator and the training of the discriminator I used the Adam optimizer with learning rate $lr = 2e - 4$ and betas $\beta = (0.5, 0.999)$.

V. Dataset

Unlike what was specified in the proposal, I decided to stick with the generation of multivariate time series and discard the dataset that only contains the daily returns. The reason for this decision is that I don't see any use case in creating univariate financial time series. For just one single feature, the amount of already available data should be enough for a good generalization. With the curse of dimensionality, the need for more training samples increases exponentially with additional features. Additionally, instead of generating new financial time series based on the Dow Jones Industrial Average, I decided to use Dogecoin as the underlying asset (see Figure 5).

Fig. 5: Dogecoin[4]

The Dogecoin financial time series data was fetched through the yfinance API[5]. The historical price data of Dogecoin that was fetched consists of the following five base features of a financial time series:

- Open: price of the first trade that happened in the sampling period
- Close: price of the last trade that happened in the sampling period
- High: highest price reached during the sampling period
- Low: lowest price reached during the sampling period
- Volume: number of shares traded during the sampling period

In total, I used used 4 years of historical price data with daily features. Figure 6 illustrates the base structure of a financial time series in a so called candlestick chart.

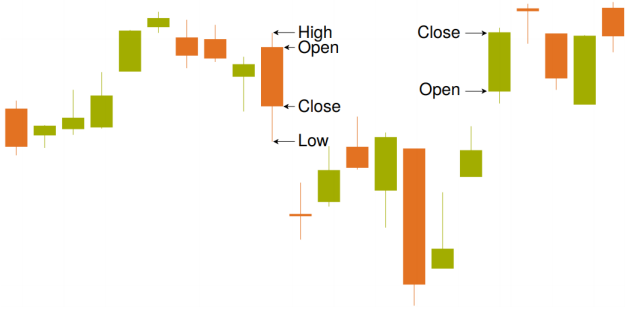


Fig. 6: Candlestick chart: each candle corresponds to one daily sample.

VI. Preprocessing

First of all, I split the dataset into 80% training and 20% test set. Since we are dealing with time series, no shuffling was performed before the data split. In the proposal, I mentioned standardizing the dataset using the mean and standard deviation of the training set. However, due to the high non-stationarity of financial time series, I switched to a min-max scaler instead. After scaling the training and test set between 0 and 1, I created sequential observations of length $\tau = 32$ with the following sliding window approach:

$$o(t) = x_{t:(t+\tau-1)}, \tag{2}$$

Figure 7 shows the candlestick plot of one input observation (without volume) after applying the sliding window approach and after rescaling.

[4]https://thetab.com/uk/2021/02/05/dogecoin-rich-194021
[5]https://pypi.org/project/yfinance/

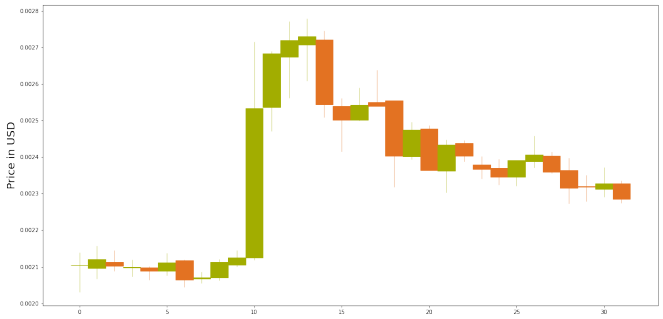


Fig. 7: Candlestick plot of one real Dogecoin observation after rescaling.

VII. Results and Discussion

Unfortunately evaluating GANs is not as straightforward as evaluating a classifier or regressor. The highly non-stationary characteristics of financial time series make the evaluation even more complicated. Train on synthetic, test on real (TSTR) would be one common method to evaluate the generated samples of a time series GAN. But the predictiveness of financial time series is in itself very controversial. In the proposal, I mentioned the use of autoregression to measure the similarity between the real financial time series and the synthetic generated financial time series. But even with that method, no particularly meaningful conclusions can be drawn because of the non-stationarity.

Having said that, I decided to focus on a qualitative evaluation. Despite the multidimensionality, candlestick charts provide a compact and intuitive representation of financial time series. In Figure 6 we can see four general properties of financial time series:

1) Green candles have a higher closing price than opening price.
2) Orange candles have a lower closing price than opening price.
3) The opening price at time $t+1$ should be similar to the closing price at time t. This property is not always true for stocks (due to exchange opening hours), but it almost always holds true for cryptocurrencies like Dogecoin (cryptocurrencies can be traded 24/7).
4) **T**he body of a candle must always be within the high and low wicks. For orange candles, the low can't be higher than the closing price and the high can't be lower than the opening price.

In order to obtain a meaningful result, I generated four random synthetical Dogecoin time-series samples (see Figure 8). The samples were generated after 2000 training epochs. The discriminator loss was around 0.5, so perfectly in line with what we would expect from the GAN.

At first glance, the generated synthetical financial time series appear to be pretty similar to a real financial time series. The GAN has definitely learned some sequential dependencies from the original training data. Furthermore, properties (1) to (4) seem to hold in most cases. Especially property (4) was never invalidated. However, when taking a closer look at the single candlestick charts, a human can spot that something is off in each of the four charts. In the first candlestick chart of Figure 8, the 7th and 29th candlestick appear to be way off. Candlestick 7 should be orange and candlestick 29 green. In the second candlestick chart, there is an uptrend between samples 17 and 25. Still, almost all of those candles are orange, while they should be green. In the third candlestick chart, candles 1 and 17 seem to be off. The opening price of candle 2 is way higher than the closing price of candle 1 and candle 17 should be orange. In the last candlestick chart, the first two

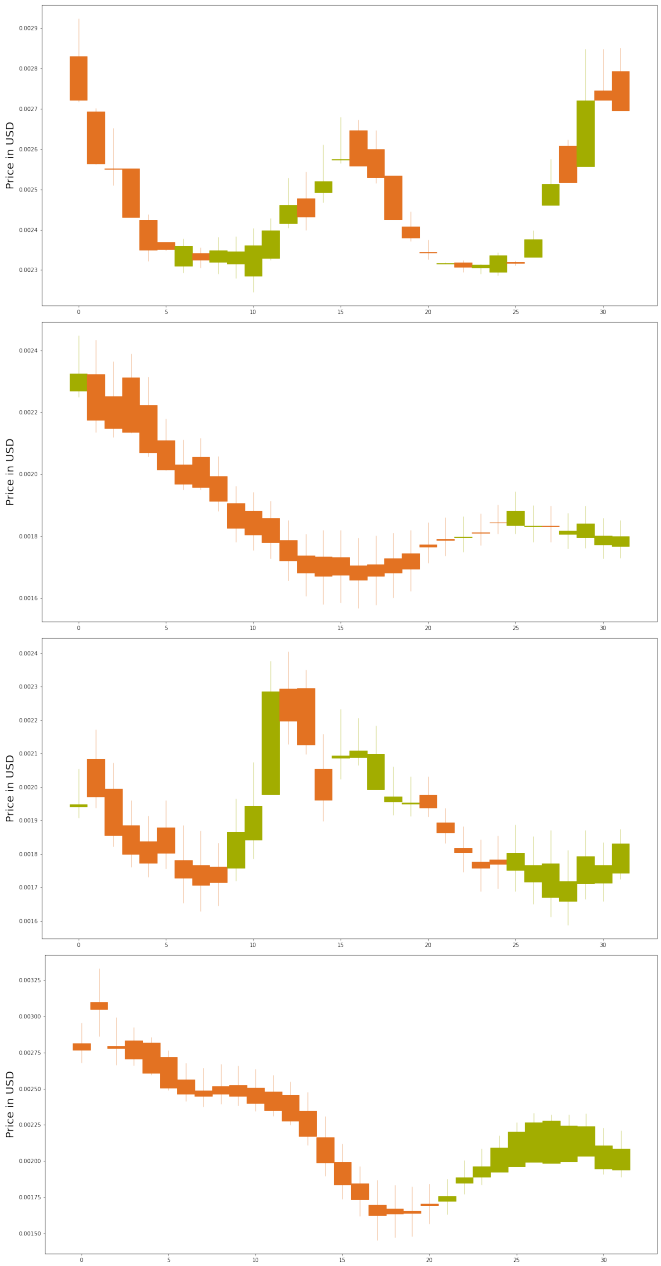


Fig. 8: Generation of new Dogecoin time series samples after 2000 epochs of training.

candles and the last seven candles appear to be wrong.

Given the high non-stationarity of financial time series, I am pretty satisfied with the quality of the first generated synthetical Dogecoin sample in Figure 8. However, as we can see not all generated samples turned out as well. Therefore, the model needs to be further optimized and improved in order to be used in real-world applications.

By experimenting with different scaling strategies, I found out that scaling each input observation independently works way better than using the whole training set as stated in section VI. The GAN was not only able to converge smoother and faster, but also the final results were significantly better. Since this method does not allow a direct rescaling of the generated samples, I will present the results of the raw generator output.

In Figure 9 we can see a scaled Dogecoin observation. Please note that due to an independent scaling of every single feature, property (4) of a financial time series doesn't hold after scaling. Nevertheless, properties (1), (2), and (3) remain intact and can be used for a qualitative evaluation.

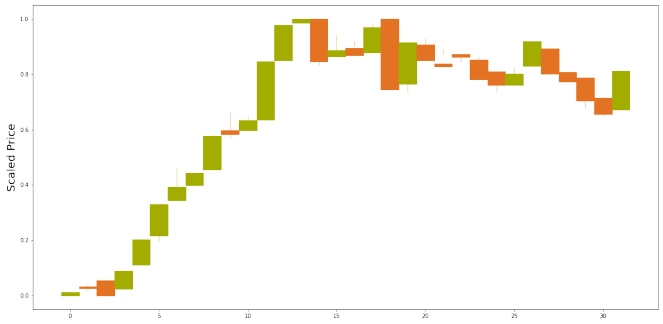


Fig. 9: Candlestick plot of one real Dogecoin observation without rescaling.

In this experiment, the GAN was only trained for 500 epochs. The four randomly generated synthetical Dogecoin samples can be seen in Figure 10. After close examination of the four candlestick charts, no invalidation of properties (1), (2) or (3) could be found.

To further validate the proposed method, the model would need to be extended so that it can learn the rescaling parameters. Only then could property (4) be evaluated appropriately.

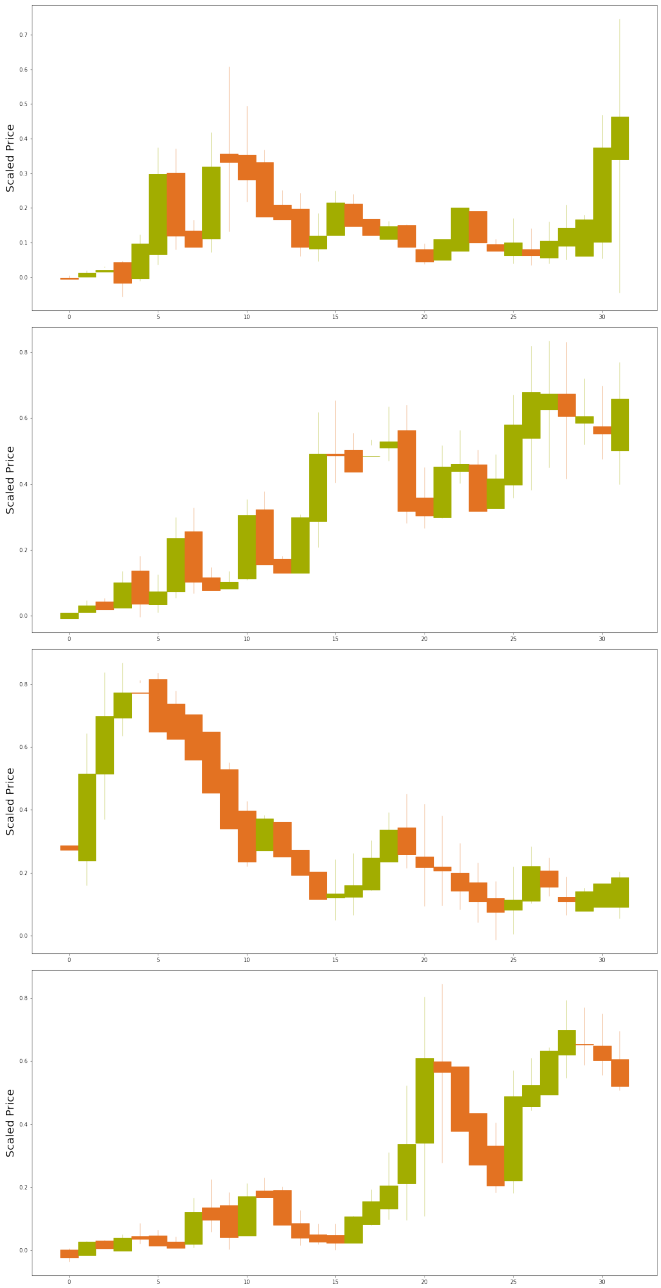


Fig. 10: Generation of new Dogecoin time series samples (without rescaling) after 500 epochs of training.

VIII. Conclusion

In this project, I build a generative adversarial network (GAN) for the generation of financial time series. To capture the sequential properties of financial time series, the generator and discriminator were implemented with a temporal convolutional network (TCN). After some model optimizations, the implemented GAN was able to generate new univariate and multivariate sine/cosine waves. Using Dogecoin as the underlying asset, the GAN was finally able to generate fairly real-looking financial time series.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein gan". In: *arXiv preprint arXiv:1701.07875* (2017).

[2] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014), pp. 2672–2680.

[3] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *arXiv preprint arXiv:1704.00028* (2017).

[4] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[5] Mario Lucic et al. "Are gans created equal? a large-scale study". In: *Advances in neural information processing systems*. 2018, pp. 700–709.

[6] Takeru Miyato et al. "Spectral normalization for generative adversarial networks". In: *arXiv preprint arXiv:1802.05957* (2018).

[7] Olof Mogren. "C-RNN-GAN: Continuous recurrent neural networks with adversarial training". In: *arXiv preprint arXiv:1611.09904* (2016).

[8] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[9] Magnus Wiese et al. "Quant gans: Deep generation of financial time series". In: *Quantitative Finance* (2020), pp. 1–22.