Generative Adversarial Networks for financial time-series

A Preprint

Ali Hirsa
IEOR Department
Data Science Institute
Columbia University
`ali.hirsa@columbia.edu`

Joerg Osterrieder[*]
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
`joerg.osterrieder@zhaw.ch`

Weilong Fu
IEOR Department
Data Science Institute
Columbia University
`weilong.fu@columbia.edu`

Branka Hadji Misheva[*]
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
`branka.hadjimisheva@zhaw.ch`

Romain Délèze
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
`delezrom@students.zhaw.ch`

Danijel Jevtic
School of Engineering
Zurich University of Applied Sciences
Winterthur, Switzerland
`jevtidan@students.zhaw.ch`

December 23, 2021

[*]Financial support by the Swiss National Science Foundation within the project "Mathematics and Fintech - the next revolution in the digital transformation of the Finance industry" is gratefully acknowledged by the corresponding author. This research has also received funding from the European Union's Horizon 2020 research and innovation program FIN-TECH: A Financial supervision and Technology compliance training programme under the grant agreement No 825215 (Topic: ICT-35-2018, Type of action: CSA). Furthermore, this article is based upon work from the COST Action 19130 Fintech and Artificial Intelligence in Finance, supported by COST (European Cooperation in Science and Technology), www.cost.eu (Action Chair: Joerg Osterrieder).
The authors are grateful to Stephan Sturm, Moritz Pfenninger, Samuel Rikli, Bigler Daniel, Antonio Rosolia, management committee members of the COST (Cooperation in Science and Technology) Action Fintech and Artificial Intelligence in Finance as well as speakers and participants of the 5th and 6th European COST Conference on Artificial Intelligence in Finance and Industry, which took place at Zurich University of Applied Sciences, Switzerland, in September 2020 and 2021.

ABSTRACT

Neural networks and deep learning have been very successful in many domains in the last years and their popularity is ever growing. However, in the finance area, substantial breakthroughs have not been seen yet. Deep learning algorithms are making progress in the field of data-driven modelling, but the lack of sufficient data to train these models is currently holding back substantial progress. One of the underlying reasons is that modelling financial data is inherently a challenging task: the data often has complex statistical properties, its inner workings are largely unknown, and the noise-to-signal ratio is generally very high. Most financial models and algorithms trying to fill the lack of historical financial time series struggle to perform and are highly vulnerable to overfitting. Generative Adversarial Networks (GANs) are a neural network architecture family that have achieved good results in image generation and are being applied to generate time series and other types of financial data. We provide a detailed overview of the evolution of GANs for modelling financial time-series and will apply a conditional Wasserstein GAN with gradient penalty to financial time-series, corresponding to a min-max two-player game. Our deep neural network, the conditional WGAN-GP, consists of a generator and discriminator function which utilize a convolutional architecture, being trained via backpropagation. To test the performance of GANs from a data-driven perspective, we are testing several GAN architectures on financial market datasets which as far as we know, have not yet been explored in such a setting. Those are: The generated synthetic time-series are visually indistinguishable from the real data and the numerical results show that the generated data are close to the real data distribution. By generating artificial time-series of E-mini S&P500 futures in 5 minutely data, we will reproduce most of its stylized facts and are then furthermore equipped to conduct thorough statistical simulations.

Furthermore, we will use the traditional risk-measures like Value-at-Risk, Expected Shortfall, Maximum Drawdown and Sharpe Ratio to quantify that. This research will help quantitative analysts, investors, and regulatory authorities to better understand the risks inherent in futures marked as well as to generate synthetic time-series data.

***K*eywords** WGAN-GP, Wasserstein Loss, Generative Adversarial Networks, financial time-series, Deep neural networks, Generator, financial data, discriminator

Contents

1 Introduction

Data is the new oil. But just as with oil (or renewable energy going forward), there is never enough of it and it needs to be refined before it can be used in applications. We look at the recently develop Generative Adversarial Networks [1] applied in a financial market setting.

1.1 Financial Markets and Financial Data

The finance industry is one of the most influential fields impacted by new developments in Artificial Intelligence. More specifically machine learning has been deployed to forecasting, customer service, risk management and portfolio management, among others. The finance industry generates huge amounts of data which offers new opportunities for models to be applied and used to obtain better knowledge and spot opportunities.

Financial markets, like the economy, are highly complex systems where it is often impossible to explain macro phenomena by a simple summation of micro processes or events. It's hard to predict the results of actions in the system and sometimes impossible to find the causes of large abnormalities, or even to single out factors that influenced an event. In the 2010 flash crash [2] for instance, plausible explanations only surfaced years after the episode, and even in specific segments, studies [3] have found that tools for market analysis show an incomplete picture. Modelling in this setting involves dealing with a great deal of uncertainty.

While there are large amounts of data generated by the banking and financial sectors, there is still a shortage of data if one wants to apply deep neural networks. This scarcity of historical and other types of data as well as the data-intensive nature of ML models has been a limiting barrier to achieve better models.

1.2 Financial time-series modelling

Modelling financial time series comes with its own challenges since the dynamics of financial markets are very complex in nature, and the mechanism generating the data, and thus its original distribution, are unknown.

Three main distinguishable streams of modelling financial time-series exist.
Notably there are the **ARCH/GARCH** [4] family of models, which rely on classical statistics and model the change in variance over time in a time series by describing the variance of the current error term as a function of past errors. Often the variance is related to the squares of the previous innovations. As AR (auto regressive) models, they heavily rely on past information to construct a prediction. They are commonly used in time series modelling where time-varying volatility and volatility clustering are present. Another main family of models are agent-based models (**ABM**s). In agent-based models, actions and interactions of autonomous agents are simulated in order to understand the behaviour of a system and what governs its outcomes. The agents are entities, typically represented computationally as objects. These agents hold a state, which can be any data that describes the agent. ABMs simulate interactions of multiple agents to re-create and predict the behaviour of complex phenomena, where the goal is to generate higher level properties from the interaction of lower-level agents. Third, by adopting a purely data-driven modelling approach, combined with neural networks, we can overcome the shortcomings of both the model-based and the agent-based approach. A large zoo of techniques to model distributions exists, see e.g. [1]. We will focus on Generative Adversarial Networks (GANs). They are a group of unsupervised learning algorithms in computer science, first introduced by Ian Goodfellow and his colleagues [1] in 2014 to build a generative adversarial Network architecture that creates new data instances that resemble the training data to create images. (Fig.2) [1] shows the first-ever generated fake images using a GAN. Nowadays, GANs are also used to produce diverse types of synthetic financial data and may provide good results, since they can generate data by sampling only from real data, often with no additional assumptions or inputs. Driven by the largely empirical aspect of financial data, the avoidance of possible human bias being infiltrated into the modelling process could possibly be a step forward in this field of study.

1.3 Neural networks

Neural networks are a subset of machine learning techniques and are at the heart of deep learning algorithms. The human brain inspires their name and structure. Deep neural networks (Fig.1) are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. In the simplest case, each node is connected to another and has an associated weight and threshold. If the output of any individual perceptron is above the specified threshold value, that perceptron is activated, sending data to the next layer of the network. Otherwise, no data passes along the next layer of the network.

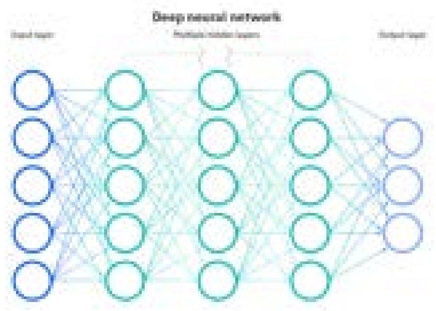


Figure 1: Structure of a deep neural network [5]

Neural networks rely on training data to learn and improve their accuracy over time. Once these learning algorithms achieve high accuracy, they are potent tools in computer science and artificial intelligence, allowing us to classify and cluster data as well as serve as universal approximation tools to unknown functions. [6]

1.4 Research Contribution

This paper gives an overview of the recent framework of Generative Adversarial Networks for generating synthetic data. The goal is to provide an understanding of how GANs work, present the current state of research and recent practical applications. Additionally, we are applying a conditional Wasserstein GAN with Gradient penalty to generate synthetic financial time series of E-mini S&P500 futures. Using those time-series we can further analyse the risks inherent in E-mini S&P500 futures for financial investors.

1.5 Structure of the paper

The rest of the paper is organised as follows: In Section 2 & 3, we introduce the background literature on GANs, current research and similar approaches. Section 4 introduces neural networks, with Section 5 focusing on Generative Adversarial Networks. Section 6 focuses on statistical methods for time-series analysis. Next, in Section 7, we set up our model and show the results of generating our financial time-series. We implement a conditional Wasserstein-GAN with gradient penalty and a convolutional architecture applied to E-mini S&P500 futures. The results are evaluated by comparing the generated synthetic distribution against the historic data distribution. Lastly, we will conclude in Section 8 and give an outlook on further research and applications.

Figure 2: First-ever created fake images by a GAN [7]

2 Literature Overview

Our research is combining three main building blocks: Financial time-series, deep neural networks, and Generative Adversarial Networks. We give a short overview of existing research in each of those three areas.

2.1 Deep learning and neural networks

Deep Learning is a subfield of machine learning that deals with algorithms inspired by the structure and function of the brain, called artificial neural networks.[8] In the meantime, there are many applications in the field of Deep Learning. In particular, Deep Learning architecture in general has grown. The most common architectures are Deep Neural Networks, Deep Belief Networks, Deep Reinforcement Learning, Recurrent Neural Networks and Convolutional Neural Networks. These algorithms are used in computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug discovery, medical image analysis, materials testing and board game programming, where they have produced results comparable to, and in some cases exceeding, the performance of human experts.[9]

2.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a relatively new class of machine learning frameworks designed by Ian Goodfellow et al. in 2014 [1]. Wang et al [10] provide a survey and taxonomy of GANs for computer vision. For additional overviews on GANs, the following papers provide an excellent starting point: [11], focusing on the underlying theory and explaining the connections between different GAN variants; [12] undertake an extensive database search to classify GAN research into its application areas, [13] introduces the main concepts and the theory of GANs, while examining their applications in computer vision; [10] analyse the relation between GANs and parallel intelligence, after having given an overview; [14] analyse regularization methods for GANs. We provide a detailed overview of the evolution of GANs in Section 3. In (Fig.3) we see an overview of the most common GANs, divided into arichitecture and their loss function.

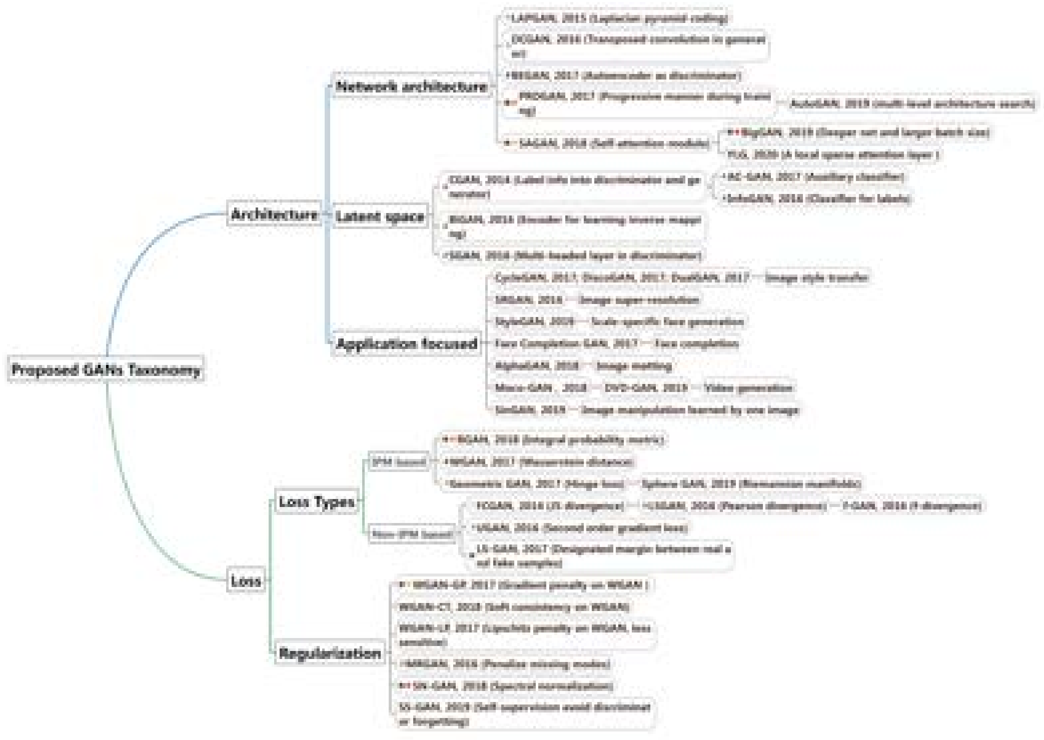

Proposed GANs Taxonomy
Architecture
Network architecture
LAPGAN, 2015 (Laplacian pyramid coding)
DCGAN, 2016 (Transposed convolution in generator)
BEGAN, 2017 (Autoencoder as discriminator)
PROGAN, 2017 (Progressive manner during training)
AutoGAN, 2019 (multi-level architecture search)
SAGAN, 2018 (Self-attention module)
BigGAN, 2019 (Deeper net and larger batch size)
YLG, 2020 (A local sparse attention layer)
Latent space
CGAN, 2014 (Label info into discriminator and generator)
AC-GAN, 2017 (Auxiliary classifier)
InfoGAN, 2016 (Classifier for labels)
BiGAN, 2016 (Encoder for learning inverse mapping)
SGAN, 2016 (Multi-headed layer in discriminator)
Application focused
CycleGAN, 2017, DiscoGAN, 2017, DualGAN, 2017
Image style transfer
SRGAN, 2016
Image super-resolution
StyleGAN, 2019
Scale-specific face generation
Face Completion GAN, 2017
Face completion
AlphaGAN, 2018
Image matting
Moco-GAN, 2018
DVD-GAN, 2019
Video generation
SinGAN, 2019
Image manipulation learned by one image
Loss
Loss Types
IPM based
RGAN, 2018 (Integral probability metric)
WGAN, 2017 (Wasserstein distance)
Geometric GAN, 2017 (Hinge loss)
Sphere GAN, 2019 (Riemannian manifolds)
Non-IPM based
FCGAN, 2016 (JS divergence)
LSGAN, 2016 (Pearson divergence)
f-GAN, 2016 (f-divergence)
UGAN, 2016 (Second order gradient loss)
LS-GAN, 2017 (Designated margin between real and fake samples)
Regularization
WGAN-GP, 2017 (Gradient penalty on WGAN)
WGAN-CT, 2018 (Soft consistency on WGAN)
WGAN-LP, 2017 (Lipschitz penalty on WGAN, less sensitive)
MRGAN, 2016 (Penalize missing modes)
SN-GAN, 2018 (Spectral normalization)
SS-GAN, 2019 (Self-supervision avoid discriminator forgetting)

Figure 3: Taxonomy of GANs, see [10]

2.3 Stylized facts and risks of financial time-series

The dynamics of financial markets can be defined as examples of complex phenomena [15], since the underlying mechanics of the processes are largely unknown. However, years of research in financial time series have shown that although they may look completely random, variations of asset prices share several significant statistical properties, which are common across a multitude of markets and timeframes. These properties are known as stylized empirical facts, or just stylized facts. The most important Stylized Facts are captured in.[16]

Stylized facts were gathered by taking statistical features of asset returns found in studies about many markets and instruments. A causal analysis of distinct economic scenarios over the globe may expect that being influenced by different events and environment would result in different statistical properties of the said returns. Still, a result of decades of empirical studies of financial markets has shown that some key characteristics are constantly present across markets with different assumed characteristics. Due to these generalizations, the obtained features end up losing in precision, giving the stylized facts a more qualitative aspect. Is important to note that these qualitative properties are not easy to reproduce via modelling of stochastic processes and end up being essential in modelling asset price dynamics, as it is expected that these models can capture/replicate these statistical features.

From the proposed stylized facts[17] a selection of five is portrayed here based on their relevance and use in the modelling literature:

• **Linear unpredictability** or **absence of autocorrelations**: Except for short intra-day time scales, it is expected that asset returns show minimal linear autocorrelations.

• **Fat-tailed distribution** or **heavy tails**: the distribution of returns seems to display a power-law or Pareto-like tail, which in practical terms means that the probability of extreme values occurring is much larger than in a normally distributed dataset.

• **Volatility clustering**: different measures of volatility display a positive autocorrelation over several days. Meaning that there is a tendency of high volatility events to cluster together. Simplifying, large price changes tend to be followed by large changes and small price changes tend to be followed by small changes.

• **Gain/loss asymmetry**: Observations have shown that large plunges in stock prices and index values do no share equally large upward movements.

• **Aggregational Gaussianity**: Over larger time scales, the distribution of returns looks like a normal distribution.

3 The evolution of GANs in Finance - Overview and State of Research

The possibility of recreating complex distributions via GANs has captured the attention of quantitative finance researchers, although GANs most notorious application lies in computer vision and image generation, there is some promising research on the field of data-driven modelling in finance, especially its application to time series analysis and generation.

There is currently a steady development undergoing in the broad field of finance, with several applications being researched for generative adversarial networks. Some of the principal developments, also portrayed in tabel 1, are in market prediction [18], tuning of trading models [19], portfolio management [20] and optimization [21], synthetic data generation [15] and diverse types of fraud detection [22].

When modelling financial time series, there are several models used and developed over the years, such as the earlier mentioned ARCH/GARCH models, agent-based models, and various additional time-series models, predominantly modelling volatility, such as the Heston model. However, until the advent of machine learning, made possible by the ever-increasing available computational power, development of financial models has been slow. Purely data-driven modelling via neural networks and machine learning is a growing sub-field of research and has the potential to improve the modelling of complex and unknown statistical dynamics present in financial data.

3.1 About the need for synthetic data

Among the different applications of GANs being researched in finance, the topic of generation of synthetic datasets deserves awareness. Synthetic data is important because it can be tailor-made for specific uses or conditions where real data may be lacking or unavailable. This can be useful in numerous cases such as:

• Privacy and compliance rules may severely limit data availability and its application [23].
• Data is often required in product testing environments and is often limited or unavailable to testers [24].
• Machine learning requires large amounts of training data, such data can be expensive and scarce [25].

The growth of data-driven processes and the new possibilities provided by this kind of analytics and modelling created a higher demand for data and data scientists in finance. Financial data-sets are often very regulated, which limits their use in developing new products and processes. As described by JP Morgans' AI Research[26], anonymization is unreliable, and encryption can cripple the use cases of data. Statistically consistent synthetic financial data can solve most of these limitations, producing high-volume artificial test data also greatly improves scalability and cooperative work with usually sensitive or limited data-sets.

With bootstrapping, the data set can be as large as we want, but we do not gain any new information. In Monte Carlo simulation, we have to assume normally distributed data, which is very rare in finance. With GANs, both problems are solved because we generate new data, we gain information and we do not have to make distributional assumptions. Any complex distribution can be simulated with GANs. This is why GANs are so valuable for such simulations. The disadvantage is that we have to develop a new GAN for every application. This means that there is no general GAN that can be used for all problems.

3.2 Main GANs in financial research

Many GAN variants, for a non-exhaustive list see e.g. [11], [12], [13], [40], [14], have been proposed in the literature to improve performance, these can mainly be divided into two types: Architectural variations and different loss functions, including their regularizations. In the architecture variants, structural changes are made to adapt the GAN to a certain purpose, ranging from different neural networks for discriminator and generator to ensemble methods, see e.g. [41], or to improve its overall performance. In loss variants, different approaches to loss functions try to improve the stability and the performance of the training phase, often focussing on the issue of non-convergence. Modifications have been made to tailor each network to its specific goal and the underlying data-set. Overall, the main topic of GAN research is and remains centred around image generation and computer vision, as well as ensuring convergence of the training process. Even so, based on the substantial focus on data and the need for substantial amounts of it for deep learning

Table 1: GANs in Finance

Field	Application	Method
Time Series Forecasting	Market Prediction	GAN-FD [18], ST-GAN [27], MTSGAN [28]
	Fine-Tuning of trading models	C-GAN [19], MAS-GAN [29]
Portfolio Management	Porfolio Optimization	PAGAN [20], GAN-MP [30], DAT-CGAN [31], CorrGAN [21]
Time Series Generation	Synthetic time series generation and Finance Data Augmentation	TimeGAN [32], WGAN-GP [33], FIN-GAN [15], Quant GAN [34], RA-GAN [35], CDRAGAN [36], SigCWGAN [37], ST-GAN [27]
Fraud Detection	Detection of market manipulation	LSTM-GAN [22]
	Detection of Credit Card Fraud	RWGAN [38], LSTM-GAN-2 [39]

models, it is clear that GANs are helping to expand the field of finance research. There have been some milestone papers which will be discussed more thoroughly in the next sub-sections.

3.2.1 FIN-GAN - 2019

FIN-GAN [15] is a proposed application of the original GAN with the goal of generating synthetic time series that replicate the main stylized facts of financial time series. There are no relevant structural changes compared to the first proposed framework. For the architecture of the discriminator and the generator, three structures were proposed, 1- A MLP (multi layer perceptron) architecture with four layers of fully connected neural networks with the hyperbolic tangent activation function. 2- A CNN (convolutional neural network) architecture with six convolutional layers and the leaky ReLU activation function. 3- A MLP-CNN which combines CNN and MLP via an element-wise multiplication of their outputs.

The algorithm remains generally unchanged from the original proposition, using the min-max loss and the Adam optimizer for updating the parameters of the two neural networks.

The main findings were that changes of the generator's neural network architecture have a greater effect on the quality of output then changes of the discriminator. The use of batch normalization (rescaling hidden vectors to keep mean and variance consistent during training) showed larges fluctuation in the generated time series and a strong autocorrelation, an indication that this common tool for image generation in deep learning is not ideal for the modelling of financial time series. The output vector size is set to a large value because the length of the time-series should be long enough to reliably calculate the statistical properties such as the probability distribution and the correlation functions.

FIN-GAN applied to finance
FIN-GAN can generate financial time series that are deemed realistic, due to the presence of the major stylized facts, showing the characteristics that are intrinsic to time series found in the real world. The approach marks an important point for GANs in finance, since it is a model that learns the properties of data without requiring explicit assumptions or mathematical formulations, something that stochastic process and agent-based modelling cannot do without non-trivial assumptions.

3.2.2 Conditional GAN (cGAN) - 2019

The conditional GAN (cGAN) was first introduced in 2015 [43], and later adapted to finance by Koshiyama [19] 2019. This GAN's architecture has been structurally extended to a conditional model (Fig.4) where the Generator and Discriminator are conditioned on some extra information y. This auxiliary information y could take the form of class labels or other data. Which is then fed into both D and G as additional input layers, in a process called conditioning.

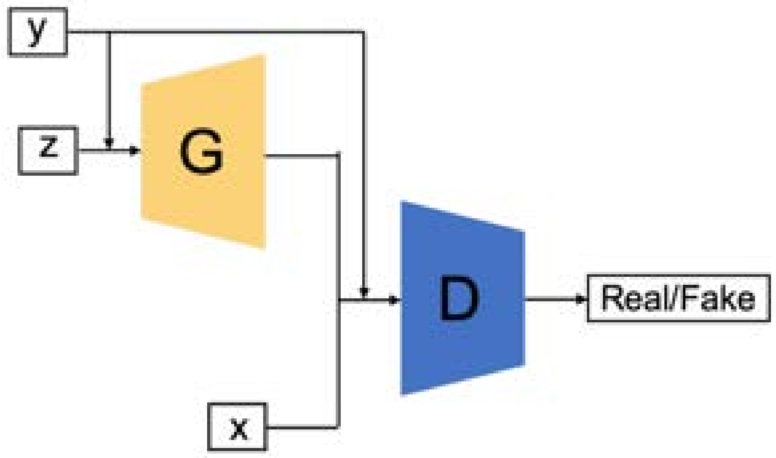


Figure 4: Conditional-GAN architecture [42]

The added input layer of one-hot-encoded labels guides the Generator to produce specific outputs. This level of control does not exist in the originally proposed GAN architecture. With the benefit from the additional encoded information, cGAN can also be trained on multimodal datasets that contain labelled data.

Like the original GAN, the Discriminator and Generator networks are MLP's where the Generator output layer activation is a linear function and the Discriminator output layer activation is a sigmoid function. The loss function has an added conditional element for both D and G, and they both are optimized via stochastic gradient descent.

Conditional GAN applied to finance
Conditioning information in finance: added vector y can represent a current/expected market condition, appropriate for modelling sequential data such as time series. It also enables the construction of "what-if" scenarios, commonly used for stress tests and other scenario-based models. The implemented cGAN has shown some new tools to improve trading strategies, specifically for finding optimal hyperparameters (fine-tuning) of trading algorithms, and combination of trading strategies, where trading models are trained on samples generated by the cGAN, and their outputs combined to get a final result.

Compared to the classical method of bootstrapping (resampling), the cGAN can generate more diverse training and testing sets. There is also the capability of drawing samples targeting stress events and an added layer of anonymization to the dataset, not achievable on other shuffling and resampling techniques.

3.2.3 WGAN-GP - 2019

The first Wasserstein GAN (WGAN [44]) proposed a new cost function using the Wasserstein distance as an alternative to the original GAN loss function, which is based on the Kullback-LeiblerL and Jensen-Shannon divergences. The Wasserstein distance quantifies the distance between two probability distributions, it is also called Earth Mover's distance because it can be interpreted as the minimum energy cost of reshaping one probability distribution into another via stepwise (in a discrete setting) increments, analogous to transporting blocks of cargo.

This modification attempts to solve the problem of vanishing gradient and mode collapse which happened with binary-cross entropy loss GANs: Often when the Discriminator started to improve faster than the Generator, it would start outputting more extreme values, giving less meaningful feedback to the Generator, stopping the learning process. With W-Loss the Discriminator is replaced by a Critic. The Critic replaces the sigmoid with a linear activation function, this way the output is not limited between 0 and 1 and the cost function continues to grow, regardless of how far apart the distributions are. This way, the gradient will not approach zero, making the GAN less prone to the vanishing gradient problem, and consequently, mode collapse.

There is a condition however, when using W-Loss, the Critic neural network needs to be 1-Lipschitz continuous. This condition implies that the Critic is differentiable almost everywhere and essentially bounded. To enforce this condition

WGAN uses weight clipping, where the weights of the Critic are forced to take values between a fixed interval. After the weights are updated during gradient descent, any weights outside of the desired interval are clipped, so values that are too high or too low are set to some fixed maximum value. While ensuring a more stable training process, forcing the weights to a limited range of values could limit the critics ability to learn and ultimately for the GAN to perform. This causes the need for additional hyperparameter tuning to adjust the model for training.

In WGAN-GP [45], another way trying to enforce the 1-Lipschitz continuity was introduced, Gradient penalty (GP), which is less restrictive way to enforce this continuity, albeit does not always guarantee Lipschitz-continuity. With GP a regularization term is added to the loss function, it penalizes the critic when its gradient norm is higher than 1. The regularization term is achieved by by an interpolation between samples from the real and generated distributions.

FIN-GAN applied to finance
The finance application of WGAN-GP [33] for the one-dimensional case proposes a Wasserstein's GAN with Gradient Penalty with 1-dimensional convolutional networks to work on time series data. It compared the relevant statistics of generated versus original time series and proposed taking a rolling window before feeding the original series to the model, aiming to induce more variability scenarios. In the results, the generated data was largely able to replicate the stylized facts of the S&P 500 while also retaining some visual similarity with visible volatility clusters.

3.2.4 Corr-GAN - 2019

CorrGAN [21] is a novel approach to generate empirical correlation matrices estimated on asset returns. This GAN is based on the DCGAN [46] architecture.

DCGAN is a milestone GAN, being the first to apply deconvolutional neural networks in the Generator (Fig.5). Some critical changes to the architecture of DCGAN compared to the original GAN were made, which enabled higher resolution modelling and stabilized training. It also introduced the learning of hierarchies of representations in natural images. Deconvolutions are also known as transposed convolutions, they work by swapping the forward and the backward passes of a convolution. There are several key aspects of DCGAN:

• The DCGAN structure replaces pooling layers with strided convolutions for Discriminator and fractional-strided convolutions for Generators.

• Batch normalization in both the Discriminator and the Generator, improving location of generated real samples that center at zero.

• A ReLU activation is used in the Generator for all layers except the output, which uses tanh, while LeakyReLU activation is used in the Discriminator for all layers. The LeakyReLU activation prevents the network from being stuck in a "dying state" situation as the Generator receives gradients from the Discriminator.

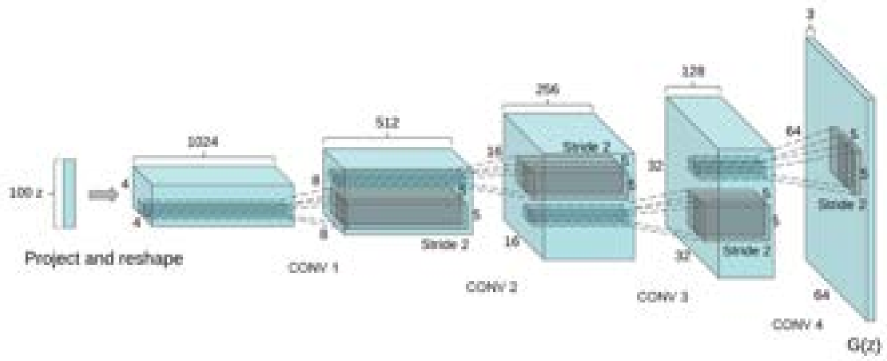


Figure 5: Generator layout of DCGAN[46]

CorrGAN applied to finance
The DCGAN architecture was originally developed to improve the quality and resolution of synthetic images. The CorrGAN paper showed an implementation of DCGAN on empirical correlation matrices estimated on S&P 500 returns. The process showed convincing results, although the generated matrices are not exactly correlation matrices (their diagonal is very close but not equal to 1), the major stylized facts were reproduced in the synthetic data. In short, the main stylized facts of financial correlation matrices are the distribution of pairwise correlations being positively shifted, the first Eigenvector having positive entries and the corresponding minimum spanning tree being scale free.

Practical financial applications of the CorrGAN framework could range from improving trading strategies to risk and portfolio stress testing. Some suggested applications include improving Monte Carlo backtesting and stress testing portfolios by conditioning on market regime variables of different macroeconomic scenarios. The researchers developed a website[47] to test if people can visually distinguish real from fake correlation matrices, currently they are indistinguishable to the naked eye.

3.2.5 QuantGAN - 2020

Introduced in 2019, applying GANs to financial time series generation, QuantGAN [34] consists of a GAN variation which utilizes temporal convolutional networks (TCNs) aiming at capturing long-range dependencies like volatility clusters. The objective was to approximate a realistic asset price simulator by using a neural network, data-driven concept.

The TCNs are convolutional frameworks that provide a unified approach to capture low and high level information in a single model. TCNs consist of a causal 1D fully convolutional network architecture, where an output depends only on past sequence elements. Another key characteristic are more dilated convolutions. Dilation refers to the distance between elements of the input sequence that are used to compute one element of the output sequence, this means that instead of increasing the size of the kernel/filter, it introduces empty spaces for more coverage, increasing the receptive field and giving a broader view, with more context information about the input.

Empirical results suggest that TCNs are better at capturing long-range dependencies in time series than other convolutional methods. TCNs have the advantage of having more controllable gradients compared to Recurrent Neural Networks, simplifying the optimization process.

QuantGAN applied to finance
QuantGAN uses TCNs as Discriminator and Generator networks. It was trained on nine years of S&P 500 data and used the Wasserstein distance as a distributional evaluation metric. As dependence scores, it used the ACF and leverage effect scores to assess accuracy based on the stylized facts. The result was that QuantGAN generated returns closely matched the real returns, this was corroborated by the sharp drops in the ACF and the negative correlation between squared and non-squared log returns at short time lags.

Finally, the authors evaluate that the proposed architecture generates competitive results, which can be used to approximate financial time series. They point to the lack of a unified metric to quantify the goodness of fit of the generated distributions, but overall the findings are a solid step in developing data-driven models in finance.

3.2.6 MAS-GAN - 2021

This new framework, developed by JP Morgan's AI Research team in a yet to be published paper [29], is a two-step method for multi-agent market-simulator calibration. It uses a GAN Discriminator to calibrate a market simulator constructed using an agent-based approach, first a calibration objective is learned with a GAN Discriminator, then the learnt calibration objective is used to simulate parameter optimization. It is the first method to use a GAN trained Discriminator as an objective function for multi-agent system optimization.

This work builds on SAGAN [48], an architectural variant developed in 2018 that introduces a self-attention layer to a Convolutional GAN. Using Convolutional Neural Networks CNNs, the Self-Attention method was designed to create a balance between efficiency and long-range dependencies (large receptive fields) by using the attention mechanism, well known from the Natural Language Processing (NLP) field of research. In a self-attention layer visualized in 6, the feature map is passed through three 1x1 convolutions, called Query, Key and Value, where the vectors generated by Query and Key multiply to create a vector of probabilities that decides how much to expose to the next layer. In other words, the Q x V multiplications' output is passed through a softmax activation function that creates a so-called attention map, which is multiplied by the value vector to create the self-attention feature map. This layer is complementary to regular convolutions and aides the network in capturing fine features from an input source.

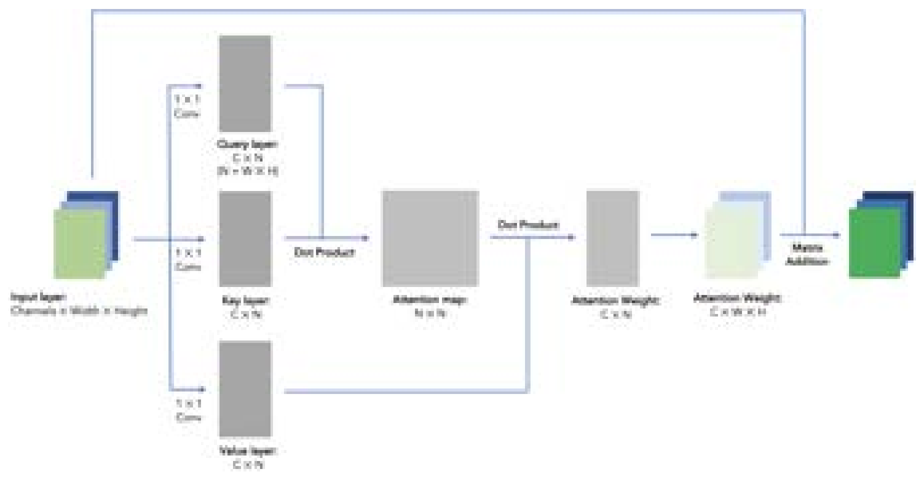


Figure 6: Self-Attention Module[49]

MAS-GAN applied to finance
The MAS-GAN model consists of a SAGAN trained on one-minute mid-price returns and cumulative one-minute traded volumes of thirty stocks traded on NASDAQ in June 2019. After the training process is done, the Discriminator is used to optimize a discrete event model that simulates a limit order book via three distinct agent behaviours. The objective is to find the most realistic agent configuration through calibration, namely, to find which configurations on a rectangular grid that can produce the simulation closest to the historical dataset. The method has shown good calibration performance for multi-market simulations, based on stylized facts and the Kolmogorov-Smirnov test. Further research into understanding behavioural explanations of the market is suggested, but not presented in the paper.

3.3 Current GAN applications in the financial industry

As shown in the previous section, the topic of Generative Adversarial Nets is progressing in different finance applications, but the extent of its use in practice by companies is presently limited or not fully disclosed. This section provides a short overview of the publicly known implementations of GANs.

3.3.1 Generation of Synthetic data by JP Morgan AI Research

With the goal of advancing AI research and development in financial services, JP Morgan's AI Research department has a branch dedicated to generating synthetic datasets. These datasets can be requested by other research groups and comprise of 1- customer related datasets for Anti Money Laundering models, 2- customer journey events, lower level client-bank interaction dataset, 3- market execution data: limit order book data describing matches of buy and sell orders of financial instruments at a public stock exchange. 4- payment data for fraud detection: several transaction types with legitimate and abnormal activities to improve detection.

The research department proposes a framework [26] for ideal representation and transfer of synthetic data. The framework suggests: 1- Privacy preserving, the specific data and context where privacy needs to be enforced. 2- Human readability, the data and its associated generative models must be readily interpretable by regulators and other agents for the sake of transparency. 3- Compactness, the representation of synthetic data should be compact and reconstructible, it should require little technical know how as to improve synthesizing it in different environments.

All the specified data is produced in-house, there is no specific information on the generating process, but there is a paper under review by JP Morgan researchers where the MAS-GAN [29] is proposed for multi-agent simulation, where the Generator is used to calibrate an agent based model as described earlier in this paper.

3.3.2 The Digital Sandbox Pilot by the FCA

The Financial Conduct Authority is a financial regulatory body from the United Kingdom. To boost innovation, they created an environment for testing of financial models, products, and services. The Digital Sandbox [50] provides an integrated, collaborative development environment for testing and scaling projects, aiming to reproduce real scenarios and perform stress tests. The initiative is in pilot stage and has already had a test run with 28 groups presenting solutions on the topics of: access to finance for SMEs, improving the financial resilience of vulnerable consumers and fraud/scam detection.

The whole Digital Sandbox heavily relies on synthetic data, as real is under strict obligations in the UK. As described in the first report [24] synthetic financial data was commissioned to leading data scientists from industry and academia. There was a two-group effort, where one group would create the synthetic data and another which would define typologies and behaviours this data was expected to have. The main approaches to this task used GANs and ABMs and the whole process was benchmarked by the Alan Turing Institute. Access to synthetic data was ranked as the most important feature of the sandbox, however, not all data was considered useful, showing that there is demand for high quality synthetic data.

The main takeaways from the first pilot were: A digital testing environment is in high demand, particularly by startups. It accelerated product development for the participating firms. The access to good synthetic data was considered extremely valuable by all participants. A second digital sandbox is planned with an expanded testing system. This promising new platform has shown potential and could be implemented in other countries if it further succeeds in the United Kingdom.

3.3.3 Fraud detection by American Express AI Labs

Fraud losses, majorly from wire transfers and credit/debit cards caused an estimated loss of 16.9 billion USD to banks, merchants, and customers [51]. Companies use their customer data to train models to predict and prevent fraud. A known issue of fraud detection on real datasets is the class imbalance: Often a dataset can represent a biased sample from reality and will inevitably lead to faulty models. Financial services multinational American Express Co. has its AI lab looking for solutions to this issue by generating synthetic data to improve their fraud detection models. Amex researchers published a paper [36] where a hybrid of Conditional and Deep Regret Analytic GANs was proposed to generate synthetic datasets.

Three tabular datasets with internal company data were used to recreate statistically similar samples. The generated data was evaluated by comparing characteristics of the real and generated data distributions, and also by the internally developed tool DataQC [52], which uses well known methods to look for anomalies in datasets and outputs a unified score of attribute anomaly levels. The generated data showed satisfactory results, but it was found that models trained on synthetic data still performed worse than those trained on real data. The research team states that further research into generating synthetic data is being done, the capacity on which the generated data is being used internally was not disclosed.

4 Neural networks

Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. The human brain inspires their name and structure. Deep neural networks (Fig.1) compromises of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node (also called perceptron) is connected to another and has an associated weight and threshold. If the output of any individual perceptron is above the specified threshold value, that perceptron is activated, sending data to the next layer of the network. Otherwise, no data passes along the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms achieve high accuracy, they are potent tools in computer science and artificial intelligence, allowing us to classify and cluster data.[6]

4.1 Recurrent Neural Network

A recurrent neural network is a type of artificial neural network that uses sequential data or time-series data. It stores the output activations from the layers of the network. Then, the next time we feed an input example to the network, we include the previously-stored outputs as additional inputs. Like feed-forward, recurrent neural networks use training data to learn. Recurrent Neural Networks, however, can memorize previous inputs when a huge set of sequential data is fed. A deep neural network assumes that inputs and outputs are independent of each other. The output of a recurrent neural network depends on the primary elements within its sequences.[5][53]

Even though RNNs are pretty powerful, they suffer from vanishing gradient problem, which hinders them from using long term information. They are useful for storing memories of 3 or 4 instances of past iterations. More enormous numbers of instances will not provide good results.[54] That is why we use a better variation of RNNs: Long Short Term Networks (LSTM). A recurrent neural network (RNN) (Fig.7 left) is a type of artificial neural network that uses sequential or time series data. These deep learning algorithms are often used for ordinal or temporal problems. Like feedforward and convolutional neural networks (CNNs) (Fig.7 right), recurrent neural networks use training data for learning. They are characterised by their "memory", as they use information from previous inputs to influence the current input and output.While conventional deep neural networks assume that inputs and outputs are independent, the output of recurrent neural networks depends on the previous elements within the sequence. While future events would also be helpful in determining the output of a particular sequence, unidirectional recurrent neural networks cannot take these events into account in their predictions.[5]

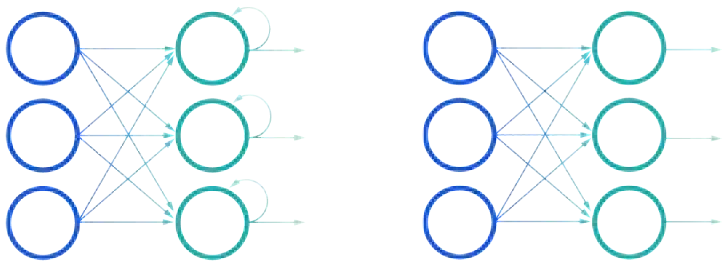
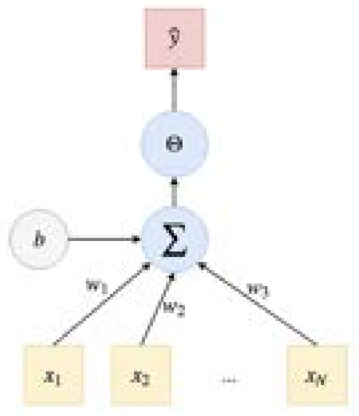


Figure 7: Comparison of Recurrent Neural Networks (on the left) and Feedforward Neural Networks (on the right) [5]

4.2 Single Layer Perceptron (SLP)

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers, deciding whether an input vector is some specific class. Frank Rosenblatt firstly introduced the simplest form of a neural network in 1958. It consists of a single artifical neuron with adjustable weights and a treshold that will convert an input vector into an output vector and represent a simple associative memory.[5][55] The neuron consists of two mathematical functions: A calculation of the network input and an activation function that decides whether the calculated net input now "light up" or not. Therefore, it is binary in its output: Think of a small light that depends on the input values and weights, an input (sum) is formed, and then a function decides whether the light is lit. This concept of output generation is called feed-forward propagation.[56] Let's consider the structure of the perceptron. It contains four key components:



1. input x_i
2. weights w_i
3. weighted sum $\sum$
4. thresholding value by using an activation function

Figure 8: Computational graph [57]

In (Fig.8) we see a simplified representation of how a perceptron (node) works. By using the weighted summing function, the perceptron becomes a learnable parameter. By adjusting the weights, the perceptron can differentiate between two classes and therefore model the classes. The weighted sum is transmitted through an activation function θ. Activation functions decide to activate or deactivate the neurons to get a desired output. The $\sum$ represents the linear combination of the inputs $X(x_1, x_2)$ of length n that is weighted by a weight vector w.

$$x_1 = x_1 * w_1 \tag{1}$$

$$x_2 = x_2 * w_2 \tag{2}$$

The output of the activation functions is the output of the perceptron. The larger the numerical value of the output will be, the higher the confidence of the prediction.[58][59][60][61] Let us simplify. The inputs are defined as x_1 and x_2. Each input x_1, x_2 gets multiplied by its assigned weight w_1 and w_2. The input vector represents the training data on which the neural network is getting trained. Once this vector is defined, each element x_{i} of the vector is assigned to a weight w_{i}. The input elements are multiplied by their associated weight, and the resulting *n* results are summed to obtain a weighted average. The significance of the impact on individual input vector elements on the weighted average depends on the individual weights. After the weighted average is calculated, a bias is added. The bias is an element that adjusts the boundary away from origin without any dependence on the input value:

$$v = (x_1 * w_1) + (x_2 * w_2) + b \tag{3}$$

Then, perceptron's output will be presented by transforming it to a (non-)linear activation function. We explain later what the activation functions are crucial.[57]

$$y = f(x_1 * w_1 + x_2 * w_2 + b) \tag{4}$$

The SLP was, in a way, the beginning of artificial intelligence and mostly inspired the development of more complex neural networks. The main limitation of the SLP models is that those perceptron models are only accurate when working with data that is linearly separable and is therefore limited.[60]

4.3 Multi layer perceptron

MLPs are distinguished from SLPs because they are hidden layers that affect the output of the model. In the MLP, the neurons are structured into layers. They belong to the class of feed-forward neural networks (FNNs), which means that nothing ever flows back during output generation, i.e. everything flows from input to output. In (Fig.9), we can see an artificial neural network with multi-layer perceptrons. The multi-layer perceptrons contain inputs x_n, hidden layers $h_n{}^{(\mathrm{n})}$, and output layers y_n.

Each neuron in the hidden layer $h_n{}^{(\mathrm{n})}$ is connected to each other neuron in $u_{n+1}{}^{(\mathrm{n+1})}$. The hidden layers consist of many neurons with connections between the layers. Each input value is associated with a weight connected to each in the

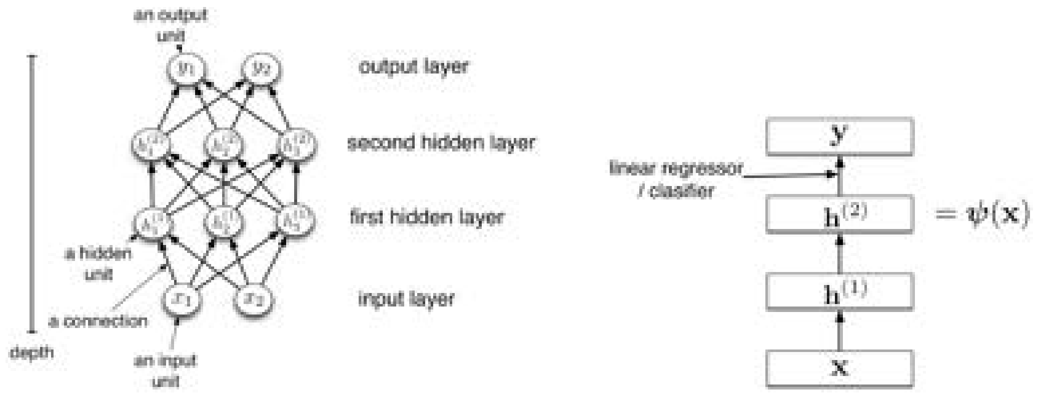


Figure 9: A multilayer perceptron with two hidden layers

hidden layers. The number of layers is known as the depth, and the number of units in a layer is known as the width. The last layer of neurons holds the individual classification or regression output.[62]
The general neuron processing unit looks like this:

$$a = \phi(\sum{}_{\mathrm{j}} w_{\mathrm{j}} * x_{\mathrm{j}} + b) \tag{5}$$

Where x_j are the inputs to the unit, the w_j are the weights, b is the bias, a is the unit's activation, and ϕ the non-linear activation function. The possibilities of machine learning are immense. Here are just some examples:

- Regression uses a linear model, so ϕ(z) = z.
- In binary linear classifiers, ϕ is a hard threshold at zero.
- In logistic regression, ϕ is the logistic function ϕ (z) = 1 = (1 + e + $^{-z}$)

A neural network is a combination of several of these presented units. Each one takes on an effortless action. In aggregation, it is capable of powerful and useful computations. As mentioned before, every unit in one layer is connected to every unit in the next layer. Each unit has its own bias and weight for every unit's connections in the two consecutive layers. We can define the first hidden layers $h_{\mathrm{i}}^{(\mathrm{n})}$ as follows:

$$h_{\mathrm{i}}^{(1)} = \phi^{(1)}\Big(\sum_{j} w_{\mathrm{ij}}^{(1)} x_{\mathrm{j}} + b_{\mathrm{i}}^{(1)}\Big) \tag{6}$$

$$h_{i}^{(2)} = \phi^{(2)}\Big(\sum_{j} w_{\mathrm{ij}}^{(2)} h_{\mathrm{j}}^{2} + b_{\mathrm{i}}^{(1)}\Big) \tag{7}$$

$$h_{\mathrm{i}}^{(3)} = \phi^{(3)}\Big(\sum_{j} w_{\mathrm{ij}}^{(3)} h_{\mathrm{j}}^{2} + b_{\mathrm{i}}^{(3)}\Big) \tag{8}$$

Consider distinguishing $\phi(1)$ and $\phi(2)$ because the different layers may have other activation functions.
Each layer contains multiple units, which represent the activations of all its units with an activation vector. There is a weight for every pair of units in two consecutive layers. We represent each layer's weight for every pair of units in two consecutive layers with a weight matrix. So the above computations could now be written in the desired vectorized form:

$$h^{(1)} = \phi^{(1)}\Big(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\Big) \tag{9}$$

$$h^{(2)} = \phi^{(2)}\Big(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\Big) \tag{10}$$

$$y_i = \phi^{(3)}\Big(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}\Big) \tag{11}$$

By applying the activation function to a vector, it independently happens to all the entries. In the feed-forward neural networks, the units are arranged into a graph without cycles, so all the computation can be done sequentially. However, in a recurrent neural network, the graph contains cycles, so the processing can feed into itself and is therefore suitable for time series.[63]

4.4 Backpropagation

Backpropagation is an algorithm of supervised learning of artificial neural networks using gradient descent. With an error function, the method calculates the gradient of the error function concerning the neural network's weights. The calculation runs backward through the whole network, with the gradient of the final layer of weights being calculated first and then the gradient of the first layer of weights. Partial computations of the gradient from one layer are reused to compute the gradient for the previous layer.[64]

4.5 Activation functions

An activation function is a unit that determines which information should be transmitted to the next neuron. Each neuron in the neural network accepts the output value of the neurons from previous layers as input and passes the processed value to the next layer. In a multi-layer neural network, there is a function between these two layers. This function is called the activation function. (Fig.10) There are several different activation functions, which can output different outputs within a range.

Using a linear function, the input of each layer will be a linear function of the output to the previous layer. The network would be limited in learning linear functions, thus modeling them. No matter how many the neural network has, the output would always be a linear combination. They determine the accuracy of a deep learning model and also the

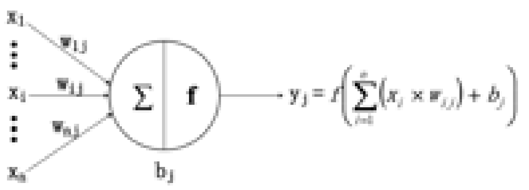

x_1
w_{1j}
x_i
w_{ij}
w_{nj}
x_n
Σ
f
b_j
$y_j = f\left(\sum_{i=1}^{n}(x_i \times w_{i,j}) + b_j\right)$

Figure 10: Structure of an activation function

computational efficiency of training a model. Many activation functions are employed, of which ReLu, sigmoid and tanh are extensively used. The $\tanh$ function outputs a natural number between [-1,1]. Because the mean value of the output is 0, it achieves a normalization to make the following layers easier to learn. The sigmoid function is a non-linear activation function used for binary classification problems by transforming the output between [0, 1]. By x approaching θ, the gradient becomes steeper. In ReLu, when x is less than 0, its function value is 0. When x is greater than or equal to 0, its function value is x itself. When x is less than 0, the gradient of ReLU is 0, which means the back-propagated error will be multiplied by 0.[65]

4.6 Optimizers - RMSprop

Optimizers are algorithms that change the attributes of weights and learning rates to minimize their losses.[66] The way of changing the weights or the learning rate is defined by the optimizer you are using. A good example helps to imagine how an optimizer is working. A mountain climber wants to climb down from a mountain but cannot see anything. She does not know where to walk, but she begins to feel her way. By taking a step, she thinks whether it is going down or not. With every step upwards, she makes a loss of progress. With every step downwards, she makes progress, because she wants to reach the bottom. If she does this over some time and only carries out the downhill steps, she will eventually get her destination at the bottom. This process is also done by optimizing a loss function. Optimization algorithms are reducing the losses and provide the most accurate results possible.

RMSprop

RMSprop divides the learning rate by an exponentially decaying average of squared gradients. It decides how much of the gradient you update. More immense steps mean that the weights are changed more every iteration to reach their optimal value faster but possibly miss the exact optimum. So with smaller steps, the weights are changed less every iteration, so it takes more epochs to reach their optimal value, but they are less likely to miss the optimum loss function's optimum. In (Fig.11), we can see that if the red dot rolls carefully with a small learning rate, we can expect consistent progress.[67]

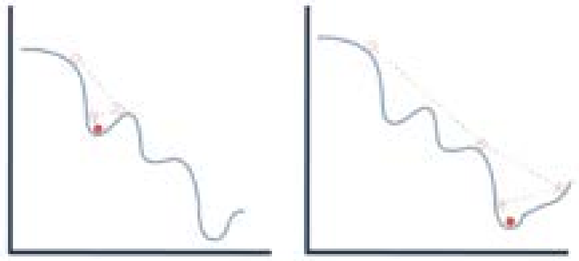


Figure 11: Learning rate

There is an option of learning rates which allows, to use of bigger steps in the first epochs and then reduce the step size when the weights have come in proximity of their optimal value. Several optimizers were introduced in the last few years, where every optimizer has its advantages and disadvantages. The players' weights are updated according to a stochastic gradient descent step with either ADAM or RMSprop optimizer to increase training convergence. It is usually used as a learning criterion of the optimization problem.[67] Remember the RMSprop optimizer, which we will later use in the Implementation. ADAM is one of the most used optimizers, but it did not give better results, which is why we do not go further into the ADAM optimizer.

4.7 Loss functions

Deep learning neural networks use a stochastic gradient descent optimization algorithm. The error of the actual state must be estimated repeatedly. That case requires a loss function to estimate the loss to update the weights, and reduce the loss on the following evaluation. The loss function takes a batch of actual samples and generated samples and then calculates the difference. The lower our loss is, the better the performance of our model. They are explained later in more detail.[68]

4.8 Learning rate

To produce stabilized GAN models, the learning rates should be low. The movement is across the gradient slope by taking smaller values, so the local minimum does not get missed. But higher learning rates might cause the gradient descent to overshoot the minimum, and as a result, the resulting model fails to converge to a minimum and leads to training failure of GANs.[69]

4.9 Batch size

The batch size determines the number of instances passed through the model, before the backpropagation for each epoch happens. A Smaller batch size results in updating the error gradients based on a smaller batch of samples and offers a regularization effect to reduce the error. Sometimes, increasing the minibatch size can improve the performance of the model. However, bigger batch size is expected to impact the performance because of the training of the Discriminator. By using a lot of samples, will end in overpowering the Generator. [69]

4.10 Long Short-Term Memory

The recurrent neural network suffers from gradient vanishing and exploding problems. But this is where the Long Short-Term Memory (LSTM) comes in. The LSTM networks are a modified version of recurrent neural networks (RNN making it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process, and predict time series. It trains the model by using backpropagation. In a LSTM network, three gates are present, as in (Fig.12):

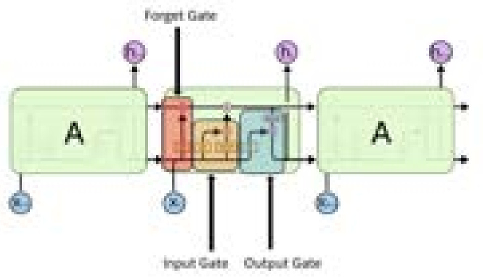


Figure 12: Structure of a LSTM

Input
The Input gate discovers which value from input should be used to modify the memory. The sigmoid function then decides which values get through [0,1]. The tanh function gives weightage to the values which are passed, deciding their level of importance ranging from [-1, 1].

$$i_{\text{t}} = \sigma(W_{\text{i}} * [h_{\text{t-1}}, x_{\text{t}}] + b_{\text{i}} \tag{12}$$

$$\tilde{C}_{\text{t}} = tanh(W_{\text{c}} * [h_{\text{t-1}}, x_{\text{t}}] + b_{\text{c}}) \tag{13}$$

Forget gate
The forget gate shows what details are discarded from the block. It is decided by the activation function. It considers

the previous state $h_{(t-1)}$ and the content input (X_{t}) and outputs a number between [0,1] for each number in the cell state $C_{(t-1)}$.

$$f_{\text{t}} = \sigma(W_{\text{f}} * [h_{\text{t-1}}, x_{\text{t}}] + b_{\text{f}}) \tag{14}$$

Output gate
The input and the memory of the block is used to decide the output. Activation functions decide which values to let through [0, 1], and then the tanh function gives weightage age to all values, which then decides its value from [-1, 1] and multiplies with the output of the activation function.[53]

$$o_t = \sigma(W_{\text{o}}[h_{\text{t-1}}, x_{\text{t}}] + B_{\text{o}}) \tag{15}$$

5 Generative Adversarial Networks (GAN)

Generative adversarial networks, GANs, are a series of generative machine learning frameworks first introduced by Ian Goodfellow and his collaborators in 2014 [1]. They gained a lot of attention due to its simplicity and effectiveness. In a short time span, considerable progress was made to the initial application of GANs - image generation, but also hundreds of different types of GANs were created to optimize for various tasks, from computer vision to fraud detection in banks, the framework presented in 2014 has come a long way. From getting good images on the MNIST dataset and recognizable human faces in the original paper, as of 2021, GANs can generate near perfect human faces with StyleGAN [70], as seen in (Fig.13), and have a constantly expanding portfolio of applications.

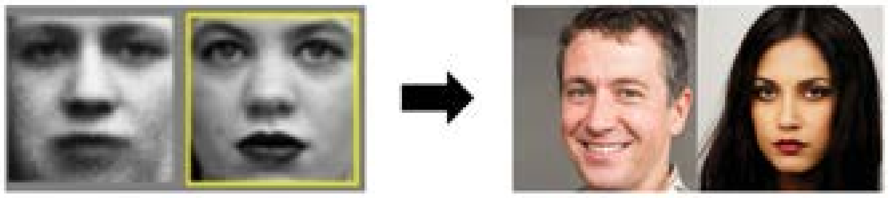


Figure 13: 2014: first generated faces [1] vs. 2020: current state of the art [70]

There are also several applications to finance and financial time series analysis, although compared to other fields, these frameworks are still considered novelties, which makes for an exciting area of research. Since the research is still in its infancy, its reasonable to assume that further applications are yet to be developed and improved upon. Presently, much of the research is still in development and concrete applications are limited, but this paper will show that the current work already shows great potential.

GANs belong to the family of generative models in machine learning ML, generative models are processes that can generate new data instances, more formally, given a observable variable X and a target variable Y, a generative model is a statistical model of the joint probability distribution on P(X|Y). Generally, the process involves discovering patterns in the input data and learning them in a way that the model can then generate new samples that retain characteristics of the original dataset.

5.1 The GAN framework

The original GAN framework estimates generative models through an adversarial process, where two models (usually neural networks) are trained in parallel as represented in (Fig.14) and described below:

There is a generative model, the Generator (G) that captures the data distribution and generates new data. The second model is a classifier, the Discriminator (D) which estimates the probability that a sample came from the training data and not from G. The training process for the Generator is to maximize the probability of its output being misclassified by the Discriminator.

The Generator is responsible for the generation of data, and the Discriminator has the task of assessing the quality of the generated data and providing feedback to the Generator. These neural networks are optimised under game-theoretic conditions: The Generator is optimised to generate data to fool the Discriminator, and the Discriminator is optimised to detect the source of the input, namely the Generator or the real data set.

5.2 Neural Network architecture

The neural network architecture is made of individual units called neurons that mimic the biological behaviour of a human brain. Simplified, we can imagine the structure of an artificial neural network as follows. The neural network model consists of nodes, also called neurons, which receive information from other neurons or from outside, modify it and output it as a result. This is done through three different layers, each of which can be assigned a type of neuron. For the input (input layer), for the output (output layer) and in the middle the (hidden layers). The information is received by the input neurons and output by the output neurons. The hidden neurons lie in between and map internal information patterns. The neurons are connected to each other by means of edges. The stronger the connection, the greater the influence on the other neuron.[71]

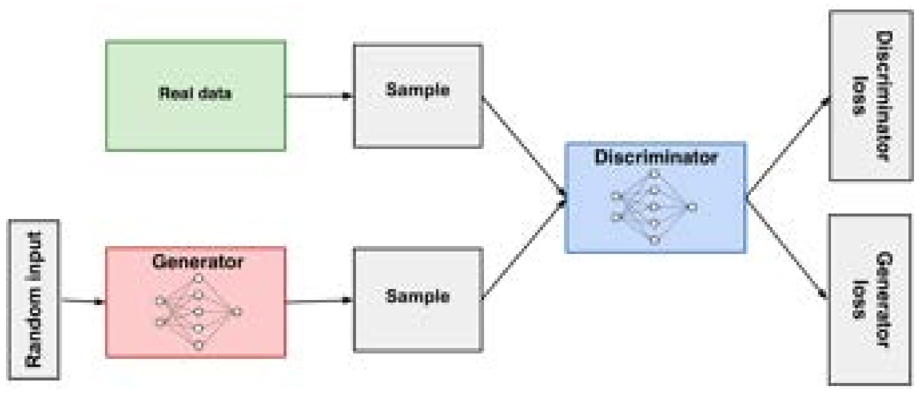


Figure 14: The originally proposed architecture for Generative Adversarial Networks

5.2.1 The Discriminator

The Discriminator is a classifier. It receives Real and Synthetic (from G) data and attempts to distinguish them. It can use several different network architectures, depending on the type of data being classified. The Discriminator network connects to two loss functions that are used in different parts of training. After classifying real/synthetic data, the Discriminator loss penalizes it for misclassifications, and its weights are updated via **backpropagation** from its loss through the network.

5.2.2 The Generator

The Generator network G uses feedback from the Discriminator D to learn to generate synthetic data that ideally resembles the original data in key aspects. Its goal is for the created data to be classified as real by the Discriminator.

The network receives a random input, some type of noise, from which it generates some output, this output is then evaluated by the Discriminator and results in a Generator loss, which then penalizes the Generator for not deceiving the Discriminator. By introducing noise, a GAN can potentially produce a wide variety of outputs by sampling from different places in the target distribution. Usually noise is introduced by sampling from the uniform distribution.

5.3 Loss functions and regularizers

The GAN training process uses loss functions that measure the distance between the distributions of the generated and real data to assess their similarity. There are many proposed methods to solve this challenge. In the original "vanilla" GAN, a so-called minimax loss was introduced.

The formulation of the minimax loss is derived from the cross entropy between the real and generated distributions by the JS divergence when the Discriminator is optimal. For the Generator, minimizing the loss is equivalent to minimizing $\log(1 - D(G(\mathbf{z})))$ since it can't directly affect the $\log D(\mathbf{x})$ term in the function. The Jensen-Shannon - JS - divergence measures the similarity between two probability distributions, it is based on the entropy of a discrete random variable being a measurement of the amount of information required on average to describe that variable.

In the original framework, the Generator and Discriminator losses come from a single measure of distance between probability distributions. The two terms are updated in an alternating fashion, depending on which network is being trained.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

• $D(\mathbf{x})$ is the Discriminator's estimate of the probability that real data instance $\mathbf{x}$ is real.
• $\mathbb{E}_{\mathbf{x}}$ is the expected value over all real data instances.
• $G(\mathbf{z})$ is the Generator's output when given noise $\mathbf{z}$.
• $D(G(\mathbf{z}))$ is the Discriminator's estimate of the probability that a fake instance is real.
• $\mathbb{E}_{\mathbf{z}}$ is the expected value over all random inputs to the Generator (in effect, the expected value over all generated fake instances $G(\mathbf{z})$).

5.4 Training and Optimizers

Optimizers update the model in response to the output of the loss function. In essence, they have control over the learning process of a neural network by finding the values of parameters such that a loss function is at its lowest. The learning rate is a key hyperparameter that scales the gradient and sets the speed at which the model is updated.

Most models use a gradient descent-based optimizer. Gradient descent is the direction of steepest descent of a function, these algorithms are used to find the local minimum of differentiable function by small iterations.

Adaptive moment estimation (Adam) is an optimization algorithm used to update network weights iteratively, it is an extension to stochastic gradient descent and has seen broad adoption in deep learning applications from computer vision to natural language processing. Adam works by storing both the exponentially decaying average of past squared gradients and exponentially decaying average of past gradients. There are a few other types of optimizers being used in GAN literature, but Adam is currently among the most popular choices.

5.5 Challenges in the training process

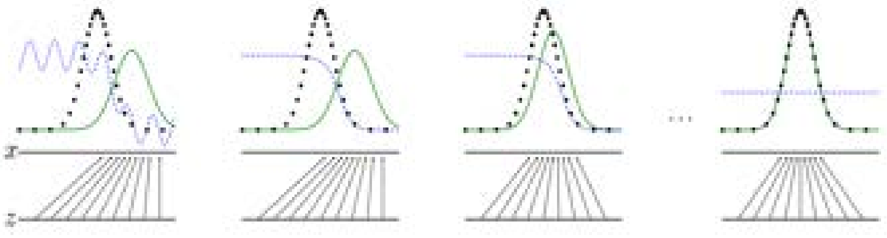


Figure 15: Theoretical evolution of GAN training [1], as the Generative distribution (green line) approaches the real data distribution, the Discriminative distribution (dotted blue line) will be unable to distinguish them and stabilizes at $D(\mathbf{x}) = ½$

The Discriminator (D) and Generator (G) networks are trained separately, by different alternating processes.

Generator - G
Sample from random noise z -> new sample is produced in G -> D classifies the sample as "Real" or "Fake" -> Loss calculated from D classification -> Backpropagate through Discriminator and Generator to obtain gradients -> Gradients used to change G weights

Discriminator - D
D classifies real data and G fake data -> Discriminator loss penalizes G for misclassifying real and fake instances -> D updates its weights through backpropagation from the Discriminator loss through the network D.

Adversarial training
Following the description above, both models are competing against each other in a way called adversarial in game theory, and they are playing a zero-sum game. This means that when the Discriminator successfully identifies a sample, it is rewarded or no update is done to the model parameters, whereas the Generator is penalized with large changes to model parameters. From the other perspective, when the Generator tricks the Discriminator, it is rewarded, or no update is done to its parameters, but the Discriminator is penalized, and its model parameters are changed.

Using the Discriminator to train the Generator
When training a neural net, weights are altered to reduce the error or loss of the output. Generative Adversarial Networks are more complex since the Generator is not directly connected to its loss function. It is the Discriminator that produces the output that will affect the Generator (Generator loss). Backpropagation then adjusts each weight by calculating the weight's impact on the output.

The impact of a Generator weight depends on the impact of the Discriminator weights it feeds into. Backpropagation starts at the output and flows back through the Discriminator into the Generator. In an optimal case, where both the Discriminator and Generator evolve at the same pace, the Generator would end up generating samples indistinguishable from those drawn from real data, in a process shown in (Fig.15).

5.6 Challenges

Although capable of generating very accurate synthetic samples [70], GANs are also known [42] to be hard to train. Training two networks simultaneously means that when the parameters of one model are updated, the optimization problem changes. This creates a dynamic system that is harder to control. Non convergence is a common issue in GAN training. Deep models are usually trained using an optimization algorithm that looks for the lowest point of a loss function, but in a two player scenario, instead of reaching an equilibrium, the gradients may conflict and never converge, thus missing the optimum minima. In other words, if the Generator gets too good too fast, it may fool the Discriminator and stop getting meaningful feedback, which in turn will make the Generator train on bad feedback, leading to a collapse in output quality.

Mode collapse is a failure mode of GANs that happens due to a deficiency in training. It can happen when the Generator maps several noise input-values to the same output region, or when the Generator ignores a region of the target data distribution. This means that if the Generator gets stuck in a local minimum generating limited samples, the Discriminator will eventually learn to differentiate the Generator's fakes, ending the learning process and leading to an undiversified output. This is a problem because in generative modelling, the goal is not only to create realistic looking samples, but also to be able to produce a wider variety of samples. Several adaptations of the original model try different adjustments to mitigate these problems.

The ongoing GAN research has shown that when the Discriminator gets too good, the training of the Generator can fail. The reason is that an optimal Discriminator does not provide sufficient feedback for the Generator to properly learn. This is called the **vanishing gradient** problem, when the gradient gets so small that in backpropagation it does not change the weight values of the Generator's initial layers, so their learning can get very slow and eventually come to a halt.

5.7 Evaluation of GANs

Another issue concerning the development of GANs, is how to evaluate their training accuracy. Since GANs have been first developed around image generation, the most used evaluation metric is currently the Fréchet Inception Distance (FID). It is based on the Fréchet Distance, a measurement that compares the statistics of two multivariate normal distributions – mean and covariance matrix - to quantify how far apart the distributions are from each other. It uses the features extracted from the imageNet dataset by the Inception v3 network. For financial time-series we are tasked with defining the proper evaluation metric. Currently, it is an ongoing research topic, how best to define one single metric to define all time-series properties and the corresponding performance of GANs.

5.8 Current research

To solve the main training issues, vanishing gradient and mode collapse, research has developed into two main approaches: The proposal of new network architectures and the use of different loss functions. Recent research [42] shows that the performance of GANs is related to the network and batch sizes, indicating that well designed architectures have a critical effect on output quality. The redesign of loss functions, including regularization and normalization has yielded improvements on training stability. It is important to note that improvements are targeted at specific applications, so presently there is not one unique fits-all solution.

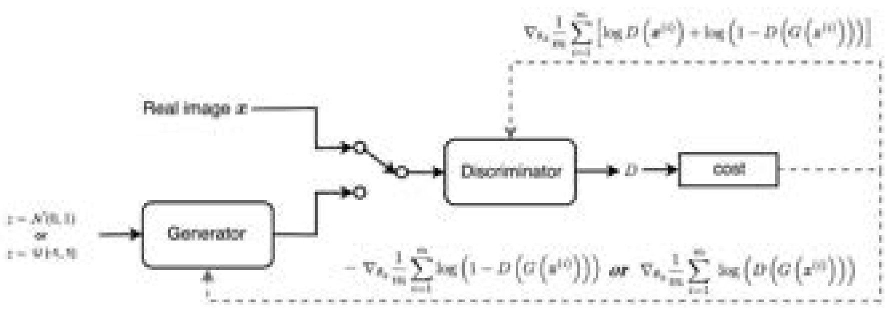

$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
Real image x
Discriminator
D
cost
$z \sim \mathcal{N}(0,1)$
or
$z \sim \mathcal{U}(-1,1)$
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$

Figure 16: Model of a WGAN

5.9 The Wasserstein GAN

The main differences between the original GAN and a WGAN:

- A WGAN uses the Wasserstein loss compared to the BCE loss.
- There is no need for a sigmoid activation in the final layer of the WGAN critic.
- The critic is trained multiple times for each update of the Generator.

The Wasserstein GAN was one of the first significant steps toward stabilizing GAN training. With some changes, the authors were able to show how to train GANs that have the following two properties, see [44]:

- A meaningful loss metric that correlates with the Generator's convergence and sample quality
- Improved stability of the optimization process

Specifically, the authors introduce a new loss function for both the Discriminator and the Generator. It is using the Wasserstein loss function instead of binary cross-entropy from the original GAN and results in a more stable convergence of the model. The Wasserstein loss is defined as the shortest average distance to move the probability mass from one distribution to another. For two distributions which are non-overlapping, the Wasserstein distance can provide a smooth representation of the space in-between, which is not always the case with the JS-Divergence.

	Discriminator/Critic	**Generator**	
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$	(16, 17)
WGAN	$\nabla_{w} \frac{1}{m} \sum_{i=1}^{m} \left[f\left(x^{(i)}\right) - f\left(G\left(z^{(i)}\right)\right)\right]$	$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} f\left(G\left(z^{(i)}\right)\right)$	(18, 19)

By removing the sigmoid activation from the final layer of the Discriminator, the predictions no longer fall in the range [0,1], but instead, they can now be any number in the range $[-\infty, \infty]$. For this reason, the Discriminator in a WGAN is usually referred to as a critic. The WGAN value function is constructed using the Kantorovich-Rubstein duality to obtain:

$$\underset{G}{min} \quad \underset{D \in D}{max} \quad \underset{x \sim \mathbb{P}_\mathrm{r}}{\mathbb{E}}[D(x)] - \underset{\tilde{x} \sim \mathbb{P}_\mathrm{g}}{\mathbb{E}}[D(\tilde{x})] \tag{20}$$

where D are 1-Lipschitz functions and $\mathbb{P}_g$ is the model distribution. Under an optimal Discriminator, minimizing the value function with respect to the Generator's parameters will minimize W($\mathbb{P}_r$, $\mathbb{P}_g$). The Wasserstein loss function trains the critic to convergence to ensure that the gradients for the Generator update are exact. Whereas in a standard GAN it is important to not let the Discriminator get too strong to avoid vanishing gradients, with WGANs, we can simply train the critic several times between Generator updates to make sure it is close to convergence. A WGAN model is shown in (Fig.16). Basically, all GANs models look the same. They differ in their functions.

5.9.1 Lipschitz Constraint

It is unusual for allowing the critic to output any number in the range $[-\infty, \infty]$, rather than applying e.g., a sigmoid function to restrict the output to the usual [0, 1] range. The authors of the WGAN paper show that for the Wasserstein loss function, it also needs to place an additional constraint on the critic. Specifically, it is required that the critic is a 1-Lipschitz continuous function. To enforce the Lipschitz constraint on the critic, it proposes to clip the critic's weights to be in a compact space. Functions that are continuously differentiable on every point are Lipschitz continuous because of its derivation.[72] The critic is a function D that converts an input into a prediction. A function is 1-Lipschitz if it satisfies the following inequality for any two inputs, x_1 and x_2:

$$\frac{D(x_1) - (x_2)}{|x_1 - x_2|} \leq 1 \tag{21}$$

x_1 and x_2 are the average absolute difference between the two inputs and the critic predictions. It requires a limit on the rate at which the predictions of the critic can change. The absolute value of the gradient must be at most one everywhere.[7]

5.9.2 Wasserstein GAN with gradient penalty term

A main criticism of the WGAN is that the capacity in weight clipping of the critic is strongly diminished to learn. Even in the original WGAN paper the authors write: “Weight clipping is a bad way to enforce a Lipschitz constraint.”
A good performance of a critic is crucial of a WGAN, because , without accurate gradients, the Generator cannot learn how to update its weights to create better samples over time. Therefore, one of the most recent extensions to the WGAN is the Wasserstein GAN Gradient Penalty (WGAN-GP) framework, see [73]. The WGAN-GP Generator is defined and compiled in the same way as the Wasserstein GAN Generator. It is only the critic that needs to be changed by including the penalty gradient term.

Gradient penalty

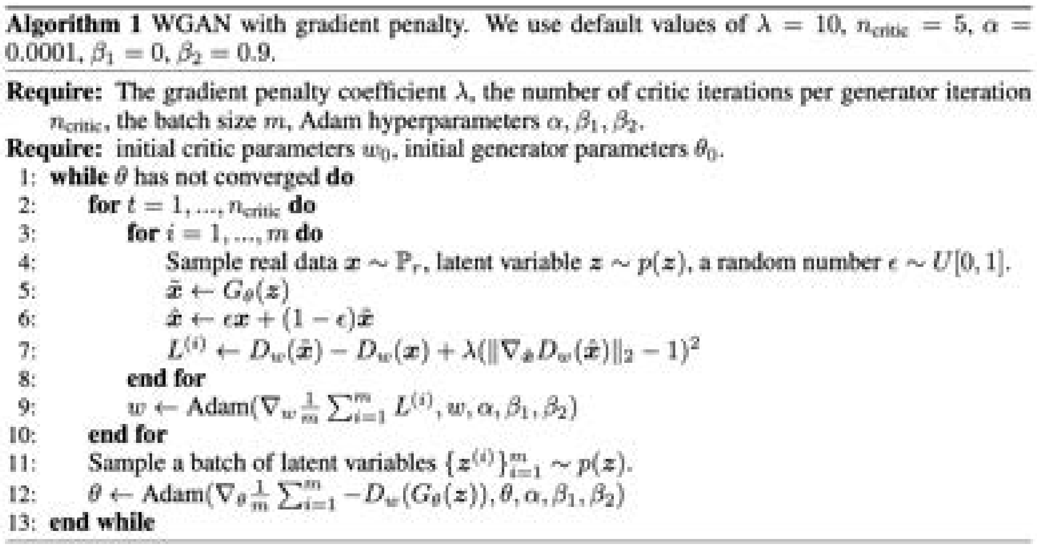
A differential function f is 1-Lipschitz if it has gradients with the norm at most 1 everywhere. So, points interpolated between the real and generated data should have a gradient norm 1 for f. Instead of applying weight clipping, the gradient penalty penalizes the model if the gradient norm moves away from its target norm value 1. In (Fig.17) we see the following formula:

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}}$$

Figure 17: WGAN with Gradient Penalty [72]

$\tilde{x}$ is sampled from $\tilde{x}$ and x is sampled t-uniformly [0,1]. Lambda (λ) is usually set to 10. Batch normalization is avoided for the critic. Batch normalization creates correlations between samples in the same batch. By penalizing the norm of the gradient of the Discriminator with respect to its inputs instead of the existing weight clipping, it can improve classification accuracy.[74][75][76]

Implementation of Wasserstein with gradient penalty



Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

```
Require: The gradient penalty coefficient $\lambda$, the number of critic iterations per generator iteration $n_{\text{critic}}$, the batch size $m$, Adam hyperparameters $\alpha, \beta_1, \beta_2$.
Require: initial critic parameters $w_0$, initial generator parameters $\theta_0$.
 1: while $\theta$ has not converged do
 2:     for $t = 1, ..., n_{\text{critic}}$ do
 3:         for $i = 1, ..., m$ do
 4:             Sample real data $x \sim \mathbb{P}_r$, latent variable $z \sim p(z)$, a random number $\epsilon \sim U[0, 1]$.
 5:             $\tilde{x} \leftarrow G_\theta(z)$
 6:             $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
 7:             $L^{(i)} \leftarrow D_w(\tilde{x}) - D_w(x) + \lambda(\|\nabla_{\hat{x}} D_w(\hat{x})\|_2 - 1)^2$
 8:         end for
 9:         $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m}\sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
10:     end for
11:     Sample a batch of latent variables $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$.
12:     $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} -D_w(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2)$
13: end while
```

Figure 18: Implementation of WGAN-GP[74]

6 Statistical Methods for time-series analysis

To better understand the statistical methods for analysing time series, the data of the E-mini S&P500 was used. With these data, the methods are explained in detail.

6.1 Q-Q Plot

The quantile-quantile (qq) plot is a graphical technique for determining if two data sets come from a common distribution. A qq-plot is a plot that sets the quantiles of the first data set against the quantiles of the second data set. By a quantile, it means the percent of points below a given value. If a reference line is plotted and the two groups come from a population with the same distribution, the points should fall close along this reference line. The greater the distance to this reference line, the higher the probability that the two given data sets come from populations with different distributions. There are several advantages of qq-Plot, such as:

- The sample sizes do not have to be the same.
- Many distributional aspects can be simultaneously tested. As example the shifts in location and scale, changes in symmetry, and the presence of outlier can be detected in a qq-Plot.

whereas questions can be answered as follows:

- Do the two distributions from populations with a common distribution?
- Are location and scale of the two datasets familiar?
- Are the shapes of the two datasets similar?
- Are there similarities in the tails?

When there are two data samples, the assumption of a similar distribution is justified. If so, the location and scale estimators can pool both data sets to obtain estimates of the common location and scale. When two samples are different, it can also be helpful to understand the differences.[77]

6.2 ACF

The random errors in the model are often positively correlated over time. Each random error is more likely to be similar to the previous random error that it would be, if random errors were independent of each other. The coefficient of correlation between two values in a time series is called the autocorrelation function (ACF). An autocorrelation of +1 represents a perfect positive correlation, which means that an increase in one time series leads to a proportionate increase in the other time series. On the other hand, an autocorrelation of -1 represents a perfect negative correlation, so an increase in one time series results in a proportionate decrease in the different time series. Even if the autocorrelation is in minus, there can still be a non-linear relationship between a time series and a lagged version of itself. In (Fig.19) we see an example of an ACF plot.

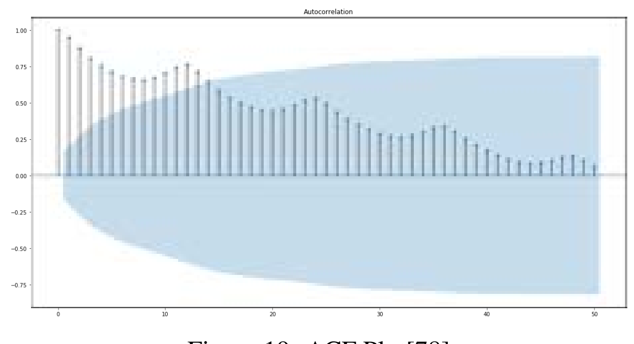


Figure 19: ACF Plot[78]

6.3 Log returns

Most methods of financial time series analysis operate with static data. A stationary time series is represented by data over time, whose statistical properties remain constant regardless of a change in the time origin. There are various ways of making time series stationary, but those ways look at the difference between values rather than the absolute values. In market data, the standard way to get stationary data is to work with log returns. They are calculated as the natural logarithm of the index today divided by the index of the day before: $\ln(\frac{V}{t-1})$.[79] (Fig.20) show the log returns as a time series of E-Mini S&P500. We observe that the E-mini S&P500 has five volatility clusters between 01.11.2021 and 06.11.2021, but the fluctuations are not very high therefore they cannot be identified as extreme clusters.

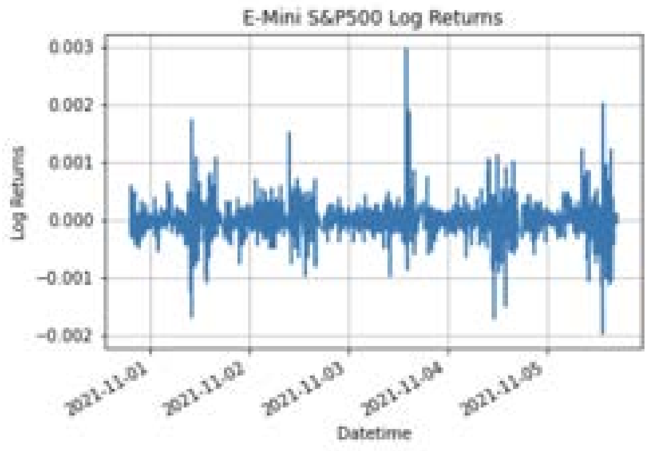


Figure 20: Log Returns of E-Mini S&P 500

6.4 Fat tails

A common stylized fact of financial time series is the exhibition of fat tails. The distribution of a variable has fat tails if its outcomes are more extreme than a normally distributed variable, where the mean and variance are equal. The qq-plot is a common graphical method that is used to analyze the tails of a distribution. (Fig.21) show the qq-plots, comparing the distribution of log returns with the expected normal distribution. We see that the E-Mini S&P500 returns generally do not correspond to a normal distribution.

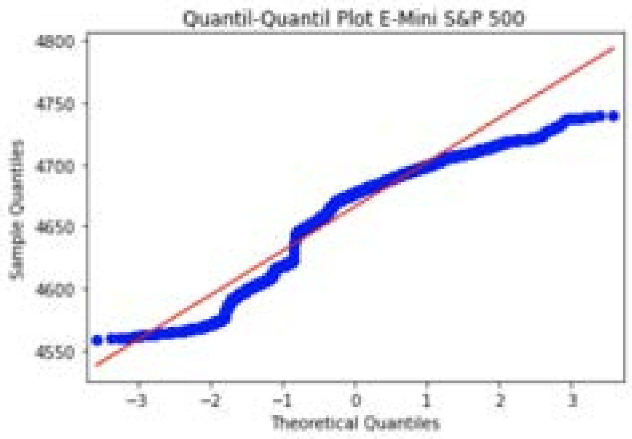


Figure 21: Quantil-Quantil Plot of E-Mini S&P 500

6.5 Volatility Clusters

A common stylized fact is volatility clustering. It means that high price volatility events tend to cluster with time. In addition, it generally means that significant price changes follow large changes in price levels, and small changes tend to be followed by small changes. Autocorrelation plots are realistic representations of volatility clustering effects. We use the (linear) autocorrelation function (ACF) to measure how the current price values and its past values correlate. (Fig.22) show that the E-mini S&P500 has no linear autocorrelation. This result confirms generally accepted knowledge that the price movements in financial markets are not exhibiting any significant linear correlations. If such correlation were statistically significant, the prediction of stock prices would overall deliver highly accurate results because there would be strong evidence for predictability. In (Fig.23), the dependency of the conditional variance can be captured using absolute log returns. Moreover it shows, that absolute log returns present significant correlations with a slow decaying trend. Therefore, we conclude that log returns are not independent, principally caused by volatility clustering.[80]

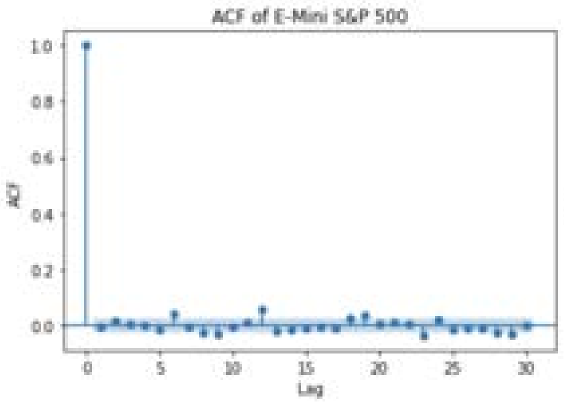


Figure 22: ACF of E-Mini S&P 500

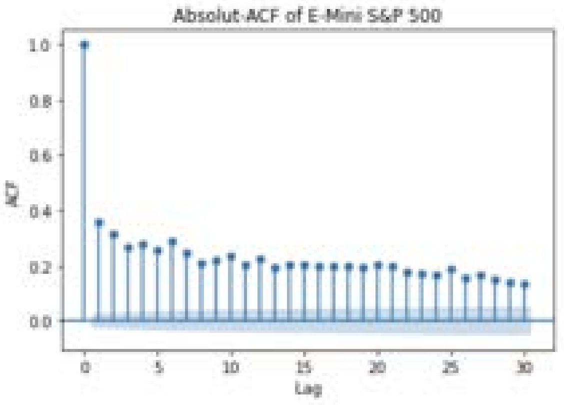


Figure 23: Absolut ACF of E-Mini S&P 500

6.6 Skewness and kurtosis

Fundamental statistic and probability theory almost exclusively deal with the first and second central moment of a random variable, expectation and variance. For the statistical analysis of financial data, the third and fourth central moments called skewness and kurtosis are often meaningful.
The skewness is the third central moment and is essential for measuring another common stylized fact called the "Gain-Loss Asymmetry". It is referred to as the observation when significant drawdowns in a stock index value appear, but not equally large upward movements. The skewness measures the symmetrical gathers around its mean, and therefore are all symmetrically characterized with a skewness value of 0. Positive skewness implies that the right tail is fatter than the left, which means that positive returns tend to occur more often than significant negative returns. The kurtosis is the fourth central moment and measures the degree of peakedness of the distribution relative to its tails. The kurtosis is a measurement mix between asymmetry and tail-weight, and hence it is more informative for symmetrical distributions. The kurtosis of the normal distribution is given by the value 3. This is the reason, why a higher kurtosis value than 3 is a signal for fat tails.
The minimal kurtosis value of a distribution is equal to 1. The probability density function of the log-returns of both investigated distributions is shown in (Fig.24) with their associated skewness and kurtosis values. At first glance, it can be seen that the log returns from the generated data are not staggeringly different from the historical data. The negative skewness values mean that both distributions have thicker left tails than right tails, while the historical data seem to be slightly more left skewed. Furthermore, we confirm that both time series have a higher kurtosis value than the normal distribution. Our analysis shows that the value of the historical data has a higher kurtosis value than the generated data.

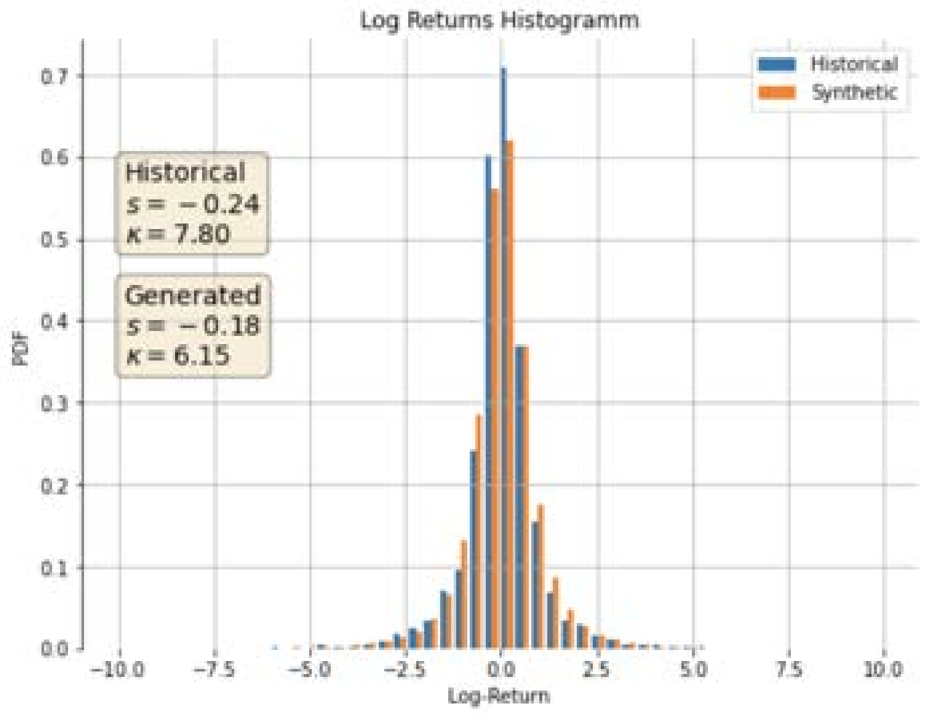


Figure 24: Empirical evaluation of E-Mini S&P 500

6.7 Analysis of skewness and kurtosis with different epoch lengths

In the previous subchapter we wrote about skewness and kurtosis. Now we will examine the distributions of the log returns of the E-mini S&P500 generated by the WGAN-GP. Four different epochs are illustrated. (Fig.25) describes the output of one epoch, (Fig.26) describes the output of 100 epochs, (Fig.27) the output of 300 epochs and (Fig.28) the output of 500 epochs.

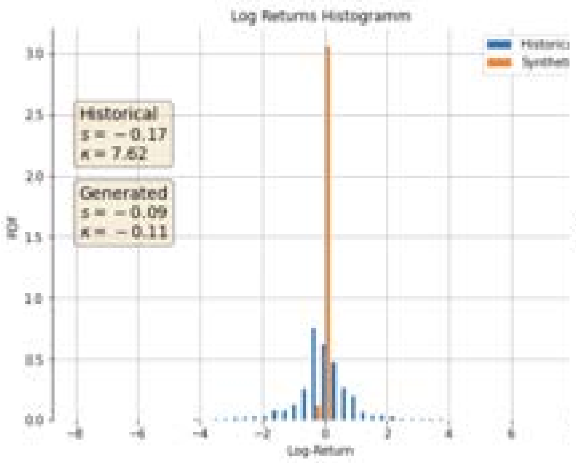


Figure 25: Log Returns Histogram by Epoch 1

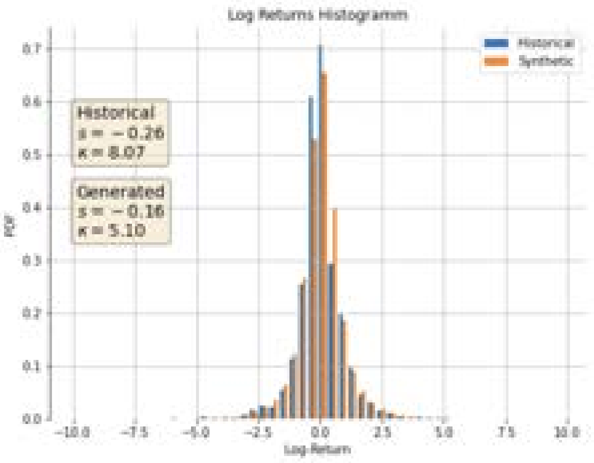


Figure 26: Log Returns Histogram by Epoch 100

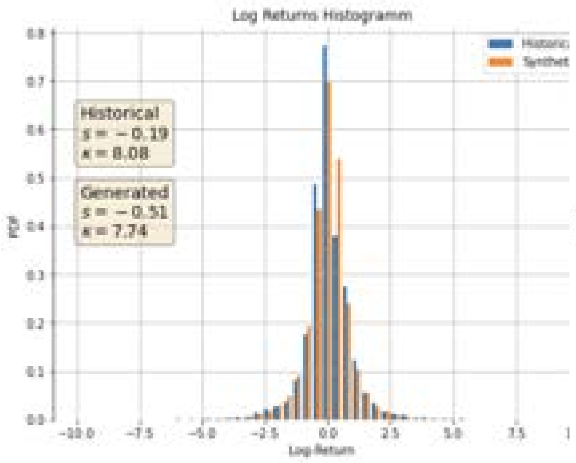


Figure 27: Log Returns Histogram by Epoch 300

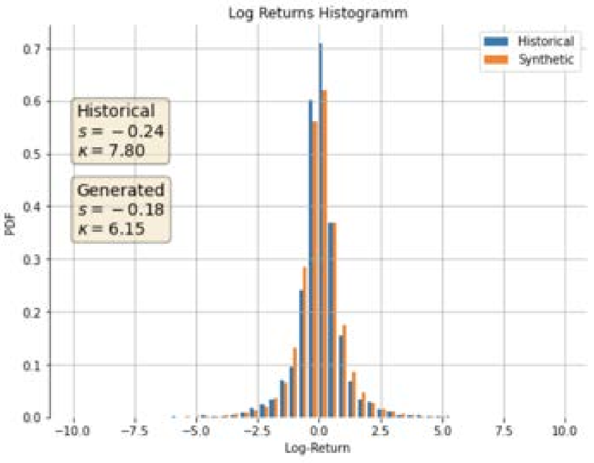


Figure 28: Log Returns Histogram by Epoch 500

If we look at the values of Skewness, we see that for all epochs the values are negative. The distributions of the Log Returns have a thicker left tail than the right, which means that negative returns tend to occur more often than significant positive returns. If we have a look on the kurtosis, as we already know, a value higher than 3 is a signal for fat tails. The kurtosis analysis shows that for all epochs except for epoch 1, fat tails have been generated.

6.8 Financial Risk Measurement

In this chapter, we will delve into the theory of various risk measures such as the maximum drawdown, the Value at Risk (VaR), the Conditional Value at Risk (CVaR or Expected Shortfall ES) and finally the Sharpe Ratio. The results generated with the data from the WGAN-GP are presented in the chapter 7.

6.8.1 Maximum Drawdown

The Maximum Drawdown is an asymmetric risk measure in banking that measures the highest price loss in financial instruments that can be achieved by an investor during an accounting period see (Fig.29). The counterpart is the maximum price gain as the "best case". If an investor has bought a stock at the highest price and sold it at the lowest price within the reference period, he has realized the worst case, and the loss in value achieved corresponds to the maximum drawdown.[81]

The extent of the losses of an investment can be analysed with the maximum drawdown. The MDD is the historically largest possible percentage decline of the capital curve that occurred in the respective time period.[82]

We are working with the log returns of financial time series and the MMD is calculated as follows:

$$MMD(T) = -\min_{\tau \in (0,T)} \left(\min_{\tau \in (0,T)} R(t, \tau) \right) \tag{22}$$

where

$$R(t, \tau) = log(x_t) - log(x_\tau) \tag{23}$$

is the Log Return.

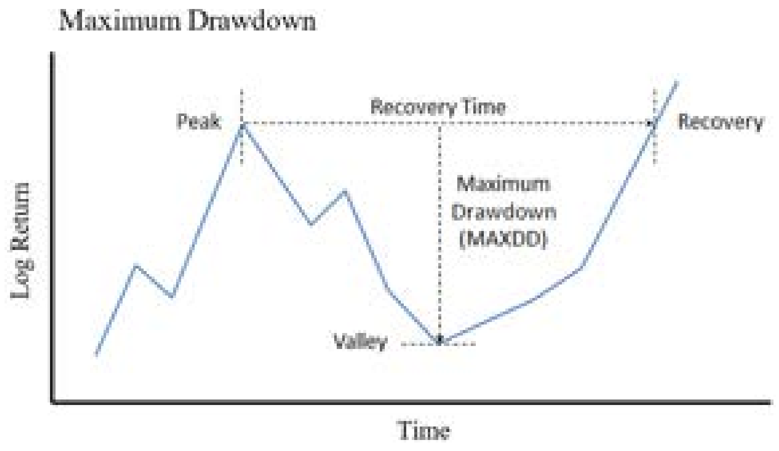


Figure 29: Maximum Drawdown example[82]

The results of the output of the WGAN-GP can be seen in chapter 7.

6.8.2 VaR and CVaR

Value-at-risk estimates how much a single financial instrument or portfolio of assets can lose with a certain probability within a certain period of time (e.g., one day). VaR has become the standard for measuring risk in the financial sector. The confidence level is usually 95% or 99%, i.e., the probability that the loss will not exceed the VaR level is 95% (or 99%). For example, we want to calculate VaR with synthetic data and compare it with VaR in historical data, those result will be discussed in chapter 7.

The Value at risk, with confidence level $\alpha \in]0, 1[$ was calculated with the following formula.

$$VaR_{\alpha}(X) = S_0(1 - e^{z_{\alpha}\sigma}) \tag{24}$$

CVaR considers losses that exceed VaR. Whether this property is good or bad depends on the application of VaR: CVaR fully illustrates the risk reflected in extreme terminal values. This is a very important feature for the correct estimation of extreme losses. Compared to VaR, CVaR shows relatively poor out-of-sample performance if the tail is not modelled correctly. The Expected Shortfall, with confidence level $\alpha \in]0, 1[$ was calculated with the following formula.

$$CVaR_{\alpha}(X) = \int_{-\infty}^{\infty} z F_X^{\alpha}(z) \tag{25}$$

$$F_X^{\alpha}(z) = \begin{cases} 0, & \text{when z} < VaR_{\alpha}(X) \\ \frac{F_X(z)-\alpha}{1-\alpha}, & \text{when z} \geq VaR_{\alpha}(X) \end{cases} \tag{26}$$

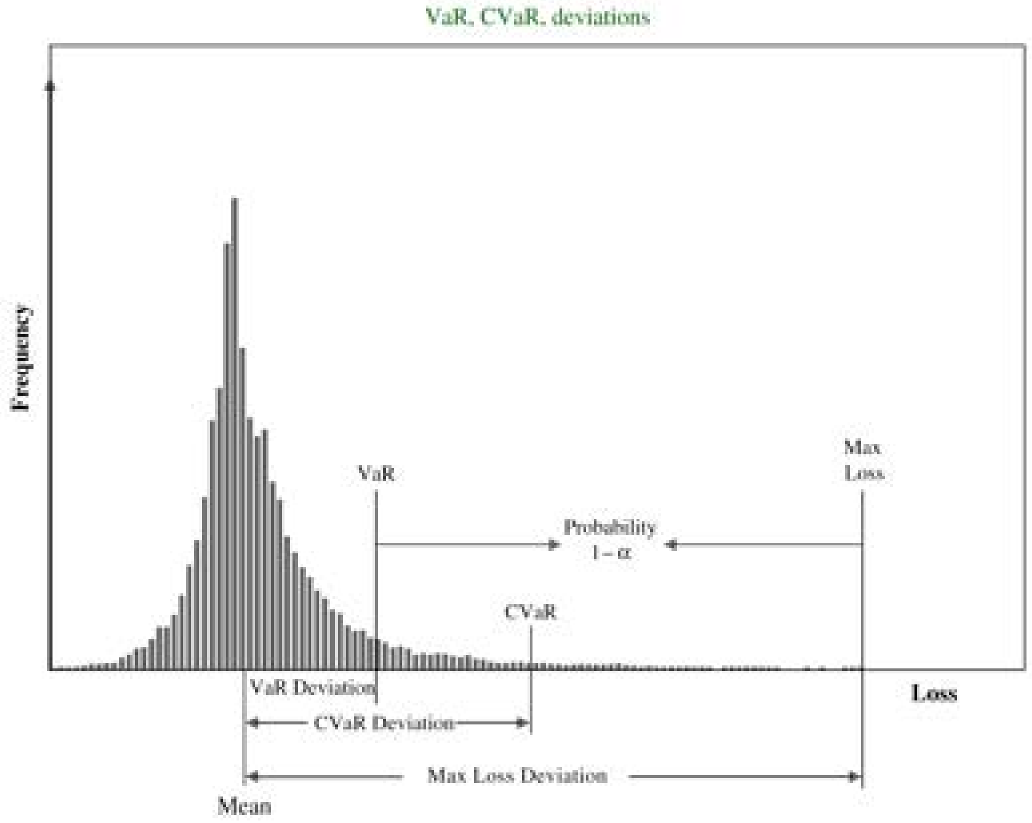


Figure 30: Var and CVaR[83]

The (Fig.30) shows the interaction of VaR and CVaR. It can be seen that CVaR is only used when the largest possible loss is of interest. The CVaR considers the probability that lies above the VaR.[83]

6.8.3 Sharpe Ratio

The Sharpe Ratio measures the fund's excess return per unit of risk. Therefore, it indicates whether the investment result represents high or low risk. A positive Sharpe ratio means that you can achieve a higher return than a risk-free money market investment. On the other hand, it shows the relationship between this excess return and the risk taken. Conversely, a negative Sharpe ratio of less than zero means that money market returns are not even exceeded. We calculated the Sharpe Ratio with the following formula:[84]

$$S = \frac{R_p - R_f}{\sigma_p} \tag{27}$$

R_p = Average Returns of the Investment/Portfolio that we are considering.
R_f= Returns of a Risk-free Investment
σ = Standard Deviation of the Portfolio/Investment

The results of the output of the WGAN-GP can be seen in chapter 7.

7 WGAN-GP results

In this chapter, we will apply the generated data to various risk measures from the financial industry. The aim is to show that the generated data can be used to perform meaningful analyses for investors. First, we will present the generated data with the corresponding statistical characteristics. Then we apply the data to the Maximum Drawdown, VaR / CVaR (Expected Shortfall) and Sharpe Ratio.

As we described in the introduction, we used in five minutely data from the E-mini S&P500. It begins at 2021-10-28 and ends at 2021-11-26. In table 2 we see the first six data point of the data set and in table 3 the last six.

Date	Open	High	Low	Close	Adj Close
2021-10-28 18:00:00	4577.00	4579.25	4576.25	4577.00	4577.00
2021-10-28 18:05:00	4577.25	4578.00	4576.25	4577.00	4577.00
2021-10-28 18:10:00	4577.00	4577.50	4575.50	4577.25	4577.25
2021-10-28 18:15:00	4577.00	4577.25	4575.75	4577.00	4577.00
2021-10-28 18:20:00	4577.00	4578.00	4576.75	4577.75	4577.75

Table 2: The first six data points of the E-mini S&P500 data set

Date	Open	High	Low	Close	Adj Close
2021-11-28 12:40:00	4615.00	4616.00	4608.25	4610.75	4610.75
2021-11-28 12:45:00	4610.50	4612.00	4598.25	4599.25	4599.25
2021-11-28 12:50:00	4599.00	4599.00	4581.25	4582.00	4582.00
2021-11-28 12:55:00	4582.00	4606.25	4581.50	4594.50	4594.50
2021-11-28 12:60:00	4594.75	4601.00	4590.75	4590.75	4590.75

Table 3: The last six data points of the E-mini S&P500 data set

The following (Fig.31 and 32) shows us the price values and the Log Returns. These two plots show the data from 1.11.2021 to 6.11.2021.

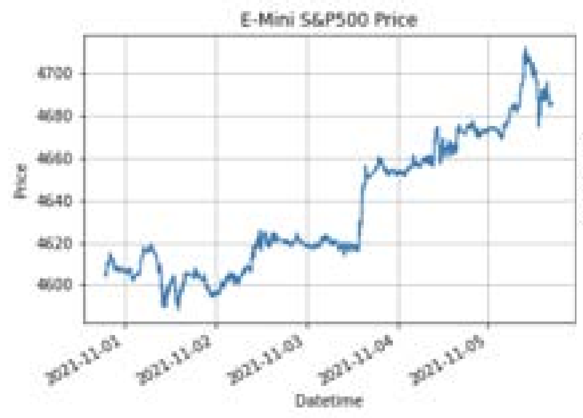


Figure 31: Price of E-Mini S&P500

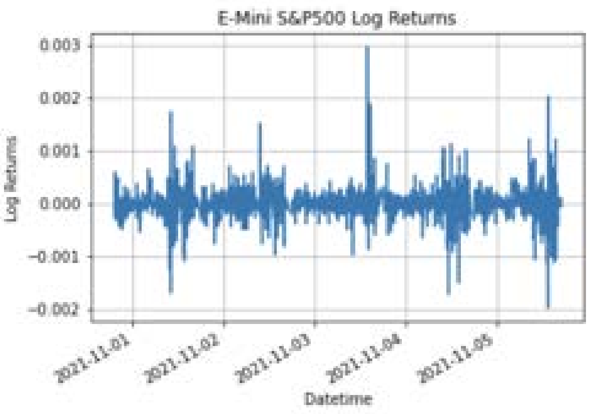


Figure 32: Log return of E-Mini S&P500

Optimisation of the discriminator loss function

It is also worth mentioning that we have optimised the discriminator loss. We have expanded the total loss with factor c. This is shown in the following formula:

$$c = \gamma * (VaR_{historical} - VaR_{generated})^2 \tag{28}$$

$$total_{loss} = real_{loss} + fake_{loss} + \lambda * gradient_{penalty} + c \tag{29}$$

Thanks to this loss optimisation, we have succeeded in achieving convergence after only 500 epochs. This means that through optimisation, the WGAN_GP can achieve good results more quickly.

7.1 Synthetic Data description

In this section, we are going to show the result of the generated data. As we know, there is still a shortage of data, if we're want to train neuronal networks. With the WGAN-GP, we have succeeded in generating synthetic data in the (Fig.33,34,35,36) the generated data are plotted in orange and the historical data in blue. Several epochs were simulated, which clearly shows the development of generated data of the WGAN-GP. The WGAN-GP we have used here generates different time series of length 256. The (Fig.33,34,35,36) show 16 such time series. They can be interpreted as 16 different scenarios. If you look at the graph with one epoch, you will see that the data is not good, the distribution of the data does not match the history. But if you increase the epochs, e.g., to 500, the distribution of the artificially generated data approaches the historical data.

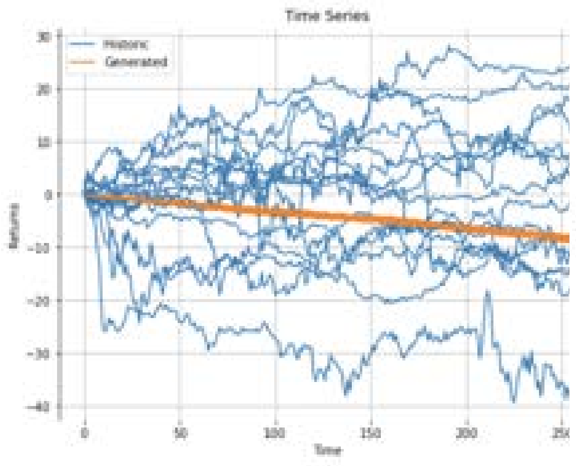


Figure 33: Synthetic data by Epoch 1

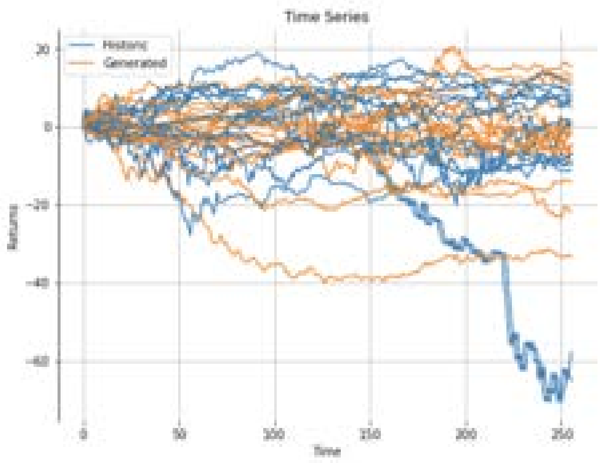


Figure 34: Synthetic data by Epoch 100

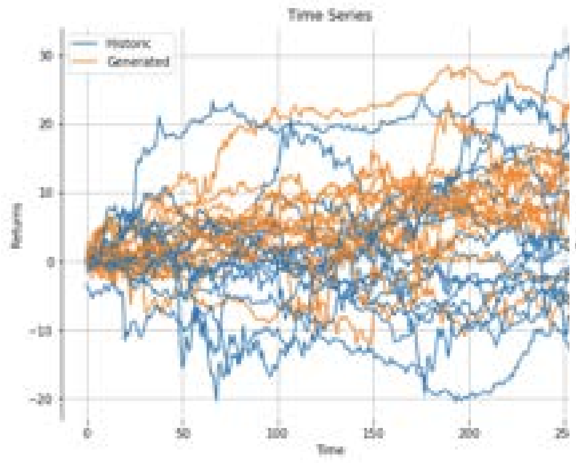


Figure 35: Synthetic data by Epoch 300

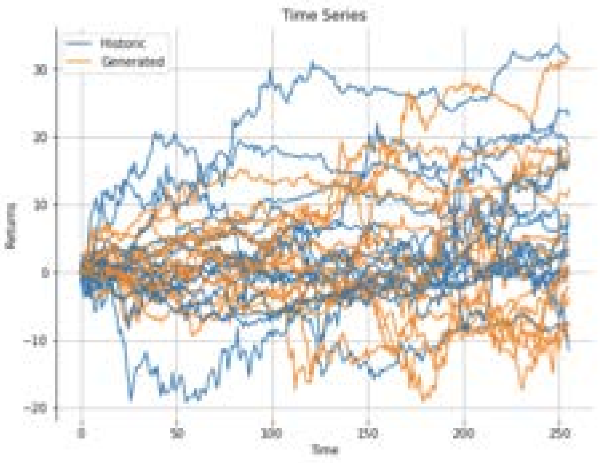


Figure 36: Synthetic data by Epoch 500

In the following, the forecasts and the confidence interval of the different epochs are examined. We here assume throughout that the data-process is stationary. In practice we require weak stationarity, that is:

$$E[x_t] = 0 \tag{30}$$

$$Var[x_t] = \sigma^2\text{fixed} \tag{31}$$

$$Cov(x_t, x_{t-k}) = R(k)\text{does not depend on t} \tag{32}$$

From a purely theoretical perspective, however, we have to (we should) require strong stationarity but we ignore these mathematical subtleties here. Obviously, many economic series are non-stationary: series may be subject to trends, shifts, increasing or decreasing volatility and so on.[85]

When we look at the (Fig.37,38,39,40) we see the mean of the 16-time series. We have calculated the mean for each time step of the 16-time series and thus produced a time series which represents the mean of all generated time series. This is the forecasts and the confidence interval of the respective epochs. It can be seen that as the number of epochs increases, the confidence intervals of the historical and generated data converge (but are not equal). The mean of the generated data corresponds to the forecast. In theory, for stationary time series, the mean is the best estimate of the point forecast.

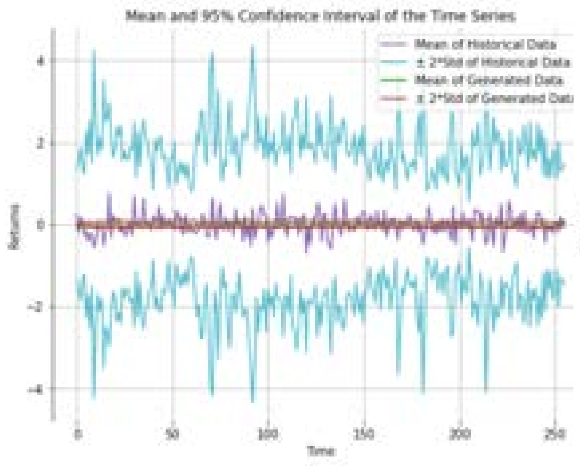


Figure 37: Average returns conf. int. by Epoch 1

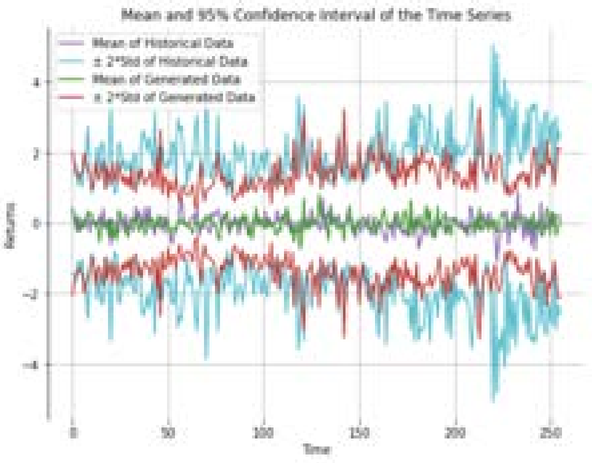


Figure 38: Average returns conf. int. by Epoch 100

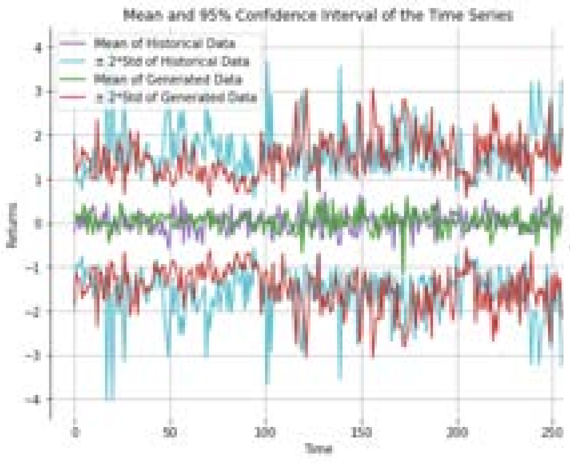


Figure 39: Average returns conf. int. by Epoch 300

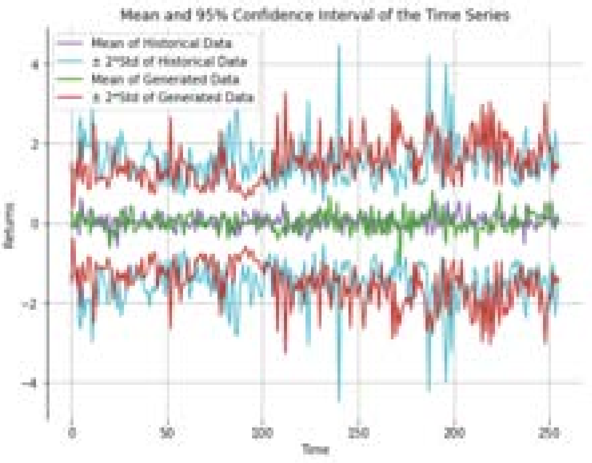


Figure 40: Average returns conf. int. by Epoch 500

7.2 Maximum Drawdown results

In this section, the MDDs were calculated from the generated and historical data. Four different epochs were examined, reflecting the calculation process under the different epochs. In the (Fig.37,38,43,44), the mean of Maximum Drawdown was calculated on historical and generated data. We are interested in the most worst-case scenario. So, we plotted only the negative part. As we can see the result of the Generated MDD in (Fig.44) with a value of -4% for the historical data and -3% for generated data. We can see all results of each epoch in table 4

	Epoch 1	Epoch 100	Epoch 300	Epoch 500
Historical Sharpe Ratio	-0.0116	0.1101	0.0881	0.0509
Generated Sharpe Ratio	-0.5560	-0.0049	0.0455	0.0251

Table 4: Values of Sharpe Ratio per each Epoch

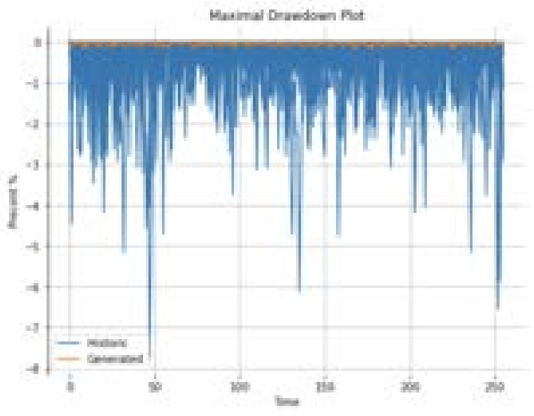


Figure 41: Maximum Drawdown by Epoch 1

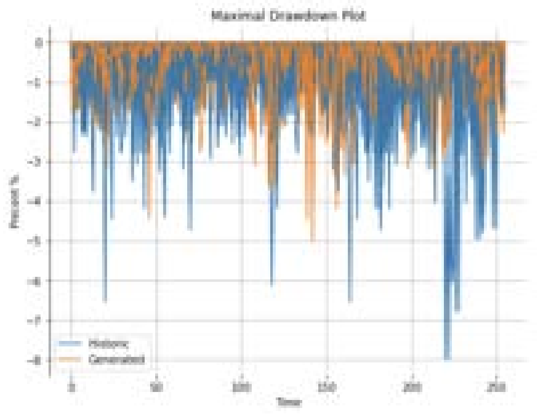


Figure 42: Maximum Drawdown by Epoch 100

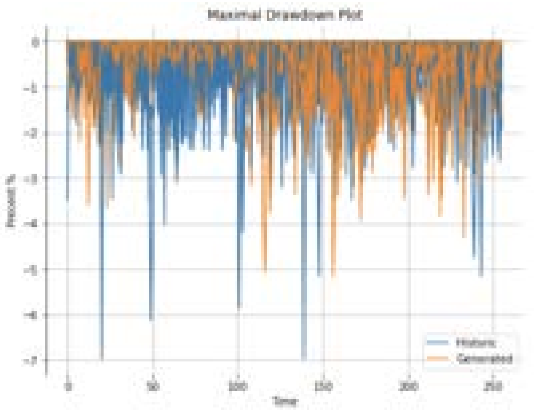


Figure 43: Maximum Drawdown by Epoch 300

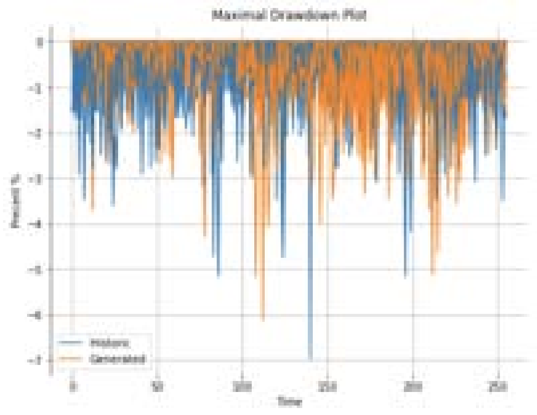


Figure 44: Maximum Drawdown by Epoch 500

If we have a look at the (Fig.45,46,47,48), the distribution of the generated and historical results is very similar but not exactly same. We cannot exactly describe the distribution, but we can see, that the largest part of the data is at the same area and the biggest drawdown of the generated and historical data are in same area of the tails.

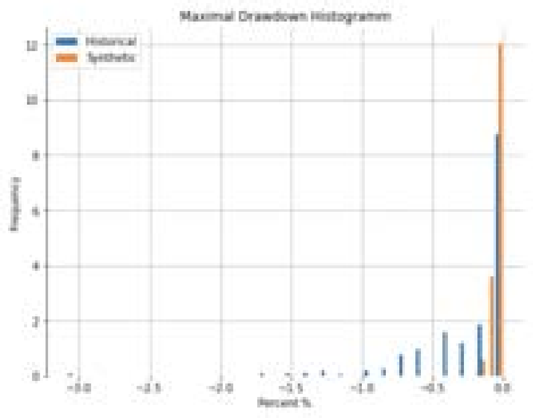


Figure 45: MDD Distribution by Epoch 1

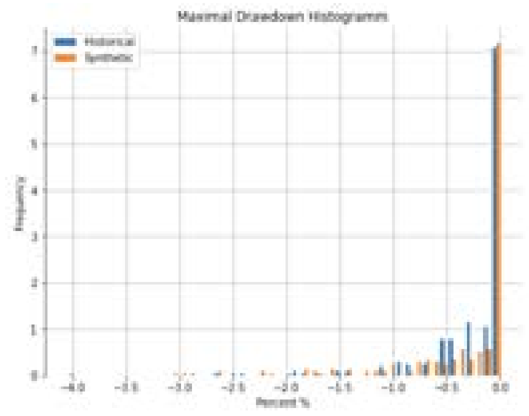


Figure 46: MDD Distribution by Epoch 100

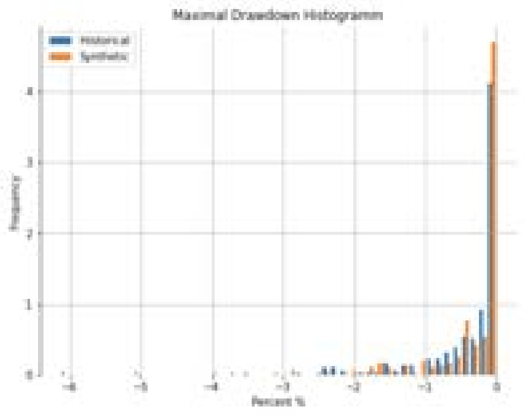


Figure 47: MDD Distribution by Epoch 300

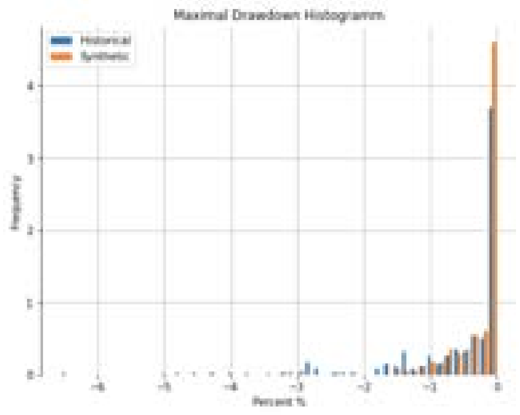


Figure 48: MDD Distribution by Epoch 500

7.3 VaR and CVaR results

In this chapter we will apply the historical and generated data to VaR and CVaR and discuss the results. Table 5 shows the VaR/CVaR results from the historical and generated data. It can be seen that by simulating more epochs, the difference between the historical VaR and the generated VaR approaches zero this also applies to the CVaR.

	Epoch 1	Epoch 100	Epoch 300	Epoch 500
Historical $VaR_{0.05}$	0.713	0.645	0.767	0.751
Historical $CVaR_{0.05}$	-0.602	-0.477	-0.927	-0.804
Generated $VaR_{0.05}$	-0.173	0.730	0.712	0.759
Generated $CVaR_{0.05}$	0.030	-0.473	-0.552	-0.720

Table 5: VaR/CVaR reuslts of generated and historical data

If we look at (Fig.49,50,51,52), it is evident that the quantile plot of the log returns of the generated data behaves similarly to that of the historical data. For this purpose, the log returns are ordered in ascending order. The first point is the value at which 1% of the log returns are smaller, the second point is the value at which 2% of the log returns are smaller and so on. The VaR from the generated data by 500 Epochs at the 5% point is 0.751 (see Table 5). Multiplying this value by the price of the share means that with 95% probability the loss will not be larger. VaR should be treated with caution, because VaR cannot provide any information about the amount of the loss if the VaR value is exceeded. This can be calculated using the Expected Shortfall (CVaR), which we will discuss next. Expected Shortfall calculates the expected loss for loss events beyond the confidence level in this example for 5% of the cases in which the value-at-risk is exceeded. It looks at losses that exceed the VaR and determines their average amount.[83][86]

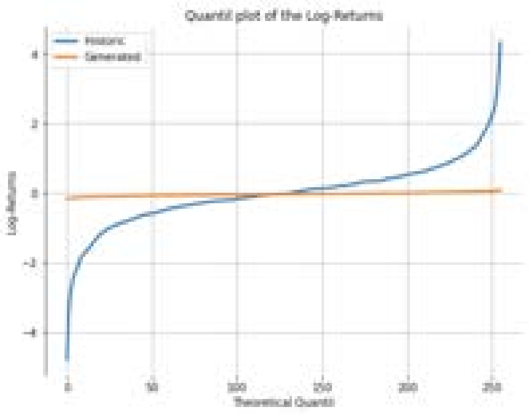


Figure 49: Q-Plot of Log Returns by Epoch 1

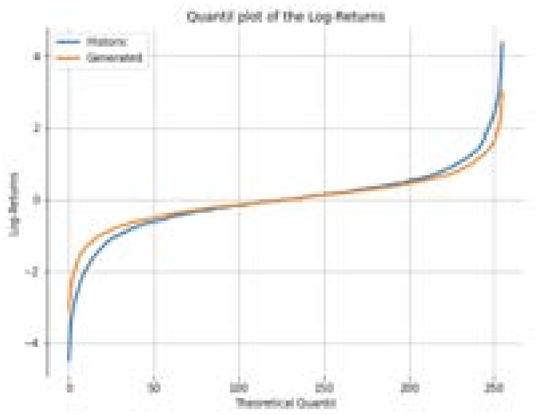


Figure 50: Q-Plot of Log Returns by Epoch 100

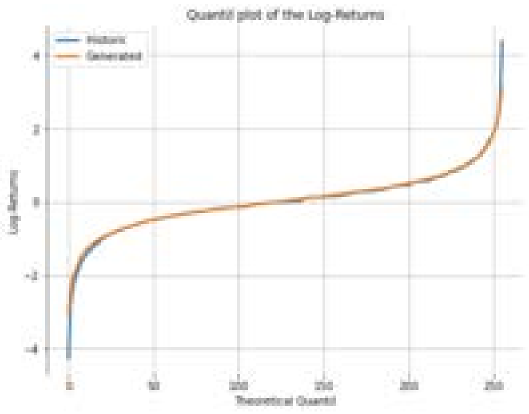


Figure 51: Q-Plot of Log Returns by Epoch 300

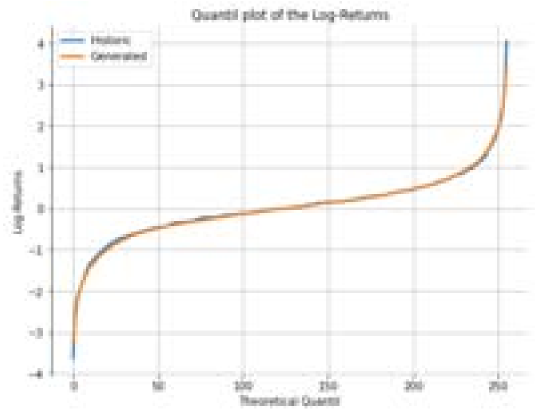


Figure 52: Q-Plot of Log Returns by Epoch 500

(Fig.53,54,55,56) shows the distribution of the log returns. It can be seen that the distribution of the synthetic data differs from the distribution of the historical data. We can also see that the more epochs we make the closer we get to the real data.

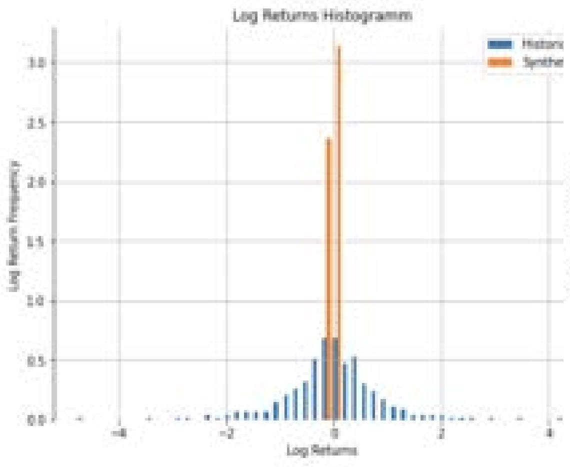


Figure 53: VaR and CVaR Distribution by Epoch 1

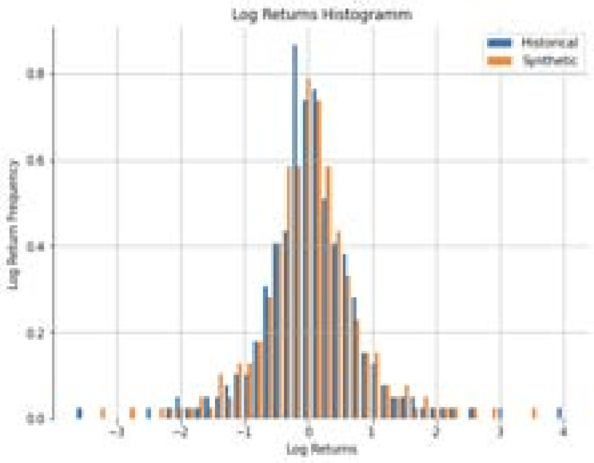


Figure 54: VaR and CVaR Distribution by Epoch 100

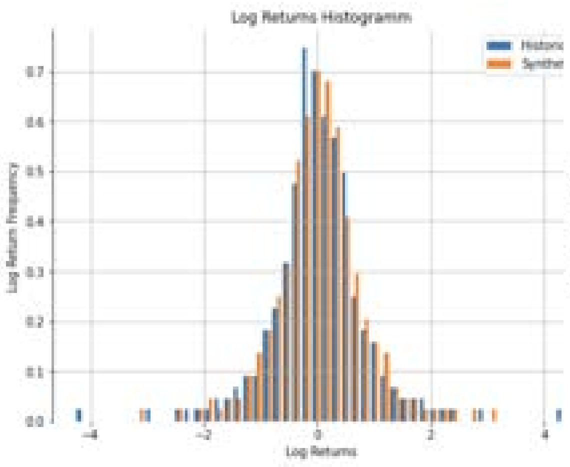


Figure 55: VaR and CVaR Distribution by Epoch 300

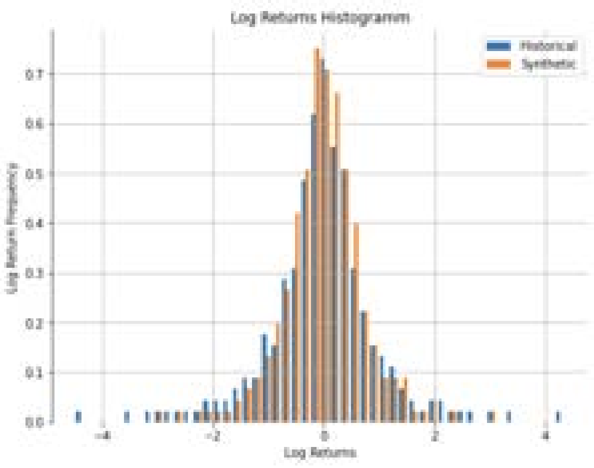


Figure 56: VaR and CVaR Distribution by Epoch 500

7.4 Sharpe ratio results

As we can see here, the difference between the historical Sharpe Ratio and the generated Sharpe Ratio becomes smaller the more epochs the GAN has. This means that if we increase the number of epochs significantly, we will achieve the same Sharpe Ratio as for the historical data. In our case, at Epoch = 500 we have a difference of (0.0258).

	Epoch 1	Epoch 100	Epoch 300	Epoch 500
Historical Sharpe Ratio	-0.0116	0.1101	0.0881	0.0509
Generated Sharpe Ratio	-0.5560	-0.0049	0.0455	0.0251

Table 6: Values of Sharpe Ratio per each Epoch

8 Conclusions - Outlook for the future of GANs in Finance

We gave an overview of Generative Adversarial Networks in finance, describing what they are, why they are used in finance, followed by some important developments in research and practical applications. To test the versatility of different frameworks, it was demonstrated with an example.The importance and wide application of synthetic data in finance was demonstrated, as well as some of the challenges faced in generating it.

8.1 Analysis of results

Active research shows that the generation of synthetic financial data using GANs is feasible and viable, other novel applications such as the calibration of trading models have also shown positive results. Research has refined the training of GANs with financial data, but it is still a complex task and further stabilisation of the training will remain an active research topic. Another problem is the lack of a consistent quantitative metric to evaluate performance on generated financial data. Most of the literature relies on reproducing stylised facts, which is a sound metric, but a more precise assessment would greatly benefit the development of these models, as noted earlier in the literature review.

Although a novel approach, GANs are making inroads into the financial industry. Some applications have been demonstrated, in particular the use of synthetic data generated by adversarial models is gaining momentum. The usefulness of good synthetic data is gaining traction as it opens up the possibility of modelling information that would otherwise be regulated for privacy reasons or too scarce for deep models. It has been shown that GANs can generate consistent data for various purposes, including retail banking and market data.

8.2 Outlook and recommendations

Overall, GANs have a large potential in solving the specific need for synthetic financial data, along with other diverse modelling tasks. Being still a novelty in finance, they are more present in research than in practice, but their potential can increase the capabilities of data-driven deep models, reaching areas where these methods previously could not be applied due to limitations in data. The development of better evaluation metrics for sample comparison is a key aspect for the further advance in the field, since providing easier tools would make it easier for use by less qualified personnel. Refinements to improve stability and reliability in the training process are in active research and there are still some improvements to be made before a state-of-the-art model is presented. Taking all of this into account, Generative Adversarial Networks are a good augmentation of the modelling toolbox, adding new possibilities to the challenging world of quantitative finance.

8.3 Summary

In this paper, we covered the structure and construction of a GAN and LSTMs in detail and summarized the most important literature. We trained our recurrent WGAN-GP model with improved discriminator loss for generating real-valued sequential data. The implementation of the Wasserstein metric ensured a stable loss of both the generator and the discriminator throughout the training. The loss quickly converged to a minimum with cycle oscillation thanks to improved discriminator loss. It is difficult to find the cause of this oscillation. LSTM models are at an early stage and the model parameters are not yet optimally set. Finding the optimal parameters to fine-tune model performance is challenging and a time-consuming task that provides a basis for further investigation. However, the evaluation has shown that a visual distinction between the generated and the real-time series is almost impossible. Thanks to the improvement of the discriminator loss, we were able to achieve significantly better results.

How closely a GAN can track and reflect the characteristics of stock prices is still difficult to demonstrate or prove due to high volatility and unexpected events in the market. Nevertheless, promising arguments and approaches argue for further experimentation in this area. GANs seem to be an excellent method to capture the dynamics of financial assets and predict future movements. Their application to derivatives pricing, portfolio hedging, and risk management could become an important tool in the future as we live in an era of big data.

References

[1] Goodfellow Ian J, Pouget-Abadie Jean, Mirza Mehdi, Xu Bing, Warde-Farley David, Ozair Sherjil, Courville Aaron, and Bengio Yoshua. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[2] Katie Martin. Flash crash — the trading savant who crashed the us stock market. *Financial Times*, May 2020.

[3] Christopher B Barrett. Market analysis methods: are our enriched toolkits well suited to enlivened markets? *American journal of agricultural economics*, 78(3):825–829, 1996.

[4] Robert F. Engle, Sergio M. Focardi, and Frank J. Fabozzi. ARCH/GARCH Models in Applied Financial Econometrics. In *Encyclopedia of Financial Models*. American Cancer Society, 2012.

[5] IBM Cloud Education. What are neural networks. url: `https://www.ibm.com/cloud/learn/neural-networks`, August 2020.

[6] Gautam Ramachandra. Generative adversarial networks. url: `https://medium.com/@gautamrbharadwaj/generative-adversarial-networks-85900be25425`, February 2017.

[7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. url: `https://arxiv.org/pdf/1406.2661.pdf`, 2014.

[8] Brownlee Jason. What is deep learning. url: `https://machinelearningmastery.com/what-is-deep-learning/`, 2019.

[9] Hu J, Niu H, Carrasco J, Lennox B, and Arvin F. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. url: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9244647`, December 2020.

[10] Zhengwei Wang, Qi She, and Tomas E. Ward. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. *arXiv:1906.01529 [cs]*, December 2020. arXiv: 1906.01529.

[11] Jie Gui, Zhenan Sun, Yonggang Wen, Dacheng Tao, and Jieping Ye. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. *arXiv:2001.06937v1*, January 2020.

[12] Alankrita Aggarwal, Gopi Battineni, and Mamta Mittal. Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, January 2021.

[13] Pegah Salehi, Abdolah Chalechale, and Maryam Taghizadeh. Generative Adversarial Networks (GANs): An Overview of Theoretical Model, Evaluation Metrics, and Recent Developments. *arXiv:2005.13178v1*, May 2020.

[14] Minhyeok Lee and Junhee Seok. Regularization Methods for Generative Adversarial Networks: An Overview of Recent Studies. *arXiv:2005.09165v1*, May 2020.

[15] Takahashi Shuntaro, Chen Yu, and Tanaka-Ishii Kumiko. Modelling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527, Apr 2019.

[16] Anirban Chakraborti, Ioane Muni Toke, Marco Patriarca, and Frédéric Abergel. Econophysics review: I. empirical facts. url: `https://hal.archives-ouvertes.fr/hal-00621058/document`.

[17] Cont Rama. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 2001.

[18] Zhou Xingyu, Pan Zhisong, Hu Guyu, Tang Siqi, and Zhao Cheng. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018, 2018.

[19] Koshiyama Adriano, Firoozye Nick, and Treleaven Philip. Generative adversarial networks for financial trading strategies fine-tuning and combination. *arXiv preprint arXiv:1901.01751*, 2019.

[20] Giovanni Mariani, Yada Zhu, Jianbo Li, Florian Scheidegger, Roxana Istrate, Costas Bekas, and A Cristiano I Malossi. Pagan: Portfolio analysis with generative adversarial networks. *arXiv preprint arXiv:1909.10578*, 2019.

[21] Gautier Marti. Corrgan: Sampling realistic financial correlation matrices using generative adversarial networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8459–8463. IEEE, 2020.

[22] Teema Leangarun, Poj Tangamchit, and Suttipong Thajchayapong. Stock price manipulation detection using generative adversarial networks. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2104–2111. IEEE, 2018.

[23] Cem Dilmegani. The ultimate guide to synthetic data in 2021. url: `https://research.aimultiple.com/synthetic-data/`, Jun 2021.

[24] Financial Conduct Authority. *Supporting innovation in financial services: the digital sandbox pilot*. FCA, Apr 2021.

[25] James Montantes. Deep learning in finance: Is this the future of the financial industry?, Jul 2020.

[26] Samuel Assefa. Generating synthetic data in finance: opportunities, challenges and pitfalls. *Challenges and Pitfalls (June 23, 2020)*, 2020.

[27] Muthukumar Pratyush and Zhong Jie. A stochastic time series model for predicting financial trends using nlp. *arXiv preprint arXiv:2102.01290*, 2021.

[28] Wu Weijie, Huang Fang, Kao Yidi, Chen Zhou, and Wu Qi. Prediction method of multiple related time series based on generative adversarial networks. *Information*, 12(2):55, 2021.

[29] Storchan Victor, Balch Tucker, and Vyetrenko Svitlana. Mas-gan: Adversarial calibration of multi-agent market simulators. Paper under review, 2021.

[30] Yerin Kim, Daemook Kang, Mingoo Jeon, and Chungmok Lee. Gan-mp hybrid heuristic algorithm for non-convex portfolio optimization problem. *The Engineering Economist*, Jun 2019.

[31] Sun He, Deng Zhun, Chen Hui, and Parkes David C. Decision-aware conditional gans for time series data. *arXiv preprint arXiv:2009.12682*, 2020.

[32] Yoon Jinsung, Jarrett Daniel, and van der Schaar Mihaela. Time-series generative adversarial networks. *NeurIPS 2019*, 2019.

[33] Fernando De Meer Pardo. Enriching financial datasets with generative adversarial networks. Master's thesis, Delft University of Technology, August 2019.

[34] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: Deep generation of financial time series. *arXiv:1907.06673v2 [q-fin.MF]*, Dec 2019.

[35] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[36] Dmitry Efimov, Di Xu, Luyang Kong, Alexey Nefedov, and Archana Anandakrishnan. Using generative adversarial networks to synthesize artificial financial datasets. *arXiv preprint arXiv:2002.02271*, 2020.

[37] Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional sig-wasserstein gans for time series generation. *arXiv preprint arXiv:2006.05421*, 2020.

[38] Akhil Sethia, Raj Patel, and Purva Raut. Data augmentation using generative models for credit card fraud detection. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pages 1–6. IEEE, 2018.

[39] Zhaohui Zhang, Lijun Yang, Ligong Chen, Qiuwen Liu, Ying Meng, Pengwei Wang, and Maozhen Li. A generative adversarial network–based method for generating negative financial samples. *International Journal of Distributed Sensor Networks*, 16(2):1550147720907053, 2020.

[40] Kunfeng Wang, Yanjie Duan, Yilun Lin, Xinhu Zheng, and Fei-Yue Wang. Generative Adversarial Networks: Introduction and Outlook. *IEEE/CAA Journal of Automatica Sinica*, 4(4):588–598, October 2017.

[41] Yaxing Wang, Lichao Zhang, and Joost van de Weijer. Ensembles of Generative Adversarial Networks. *arXiv:1612.00991 [cs]*, December 2016. arXiv: 1612.00991.

[42] Zhengwei Wang, Qi She, and Tomas E Ward. Generative adversarial networks in computer vision: A survey and taxonomy. *arXiv preprint arXiv:1906.01529*, 2019.

[43] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[44] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[45] Gulrajani Ishaan, Ahmed Faruk, Arjovsky Martin, Dumoulin Vincent, and Courville Aaron. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[46] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[47] Gautier Marti. corrgan.io. url: `http://www.corrgan.io/`, 2019.

[48] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR, 2019.

[49] Frank Xu. Building your own self-attention gans. url: https://towardsdatascience.com/building-your-own-self-attention-gans-e8c9b9fe8e51, Jul 2020.

[50] FCA and City of London. Digital sandbox pilot. url: https://www.digitalsandboxpilot.co.uk/, 2021.

[51] TJ Horan. Credit card fraud: It's still a thing (and as big as ever), Jan 2021.

[52] David Love, Nalin Aggarwal, Alexander Statnikov, and Chao Yuan. An automated system for data attribute anomaly detection. In *KDD 2017 Workshop on Anomaly Detection in Finance*, pages 95–101. PMLR, 2018.

[53] Kaggle. Intro to recurrent neural networks lstm. url: https://www.kaggle.com/thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru, 2018.

[54] Kaggle. Funktionsweise künstlicher neuronaler netze. url: https://data-science-blog.com/blog/2018/08/31/funktionsweise-kunstlicher-neuronaler-netze/, August 2018.

[55] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. url: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf, 1958.

[56] Harshit Dwivedi. Understanding gan loss functions. url: https://neptune.ai/blog/gan-loss-functions, August 2018.

[57] Victor Zhou. Machine learning for beginners: An introduction to neural networks. url: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9.

[58] SAS Institut Inc. Neural networks - "what they are and why they matter. url: https://www.sas.com/en_sa/insights/analytics/neural-networks.html, August 2020.

[59] Lalithnaryan C. Perceptron algorithm - a hands on introduction. url: https://www.section.io/engineering-education/perceptron-algorithm/, November 2020.

[60] Robert E. Schapire Yoav Freund. Large margin classification using the perceptron algorithm. url: http://cseweb.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf, November 1999.

[61] Nico Litzel Dipl.-Ing. (FH) Stefan Luber. Was ist ein perzeptron? url: https://towardsdatascience.com/https-medium-com-francesco-cicala-whats-whys-and-hows-of-perceptron-f87c66f512c5, August 2018.

[62] Craig Glastonbury. A primer on artificial neural networks and convolutions for image data. url: https://glastonburyc.github.io/neuralprimer.html, 2017.

[63] Roger Grosse. Multilayer perceptrons. url: https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec05.pdf, 2018.

[64] Jonathan Hui. Backpropagation step by step. url: https://hmkcode.com/ai/backpropagation-step-by-step/, November 2019.

[65] Zewen Li, Wenjie Yang, Shouheng Peng, and Fan Liu. A survey of convolutional neural networks: Analysis, applications, and prospects. url: https://arxiv.org/pdf/2004.02806.pdf, April 2020.

[66] Sanket Doshi. Various optimization algorithms for training neural network. url: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6, January 2019.

[67] Nagesh Singh Chauhan. Optimization algorithms in neural networks. url: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html, January 2020.

[68] Google developers. Loss functions. url: https://developers.google.com/machine-learning/gan/loss, February 2020.

[69] David Foster. Chapter 4. generative adversarial networks. url: https://www.oreilly.com/library/view/generative-deep-learning/9781492041931/ch04.html.

[70] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[71] Wuttke Laurenz. Was ist ein künstliches neurales netzwerk. url: https://datasolut.com/neuronale-netzwerke-einfuehrung, 2020.

[72] Jonathan Hui. Gan — wasserstein gan and wgan-gp. url: https://jonathan-hui.medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490, June 2018.

[73] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. *arXiv:1704.00028 [cs, stat]*, December 2017. arXiv: 1704.00028.

[74] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. url: `https://arxiv.org/pdf/1704.00028.pdf`, December 2017.

[75] Lilian Weng. From gan to wgan. url: `https://arxiv.org/pdf/1904.08994.pdf`, April 2019.

[76] Luc Van Gool Zhiwu Huang, Jiqing W. Manifold-valued image generation with wasserstein generative adversarial nets. url: `https://arxiv.org/pdf/1712.01551.pdf`, January 2019.

[77] Quantile-quantile plot. url: `https://www.itl.nist.gov/div898/handbook/eda/section3/qqplot.htm`.

[78] Krzysztof Drelczuk. Acf (autocorrelation function) — simple explanation with python example. url: `https://medium.com/@krzysztofdrelczuk/acf-autocorrelation-function-simple-explanation-with-python-example-492484c32711`, May 2020.

[79] Ms. Archana Bathula Mr. Karthik Jilla, Dr. Sarat Chadra Nayak. tylized facts of financial time series: A comprehensive analysis. url: `https://www.ijrter.com/published_special_issues/01-12-2017/stylized-facts-of-financial-time-series-a-comprehensive-analysis.pdf`, 2017.

[80] Rama Cont. Volatility clustering in financial markets: Empirical facts and agent-based models. url: `https://link.springer.com/chapter/10.1007/978-3-540-34625-8_10`, 2017.

[81] Choi Jaehyung. Maximum drawdown, recovery, and momentum. url: `https://arxiv.org/pdf/1403.8125.pdf`, 2021.

[82] Boccadoro Charles. Recovery time. url: `https://www.mutualfundobserver.com/2014/08/recovery-time/`, 2014.

[83] Sarykalin Sergey, Serraino Gaia, and Uryasev Stan. Value-at-risk vs. conditional value-at-risk in risk management and optimization. url: `https://www.ise.ufl.edu/uryasev/files/2011/11/VaR_vs_CVaR_INFORMS.pdf`, 2008.

[84] Sharp William F. The sharpe ratio. url: `https://web.stanford.edu/~wfsharpe/art/sr/SR.htm`.

[85] Marc Wildi. An introduction to conditional volatility models. *ZHAW School of Engineering*, page 3, 2020.

[86] Northstar risk. Expected shortfall. url: `https://www.northstarrisk.com/expected-shortfall`.