University College London

Department of Computer Science

FTS-GAN: Generating Multidimensional Financial Time Series

Candidate Number

KRSJ5

Academic Supervisor

Dr Denise Gorse

Department of Computer Science

University College London

Industrial Supervisor

Mr Federico Fontana

Quantitative Strategies

XAI Asset Management

September 13, 2021

This dissertation is submitted as partial requirement for the MSc Computational Finance degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text.

Abstract

In an ever growing world driven by data, generative models have found a use when datasets are hard to come by or there is limited available data. Small datasets can impair deep learning models as they will suffer from the problem of overfitting. Synthetic data can be used to mitigate this problem. This project proposes a novel framework, Financial Time Series GAN (FTS-GAN), which is a Generative Adversarial Network (GAN) that 'stacks' Temporal Convolutional Nets (TCNs) to correctly capture the temporal dependence of multiple financial time series. It learns the underlying distribution of some input data and can generate financial series that are close to this distribution. FTS-GAN is one of the first frameworks based on TCNs to be used in multi series generation of time series. The model is tested to produce synthetic datasets for S&P 500 and VIX data as well as S&P 500, Gold and T-Note indexes in order to push the model and see its performance for multiple assets. FTS-GAN produces plausible results and although has limitations in some areas it provides a potential avenue for TCN based network architectures and multidimensional data.

Keywords: Generative Adversarial Networks, FTS-GAN, GAN, TCN, CNN.

Acknowledgements

I would like to thank my industry supervisor, Federico Fontana, for helping me through the problems I encountered and ensuring that I had all the tools required to carry out my dissertation.

I would also like to thank XAI for welcoming me into their company and allowing me to undertake my project with them. Without the Virtual Machine as well as the data that they provided, this project would have been a real struggle.

Thank you to Denise Gorse for guiding me through the Thesis process and ensuring that I was on the right track.

Contents

1 Introduction **8**

1.1 Why is there a need for synthetic data? 8

1.2 Aims 9

1.3 Structure 10

2 Background & Related Work **11**

2.1 Financial Time Series 11

2.1.1 Statistical Properties 12

2.2 Background 15

2.2.1 Neural Networks 15

2.2.2 Convolutional Neural Networks 21

2.2.3 Temporal Convolution Network 23

2.2.4 Generative Adversarial Network 24

2.3 Related Work 28

3 Methodology **34**

3.1 FTS-GAN Network Architecture 34

3.2 Conditional FTS-GAN 41

3.3 Challenges 42

3.4 Additional GAN improvements 43

3.5 Data Preprocessing 43

3.6 Hyperparameter Selection 44

3.7 Evaluation Methods 45

3.8 Hardware and Setup 46

3.9 Alternative Models Considered 46

4 Results **48**

4.1 Preliminary Experiments 48

4.2 Benchmarking on a Single Time Series 49

4.3 Datasets and Challenges 52

4.4 S&P 500 and VIX 54
4.5 S&P 500, Gold and Government Bonds 59
4.6 OHCL 64
4.7 Effect of the Dataset's Time Frame 65
4.8 Testing a Conditioned FTS-GAN 66
4.9 Train on Synthetic, Test on Real Data 67

5 Conclusions & Further Research **69**
5.1 Conclusions 69
5.2 Further Research 70

A **78**
A.1 S&P and VIX 78
A.2 S&P 500, Gold and T-Note 80
A.3 Project Summary 82

List of Figures

2.1 Stylised facts of financial markets. The plots here are for the S&P 500 over the period 1992 to 2020 14
2.2 A fully connected neural network. 16
2.4 Diagram of a 2D convolution. 22
2.5 Dilated convolutions with increasing dilation size (1, 2, 4). The number of parameters for each layer is the same but the amount of coverage they are gaining is exponentially larger[20]. 23
2.6 Diagram of a 1D convolution [21]. 23
2.7 Diagram of the setup of a standard GAN [23]. 24
2.8 The evolution of GAN training [24]. 26
2.9 Advancements in image synthesis between 2014 to 2017 [38] 29

3.1 Wavenet architecture [58]. 35
3.2 A stack of causal convolutions [58]. 36
3.3 A stack of dilated causal convolutions, with dilation increasing by factor of 2 each layer [58]. 36
3.4 Network diagram for a section of FTS-GAN for two time series. . . . 38
3.5 ECA Module [61]. 39
3.6 Temporal convolution Block 39
3.7 Diagram of the ResNet classifier used in checking the quality of the synthetic data [59]. 46

4.1 Results for WGAN-GP on S&P 500. 49
4.2 10 synthetic samples for the S&P 500. 50
4.3 Autocorrelations for returns of S&P 500. 51
4.4 Return distributions for S&P 500. 51
4.5 Leverage effect for the S&P 500. 51
4.6 Single example of a synthetic sample for the S&P 500 and VIX. . . . 54
4.7 Serial autocorrelations for synthetic S&P 500 and VIX samples. . . . 55
4.8 Absolute autocorrelations for synthetic S&P 500 and VIX samples. . 56

4.9 Histogram ffor synthetic S&P 500 and VIX samples. 56
4.10 Log histogram for synthetic S&P 500 and VIX samples. 57
4.11 Leverage effect for synthetic S&P 500 and VIX samples. 57
4.12 Serial autocorrelations for synthetic S&P500, Gold and T-Note samples. 59
4.13 Serial autocorrelations for synthetic S&P500, Gold and T-Note samples. 60
4.14 Absolute autocorrelations for synthetic S&P500, Gold and T-Note samples. 61
4.15 Histogram for synthetic S&P500, Gold and T-Note samples. 62
4.16 Log histogram for synthetic S&P500, Gold and T-Note samples. . . . 62
4.17 Leverage effect for synthetic S&P500, Gold and T-Note samples. . . . 63
4.18 Comparison of synthetic OHCL samples and the real data. The synthetic samples are the red, green, orange and blue lines while the rest are for the real data. 64
4.19 PCA plot to show the interactions between the features and compare the synthetic data to the real data 65
4.20 Serial autocorrelations for the three timeseries 66
4.21 Histogram for synthetic samples of the S&P 500, Gold and T-Note where the model has been conditioned on the drawdown of the training data. 67
4.22 Results for the 'train on synthetic, test on real' evaluation method on VIX data. 68

List of Tables

3.1 FTS-GAN: Example implementation of 6 hidden layers which would give a lookback of 127. 40
3.2 ResNet Block. The model is made up of three of these with a residual connection which is added at to the output of each block. 45
3.3 Generator for WGAN-GP 47
3.4 Critic for WGAN-GP 47

4.1 Correlation comparison of the real and fake data for the S&P500 and VIX. 56
4.2 FTS-GAN: Parameter selection for S&P 500 and VIX model 58
4.3 Correlation comparison of the real and fake data for the S&P500, Gold and T-Note. 61
4.4 FTS-GAN: Parameter selection for S&P 500, Gold and T-Note model 61

A.1 Project Summary 82

Chapter 1

Introduction

1.1 Why is there a need for synthetic data?

The requirement for generating synthetic data is growing due to the rapid acceleration into a data driven world. Data has become a powerful commodity that is being utilized in all sorts of industries. With the advancements in machine learning and businesses wanting to leverage data to derive value for their company, synthetic data poses a valuable asset in industries where access to data can be problematic. This is important within the financial industry where data can be limited but there is a desire to apply deep learning techniques, and so to mitigate overfitting, datasets can be expanded by way of synthetic data.

Problems with sourcing data can either be due to there physically not being enough data, such as the release of a new product or new market. It can also be due to privacy as the dataset could contain personal information. Getting access to use this sort of data can be problematic, taking time and hurdles to overcome. This is where synthetic data can be used to help train models when the access to data is limited. The main focus of this report is on the generation of market data; however, finance is not the only potential application area and synthetic data could be used in other applications such as fraud detection. It also opens up the possibility of new potential revenue streams for financial companies. If they have data that is GDPR protected they cannot sell it, but if they could take their dataset and synthesize it so it had similar statistical properties as the original, then this could be sold to other institutions. This is key for large financial organistations which have huge amounts of customer and financial data, but for security reasons the data is heavily protected and unobtainable to the public and to other institutions.

For financial market data the synthetic datasets would allow researchers to be

able to combat overfitting when applying deep neural networks to their data. When building trading strategies it is very important that the researcher limits the amount of exposure to the test data as too much testing can lead to overfitting, which produces good results in backtesting but may mean once the model is deployed in a live environment it will fail. By using synthetic data they can run more tests without the concerns for the potential of overfitting. This could be beneficial in markets where data is limited; some markets can have incomplete or poor past data and so the size of the useable dataset is small.

GANs have been some of the front runners when it comes to generating synthetic data. Although originally developed for use in generating images, researchers have managed to apply them to time series data. In imaging, GANs continue to perform well with a huge range of literature that provides alterations to the classical GAN to fit certain goals, such as converting one style of picture to another [1], or image translation by changing pictures of horses to zebras [2].

Some of the first time series GANs were developed for medical data [3] as this form of data is generally private and so the use of it can be in violation of GDPR and other government regulations. So synthetic data provides a way for third parties to train models on the data without actually exposing any private information to them. More recent work has been to apply GANs to financial data. Financial returns possess common statistical properties that can be learnt and recreated even though the prices move in a seemingly random way [4].

This project proposes Financial Time Series GAN (FTS-GAN), which is a novel GAN that uses a 'stacked' TCN architecture in order to generate multiple linked time series such as the S&P500 and VIX. The advantage of using a GAN is that it requires no additional input to guide the training and so hopefully should be able to learn the known and unknown characteristics of a market and, therefore, outperform parametric models. If the model is successful the generated data should be able to be used in a variety of machine learning tasks and help the models within asset management as well as other branches of finance.

1.2 Aims

The aims of this project are as follows:

- Implement a model from the literature that can generate high quality synthetic financial data and validate the results for a single time series.
- Improve on the current weaknesses of GAN frameworks applied to time series.

- Extend the model to a pair of correlated time series such as the S&P 500 and VIX. This would ensure the model could produce both mean-reverting and non mean-reverting time series.
- Produce a model that can be used for the generation of 3 or more synthetic series.
- Provide a method to allow the 'conditioning' of the model so that scenarios can be built into the synthetic data.

1.3 Structure

The thesis will be structured in the following way. In Chapter 2, the characteristic features of financial time series data are introduced and these will form part of the evaluation method of the synthetic data. Background material is introduced relating to the working and training of neural networks and Convolutional Neural Networks (CNNs) and how this applies to the GAN framework used for data synthesis. Finally, it will include an up to date review of the current literature for GANs as well as how they have been adapted for time series and financial datasets. Chapter 3 introduces FTS-GAN with a detailed explanation on how Temporal Convolution Networks (TCNs) are applied in the case of generating multidimensional time series. This chapter also includes general GAN improvements that can be made as well as a brief explanation of the other models that have been tested. In Chapter 4 the results are presented for the two datasets and the performance is analysed based on the characteristics laid out in Chapter 1. The results for a conditional FTS-GAN as well as the generation of OHCL data are also explored. Chapter 5 summarises the work conducted within this project and discusses the possible directions of future research and potential improvements to the model.

Chapter 2

Background & Related Work

This chapter provides the necessary background to understand financial time series and the statistical properties that arise from the returns. It will also cover the fundamentals of neural networks and introduce variants such as the Convolutional Neural Network (CNN) that will aid in the understanding of this project and the models that are used. The GAN framework will be explained, which will lay the foundations for FTS-GAN. Finally, a literature review will be carried out that covers the inception of the GAN in the use of generative data and how it has been applied to financial time series to date.

2.1 Financial Time Series

Although in the short term the evolution of a price series is a random walk, a lot of research has shown that the returns of most assets possess very similar characteristics or *stylised facts*. These stylised facts are constantly present across different markets all over the globe. These include [4]:

- *Absence of autocorrelations:* As the asset returns evolve as a random walk the returns should be uncorrelated.
- *Heavy tails:* Research has shown that the tails of the returns usually exhibit a Pareto distribution.
- *Gain/loss asymmetry:* The common saying 'the markets take the stairs up and the elevator down' should be present in the returns with upwards movements being less severe then downward trends.
- *Volatility clustering:* Large returns and small returns generally group together to form clusters, and so periods of low volatility lead to more low volatility

and the same for high volatility.

- *Slow decay of autocorrelations in absolute returns:* The absolute returns slowly decay with a power law.
- *Leverage effect:* The measure of volatility of an asset is inversely correlated to the returns.
- *Asymmetry in time scales:* Using coarse measures of volatility better predicts volatility on a fine scale rather than the reverse.

These stylised facts need to be well reproduced in the synthetic data and so when evaluating the generated datasets it is important that these features are present. This is so that machine learning algorithms can be trained on the synthetic data without worry that the algorithm will perform poorly when presented with real data. It is also desirable that the GAN can learn to replicate the properties without having to be fed prior assumptions. This will potentially help to ensure that properties of markets that are unknown can be captured by the model as there is little guidance given by the user to tell the model what to expect.

2.1.1 Statistical Properties

In this section the main stylised facts are covered in more detail and these will form the metrics used when assessing the quality of the synthetic data. Figure 2.1 shows plots that demonstrate the stylised facts. One of the key focuses is the volatility clustering which can be monitored by the decay of the absolute returns. It is important to capture this as markets exhibit heteroskedastic behavior and this is usually poorly replicated by some of the simplistic models.

Uncorrelated Returns

One of the most important statistical properties of financial returns is that there are no autocorrelations in the time series. There should be no linear predictability, i.e. one cannot predict the future price based on the current price. The autocorrelation of the price change is:

$$C(\tau) = \ \mathrm{corr}(r(t, \Delta t), r(t + \tau, \Delta t)) \approx 0 \tag{2.1}$$

The value of $C(\tau)$ rapidly tends to zero after a few minutes and so one can assume that it is essentially zero [4]. From this one can make the assumption that the markets are relatively efficient, and this is usually used as evidence to support the

efficient market hypothesis (EMH) [5]. This is important to get right, as if there are correlations, any model that is trained on the data will be able to make predictions of the future price from the past which will allow trading strategies to seem profitable when they are not.

Heavy Tails

Fat tails, or heavy tails, have been studied by many authors (see for example Mandelbrot [6] or Gopikrishnan *et al* [7]), who have shown that the tails decay as a power-law and so the distribution of the returns is not normal like a lot of parametric models assume. A series has a power-law decay if it has the following:

$$P(r) =\propto r^{-\alpha} \tag{2.2}$$

There is no consensus on the origin of the heavy tails; however, one of the most plausible is volatility clustering which is discussed in the next section. This is important to capture as it means that in the synthetic data the volatility should change over time as markets are constantly evolving.

Volatility Clustering

Noted by Mandelbrot in 1963 [6], volatility clustering is that 'large changes tend to be followed by large changes of either sign and small changes tend to be followed by small changes'. It can be described by the following:

$$C_2(\tau) = \ \text{corr}(|r(t+\tau, \Delta t|^2, |r(t, \Delta t)|^2) \tag{2.3}$$

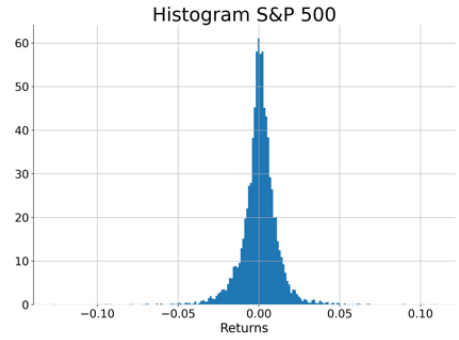
As there is a decay in the autocorrelations of the absolute returns it shows that there is long range temporal dependence in the time series.

Leverage Effect

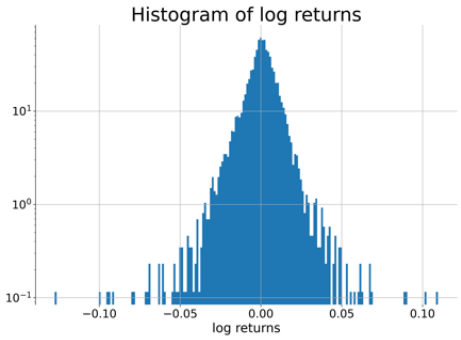
The leverage effect is the observed phenomenon that describes the interaction between the price and volatility of the data:

$$L(\tau) = \ \text{corr}(|r(t+\tau, \Delta t)|^2, r(t, \Delta t)) \tag{2.4}$$

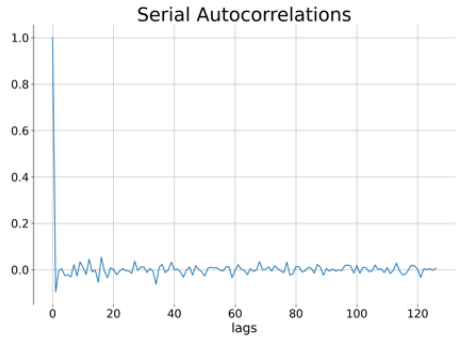
For indexes such as the S&P 500 there is generally a negative correlation between the returns and the volatility, which is intuitive as when markets start to drop investors panic, causing large and rapid movements of assets.



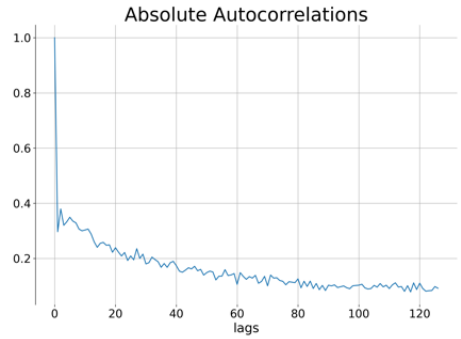
(a) Histogram of returns.



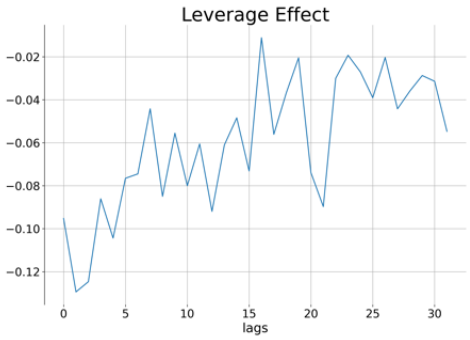
(b) Histogram of log returns.



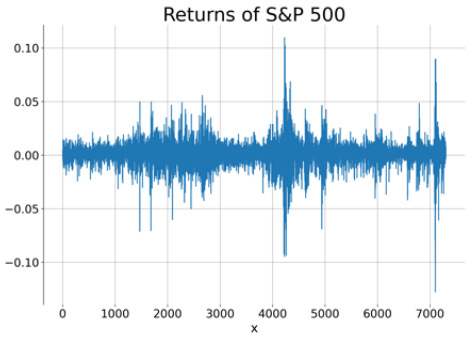
(c) Serial autocorrelations.



(d) Autocorrelations of the absolute returns.



(e) Leverage effect.



(f) Series returns.

Figure 2.1: Stylised facts of financial markets. The plots here are for the S&P 500 over the period 1992 to 2020

2.2 Background

2.2.1 Neural Networks

The Basics

Neural networks are named as such as they try to replicate what occurs in the brain's neurons. They work by taking an input and passing it through a linear layer and then passing it through a non-linear activation function. This is the make up of a single layer and forms what is known as a single-layer perceptron. For a vector of inputs, $(x_1, \ldots, x_D)^\intercal$, a single layer perceptron would undertake the following transformation on the input:

$$z_k = h(\sum_{i=1}^{D} w_{ki} x_i + w_{k0}) \tag{2.5}$$

where $k = 1, \ldots, M$. The w_{ki} refers to the weights and w_{j0}, the biases. The biases are similar to an intercept in linear regression. They are added at each layer to shift the activation function. $h(\cdot)$ represents an activation function, such as a ReLU or Sigmoid. These are a non-linear functions that can be differentiated. Multiple single layers can be stacked together to produce a full architecture with the following function:

$$y_j(\boldsymbol{x}, \boldsymbol{w}) = \sigma \left(\sum_{k=1}^{M} w_{jk}^{(2)} h \left(\sum_{i=1}^{D} w_{ki}^{(1)} x_i + w_{k0}^{(1)} \right) + w_{j0}^{(2)} \right) \tag{2.6}$$

where $h(\cdot)$ and $\sigma(\cdot)$ represent activation functions and $\boldsymbol{w}$ represents a matrix of the biases for the two layers. The superscripts (1) and (2) represent the layer numbers. Figure 2.2 shows an example of a fully connected network with one input layer, one hidden layer and one output layer.

Parameter Setting & Backpropagation

The parameters are found by training the model. For the parameters θ in a network the values should be chosen so that they minimise the loss function.

$$\min_{\theta} J(\theta) \tag{2.7}$$

For regression one would want to minimise the sum of the squares:

$$J^{(i)}(\theta) = \frac{1}{2} \sum_{i=1}^{N} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{2.8}$$

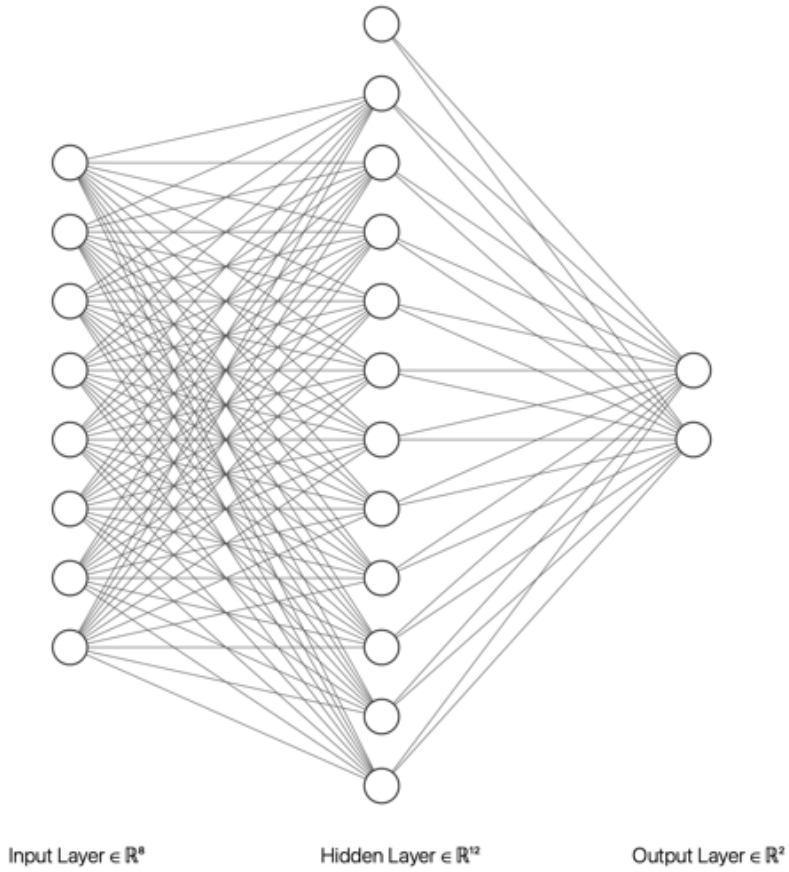


Figure 2.2: A fully connected neural network.

while for a classification task one could use binary cross-entropy:

$$J^{(i)}(\theta) \ = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta)(x^{(i)}))] \tag{2.9}$$

where m is the number of classes. The smallest value of $J(\theta)$ will occur when the gradient is $\Delta J(\theta) = 0$, due to it being a smooth continuous function w.r.t θ. However, real data is not so easy and there is a non-linear dependence so that there are multiple local minima that occur for different combinations of the parameters. The minimum that one is interested in is the *global* minimum, which is the lowest value of $J(\theta)$ for a given set of parameters. Unfortunately, an analytical solution to the loss function is not possible and so a numerical method must be applied. The popular way is to apply a gradient descent algorithm. This involves starting with randomly intialised weights and then updating them based on the gradient of the loss function:

$$\theta := \theta - \alpha\nabla_\theta J(\theta) \tag{2.10}$$

where α is the learning rate. This changes the rate at which the parameters are updated.

To implement the gradient descent one must calculate the gradient of the neural network and this can be calculated by the backpropagation algorithm. A multi-layer fully connected network and its quantities can be defined as follows:

$$a_j^l = h(z_j^l) = h(\sum w_{jk}^l a_k^{l-1} + b_j^l) \tag{2.11}$$

where a_k^{l-1} is the activation of a unit in the previous layer and w_{jk}^l is the weight of the connection. The error propagation and bias will be ignored, b_j^l, for this derivation. δ_j^l is introduced, which is the error in a layer, l, and neuron, j, defined as:

$$\delta_j^l \equiv \frac{\partial J}{\partial z_j^l} \tag{2.12}$$

For calculating the backpropagation one must run a forward propagation on the network to calculate the values of all the activation units. One first starts by calculating the cost change in the outer layer L. By applying the chain rule one can get:

$$\frac{\partial J}{\partial z_j^L} = \frac{\partial J}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \tag{2.13}$$

To calculate the propagation of the error in the output unit, the first derivative of the lost function is taken. In the case of the sum of squares, the following is produced:

$$\frac{\partial J}{\partial a_j^L} = (a_j^L - y_j) \tag{2.14}$$

So that the error can move backwards in the network, the relationship between δ_j^l and δ_j^{l+1} must be found. Using the chain rule for partial derivatives one can get to the following:

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} = \sum_k \delta_k^{l+1} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{2.15}$$

which gives the backpropagation formula:

$$\delta_j^l = \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \cdot h'(z_j^l) \tag{2.16}$$

The errors can be related to the weights and so the following relationship is produced:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.17}$$

An overview of the algorithm can be seen in Algorithm 1, where a 'minibatch' is a

Algorithm 1: Backpropagation Algorithm

1. Apply the input from a minibatch and forward pass through the network updating the inputs z and activation units a.
2. Evaluate δ for the output units.
3. Back propagate through the network and find the δs for each layer.
4. Use equation 2.17 to calculate the derivatives.
5. Repeat for each sample in the minibatch.
6. Update the weights by the following $-\eta \cdot \nabla_w J$.
7. Repeat over every minibatch.

subset of the dataset, which the gradient and updated weights are calculated over.

Activation nodes

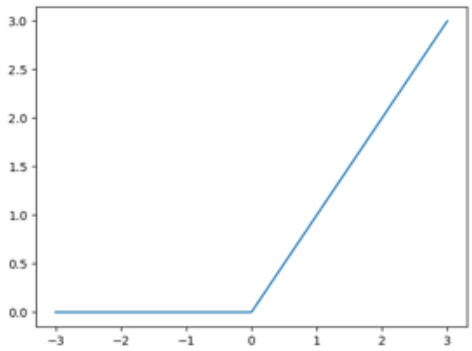
There are many different activation nodes that can be implemented into a neural network; however, the focus will be on ReLU, PReLU and LeakyReLU as these are common within the architectures used for GANs[1]. The simplest one is the ReLU or Rectified Linear Unit. It was proposed by Hinton and Nair [9], and has become a popular activation function for deep learning applications. It is given by the following:

$$y = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{else} \end{cases} \tag{2.18}$$

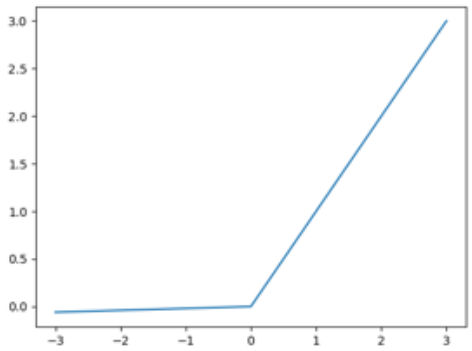
However, the ReLU can suffer from vanishing gradients due to the function being 0 when $x < 0$. This means that some gradients will not be possible and so activation nodes can go to 0 meaning that the learning becomes poor. To solve the problem with vanishing gradient the LeakyReLU was proposed by Maas *et al* [10]. This has the following equation:

$$y = ax + x = \begin{cases} x & \text{if } x > 0 \\ ax & x \leq 0 \end{cases} \tag{2.19}$$

[1]For an overview of activation functions used in neural networks see Nwankpa *et al* [8]



(a) ReLU



(b) Leaky ReLU

One can also control the parameter of the slope when $x < 0$ by using a PReLU. This allows the tuning of the slope for each individual feature. It has an equation given by:

$$y_i = \begin{cases} x_i & \text{if } x_i > 0 \\ a_i x_i & \text{if } x_i \leq 0 \end{cases} \tag{2.20}$$

Nwankpa find that PReLUs outperform ReLU on large scale image tasks while He *et al* use them in image classification and find they manage to 'surpass human-level performance' [11].

Batch Normalisation

The training of deep neural networks is challenging as the parameters in each layer are changing on every batch, and so the update procedure is chasing a moving target. This means that convergence can become slow. Batch normalization is a standardisation technique that is employed to help with updating problems. The output of each layer is standardized which means that the next layer in the network does not have to change its assumptions about its input distribution. The method is shown in Algorithm 2.

Spectral Normalisation

Spectral normalisation is a technique for normalising the weights in a neural network. It is used in GAN training on the discriminator to stabalise the training process [13]. The advantage of spectral normalization over other stabalising techniques is that it is not as computationally expensive while producing equal or better results when applied to different GANs.

Algorithm 2: Batch Normalisation algorithm from Ioffe and Szegedy [12]

input : Values of x over a mini-batch $\mathcal{B} = \{x_{1...m}\}$
Parameters to be learned: γ, β
output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$ Mini-batch mean

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$ Mini-Batch variance

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$ Mini-Batch variance

$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$$ Mini-Batch variance

Optimizers

Basic gradient descent would not be suitable for deep neural networks; as they become more complex the convergence times increase as well as the potential for the model not to converge. Therefore, optimizers that use more information about the gradient can be implemented, these include ADAM [14], ADAMW [15] and RMSProp [16]. ADAM and RMSProp are the two most common optimisers used in GAN frameworks, while ADAMW is a variant to the ADAM optimiser that has been shown to give better performance.

- *ADAM:* The ADAM optimizer combines the benefits of RMSProp and AdaGrad [17]. RMSProp adapts its learning rate based on the average of the recent gradients for each parameter. AdaGrad has the advantage of having a learning rate that works well on sparse gradients. ADAM adapts its gradient on squared gradients, v_t, and also the past gradients, m_t. This is similar to momentum. It has two parameters β_1 and β_2 that control the decay rates of the exponential averaging of the moments. The decaying averages are defined as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

  As the vectors m_t and v_t are initialised with 0's, the authors realised that they had a bias towards 0. To mitigate this problem the following correction is

applied to the moments:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

The same update rule for the parameters that is used in RMSprop is used for ADAM:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$$

- *ADAMW:* This is an improved version of the ADAM optimizer where the weight decay is decoupled from the optimization steps. It can produce large performance benefits over the original ADAM's performance. The two adjustments are the addition of L_2 regularisation and the decoupled weight. In the original method g_t is given by $\nabla f_{t+1}(\theta_t)$ but for ADAMW it is now $\nabla f_{t+1}(\theta_t) + \lambda\theta_t$. The decoupled weight is applied to the update rule to give:

$$\theta_{t+1} = \theta_t - \eta_{t+1}\left(\frac{\alpha\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} + \lambda\theta_{t-1}\right)$$

- *RMSProp:* This is a method that uses an adaptive learning rate. It is an unpublished method conceived by Geoff Hinton that builds upon gradient descent. It overcomes the limitations of AdaGrad by using a decaying moving average, which means that earlier gradients are forgotten and so only the more recent ones are used in the calculation [18]. The update rule is given by the following:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t+\epsilon}}g_t$$

A suggested learning rate is $\eta = 0.001$, while γ should be 0.9.

2.2.2 Convolutional Neural Networks

Convolutional Neural Nets (CNNs) are a type of deep neural network that has been heavily applied to image analysis. They are made up of a sequence of convolutions that transform an input to the output. Generally a CNN will be made up of a mixture of different layers, such as discrete convolutions, activation functions (ReLU) and pooling.

Discrete Convolutions

A convolution layer is a linear operation in which the input is multiplied by weights, known as a kernel. The kernel usually has a size that is smaller than an input and applies a dot product to each section of it. As it was originally pioneered for images, 2d convolutions are very common in CNNs. One of the key features of convolutions is the weight sharing. Across all the neurons in each layer the same kernel is used which provides the following characteristics [19]:

1. Translational invariance, meaning that the model can detect local feature patterns regardless of the position on the input.

2. Ability to learn spatial hierarchies by down sampling and pooling, meaning that the model learns a large field of view.

3. Increases the efficiency of the model as the same layers share the same kernel and biases, which massively reduces the number of parameters that must be learnt. This means that training will be faster.

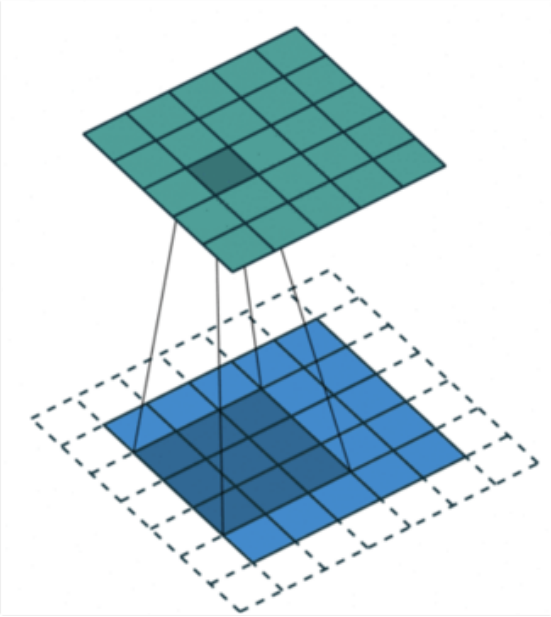


Figure 2.4: Diagram of a 2D convolution.

Figure 2.4 shows the blue input being mapped to a green feature space. The kernel moves across the input to produce this. The stride and dilation are two parameters that can be set. The stride sets how many elements the kernel should move for each filter. The dilation sets the size of the receptive view by inserting blank holes into the kernel to 'inflate' it. This allows an exponentially increasing kernel size without 'loss of resolution or coverage' [20] (See Figure 2.5). Padding is also

usually applied, with zero padding being most common. By applying a convolution the size of the feature map is smaller than the input, so to overcome this problem rows and columns of zeros are added to the input tensor so that the outputs keep the same in-plane dimensions. These are the dotted squares in Figure 2.4. If no padding were to be used, each feature map would be smaller and smaller, so in order to use models with large depths, padding must be applied.

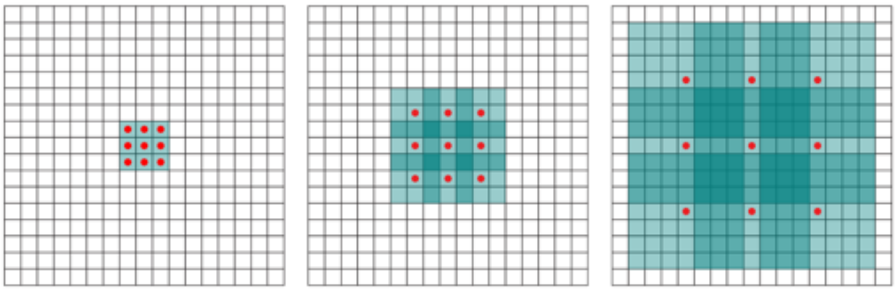


Figure 2.5: Dilated convolutions with increasing dilation size (1, 2, 4). The number of parameters for each layer is the same but the amount of coverage they are gaining is exponentially larger[20].

For time series analysis 1d convolutions are mainly used. The advantage of convolutions is that they preserve the ordering of the data and so it is intuitive to use them for time series data as each time step will be more dependent on nearby values. As shown in a later section it can be extended so that it only relies on previous data points, so that time steps are not conditioned on the future. Figure 2.6 shows an example of a 1D convolution; the kernel size is 3 and the stride is 1. The filter moves along and dot products itself with the values in the sequence.

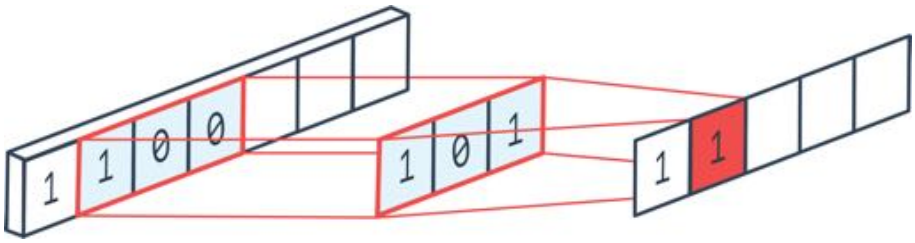


Figure 2.6: Diagram of a 1D convolution [21].

2.2.3 Temporal Convolution Network

Temporal convolution networks are a relatively new design of neural network architecture that have been seen to provide better performance over recurrent neural networks (RNNs). They are made up of a series of causal convolutions and are used in the modeling of time-series that possess long range dependence. They have the following advantages for modelling time series [22]:

- *Stable Gradients:* TCNs avoid the problem of exploding gradients due to the backpropagation being in a different direction to the temporal direction.
- *Flexible receptive field size:* The receptive field size of a TCN can be changed in many ways, such as increasing the filter size, increasing the dilation factor and stacking more layers, which allows the user to set the lookback while also managing their memory usage.
- *Parallelism:* TCNs do not have to wait for the calculation of previous time steps but can do all the transformations at the same time. This means that a long time series can be processed simultaneously, while RNNs would have to sequentially calculate the time steps.
- *Low memory requirement:* As they are using convolutions TCNs benefit from the lower memory requirement due to the sharing of weights across filters.
- *Variable length inputs:* As TCNs just slide the kernel over a 1d sequence any input length can be applied, unlike a multi layer perceptron where the input size is fixed to the model.

The implementation of TCNs will be explored further in Section 3.1, where it is shown how they are applied to FTS-GAN.

2.2.4 Generative Adversarial Network

The aim of a GAN is to learn the distribution behind a set of data, $\mathbb{P}_{\text{data}}$ and then be able to produce new data $\mathbb{P}_{\text{model}}$ from a random prior $\mathbb{P}_z \sim N(0, 1)$ that matches as closely as possible the original data.

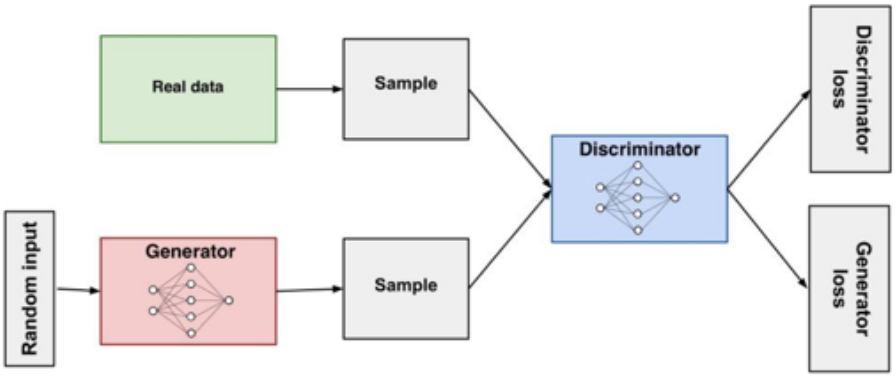


Figure 2.7: Diagram of the setup of a standard GAN [23].

The basic principle of Generative Adversarial Networks is that there are two neural networks that are in competition against each other in a 'game'. The original

GAN proposed by Goodfellow [24] is made up of a generator (G), where G is a neural network that represents a differentiable function with parameters θ, and a discriminator (D) that outputs a single value that is the probability of whether it thinks the data it has been fed comes from the real data rather than $\mathbb{P}_z$. This will be referred to as standard GAN or SGAN further on. The aim of the generator is to produce samples from random noise that can fool the discriminator into believing they are real. On the other hand, the discriminator is fed a mixture of real and fake data and it must identify what is real and fake.

At the start of the process both neural networks are untrained and so the samples produced by the generator are obviously fake. If the example of a GAN that produces images is used, the generator would start with an unrecognisable image. However, the untrained discriminator will perform poorly at deciding what is real or fake. By using a suitable loss function the two neural networks can be trained simultaneously so that they both improve together. As the generator produces more accurate images the discriminator must also improve so that it can tell the images apart, and vice versa. Both of these neural networks have a separate set of weights, θ, for the generator and γ for the discriminator which are independently learned. This is done via a chosen loss function. The overall loss function for SGAN is the minimax loss function:

$$\min_{\theta} \max_{\gamma} V(G_\theta, F_\gamma) = E_{x \sim p_{\text{data}}}[\log D_\gamma(x)] + E_{z \sim p_z}[\log(1 - D_\gamma(G_\theta(z)))] \tag{2.24}$$

This comes from a distance measure between probability distributions and is derived from the cross entropy between the generated and real distributions. For the generator it is only minimising $\log(1 - D_\gamma(G_\theta(z)))$ as it cannot affect $\log D_\gamma(x)$. In Section 2.2.4 another function will be discussed that can aid in the training of a GAN. Figure 2.8 shows the evolution of the training process. The green line represents the distribution produced by the generator and the dotted line is the discriminator. As training continuous, the generator's distribution becomes closer to that of the real distribution while the discriminator becomes more stable.

An optimizer is applied such as the ones suggested in Section 2.2.1 to implement iteratively the stochastic gradient descent. There is an option to run the discriminator k times for every update of the generator, as when using certain loss functions, such as Wasserstein, convergence is much better when the discriminator has more training than the generator.

GENERATOR The generator produces samples from noise which is usually taken from a Gaussian distribution. The samples are then evaluated by the discriminator

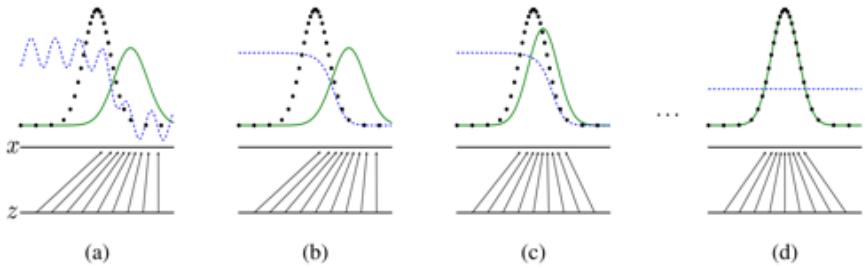


Figure 2.8: The evolution of GAN training [24].

and assigned a loss which is passed back to the generator so that it can update itself accordingly.

Discriminator For the SGAN the discriminator is just a classifier. It is fed a combination of real data and the synthetic data produced by the generator and tries to establish which is real or fake. The discriminator produces two losses, one for training itself and the other for training the generator via backpropagation.

Conditional GAN

Mirza and Osindero [25] proposed the Conditional GAN. The architecture is the same as the original GAN but introduces a new input into both the generator and discriminator. This could be a class label or some other sort of information. For example, on the MINST dataset, class labels would allow the user to choose which number they would like to generate. The standard way to introduce the conditioning is via an embedding layer; however, other methods such as concatenation can be used. The conditioned GANs generally produce higher quality samples as the guidance from the labels ensures that the right mode is selected. Section 3.2 will cover how one could potentially apply conditioning to FTS-GAN and the difficulties posed due to the requirement of a flexible input size.

Loss Functions

The loss function is an important part of the GAN as it impacts the way in which the error of the model is measured and can have an effect on the convergence of a GAN. In this section the theoretical analysis of the loss function and a suggestion for another loss function is explored.

Definition 2.2.1 (Kullback-Leibler Divergence) *The Kullback-Leibler divergence can be uses to measure how different two probability distributions are from*

each other. It is given by:

$$D(P||Q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i)) \tag{2.25}$$

Definition 2.2.2 (Jensen-Shannon Divergence) *The Jensen-Shannon divergence is used for measuring the statistical similarity between two probability distributions. It is a smoothed version of the Kullback-Leibler divergence $D(P||Q)$ and is given by:*

$$JSD(P||Q) = \frac{1}{2}D(P||M) + \frac{1}{2}D(Q||M) \tag{2.26}$$

where $M = \frac{1}{2}(P + Q)$

Training a GAN that uses the minimax loss function (Equation 2.24) is the same as minimisng the Jensen-Shannon divergence (Definition 2.2.2). The discriminator is trying to maximise the function V, with respect to the parameters γ. The optimal discriminator will be given by:

$$D^*(\boldsymbol{x}) = \frac{p_{\text{data}(x)}}{p_{\text{data}(x)} + p_{\text{model}}} \tag{2.27}$$

By plugging this into the objective we can show that:

$$V(G, D^*) = 2 \text{ JSD}(p_{\text{data}}||p_{\text{model}}) - \ln 4 \tag{2.28}$$

This means that for the optimal generator and discriminator the best the objective function can be is $-\ln 4$. However, for a lot of cases the distributions are not continuous due to disjoint supports as p_{data} is concentrated on a low dimensional manifold [26]. This means that if the discriminator is trained to convergence its error will be 0 and so the JS Divergence will be maximised [27]. This can lead to GAN instability and non-convergence. Arjovsky *et al* [28] introduced a proposal for a GAN that uses the Wasserstein distance for training the neural networks. This is the aptly named Wasserstein GAN (WGAN). The discriminator is replaced with a critic that, rather than classify whether the data is real or fake, scores how real or fake an image is.

Wasserstein Distance

Due to the Wasserstein objective function being one of the more popular loss functions for training GANs, and used for some of the initial models tested for this project, the theory behind it is presented for completeness. It is an alternative way

of training a GAN that aims to provide better convergence. It comes from a mathematical argument that the GAN should be aiming to reduce the distance between the real and generated data. The Wassertein objective function is based on the *earth-mover* distance rather than the Jensen-Shannon Divergence, which can be informally explained as the number of buckets of 'earth' multiplied by the distance to transform one distribution to another. The Wassertstein-1 distance can be written as such:

$$W(p,q) = \inf_{\gamma \in \prod(p,q)} E_{(x,y)\in\gamma} \|x - y\| \tag{2.29}$$

The equation shows how much 'earth' must be moved from x to y. To be valid the constraints $\sum_y \prod(x,y) = q(x)$ and $\sum_x \prod(x,y) = q(y)$ must hold.

2.3 Related Work

Finding models that can explain the distribution of data is a big part of financial analysis. It gives the user an understanding of what is happening and allows one to make predictions on the data. However, data can contain complex statistical properties and very rarely fits to simple models. For financial time series there are usually two main parametric model types: stochastic process models and agent-based models. Stochastic process models, such as GARCH, model the time steps as random variables with dependence on the past. Agent based models try to explain the interactions of agents within a market and make predictions based on this. However, this usually comes with a lot of assumptions and parameters that can be difficult to calibrate. The traditional methods for simulating price data using parametric models that rely on stochastic processes tend to be rather poor approximations of reality. In order to be useful they must be simple but that means they only capture some of the key aspects of a very intricate market setup. In Monte Carlo simulations when generating risk for multiple factors it is common to model independently and then apply the dependence by ensuring the brownian motion of each factor is correlated to some degree. GARCH models which are commonly used as benchmarks by researches checking financial Generative Adversarial Networks (GANs) usually struggle to properly model some of the more complex financial properties such as volatility clustering or the leverage effect [29].

Therefore, more recent methods for reproducing market data are non-parametric models where one tries to learn the distributions directly from a dataset. This has been pioneered in the GAN as well as other types of models such as Restricted Boltzmann Machine (RBM), Deep Boltzmann Machines (DBMs) and Deep Belief Networks (DBNs). Work by Kondratyev and Schwarz [30] introduce the Bernoulli

RBM to model currency pairs and find a strong potential use method for use in VaR engines. One of the big problems, especially with GANs related to finance, is the lack of open source code available to check some of the implementations and generate benchmarks, which means it is hard to find a gold standard for the best way to generate market data.

The first iteration of the GAN was proposed by Goodfellow *et al* [24] and found use within image generation. This includes work such as generating MINST numbers to changing photos to be styled so that they look like they were painted by famous artists [1]. The advancements of the image quality from GANs between 2014 to 2019 is seen in Figure 2.9, where the initial iterations were very poor, up to more modern implementations that are becoming indistinguishable to the human eye. In medicine, researchers have applied GAN to health record datasets. Usually health records contain personal information and so access to these datasets can be tricky for researchers due to privacy concerns. Choi *et al* [31] propose the 'medGAN' algorithm that allows a small sample of electronic health record (EHR) data to be increased by the addition of synthetic data that achieves sufficient performance when research such as predictive modelling is being undertaken. As well as the aforementioned use cases GANs have been applied to a range of data, not limited to: audio [32] & music [33] [34], DNA Sequences [35] [36] and text generation [37].

Figure 2.9: Advancements in image synthesis between 2014 to 2017 [38]

More recently the use of GANs for the generation of *time series* data has been explored both in the medical world [3] as well as finance [39][2]. Additions and improvements have been made to GANs which lend themselves well to time series. At the moment the obvious uses for GANs within finance are for generating synthetic data, as the financial returns all share similar characteristics [4] which can be learnt by a model. Therefore, it is not surprising that this is currently where most of the literature is focused. GANs can help overcome problems such as data scarcity, data costs and reduce overfitting when running backtesting on trading algorithms. Koshiyama *et al* [40] show the use of conditional GANs (cGAN) in the calibration

[2]For an extensive review of the usage of GANs in finance see Eckerli and Osterrieder [23]

of trading strategies. The use of conditional GANs opens up the opportunity to run specific scenarios allowing for investors to stress test their algorithms [40]. As well as generating time series other possible uses for GANs have been found other possible uses within finance, such as fraud detection (either credit fraud [41] or market manipulation [42]); portfolio management and optimisation [43] [44] [45] [46]; training and tuning of trading models [40] and also in market prediction and time series forecasting [47] [48]. Outside of finance, anomaly detection in time series has been explored with the proposal of 'TadGAN' [49]. This model performs well, producing better performance then a range of baseline models such as LSTM.

The original GAN has been improved by altering the loss function to use the Wasserstein loss function [28]. This means that one is measuring *how* real or fake an image is rather than trying to classify if they are real or fake. The benefit of this is that training becomes more stable and not as sensitive to the choice of hyperparameters. A gradient penalty term can also be added to the loss function to improve training [50]. This applies a term to the loss function to enforce the Lipschitz constraint. This algorithm has been adapted for use in time-series data as shown by de Meer Pardo [27]. They also provide a method for jointly generating VIX and S&P 500 by the use of a Relativistic average GAN (RaGAN). This is a conditional method that uses a 'base' time series of S&P 500 data to produce an 'associated' VIX dataset. The rationale behind this method over a purely multidimensional GAN is that it reduces the chance of suffering from mode collapse. The RaGAN was proposed by Jolicoeur-Martineau [51] and it adjusts the standard GAN or (SGAN) for a probabilistic loss function. The problem with the WGAN is it fails to properly capture the absolute autocorrelations, and De Meer states in his paper that 'the architectures listed perform well on experiments of Appendix A but fail in real datasets' and that more complex proprietary models are used for his results. This can be one of the problems within the research of financial GANs with many companies: research not disclosing a proper working implementation. Wiese *et al* [52] suggests the 'QuantGAN' framework that utilises Temporal Convolution Networks (TCNs) in order to capture the long range dependencies. FinGAN [29] utilise a multi-layer perceptron (MLP) and convolutional neural network (CNN) together for their generator. Another proposed algorithm for time-series data is the TimeGAN [53]. This method uses an adapted GAN framework that includes four neural networks.

Apart from de Meer Pardo's work there are few results on the generation of multiple financial series such as S&P 500 and VIX and so knowing which model architectures would scale well to this application is tricky. Unfortunately, due to de Meer's actual model being proprietary his work cannot be expanded on. There is

also concern that scaling a framework to do completely random multidimensional generation could lead to mode collapse (defined in Section 3.3). Below, a deeper explanation is given for four models that are designed for time series data. The first two, QuantGAN and FinGAN are both for single time series data while the latter two, TimeGAN and MTSS-GAN, have been used for some multivariate series; however, there are no results to show how they would reproduce the statistical properties of the financial data that are desired.

Quant GAN

The author of this paper chooses to use TCNs over other architectures such as Long short-term memory (LSTM) or other recurrent neural networks (RNN) as TCNs have been found to produce very good results, outperforming some of the more traditional architectures when trying to model long-range dependencies [22]. At the moment the model is only implemented for a single time series and while this was done well and manages to capture the required statistical properties as outlined in Section 2.1 it would be useful to have the ability to generate correlated time series. Due to the documented success of this model, this model is used as a starting point to build FTS-GAN.

FinGAN

This paper covers the use of convolutional neural networks (CNNs) and multi-layer perceptrons (MLPs) in GANS for the modelling of financial data. They test on three models: one with just CNN, one with just MLP and the final model which has an architecture which is both the CNN and MLP. Their findings show that using the combination of the MLPs and the CNNs gives the best performance. The model uses the same GAN setup as in the original Goodfellow GAN. They find that adjusting the architecture of the generator has a lot more sway on the quality of the samples. They manage to produce good statistical properties for their generated data but due to the use of MLPs they must hard code the length of the synthetic series that they would like to output, which also leads to lots of parameters. This differs from Quant GAN where the training input and output dimensions can be independent. This method has also only been applied to a single time series.

TimeGAN

This GAN was originally developed for medical time series data and adds to the original GAN by the addition of a stepwise supervised loss. It is made up of 4

separate components, a generator and discriminator such as in a traditional GAN, but also introduces an embedding function and recovery function. The addition of these two networks allows the model to combine supervised and unsupervised learning which allows more control over the dynamics of the network. However, this paper has no information on performance for reproducing the statistical properties of financial data.

MTSS-GAN

This GAN is based off stacked GANs and is aimed at producing multivariate time series where the user has control over the relationships within the data so that they can generate custom scenarios. The author shows that they can produce good results in replicating the dependencies of the different features but there is little done on production of real time series and ensuring that the returns match the statistical properties of the market.

Evaluation Metrics

Unlike other neural networks, GANs do not possess any overall objective function to minimise as they are trying to constantly find an equilibrium between the generator and the discriminator. This means that evaluating models is not necessarily straightforward and ensuring that the generated time series is actually indistinguishable from the real data can be difficult. For an overview of some of the suggested metrics for evaluation see Borji [54]. For financial data there is the benefit of having certain characteristics that would like to be achieved and so using these one can compare the sample data's statistical properties with the real data and use that to produce a closeness score as done by Wiese *et al* [52]. Other proposed metrics are 'nearest neighbour adversarial accuracy' [55] that compares the points between a target and source distribution using a nearest neighbour method, or geometry score proposed by Khrulkov and Oseledets [56] that compares the geometric properties of the data to produce a quality score. For time series data other methods make use of another independent discriminator to check the data. Yoon *et al* [53] introduce a 2-layer LSTM to predict whether their samples are real or fake and this can be used to assess the quality of the samples and so how well the model performs. A slightly different approach is that of Esteban *et al* [3] who introduce a method of 'train on synthetic, test on real' (TSTR). De Meer Pardo implements this for financial VIX by way of a ResNet. By training a classifier on the synthetic data one can see how well it would compete if one was instead to use real test data. Ideally

the use of synthetic data should have no effect on the classification ability of the ResNet. For multivariate data samples one can use visualisation to ensure that the correlations between the variables are correct by using a data visualisation method such as t-SNE [57]. However, this only provides a qualitative result.

Chapter 3

Methodology

This chapter covers the methods applied in this dissertation. The network architecture of FTS-GAN as well as the implementation are explained and how the parameter selection and evaluation of synthetic samples is carried out. A brief explanation of some of the other methods trialed will be given in the final section.

FTS-GAN uses the GAN framework as proposed by Goodfellow which is covered in detail in Section 2.2.4. Although there have been adaptations to the underlying framework for time series generation some of the better performing models still use the basic design but alter either the loss function or the neural networks used for the generator and discriminator. The novel part of FTS-GAN is in the choice of the neural network. The networks are based on Temporal Convolution Nets (TCNs) and by 'stacking' them on top of each other one can generate multiple correlated time series.

3.1 FTS-GAN Network Architecture

FTS-GAN is an extension to Quant GAN, which in turn was inspired by a different generative framework, Wavenet, designed to generate synthetic human speech [58]. However, FTS-GAN allows for the generation of multiple correlated time series. The core component of Wavenet is the block made up of the dilated convolution, 1x1 convolution and the addition of the residual (See Figure 3.1). For each of these layers the value from the 1x1 convolution is also used as the 'skip'. These 'skips' are added up for each of the layers and is what is passed on for the final output. The inclusion of all these components such as causal convolution, dilation and the residuals means that the model is sparsely connected and so it generates a long lookback, while ensuring no data leakage occurs, while the residual connection means

that information is preserved across the layers. Wavenets form the foundation of Temporal Convolution Nets and so this architecture will underpin FTS-GAN. How it works and is used within this project will be covered in the next part.

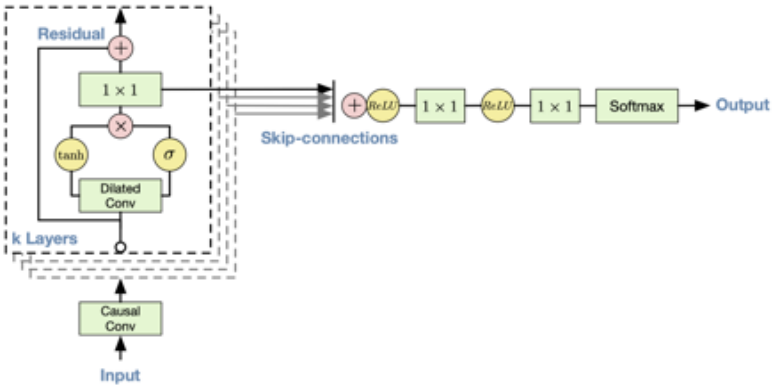


Figure 3.1: Wavenet architecture [58].

The generator of FTS-GAN is trying to generate new values that are dependent on the input noise. This means that the new returns will have a temporal dependence and so capture the statistical properties such as the absolute correlations in the returns. The discriminator is not trying to directly create new values but by applying the same architecture a value can be produced that distinguishes between real or fake based on the time series that is being tested. The joint probability of the new returns $\boldsymbol{x}$ can be factorised as a product of conditional probabilities.

$$p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t | x_1, \ldots, x_{t-1}) \tag{3.1}$$

These conditional probabilities can be modelled by TCNs. FTS-GAN's generator and discriminator both use a similar network architecture that is based on a TCN framework. The design of TCNs is very good for capturing the temporal dependence for FTS-GAN, as well as allowing the continuous generation of data, which gives flexibility in the output dimensions of the synthesised data. The key element of a TCN is the dilated causal convolutions. Causal convolutions are a method of convolution where the size of the convoluted output is kept the same as the input by the addition of padding with 0s only to the left hand side. The padding is of size (Kernel $-$ 1). The addition of the padding to only the left ensures that there is no data leakage of information from the future into data points that are in the past. This means that the prediction of the next time step at t does not include any information from a future time step $x_{t+1}, x_{t+2}, \ldots, x_T$. As all the inputs are known while training this means the convolutions for each node can be calculated in

parallel, which is advantageous over traditional time sequence modelling networks such as LSTMs.

The next addition is to use dilations as one goes deeper into the network. A standard causal convolution's lookback scales linearly with the depth of the network. In order to have a large lookback, many layers will be required which increases memory usage and computational cost. Dilated convolution involves skipping a

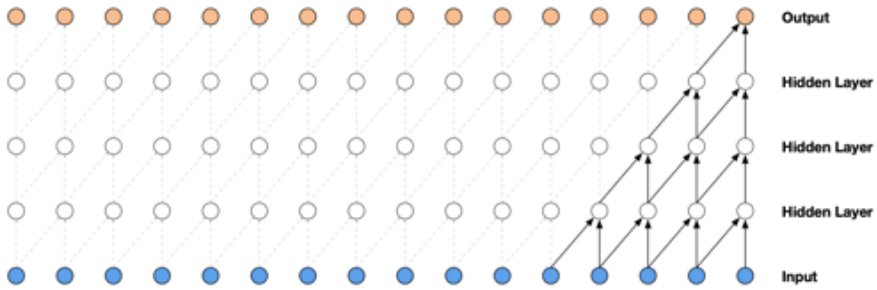


Figure 3.2: A stack of causal convolutions [58].

certain number of steps, which gives the node a larger but coarser lookback. For TCNs it is common for the dilation to be doubled for every layer so that receptive field size grows exponentially with the depth of the network [59]. This is shown in Figure 3.3 where the dilation factor doubles for each layer. The receptive field size has gone from 5 in Figure 3.2 to 16 in Figure 3.3. This is important for FTS-GAN as ideally the lookback will be as long as possible, but at the same time one does not want too many parameters at it becomes computationally expensive as well as requiring a large amount of memory. The receptive field size can be calculated by

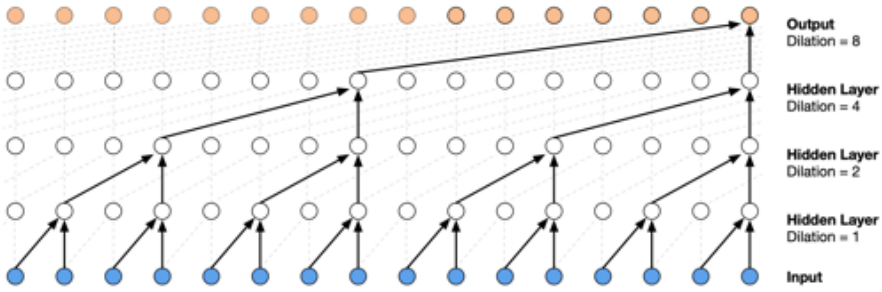


Figure 3.3: A stack of dilated causal convolutions, with dilation increasing by factor of 2 each layer [58].

the following:

$$R_{\text{field}} = 1 + 2(K_{\text{size}} - 1)N_{\text{stack}} \cdot \sum_i d_i \tag{3.2}$$

One can go further and also apply an external conditioning series to Equation 3.1

so that for each layer of the TCN and external feature can be applied:

$$p(\boldsymbol{x}|\boldsymbol{h}) = \prod p(x_i|x_1, \ldots, x_{i-1}, \boldsymbol{h}) \tag{3.3}$$

For FTS-GAN correlated times series generation is the goal and so some form of conditioning will be required between the series to learn this. The conditioning can be accomplished in a range of different ways. Wavenet utilises a gated activation output from its dilated convolution as seen in Figure 3.1:

$$\boldsymbol{z} = \tanh(W_{f,k} * \boldsymbol{x} + V_{f,k}^T \boldsymbol{h}) \odot \sigma(W_{g,k} * \boldsymbol{x} + V_{g,k}^T \boldsymbol{h}) \tag{3.4}$$

Börjesson and Singull [60] implement a TCN based network architecture for financial forecasting and use conditioning series but instead use a single activation function:

$$\boldsymbol{z} = SeLU(\boldsymbol{w}_k * \boldsymbol{x} + \boldsymbol{v}_k * \boldsymbol{h}) \tag{3.5}$$

The key difference between FTS-GAN and the two examples above is that FTS-GAN is more complex due to being a 'many to many' problem while Wavenet and Börjesson and Singull's forecasting model use exogenous series to only generate a singular output. In the case of Wavenet this is a single audio wave while for Börjesson and Singull they use external series to forecast a single time series. FTS-GAN uses an approach where for each time series a TCN network is run in parallel to the others and for each layer the residuals are fed into the input of another time series's subsequent layer, which is where the 'stacked' description comes from. This allows the production of correlated time series. The diagram for the series conditioning, shown in Figure 3.4, is a section of the network and for the final model 6-8 of these are added together, with each 'TC Block' having a larger dilation than the previous to produce the effect as seen in Figure 3.3. The make up of the 'TC Blocks' will be covered in more detail later in this section. Figure 3.4 is an example of a section of FTS-GAN setup for two time series, but it can be easily adapted for three or more time series by stacking more 'TC Blocks', PReLU and 1x1 convolution below and joining the residuals together.

Rather than just adding the residuals together, such as in Equation 3.4 and 3.5, a channel attention module is introduced using the implementation provided by Wang *et al* [61] called Efficient Channel Attention (ECA) (Figure 3.5). Attention modules try to simulate cognitive attention. They do this by making the more important features more prominent and fading out the less important ones. For example, in models that examine text an attention model would give more weight

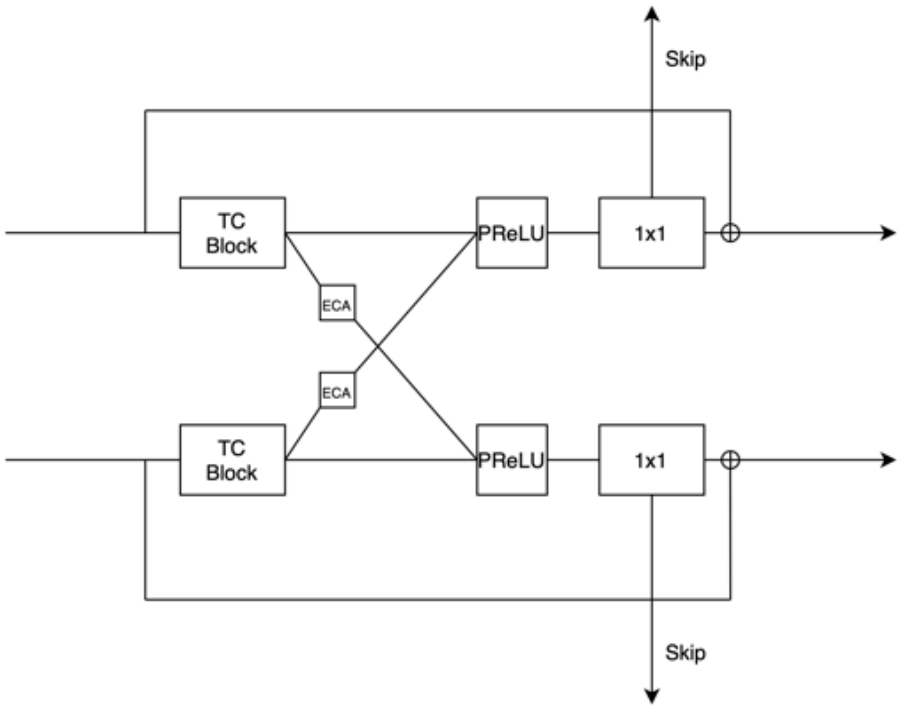


Figure 3.4: Network diagram for a section of FTS-GAN for two time series.

to certain words over others if they are more important, and this helps improve the accuracy of the model. Generally attention modules are applied to the features of the data; however, in the case of FTS-GAN they are applied to the channels so that only the important filters are transferred through. The ECA module is a computationally efficient implementation that can be easily applied and does not massively increase the complexity of FTS-GAN. The module uses three layers, an average pool, a 1d convolution and a sigmoid activation function. The module works out the important channels and then produces a vector, with channels being assigned a 1 if they are important and 0 if they are not. This vector can then be multiplied with the original input to produce an output that has the same dimensions as the input but only contains the important channels. Therefore, in FTS-GAN only the selected channels from the conditioning series will be passed along, which will help ensure only information that is critical to the correlation of the time series is used. The conditioning across all time steps is given by:

$$\boldsymbol{z} = \mathrm{ReLU}(\boldsymbol{w}_k * \boldsymbol{x} + C(\boldsymbol{v}_k * \boldsymbol{h})) \tag{3.6}$$

where $C(\cdot)$ is the channel attention module.

Each section of FTS-GAN (see Figure 3.4) includes a 'TC Block' or 'Temporal Convolution Block', which is shown in Figure 3.6. These are similar to the ones used in Quant GAN. They are added to increase the number of non-linear operations and

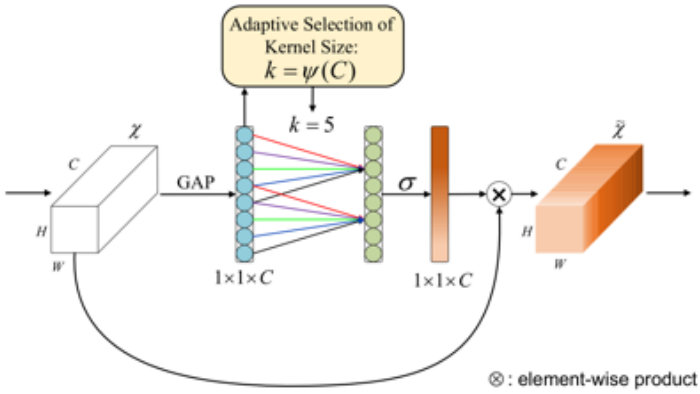


Figure 3.5: ECA Module [61].

are used in the same way in FTS-GAN. They are constructed of two convolutions and two activation functions such that the process below occurs to the input:

$$f(X) = \phi_2 \circ w_2 \circ \phi_1 \circ w_1(X) \tag{3.7}$$

w_1 and w_2 are left padded and dilated 1D convolutions while ϕ_1 and ϕ_2 represent PReLUs or any non linear activation function.

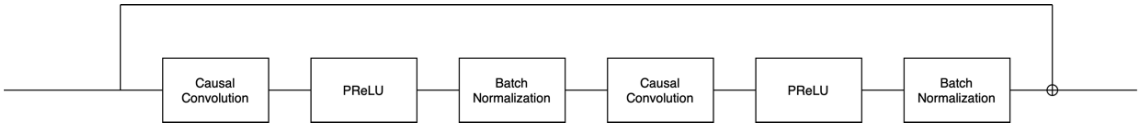


Figure 3.6: Temporal convolution Block

The 'Temporal Convolution Blocks' are what set the dilation factor for each section of FTS-GAN to produce the dilated causal convolutions. In Figure 3.6 there is also a batch normalization layer, but other weight normalising layers or dropouts could be added. For FTS-GAN, spectral normalisation is applied to the causal convolutions. It is found that applying batch normalization only to the generator works best, as when applied to the discriminator convergence either takes longer or the model fails to converge entirely. For each of the TCN blocks x is passed to the output. This is so that that the convolutions are actually learning the residuals. Residual blocks are used to improve the training of deep neural nets and have been pioneered in image recognition such as by He *et al* [62]. If $\mathcal{H}(x)$ is the underlying distribution that is trying to be approximated the residuals can be denoted as $R(x) = \mathcal{H}(x) - x$. This is rearranged to $\mathcal{H}(x) = R(x) + x$ and so means that the TCN block approximates the distribution by trying to find the residual. The reason behind approximating the residuals is that it has been shown that learning them is much easier. It also solves the problem of vanishing gradients

as the 'skip' can help propagate errors back to the initial layer. Vanishing gradients become more of a problem as a network becomes deeper and so the residual blocks can remove this issue.

At the end of each section a 'skip' connection is produced (see Figure 3.4) and these are added up element-wise to form the output of the network. The final output layer for the network is a block of 1x1 1D convolutions that ensures the output has the correct channel size. Spectral normalisation is applied to these convolutions. An example of an architecture for 6 hidden layers is shown in Table 3.1. The input into the generator is a 3 dimensional normally distributed noise prior where the second dimension is a user chosen size (typically 3 is the default value). The first dimension is the batch size and the last is the generator receptor size + discriminator receptor size - 1. This is to ensure that each data point from the generator has had a full lookback.

Layer	Layer Type	Parameters	Output Shape
1	TCN Block	Filters = K, Dilation = 1, Kernel = 1, Spectral Norm	(Batch, K, t)
2	PReLU		(Batch, K, t)
3	1x1 Block	Filters = K	(Batch, K, t)
4	TCN Block	Filters = K, Dilation = 1, Kernel = 2, Spectral Norm	(Batch, K, t)
5	PReLU		(Batch, K, t)
6	1x1 Block	Filters = K	(Batch, K, t)
7	TCN Block	Filters = K, Dilation = 2, Kernel = 2, Spectral Norm	(Batch, K, t)
8	PReLU		(Batch, K, t)
9	1x1 Block	Filters = K	(Batch, K, t)
10	TCN Block	Filters = K, Dilation = 4, Kernel = 2, Spectral Norm	(Batch, K, t)
11	PReLU		(Batch, K, t)
12	1x1 Block	Filters = K	(Batch, K, t)
13	TCN Block	Filters = K, Dilation = 8, Kernel = 2, Spectral Norm	(Batch, K, t)
14	PReLU		(Batch, K, t)
15	1x1 Block	Filters = K	(Batch, K, t)
16	TCN Block	Filters = K, Dilation = 16, Kernel = 2, Spectral Norm	(Batch, K, t)
17	PReLU		(Batch, K, t)
18	1x1 Block	Filters = K	(Batch, K, t)
19	TCN Block	Filters = K, Dilation = 32, Kernel = 2, Spectral Norm	(Batch, K, t)
20	PReLU		(Batch, K, t)
21	1x1 Block	Filters = K	(Batch, K, t)
22	PReLU		(Batch, K, t)
23	1x1 Block	Filters = K, Spectral Norm	(Batch, K, t)
24	PReLU		(Batch, K, t)
25	1x1 Block	Filters = C_{out}, Spectral Norm	(Batch, C_{out}, t)

Table 3.1: FTS-GAN: Example implementation of 6 hidden layers which would give a lookback of 127.

The architecture can then be used for both the discriminator and generator, albeit with a few modifications in the settings such as the batch normalisation not being used for the discriminator. A final list of parameters will be shown in Chapter 4

for each of the datasets tested upon. As mentioned at the start of this chapter the standard GAN framework will be used and so FTS-GAN will be implemented using the method in Algorithm 3. The loss function chosen for FTS-GAN is the minimax as used in SGAN by Goodfellow [24] and described in Section 2.2.4.

Algorithm 3: GAN training algorithm

for *number of training iterations* **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$
- Update the discriminator by ascending its stochastic gradient

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$$

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from a noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

$$\nabla \theta_g \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$$

end

3.2 Conditional FTS-GAN

In Section 2.2.4 the conditional GAN was introduced and its ability to be used in shaping the outcome of the generated samples. A similar method was tried for FTS-GAN so that the type of samples being produced can be controlled. The problem with conditioning for FTS-GAN was that the advantage of flexible inputs now becomes a problem when considering conditioning. In imaging GANs the input is a fixed dimension of pixels, and so when applying labels a FCN can be used to scale the labels from a single value to a group of features that can be easily concatenated with the noise input for the generator and the synthetic data for the discriminator. The first implementation applies a latent matrix to each layer so that Equation 3.6 would become the following:

$$\boldsymbol{z} = \text{ReLU}(\boldsymbol{w}_k * \boldsymbol{x} + C(\boldsymbol{v}_k * \boldsymbol{h}) + \boldsymbol{w}_\gamma l) \tag{3.8}$$

where l is the matrix of latent variables. The second one forms a vector by taking the label and expanding it to the size of the hidden layers by using a FCN. This vector is then added onto the start of the input series.

3.3 Challenges

There are several problems that GANs are very susceptible to and can cause issues when trying to train them and so will be encountered during this research. Where possible adjustments will be made to FTS-GAN to mitigate these problems. Although covered later within Section 3.7, one of the problems with GANs is the lack of evaluation metrics and this can be a problem when trying to tune and compare them. Two main issues that arise during the training process are discussed below and highlight how GAN training can be quite finicky.

Non-converegnce

Usually for an optimisation problem the training will cause the model to get closer and closer to the result. However, in the case of GANs there are two neural networks being updated independently and so the optimization is always changing. As the system is dynamic it can be difficult to control. If one of the networks learns at a faster rate it can cause issues and the optimum objective is never reached. For example, if the generator learns faster and can always fool the discriminator the discriminator cannot produce a loss function that allows for further useful training.

Mode Collapse

Mode collapse, also known as a Helvetic Scenario, is a deficiency in training and happens when the noise input is mapped to few or a single output value. This means the GAN misses the ability to represent regions within the target distribution and so is stuck in a local minimum. The discriminator can then easily identify the fake samples from the real and no further improvements in the quality of the generated samples will be possible. This leads to synthetic data that is useless for any application, as ideally the samples should be a spectrum of different scenarios that all possess the same underlying characteristic properties. Remedies, including changing the loss function to something such as the Wasserstein loss, have been documented to improve the stability. It is also noted that the choice of network architecture has a critical impact on the quality of the samples as well as training parameters such as batch size. An example of a model that has been adapted specifically to target stability is the Unrolled GAN [63].

3.4 Additional GAN improvements

This section briefly looks into other improvements that will be tried on FTS-GAN to improve the synthesis quality without directly changing the framework or the neural nets. One of the areas that has been researched is whether information from the discriminator can be used to help in the generation of samples after the model has been trained. Around half of the training time is spent training the discriminator but then when it comes to the generation process the discriminator is disregarded. Therefore, Azadi *et al* [64] propose a method called 'Discriminator Rejection Sampling' (DRS) which is a method that applies reject-accept sampling on the generated samples based on the probabilities that the discriminator assigns each sample during classification. They find this boosts the quality of their samples.

Another method suggested by Sinha *et al* [65] is the 'Top-k' training of GANs. This method is applied during training and tries to use more information from the discriminator to help the quality of the samples. This method works by ranking the quality of the samples for each iteration by the score assigned by the discriminator. Then only the top 'k' samples are used in the loss function to backpropagate the error and update the generator's parameters. Sinha *et al* find that they produce nearly a 10% improvement in the quality of their generated samples. This can be a tunable parameter that is selected during hyperparameter optimisation.

3.5 Data Preprocessing

To train the model on the data it must go through a few preprocessing steps. First the data must be made stationary. This can be done by taking the log returns of the prices:

$$r_t = \log\left(\frac{p_t}{p_{t-1}}\right) \tag{3.9}$$

The data must then be standardised as neural networks generally converge better on standardised inputs [66]. Similar to Wiese *et al* [52], one can also apply an inverse lambert W transform on the data. This method is used to gaussianise the data and remove the fat tails. The idea is that, if the data conforms closer to a normal distribution it will be easier to replicate. This is implemented via a script from Github [67].

The final step is to to produce rolling windows of the data. The length of the rolling window is based on the length of the field-receptor, T_f. The window moves along the time series with a stride length of 1. Therefore for a time series of length

T, we will produce $T - T_f$ samples. The windows can be defined as:

$$r_{1:T_f}^{(t)} = r_{t:(T_f+t-1)}^{\text{Series}} \tag{3.10}$$

The dimension for the data input was (Batch Size, No. Variables, No. Time series, No. Time steps). The number of variables is usually 1 but this project briefly explores the potential for inputting OHCL data for each individual series. OHCL data is multivariate data for an asset that gives the open, high, close and low for each day. The number of time steps is dependent on the receptor size of the model and is set based on the required lookback of the model. In most cases this would either be 127, 255 or 511 (these come from Equation 3.2) as most indices showed absolute autocorrelations that spanned this distance. Longer lookbacks are possible but they cause the size of the model to grow very quickly, meaning that memory becomes an issue as well as the computational time being longer. The time period of the data affects the lookback period. More recent data for the S&P showed shorter decay time for the autocorrelations than older data and so this means the time period of the data affects the choice of model parameter. This will be covered further in Section 4.7.

3.6 Hyperparameter Selection

There are multiple parameters that can be set for the model which have the potential to improve the speed of convergence and quality of the samples. For the model the size of the receptor field as well as the hidden channel dimensions are the two main parameters. When considering these parameters one must also balance them with the batch size so that the model is not too large, and the GPU runs out of memory.

There is also the choice of the optimizer as well as its parameters. To find the optimal hyperparameters the python module Optuna [68] is used, which utilises Bayesian optimization. The evaluation of models is difficult with GANs, as there is no overall objective function we must come up with our own. Using some of the statistical metrics, one can average them to produce an objective metric and use this to tell Optuna how well its search is going. The model is told to stop training once a certain threshold is reached by the metrics so that the optimal parameters can then be used in the generation process.

3.7 Evaluation Methods

As previously covered in Section 2.3, evaluation of results produced from a GAN can be very difficult. For financial data it is known what sort of statistical properties that the synthetic data should possess so this can be used as an initial starting point for analysing the quality of our generated data. For training, a metric is produced that calculates the difference between the statistical properties of the synthetic data and the real data in order to see how the training of the model is coming along. Comparing the interactions between time series is more tricky and the main one is to use the correlation between time series.

One can also use another classifier network such as a ResNet classifier [59]. A ResNet classifier is similar to a TCN in regards to having residual connections that pass between the layers, and these are found to improve the performance. ResNet is chosen as a classifier due to its high performance in classification across a range of datasets [69]. The reason for using another neural network to test the data is it gives a real world application of the data. Although the test is basic, one of the main use cases for the synthetic data is to reduce overfitting during training. Using the 'train on synthetic, test on real method' the classifier is first trained on a subset of the real data and also tested on the rest of the data. Then the test is repeated but the training is now undertaken on just the synthetic data and the testing is done on the real data. If the synthetic data is of comparable quality to the real data the accuracy scores from the classifier should be the same and in most cases we should expect a better accuracy as the number of synthetic samples will be more which will reduce the overfitting during the training phase.

Layer	Layer Type	Parameters	Output Shape
1	Conv1D	Kernel = 8, Spectral Norm	(Batch, Filters, 30)
2	Leaky ReLU	Parameter = 0.2	(Batch, Filters, 30)
3	Conv1D	Kernel = 5, Spectral Norm	(Batch, Filters, 30)
4	Leaky ReLU	Parameter = 0.2	(Batch, Filters, 30)
5	Conv1D	Kernel = 3, Spectral Norm	(Batch, Filters, 30)
6	Leaky ReLU	Parameter = 0.2	(Batch, Filters, 30)

Table 3.2: ResNet Block. The model is made up of three of these with a residual connection which is added at to the output of each block.

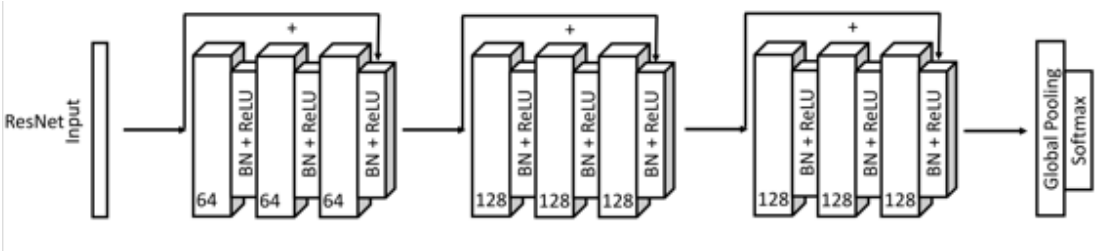


Figure 3.7: Diagram of the ResNet classifier used in checking the quality of the synthetic data [59].

3.8 Hardware and Setup

All the research was carried out in Python. FTS-GAN uses the Pytorch [70] package to build the temporal convolutional networks. Pytorch has the ability to allow one to run the model on either the CPU or use the CUDA extension to run on Nvidia GPUs to reduce the computational times of the model. The source code for FTS-GAN can be found at `https://github.com/Droyd97/fts-gan`. For the data cleaning, processing, analysis and the rest of the functionality of the package the following libraries were also used: Numpy [71], Scipy [72] and Pandas [73] [74].

FTS-GAN is computationally expensive due to complexity of the convolutional nets. Therefore, the training had to be done on a windows virtual machine with 6 vCPUs and a Nvidia P100 GPU. The access to a GPU was invaluable to the research as it reduced compute times from ∼70 sec/iteration to ∼1.5sec/iteration for 3 series. Although exact times depended on the hyperparameter selection and the number of series this produced a huge time saving that allowed us to even expand the lookback range to 511 time steps. This allowed for much faster development times allowing for a wide range of parameters and improvements to be tested that would be inaccessible on a laptop.

3.9 Alternative Models Considered

The inititial research for this project involved implementing the WGAN-GP model described in De Meer Pardo [27] and also implemented in Eckerli and Osterrieder [23]. This uses the normal GAN setup (shown in section 2.2.4) with the Wasserstein loss function and a gradient penalty to enforce the Lipschitz constraint. The architecture used is an encode-decoder style with 1D convolutions. The Generator network setup is shown in Figure 3.3 and the critic setup is shown in Figure 3.4. As well as WGAN-GP the DCGAN and SAGAN (as implemented by Eckerli and Osterrieder [23]) were tested. For multiple time series the Relativistic Average GAN

was implemented which was used for the generation of VIX samples conditioned off of S&P 500 data. This is achieved by use of 2D convolutions. In Section 4.1 the shortfall of these models will be covered and it will be explained why they were not used for the final model.

Layer	Layer Type	Parameters	Output Shape
1	Linear	Input = 50, Output = 100	(Batch, 100)
2	LeakyReLU	Parameter = 0.2	(Batch, 100)
3	Add Dimension		(Batch, 1, 100)
4	Conv1D	Filters = 32, Padding = 1, Spectral Norm	(Batch, 32, 100)
5	Upsample	Size = 200	(Batch, 32, 200)
6	Conv1D	Filters = 32, Padding = 1, Spectral Norm	(Batch, 32, 200)
7	LeakyReLU	Parameter = 0.2	(Batch, 32, 200)
8	Upsample	Size = 400	(Batch, 32, 400)
9	Conv1D	Filters = 32, Padding = 1, Spectral Norm	(Batch, 32, 400)
10	LeakyReLU	Parameter = 0.2	(Batch, 32, 400)
11	Upsample	Size = 800	(Batch, 32, 800)
12	Conv1D	Filters = 1, Padding = 1, Spectral Norm	(Batch, 1, 800)
13	LeakyReLU	Parameter = 0.2	(Batch, 1, 800)
14	Squeeze Dimension		(Batch, 800)
15	Linear	Input = 800, Output = 100	(Batch, 100)

Table 3.3: Generator for WGAN-GP

Layer	Layer Type	Parameters	Output Shape
1	Add Dimension		(Batch, 1, 100)
2	Conv1D	Filters = 32, Padding = 1, Spectral Norm	(Batch, 32, 100)
3	LeakyReLU	Parameter = 0.2	(Batch, 1, 100)
4	Max Pool	Pool size = 2	(Batch, 1, 50)
5	Conv1D	Filters = 32, Padding = 1, Spectral Norm	(Batch, 32, 50)
6	LeakyReLU	Parameter = 0.2	(Batch, 1, 50)
7	Max Pool	Pool size = 2	(Batch, 1, 25)
8	Conv1D	Filters = 32, Padding = 1, Spectral Norm	(Batch, 32, 25)
9	LeakyReLU	Parameter = 0.2	(Batch, 32, 25)
10	Flatten		(Batch, 800)
11	Linear	Input = 800, Output = 50	(Batch, 50)
12	LeakyReLU	Parameter = 0.2	(Batch, 50)
13	Linear	Input = 50, Output = 15	(Batch, 15)
14	LeakyReLU	Parameter = 0.2	(Batch, 15)
15	Linear	Input = 15, Output = 1	(Batch, 1)

Table 3.4: Critic for WGAN-GP

Chapter 4

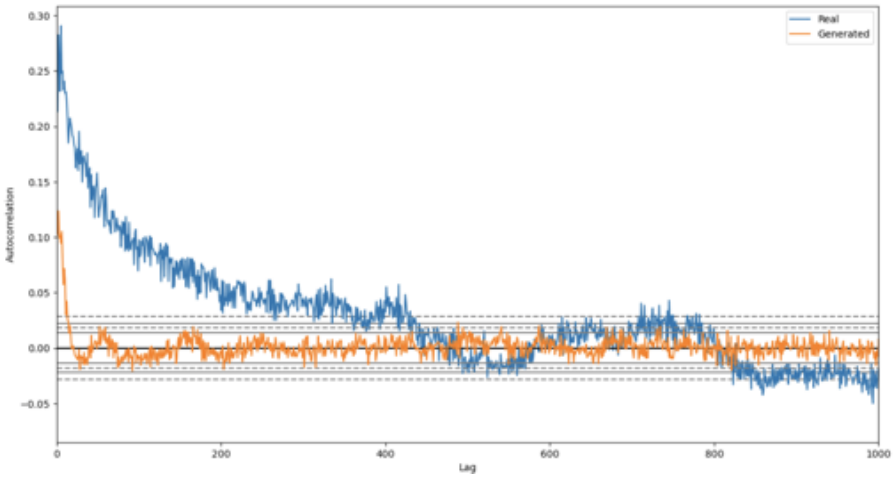
Results

In this chapter the results are presented for the testing that was undertaken during the project. The performance of WGAN-GP will be first covered as well as the rationale behind not proceeding with the alternative models. Results for just the S&P 500 will be used to validate the Quant GAN model and produce a benchmark of a single time series for comparison with the multidimensional tests. The multi series datasets will be introduced and the challenges that arose with FTS-GAN. Finally, the performance of the model will be shown with analysis into the statistical properties of its output.

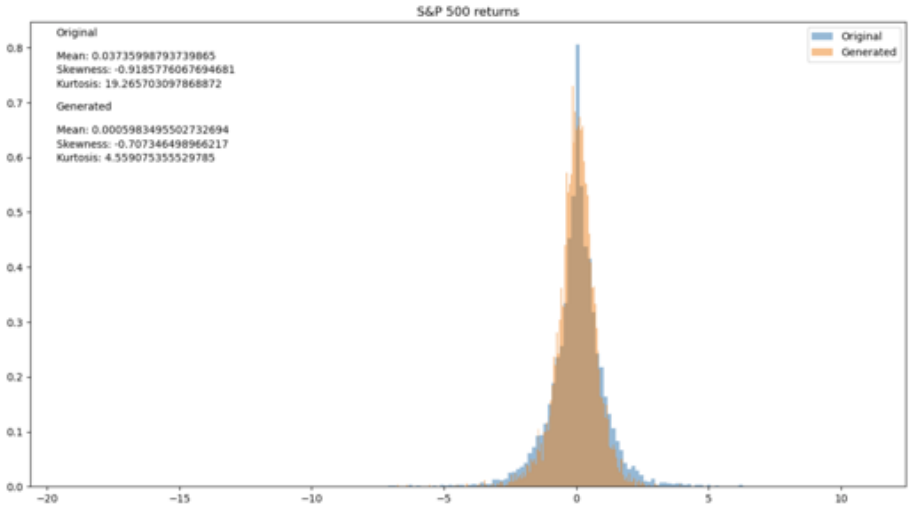
4.1 Preliminary Experiments

To highlight the deficiencies that occurred with some of the initial models, the results for the WGAN-GP model are presented in Figure 4.1a and Figure 4.1b. The recreation of the returns distribution is pretty close to the real data and it was possible to replicate this with a range of different models used such as Relativistic Average GAN (RaGAN), DCGAN and SAGAN. This was a common trait amongst most of the GANs that were designed for imaging, and the ones tested had little adaptation to their network architecture for the processing of time series data. However, the models could not reproduce the decay in the absolute returns and so the research turned to time series specific architectures such as RNNs and TCNs as they perform better as they produce continuous results and don't require the output to be 'stitched' together. WGAN-GP only outputs samples with a specific size so to create a longer time series the samples must be joined together; however, this means there is no longer correlation between the samples. This can be improved by some form of 'conditioning'; however, a time series specific architecture removes this com-

plication as well as having a better overall performance. RaGAN is an example of a multi series model that produces a second series from a first series. However, this meant that one would have to use another model to create the input into RaGAN leading to a slow generation process. As the results for the alternative models are poor it justifies the use instead of Quant GAN as the initial model which has been developed into FTS-GAN. The remainder of this chapter will focus on results that were produced by FTS-GAN.



(a) Absolute autocorrelations of the returns.



(b) Histogram of the returns.

Figure 4.1: Results for WGAN-GP on S&P 500.

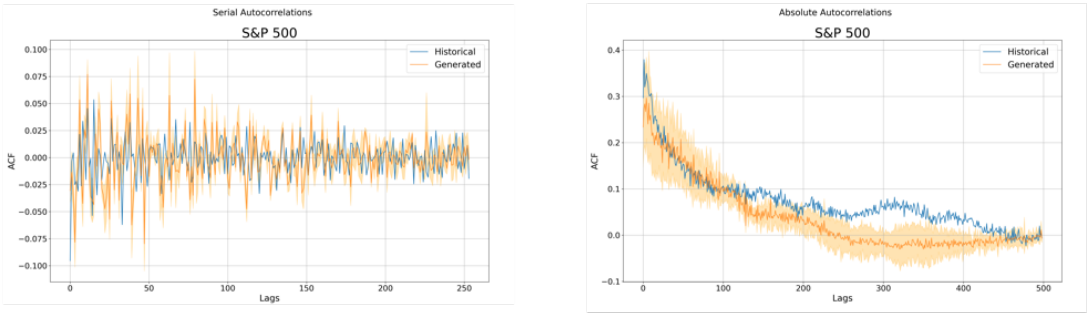
4.2 Benchmarking on a Single Time Series

The results for FTS-GAN for a single time series, the S&P 500, are presented in order to produce a benchmark in quality of the statistical properties for a single time series so that a comparison of performance can be made when more time series are

Figure 4.2: 10 synthetic samples for the S&P 500.

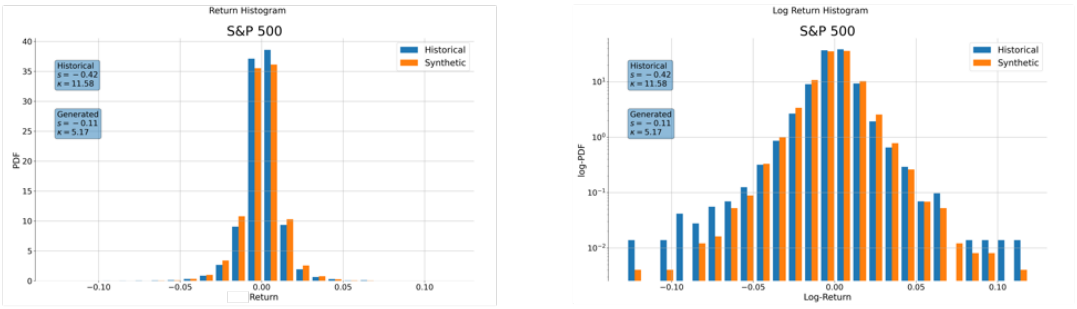
added. Figure 4.2 shows 10 samples from the generation process. A wide range of scenarios can be seen and overall the data looks plausible. At some of the extremes it could be argued that the drawdowns are a little too large or the bull market is too strong. However, it is worth noting that although the aim is to compare against the real data, this is only one version of history and the model is built to produce other potential futures that allow for robust testing of trading strategies or other statistical models. It also allows for comparison against other work in the literature and to show the ability of TCN network based GANs. The model is able to accurately reproduce uncorrelated returns (see Figure 4.3a) while for the first 100 lags it is able to reproduce the absolute autocorrelations shown in Figure 4.3b. The absolute autocorrelations become weaker after about 100 lags. The distribution of returns is modeled well with the synthetic data even managing to recreate the 'fat tails' to some degree. However, it only takes a few shocks in the data for the kurtosis to become large. The historical value is 11.58 and the generated value is 5.17 for the kurtosis, while the skew values are less far apart with the historical being -0.42 and the synthetic having a skew of -0.11. Although properly fitting the tails is tricky usually as the occurrences are very low and the model prefers it when the return distribution is continuous. Finally the leverage effect is well followed with the synthetic data managing to follow most of the movements of the real data.

After carrying out this benchmark the two datasets that will be used for testing will be introduced as well as the challenges faced with FTS-GAN. This forms the novel part of this project as although single time series have had a lot of research results, so only used as a benchmark in this project, multi time series generation is not as well documented especially between series such as the S&P 500 and VIX.



(a) Serial autocorrelations. (b) Absolute autocorrelations.

Figure 4.3: Autocorrelations for returns of S&P 500.



(a) Histogram for the returns. (b) Log histogram for returns.

Figure 4.4: Return distributions for S&P 500.

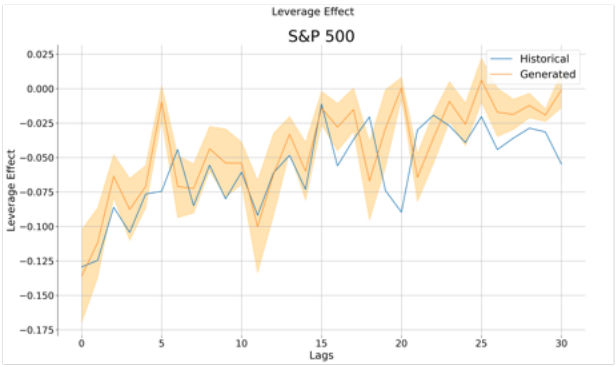


Figure 4.5: Leverage effect for the S&P 500.

4.3 Datasets and Challenges

There are two main datasets used for the testing of FTS-GAN. All the data used contain the daily prices of each respective index. The first is the S&P 500 and VIX. This is tested as they are two highly correlated time series. The VIX is a measure of the market's expectation of volatility over the next 30 days and is calculated from the prices of SPX index options. Investors can invest in the VIX through options, futures and ETFs to speculate on the volatility of the market. One of the main characteristics of the VIX is that it is mean-reverting so that if the price of the VIX is high future values should be lower. The second dataset is made up of three time series: the S&P 500, the Gold index and the US treasury bond index. There is very little literature that provides proper results and analysis for more than two synthetic financial indexes. The ability to expand the model for more and more time series is useful for practitioners who may want to test strategies that use decisions based on multiple exogenous variables. However, this comes with more computational complexity as well as the GAN struggling to fit all the data.

The development of FTS-GAN was an iterative process with the different models trialed as described in Section 3.9; however, these models underperformed in comparison to TCN based architectures as shown in Section 4.1. After finding poor results trying to capture the autocorrelations of the absolute returns, the work turned towards looking at more time series focused architectures and settled on the TCNs. Wiese has applied this for the 1D case and so methods were tried to extend it for multiple time series. The first iteration was to follow a similar method to de Meer Padro's Relativistic GAN and apply a conditioning to a model so that correlated time series could be created. Initially 2D convolutions were tried but this greatly increased the computational requirements and produced poor results. The final iteration of the conditioned time series was to apply the conditioning series in place of one of the noise channels which allowed the VIX to be produced for the S&P. This model then evolved into the 'stacked' network architecture that is used in the final form of FTS-GAN. This was further modified by adding the ECA modules to try and increase the quality of the correlations between the time series. Other improvements to FTS-GAN were trialed throughout the project such as different loss functions as well as the improvements covered in Section 3.4. Top-K did produce some improvements with it being used as a hyperparameter while the benefits of DRS were harder to see. Although lots of work in the literature points towards using the Wasserstein loss as it improves the performance, as highlighted in Section 2.2.4, for this model it provided worse results.

One of the problems encountered while training the models was finding the optimal generation point. Throughout training a GAN, its quality of synthetic samples changes and training for longer does not guarantee better quality data. In fact quite the opposite happens, which is that as the GAN is trained longer it is found that the samples start to get worse again. This is one of the many shortcomings of GANs and their difficulty to train. This means that training must be ended early to ensure that the best model weights are saved. This was initially implemented by way of comparing the metrics of the synthetic samples to the historical samples and setting a threshold for when the values reach a certain degree of accuracy. This method works well for one time series, and even two in most cases, but for three time series the model struggles to produce samples that have all three series within the threshold and so the early stopping is not triggered even though the optimal parameters of the model have been reached. Therefore, to solve this problem an objective value was produced for each iteration that was a combination of some of the metrics. If the objective value is better than the previous best the model is saved. This means at the end of training the optimal model can be reloaded and used for generating data.

Another challenge faced was the actual recreation of the sample paths. The generation of returns is one part of the process, but the ability to convert these into realistic but different sample paths could be tricky, causing some of the synthetic paths to 'explode' to infinity. This was due to the inverse transformation of the paths. Before the inverse transformation of the returns can take place they must be normalized, but rather than normalize each path independently it is better to normalize using the mean and standard deviation of all samples. If each path is done independently the mean of all the samples becomes the same and would mean that the model would produce results in a similar fashion to a Brownian bridge. This is not ideal and so by using the average of the population the samples will have different endpoints. Although this method worked well for the non mean-reverting time series, such as the S&P 500, it produced results that are orders of magnitude out for the VIX samples and so they had to be independently normalized. This was also a tricky area for the lambert W transform with the transformation back to the original samples causing large outliers that created massive skews in the data and produced a large amount of kurtosis. Other noise priors were tested such as a Cauchy distribution, to see if using a distribution with fatter tails would help the fitting, but unfortunately the model struggled with non-normal inputs.

While the hyperparameter tuning was able to run in the case of two series, it became harder and harder to search for a wide net when applying it to three series

due to the memory limitations of the GPU. This meant that the testing had to be constrained to a few depths and lookback parameters and so it became less of an automated process. It became an optimization problem because the lookback, the number of hidden channels and the batch size all affected the size of the model. Generally the lookback was set to be just bigger than the decay of the absolute autocorrelations of the series but, as will be seen in section 4.5, sometimes the number of lags of the decay is too large to capture the whole extent of the decay. It was found that a larger batch size produced better results as the model would generalise more.

4.4 S&P 500 and VIX

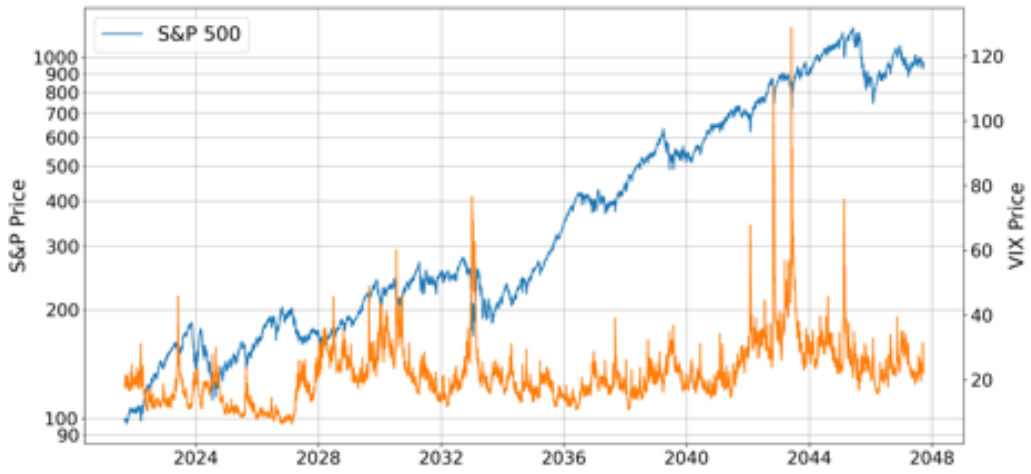


Figure 4.6: Single example of a synthetic sample for the S&P 500 and VIX.

The first multidimensional test was on the VIX and S&P 500. Looking at Figure 4.6 one can see an example of a better performing sample. See Appendix A.1 for more examples. Across the board the S&P 500 data was strong, with it showing three different modes across the 20 years of synthetic data. It possesses regions of large drawdowns similar to economic crises. It has areas that would be labeled as 'bull' markets, as well as having areas with lower volatility. The VIX data is not as realistic in how it looks, with many of the samples producing values that have not been seen in the VIX before. However, if the metrics are analysed for the statistical properties that are trying to be recreated, both the S&P 500 and VIX perform very well. The absolute autocorrealtions are a bit weak but the accuracy is affected by the model parameters, setup and the length of the real data. The time period

used for the training is very important as it has a large impact on the training of the model and the statistics that it will capture; this will be analysed further in section 4.7. The S&P 500 samples hold up well against the 1D generated data. The absolute autocorrelation returns also manage to hold onto about 100 days, although for the 2 time series data the spread is slightly larger. For the distributions the S&P 500 actually manages to have a more accurate Skew and Kurtosis value over the 1D data. The correlation between the time series is well captured; looking at the sample the VIX can clearly be seen to increase as the S&P 500 goes through periods of drawdowns, and this is supported by the correlation results in Table 4.1. The VIX data is tested using another method in Section 4.9.

As a way to rectify the issue with the VIX samples not being an accurate reconstruction of real life, the model was tested on price data rather than the returns. Although the sample outputs looked more realistic the return distribution had very heavy tails meaning there were huge movements in the price, and so this was not researched into anymore and left for further research.

The next section introduces the three time series dataset.

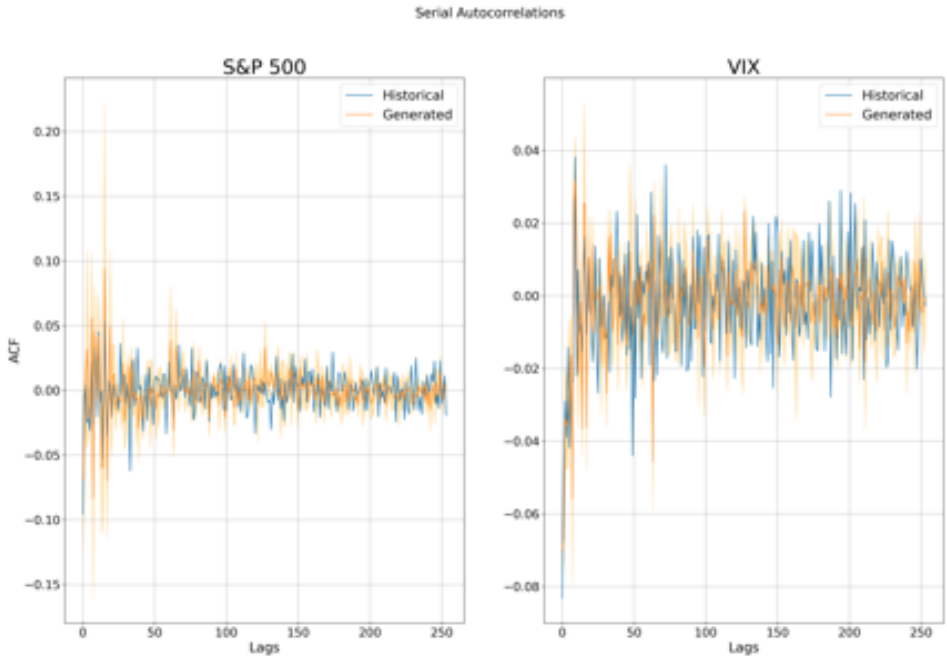


Figure 4.7: Serial autocorrelations for synthetic S&P 500 and VIX samples.

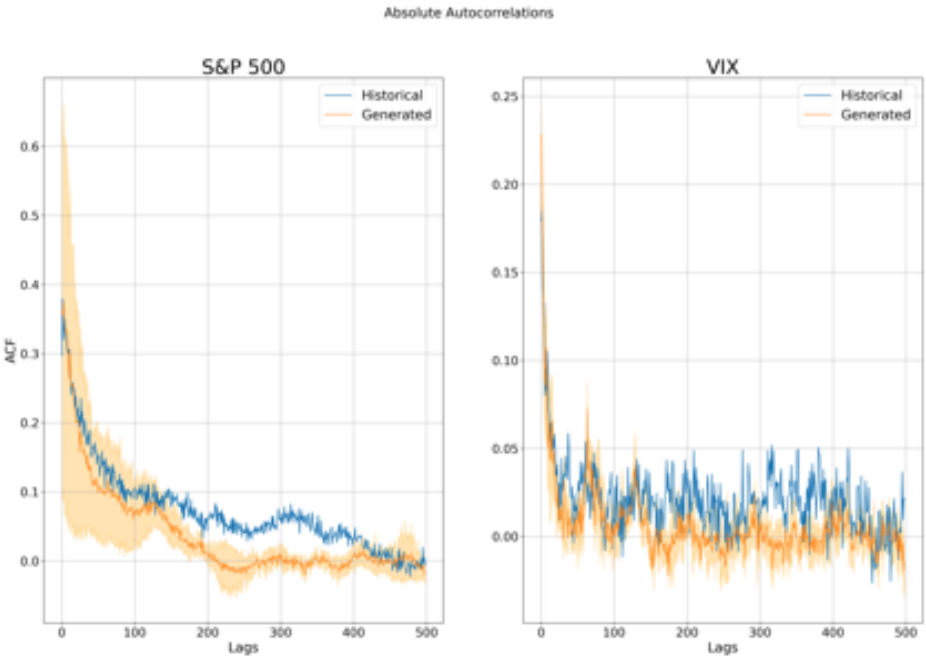


Figure 4.8: Absolute autocorrelations for synthetic S&P 500 and VIX samples.

Series	Real Correlation	Synthetic Correlation
S&P 500 and VIX	-0.71	-0.73

Table 4.1: Correlation comparison of the real and fake data for the S&P500 and VIX.

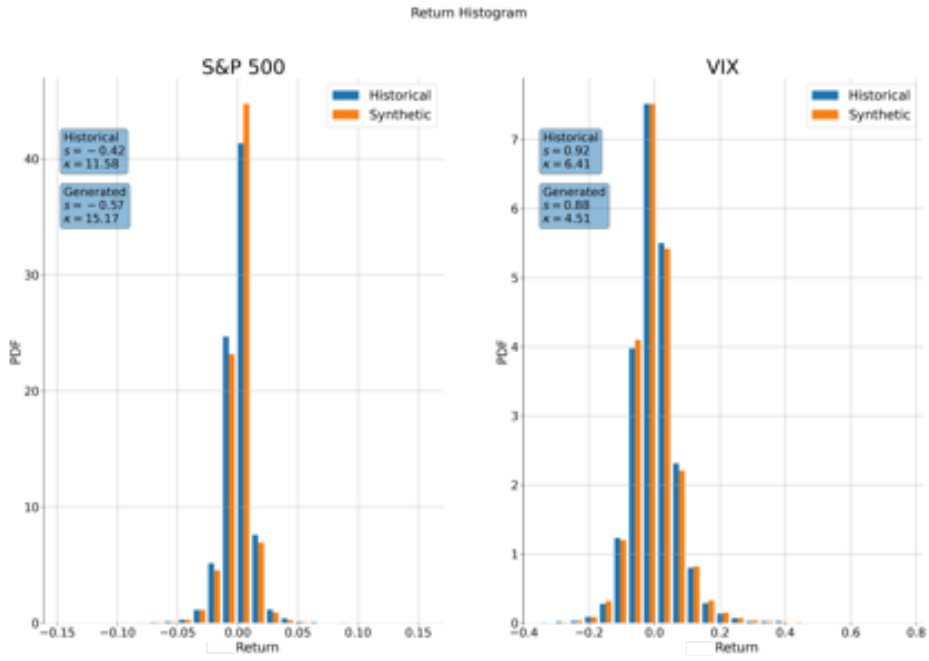


Figure 4.9: Histogram ffor synthetic S&P 500 and VIX samples.

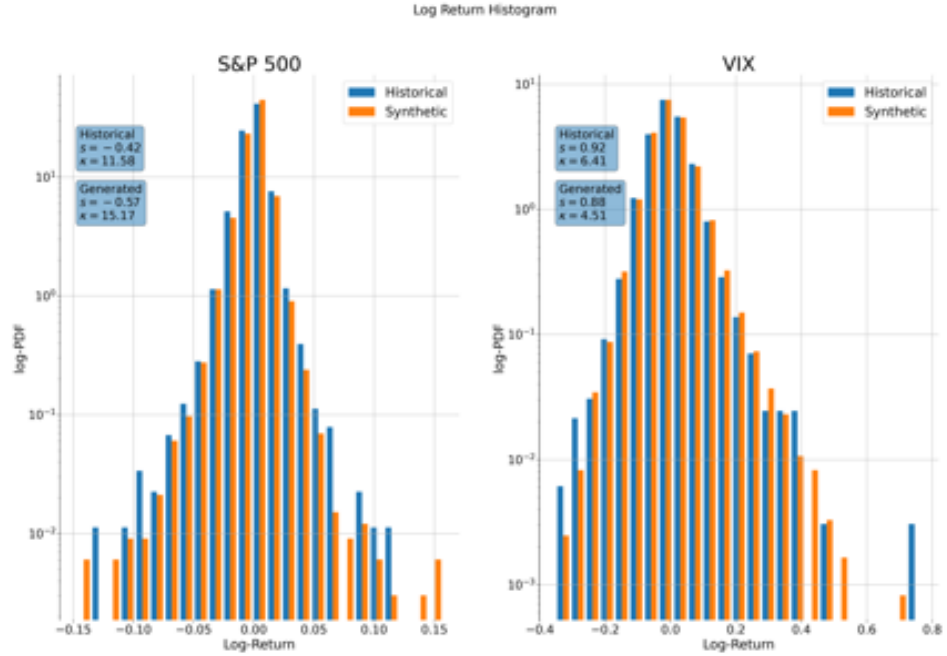


Figure 4.10: Log histogram for synthetic S&P 500 and VIX samples.

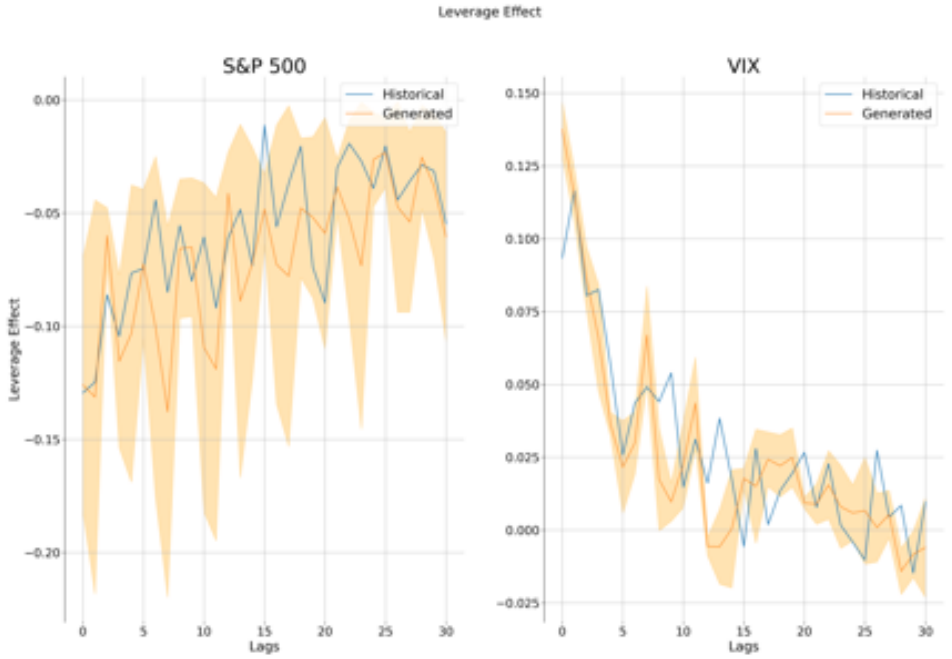


Figure 4.11: Leverage effect for synthetic S&P 500 and VIX samples.

Parameter	Generator Value	Discriminator Value
In-channel	7	1
Out-channel	1	1
Hidden layers	100	100
No. of hidden layers	7	7
Batch normalization	True	False
Optimiser	ADAMW	ADAMW
Optimiser learning rate	1e-4	3e-4
Optimiser Betas	(0.0, 0.9)	(0.0, 0.9)
Optimiser EPS	1e-08	1e-08
Batch Size	256	
Top-K	200	
Objective Features	'acf_abs', 'acf_id', 'kurtosis'	

Table 4.2: FTS-GAN: Parameter selection for S&P 500 and VIX model

4.5 S&P 500, Gold and Government Bonds

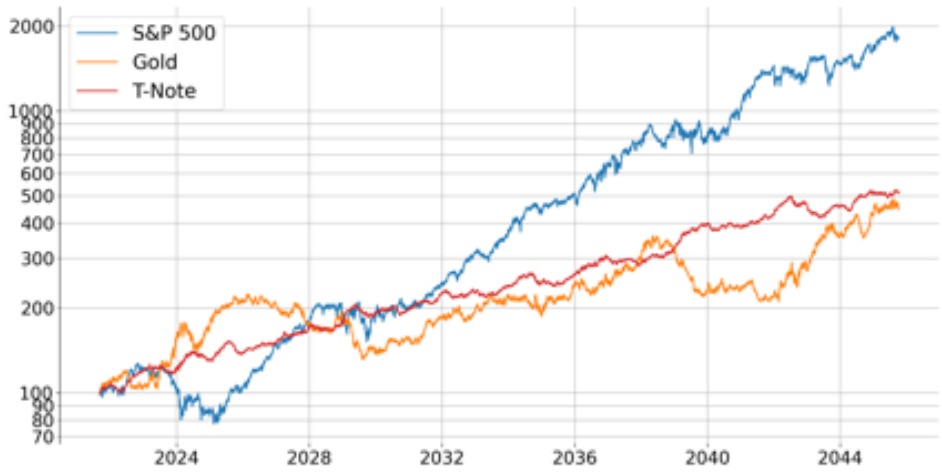


Figure 4.12: Serial autocorrelations for synthetic S&P500, Gold and T-Note samples.

The second dataset provided a good insight into how FTS-GAN faired for more than two timeseries. Figure 4.12 shows a single generated sample from the model for all three time series. Further samples generated from the model can be found in Appendix A.2. All three samples show distinct patterns that are characteristic of their respective index. If the series are analysed independently, they all perform well for each metric. Although the metrics may look a bit weak when compared to the previous ones for the S&P 500 and VIX samples this is probably due to the time period and the shocks affecting the correlation between different asset classes differently and so the model has to generalise for the whole period of 1992 - 2019. In a later section it will be shown how these results are different when a time frame with more consistent statistical properties is analysed.

All three samples possess no serial correlation as seen in Figure 4.13. For some parameters there were problems with the Gold index possessing autocorrelations in the returns for a few lags. This is a problem as synthetic market data with correlations is unusable due to trading strategies being able to easily predict the future price based on the past. However, with some parameter tuning this issue could be resolved. The absolute autocorrelations (Figure 4.14) are present for all indexes, although the S&P 500 has a faster decay in comparison to the real data. This is probably due to the real data's decay being slower, due to containing many more shocks in its time frame. For the Gold index, the model struggles to capture the end of the correlations but this is due to the real data still not reaching zero after

500 lags. If the model cannot have a lookback that covers the whole decay period it struggles to fit the absolute autocorrelations. The problem is extending the lookback over 500 lags causes the model to run out of memory and so the number of channels in the network would have to be reduced leading to even worse performance, as it was found that as the number of timeseries increased the number of channels should also increase. There is no hard rule on this but it is expected that with more timeseries there will be more features to capture and pass onto the other series, so more channels allow performance to stay high.

For two of the three correlations between the samples the results are close to the real data (see Table 4.3); however, the correlation for the Gold and T-Note is very far off the real data. At the moment there is currently no explanation why it works for the other two but fails on the final one.

Having used two datasets with a different number of indexes and getting good performance, testing on a dataset of the S&P 500 and VIX with the OHCL features will now be introduced.

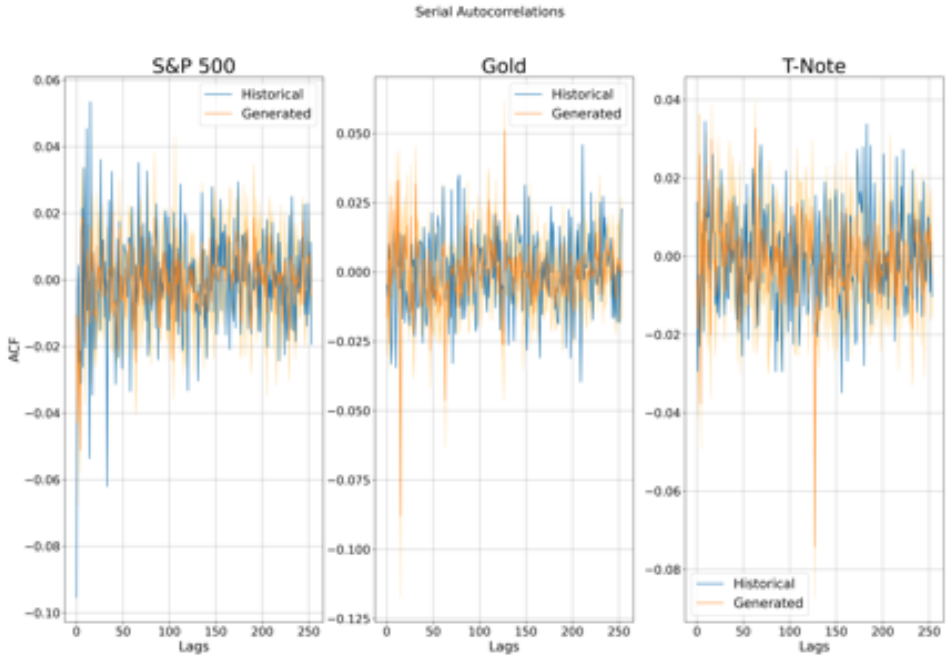


Figure 4.13: Serial autocorrelations for synthetic S&P500, Gold and T-Note samples.

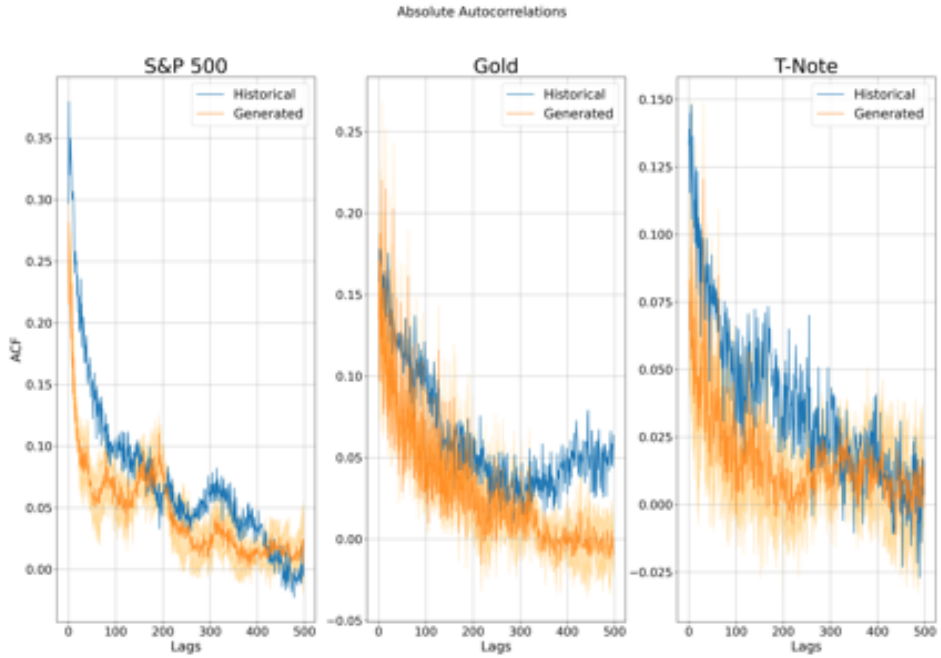


Figure 4.14: Absolute autocorrelations for synthetic S&P500, Gold and T-Note samples.

Series	Real Correlation	Synthetic Correlation
S&P 500 and Gold	-0.025	-0.060
S&P 500 and T-Note	-0.25	-0.29
Gold and T-Note	0.088	0.21

Table 4.3: Correlation comparison of the real and fake data for the S&P500, Gold and T-Note.

Parameter	Generator Value	Discriminator Value
In-channel	3	1
Out-channel	1	1
Hidden layers	100	100
No. of hidden layers	8	8
Batch normalization	True	False
Optimiser	ADAMW	ADAMW
Optimiser learning rate	1e-4	3e-4
Optimiser Betas	(0.0, 0.9)	(0.0, 0.9)
Optimiser EPS	1e-08	1e-08
Batch Size	80	
Top-K	80	
Objective Features	‘acf_abs’, ‘acf_id’, ‘correlation’	

Table 4.4: FTS-GAN: Parameter selection for S&P 500, Gold and T-Note model

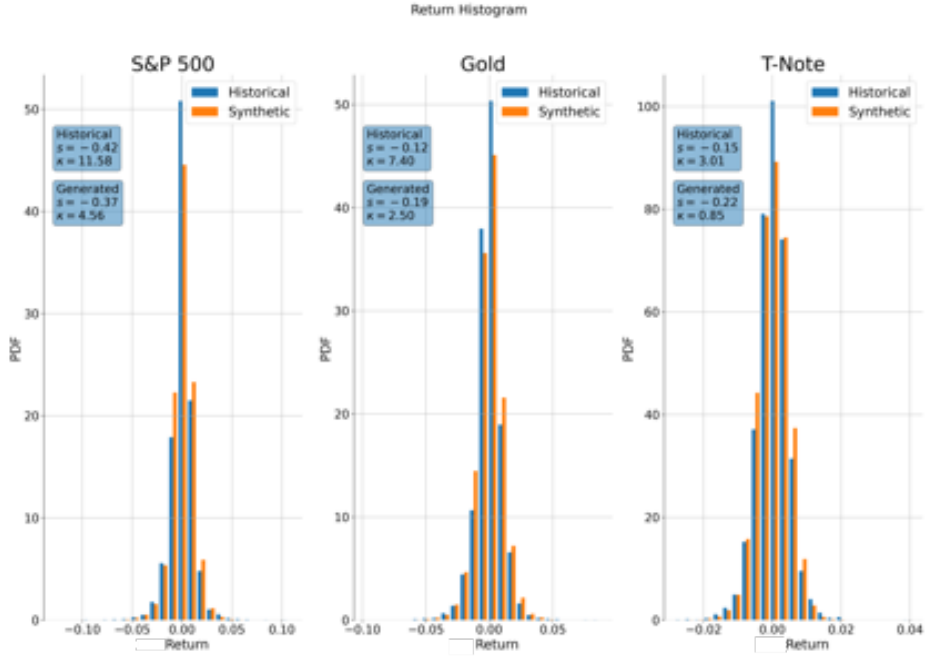


Figure 4.15: Histogram for synthetic S&P500, Gold and T-Note samples.

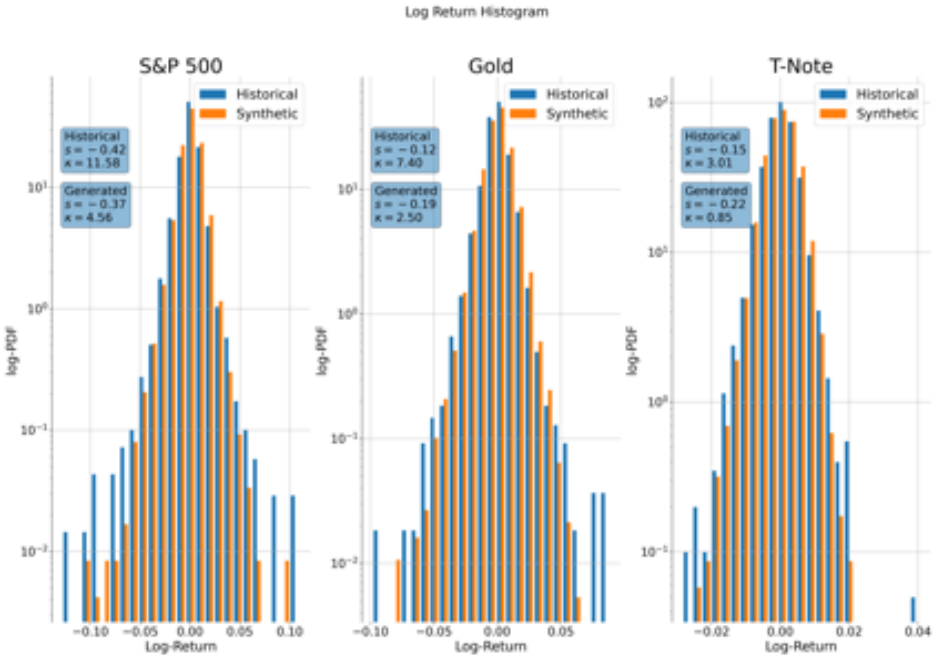


Figure 4.16: Log histogram for synthetic S&P500, Gold and T-Note samples.

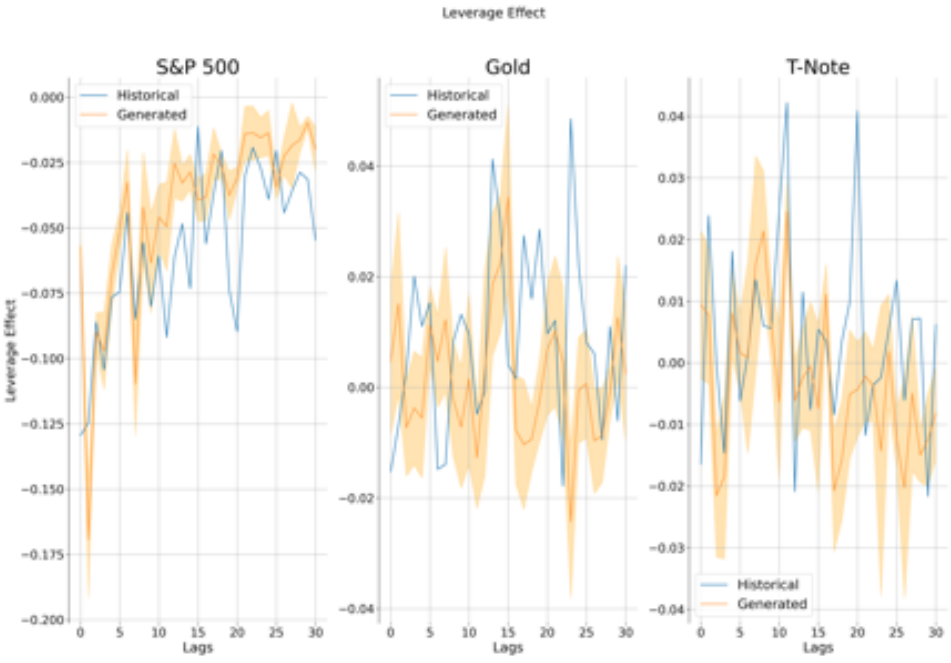


Figure 4.17: Leverage effect for synthetic S&P500, Gold and T-Note samples.

4.6 OHCL

As FTS-GAN produced plausible samples with strong statistical properties it was decided to push FTS-GAN by introducing a dataset of S&P 500 and VIX data that included the Open, High, Close and Low features of each index. There was no aim for the project to produce multivariate data for each time series, but it is an interesting area to explore and it would be an advantage if the model could produce multivariate data with minimal adjustments. The model was in no way adapted for the task except that the output of the generator was expanded to 4 channels instead of 1 and the same thing for the discriminator's input. An example of some generated sample paths are shown in Figure 4.18. As one can see, the spread between each variable is too large in comparison to the real data. The synthetic data in some cases has some huge intraday movements that are very unrealistic. Looking at the PCA plot in Figure 4.19 the comparison of the synthetic and real data features can be seen. The synthetic data has a lot more variety between the different features while the real data is very consistent, which would explain why it moves in a very similar fashion and so the intraday difference is a lot less.

Having analysed the datasets as well as providing a more difficult test, the effect of the dates chosen for the datasets is presented in the next section as this can have a large impact on the generated data.

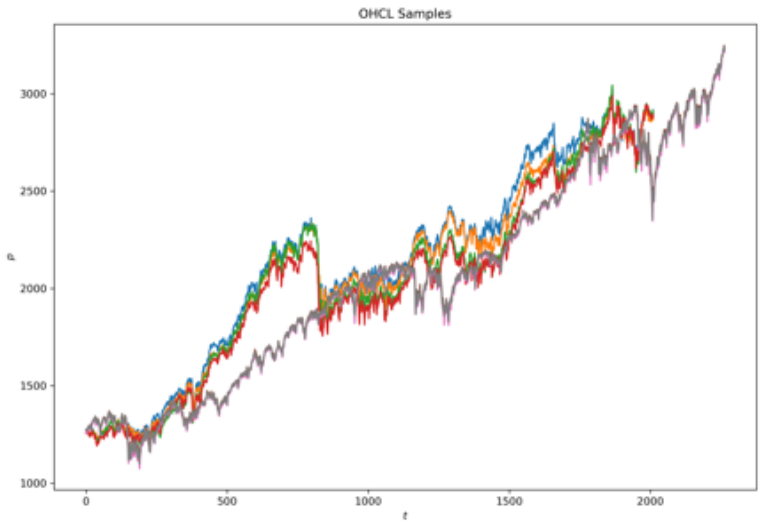


Figure 4.18: Comparison of synthetic OHCL samples and the real data. The synthetic samples are the red, green, orange and blue lines while the rest are for the real data.

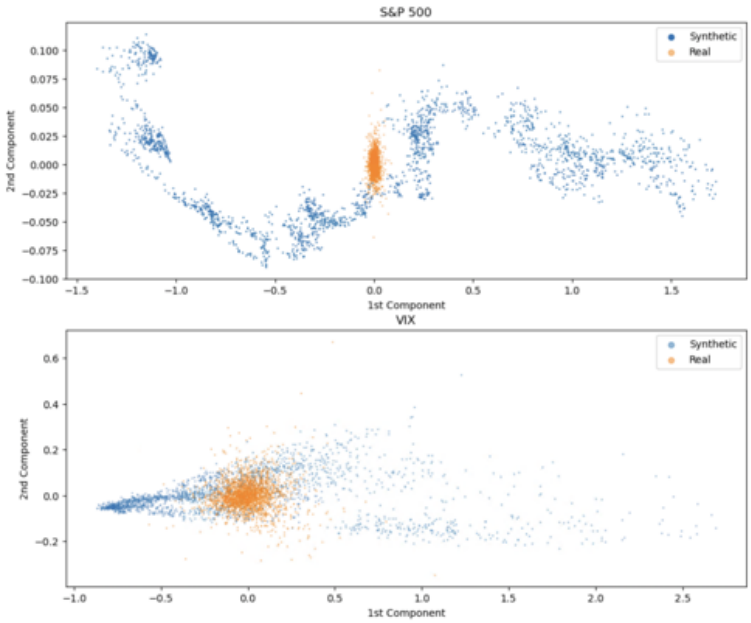


Figure 4.19: PCA plot to show the interactions between the features and compare the synthetic data to the real data

4.7 Effect of the Dataset's Time Frame

For a few of the models in the literature they use time frames that encompass only a 'bull' market or a time period that have few shocks such as between 2012 to 2019. However, this means that the model will never have seen an economic crisis and so if this is the case can one reasonably test their strategies on the synthetic data and be confident that it correctly represents shocks? Large shocks can cause huge losses to quant funds if the strategies have never been tested through such scenarios. Therefore, for the main results that are presented above, the training data extends between 1992 to 2020, but it is important to highlight that different time periods possess different statistical properties which can alter the model's ability to replicate the data and also means when comparing models with other work in the literature it becomes tricky as the quality of the model can be over inflated by choosing nicer regions. For a longer time period there will be many different scenarios and so it is no surprise that the model performs worse when the training data is in fact not uniform in its statistical properties. If the results are compared to that of a model trained on only data from 2011 to 2019, as shown in Figure 4.20, there is a stark contrast in performance. For the model trained on data from 2011 to 2019 the absolute autocorrelations are nearly spot on. As the model, in simple terms, is just some weights that slide across a noise prior and perform convolutions to produce temporal dependence, it will struggle to fit a distribution which changes over time. This is due to it having to find a set of weights that maximises the performance

for all scenarios and so it is expected that when comparing the metrics with the real data that the synthetic data may seem to perform poorly. However, from a practitioner's point of view one can be more confident in the synthetic data being able to produce scenarios that more closely represent shocks.

For all previous results FTS-GAN has been the same, but the next section now uses an altered version that allows a conditioned input, as explained in section 3.2, with the aim of improving the quality of the results and being able to adjust the scenarios of the synthetic data.

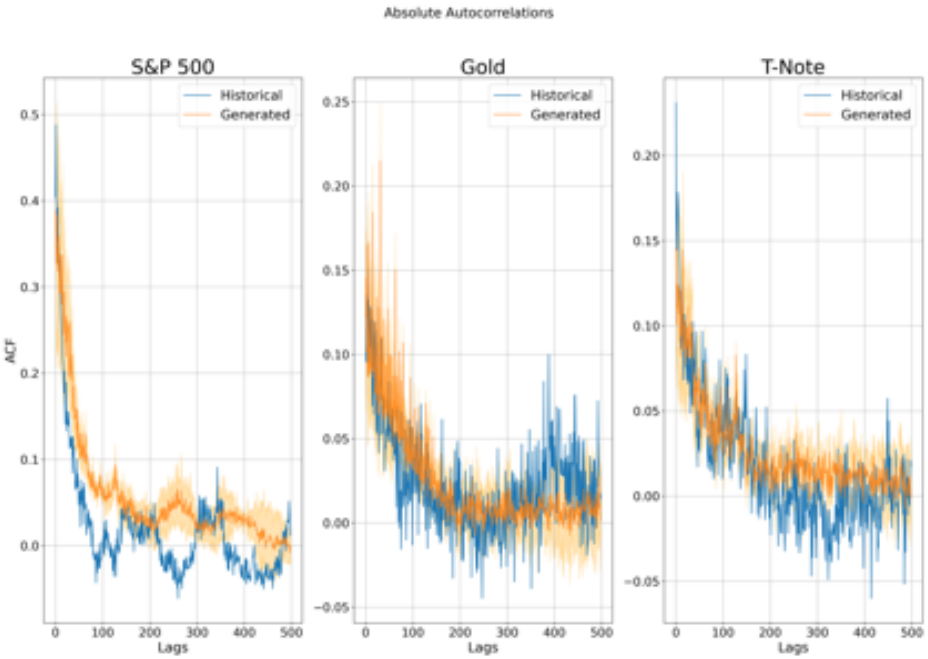


Figure 4.20: Serial autocorrelations for the three timeseries

4.8 Testing a Conditioned FTS-GAN

One of the aims of the project was to implement some form of conditioning to the model so that a user would be able to apply labels to the data to guide the training in a specific way. For the testing in this project the drawdown was used. For each window in the training data it was assigned a value of 0 or 1 depending on whether the maximum drawdown for that sample surpassed a specific threshold. The hope was that when it came to generating the synthetic samples the size of drawdown that occurred could be changed and so scenarios with large or small shocks could be generated. However, this was not the case and the synthetic samples possessed no easily distinguishable difference depending on our input feature, but it was found that training the model with the labels produced results that had

statistical properties that more closely resembled the real data. In Figure 4.21 the return distributions are shown for the conditional case. If this is compared to the results in Figure 4.15 it can be seen that there is an improvement with the data that has been 'conditioned', has a much better distribution. Although the conditioning may not have worked as anticipated it shows that there is potential for increasing the performance of the model by way of some external information; one of the reasons that conditioning was introduced into GANs was that it yielded much better performance.

At the moment all the testing has focused on comparing the statistical properties of the synthetic data with the real data, but now another test will be performed on the VIX data to test its use for deep learning applications.

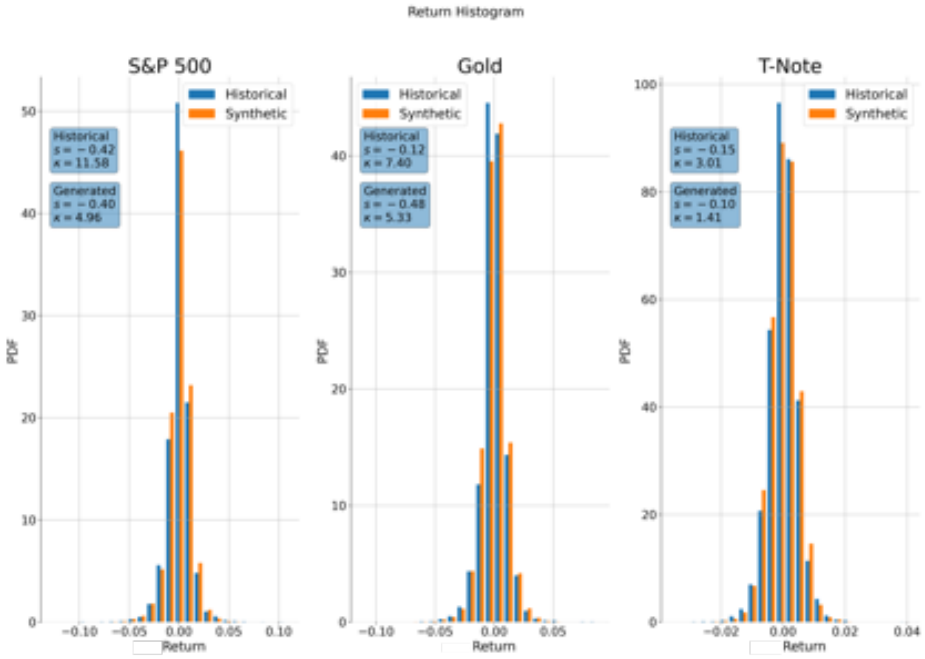
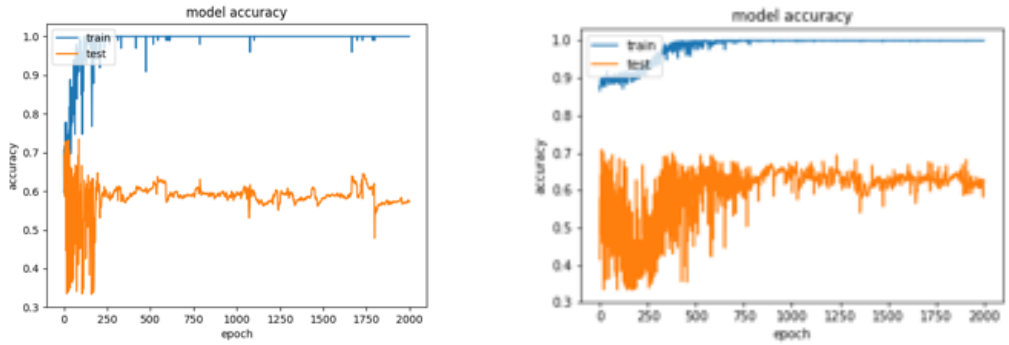


Figure 4.21: Histogram for synthetic samples of the S&P 500, Gold and T-Note where the model has been conditioned on the drawdown of the training data.

4.9 Train on Synthetic, Test on Real Data

The 'train on synthetic, test on real' method is now presented, showing both the quality of the VIX data but also highlighting that this method can be used in conjunction with the metrics to rate the quality of synthetic time series data. The experiment was carried out upon the VIX data where it was partitioned into windows of 30 days with the aim of the classifier trying to predict whether the VIX would be over a certain value in 20 days time. The accuracy for the training and testing on

the real data is about 0.6, shown in Figure 4.22a. The synthetic data used for the next part is the set shown in section 4.4. When the synthetic data is then added to the training process the accuracy is marginally higher as seen in Figure 4.22b. This is a positive sign as it first highlights that the synthetic data is in fact of good enough quality to be used in applications where there is limited data but also, as the accuracy is higher, it can be utilised in cases where one wants to mitigate overfitting.



(a) Accuracy for training and testing on real data.

(b) Accuracy for training on real, testing on synthetic.

Figure 4.22: Results for the 'train on synthetic, test on real' evaluation method on VIX data.

Chapter 5

Conclusions & Further Research

5.1 Conclusions

This project has produced a novel GAN framework, FTS-GAN, that utilises Temporal Convolutional Networks to capture the temporal dependence of multiple correlated time series and be able to generate synthetic samples that follow the learnt distribution. Although GANs can be difficult to train, with many problems as highlighted throughout this project, they produce excellent results and are one of the stronger type of generative frameworks available, especially when it comes to financial time series. FTS-GAN has built upon and further extended the use of TCNs on multiple time series, and produced a model that gives a strong foundation for future work on the generation of correlated assets without any issue of mode collapse.

FTS-GAN has been tested on two datasets and has shown good results in both cases. For the S&P 500 and VIX case the model can accurately recreate the statistical properties, as well as correctly reproducing the correlations between the two time series. However, the recreation of the VIX prices is currently weak and a lot of price data looks less realistic than one would like. The model may need some external conditioning to improve the shape but this is left for further research. Although the price data has some extreme samples, when fed into the ResNet classifier it produces strong results and even manages to marginally increase the accuracy. Therefore, the main takeaway from this dataset is that the model performs well at replicating the statistics and creating samples for trend following assets but, struggles for the mean-reverting series.

The second dataset posed a more difficult challenge due to the time series having changing statistical properties, both individually and together, that change throughout the time period used for training. This meant that the model had a harder time

to fit the model as it tries to generalise the temporal dependance of the returns. However, the results are plausible and for a shorter time period without shocks the model has no problem being able to perfectly fit the absolute autocorrelations. This poses a question about synthetic financial data and how many years of data should a practitioner be training their model on, as if the statistics of the market are changing data from over 30 years ago may not be still applicable; this is left for further research. Correlations between the time series are strong for two of them but are weak for the Gold index and the T-Note; however, FTS-GAN does produce results that are plausible and could be used to easily create daily data for the back testing of trading strategies. The S&P 500 performs worse in the 3 series case than the 2 series case, but this may be down to the indexes it is being grouped with, and not just the increase in dimension. The S&P 500 and VIX dataset are always going to have relatively consistent correlations across many years, due to the VIX being priced based off the SPX, while for the 3 time series case the correlations are less and will vary a lot more over the time frame. Applying conditioning to FTS-GAN for this dataset helped improve the statistical properties but did not give the user any control over the generation process. Further research that could be done will be covered in the next section.

5.2 Further Research

Although FTS-GAN has laid the foundations for a possible model that can be used for generating correlated time series data there are potential improvements that could be researched to further improve the quality of the data. Outside of the actual neural network and framework there are two other areas that could be focused upon to improve the model. First, the input into the model; this could either be done by conditioning or whether some other random prior could be used to improve fat tails or other aspects of the generation process. The second area is the post processing of the returns that come out of the model. A possible method such as DRS was tried, but whether further filtering can be applied to reduce the samples that look most unrealistic would be an interesting area to explore, especially as one would need to investigate the potential biases that could occur, such as a practitioner only choosing samples that have a certain level of volatility. However, this means that they may remove samples that possess interesting scenarios of 'black swan' events, which would be very useful to train their model on.

Although attempted within this research, being able to apply conditioning to TCN based models is a key area to be further explored so that the users have the

ability to adjust the scenarios generated. This would allow one to tailor data to certain specifications, which would provide the ability to generate many more datasets. Although in this project the results were less than optimal it has emphasised that the use of external features improves the ability of the model to fit data. FTS-GAN was not designed around producing OHCL data but the results may be improved by incorporating some of the architecture from GANs such as MTSS-GAN [75] that are designed to deal with multivariate data.

Finally, the last suggested area of research would be a method for being able to evaluate the quality of multidimensional time series data either via some neural network or a metric that is not as general as just the correlation. This would allow us to ensure that the time series interact correctly, like they would in the real markets. This could be implemented by a more complex classifier or could even be a network such as Börjesson and Singull's [60] and see how the forecasting ability changes based on synthetic data; however, a network such as this can be tricky, with a lot of parameters, and ideally a more simple approach would be preferred for quicker testing. An improvement in this field would help advocate the use of generative technique for time series data and show off the benefits it can provide.

Bibliography

[1] X. Chen, C. Xu, X. Yang, L. Song, and D. Tao, "Gated-GAN: Adversarial gated networks for multi-collection style transfer," *IEEE Transactions on Image Processing*, vol. 28, no. 2, pp. 546–560, 2019, ISSN: 10577149. DOI: 10.1109/TIP.2018.2869695.

[2] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2242–2251, 2017, ISSN: 15505499. DOI: 10.1109/ICCV.2017.244.

[3] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs," 2017. arXiv: 1706.02633.

[4] R. Cont, "Empirical properties of asset returns: stylized facts and statistical issues," *Quantitative Finance*, vol. 1, no. 2, pp. 223–236, 2001, ISSN: 1469-7688. DOI: 10.1080/713665670.

[5] E. F. Fama, "Efficient Capital Markets: II," *The Journal of Finance*, vol. 46, no. 5, pp. 1575–1617, Dec. 1991, ISSN: 00221082. DOI: 10.1111/j.1540-6261.1991.tb04636.x.

[6] B. Mandelbrot, "The Variation of Certain Speculative Prices," *The Journal of Business*, vol. 36, no. 4, p. 394, 1963, ISSN: 0021-9398. DOI: 10.1086/294632.

[7] P. Gopikrishnan, V. Plerou, L. A. Nunes Amaral, M. Meyer, and H. E. Stanley, "Scaling of the distribution of fluctuations of financial market indices," *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, vol. 60, no. 5, pp. 5305–5316, 1999, ISSN: 1063651X. DOI: 10.1103/PhysRevE.60.5305.

[8] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," pp. 1–20, 2018. arXiv: 1811.03378.

[9] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," 2010.

[10] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, vol. 28, 2013.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 1026–1034, 2015, ISSN: 15505499. DOI: 10.1109/ICCV.2015.123.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, 2015. arXiv: 1502.03167.

[13] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018. arXiv: 1802.05957.

[14] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015. arXiv: 1412.6980.

[15] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *7th International Conference on Learning Representations, ICLR 2019*, 2019. arXiv: 1711.05101.

[16] S. Ruder, "An overview of gradient descent optimization algorithms," pp. 1–14, 2016. arXiv: 1609.04747.

[17] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," 2011.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, Aug. 2018, ISSN: 1869-4101. DOI: 10.1007/s13244-018-0639-9.

[20] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," Nov. 2015. arXiv: 1511.07122.

[21] "1d convolution." (2020), [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-convolution (visited on 08/20/2021).

[22] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," 2018. arXiv: 1803.01271.

[23] F. Eckerli and J. Osterrieder, "Generative Adversarial Networks in finance: an overview," vol. 825215, no. 825215, pp. 1–10, Jun. 2021. arXiv: 2106.06364.

[24] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, Jun. 2014, ISSN: 0001-0782. DOI: 10.1145/3422622.

[25] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," pp. 1–7, 2014. arXiv: 1411.1784.

[26] L. Weng. "From gan to wgan." (2017), [Online]. Available: http://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html.

[27] F. De Meer Pardo, "Enriching Financial Datasets with Generative Adversarial Networks," no. July, 2019.

[28] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017. arXiv: 1701.07875.

[29] S. Takahashi, Y. Chen, and K. Tanaka-Ishii, "Modeling financial time-series with generative adversarial networks," *Physica A: Statistical Mechanics and its Applications*, vol. 527, p. 121 261, 2019, ISSN: 03784371. DOI: 10.1016/j.physa.2019.121261.

[30] A. Kondratyev and C. Schwarz, "The Market Generator," *SSRN Electronic Journal*, pp. 1–16, 2019, ISSN: 1556-5068. DOI: 10.2139/ssrn.3384948.

[31] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart, and J. Sun, "Generating Multi-label Discrete Patient Records using Generative Adversarial Networks," vol. 68, pp. 1–20, 2017. arXiv: 1703.06490.

[32] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–16, 2019. arXiv: 1802.04208.

[33] O. Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training," no. Nips, 2016. arXiv: 1611.09904.

[34] L. C. Yang, S. Y. Chou, and Y. H. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*, pp. 324–331, 2017. arXiv: 1703.10847.

[35] A. Gupta and J. Zou, "Feedback GAN (FBGAN) for DNA: a Novel Feedback-Loop Architecture for Optimizing Protein Functions," pp. 1–15, 2018. arXiv: 1804.01694.

[36] N. Killoran, L. J. Lee, A. Delong, D. Duvenaud, and B. J. Frey, "Generating and designing DNA with deep generative models," 2017. arXiv: 1712.06148.

[37] Y. Zhang, Z. Gan, and L. Carin, "Generating Text via Adversarial Training," 2016, ISSN: 00071447. DOI: 10.1136/bmj.2.3910.1177.

[38] M. Brundage, S. Avin, J. Clark, H. Toner, P. Eckersley, B. Garfinkel, A. Dafoe, P. Scharre, T. Zeitzoff, B. Filar, H. Anderson, H. Roff, G. C. Allen, J. Steinhardt, C. Flynn, S. Ó. HÉigeartaigh, S. Beard, H. Belfield, S. Farquhar, C. Lyle, R. Crootof, O. Evans, M. Page, J. Bryson, R. Yampolskiy, and D. Amodei, "The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation," *arXiv preprint arXiv:1802.07228*, no. February 2018, p. 101, Feb. 2018. arXiv: 1802.07228.

[39] L. Simonetto, "Generating spiking time series with Generative Adversarial Networks: an application on banking transactions," 2018.

[40] A. Koshiyama, N. Firoozye, and P. Treleaven, "Generative adversarial networks for financial trading strategies fine-tuning and combination," *Quantitative Finance*, vol. 21, no. 5, pp. 797–813, 2021, ISSN: 14697696. DOI: 10.1080/14697688.2020.1790635.

[41] A. Sethia, R. Patel, and P. Raut, "Data augmentation using generative models for credit card fraud detection," *2018 4th International Conference on Computing Communication and Automation, ICCCA 2018*, pp. 1–6, 2018. DOI: 10.1109/CCAA.2018.8777628.

[42] T. Leangarun, P. Tangamchit, and S. Thajchayapong, "Stock Price Manipulation Detection using Generative Adversarial Networks," *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018*, pp. 2104–2111, 2019. DOI: 10.1109/SSCI.2018.8628777.

[43] Y. Zhu, G. Mariani, and J. Li, "Pagan: Portfolio Analysis with Generative Adversarial Networks," *SSRN Electronic Journal*, 2021. DOI: 10.2139/ssrn.3755355.

[44] Y. Kim, D. Kang, M. Jeon, and C. Lee, "GAN-MP hybrid heuristic algorithm for non-convex portfolio optimization problem," *Engineering Economist*, vol. 64, no. 3, pp. 196–226, 2019, ISSN: 15472701. DOI: 10.1080/0013791X.2019.1620391.

[45] G. Marti, "CORRGAN: Sampling Realistic Financial Correlation Matrices Using Generative Adversarial Networks," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2020-May, pp. 8459–8463, 2020, ISSN: 15206149. DOI: 10.1109/ICASSP40776.2020.9053276.

[46] H. Sun, Z. Deng, H. Chen, and D. C. Parkes, "Decision-Aware Conditional GANs for Time Series Data," pp. 1–24, 2020. arXiv: 2009.12682.

[47] X. Zhou, Z. Pan, G. Hu, S. Tang, and C. Zhao, "Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets," *Mathematical Problems in Engineering*, vol. 2018, 2018, ISSN: 15635147. DOI: 10.1155/2018/4907423.

[48] W. Wu, F. Huang, Y. Kao, Z. Chen, and Q. Wu, "Prediction method of multiple related time series based on generative adversarial networks," *Information (Switzerland)*, vol. 12, no. 2, pp. 1–16, 2021, ISSN: 20782489. DOI: 10.3390/info12020055.

[49] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni, "TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks," *Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020*, pp. 33–43, 2020. DOI: 10.1109/BigData50022.2020.9378139.

[50] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein GANs," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 5768–5778, 2017, ISSN: 10495258. arXiv: 1704.00028.

[51] A. Jolicoeur-Martineau, "The relativistic discriminator: A key element missing from standard GaN," *7th International Conference on Learning Representations, ICLR 2019*, 2019. arXiv: 1807.00734.

[52] M. Wiese, R. Knobloch, R. Korn, and P. Kretschmer, "Quant GANs: deep generation of financial time series," *Quantitative Finance*, vol. 20, no. 9, pp. 1419–1440, 2020, ISSN: 14697696. DOI: 10.1080/14697688.2020.1730426.

[53] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, pp. 1–11, 2019, ISSN: 10495258.

[54] A. Borji, "Pros and cons of GAN evaluation measures," *Computer Vision and Image Understanding*, vol. 179, pp. 41–65, 2019, ISSN: 1090235X. DOI: 10.1016/j.cviu.2018.10.009.

[55] A. Yale, S. Dash, R. Dutta, I. Guyon, A. Pavao, and K. P. Bennett, "Assessing privacy and quality of synthetic health data," *ACM International Conference Proceeding Series*, 2019. DOI: 10.1145/3359115.3359124.

[56] V. Khrulkov and I. Oseledets, "Geometry score: A method for comparing generative adversarial networks," *35th International Conference on Machine Learning, ICML 2018*, vol. 6, pp. 4114–4122, 2018. arXiv: 1802.02664.

[57] C. R. García-Alonso, L. M. Pérez-Naranjo, and J. C. Fernández-Caballero, "Multiobjective evolutionary algorithms to identify highly autocorrelated areas: the case of spatial distribution in financially compromised farms," *Annals of Operations Research*, vol. 219, no. 1, pp. 187–202, Aug. 2014, ISSN: 0254-5330. DOI: 10.1007/s10479-011-0841-3.

[58] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," pp. 1–15, 2016. arXiv: 1609.03499.

[59] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," *Proceedings of the International Joint Conference on Neural Networks*, pp. 1578–1585, 2017. DOI: 10.1109/IJCNN.2017.7966039.

[60] L. Börjesson and M. Singull, "Forecasting financial time series through causal and dilated convolutional neural networks," *Entropy*, vol. 22, no. 10, pp. 1–20, 2020, ISSN: 10994300. DOI: 10.3390/e22101094.

[61] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "ECA-Net: Efficient channel attention for deep convolutional neural networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 11 531–11 539, 2020, ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.01155.

[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 770–778, 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

[63] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled Generative Adversarial Networks," *Advances in Neural Information Processing Systems*, pp. 469–477, Nov. 2016, ISSN: 10495258. arXiv: 1611.02163.

[64] S. Azadi, A. Odena, C. Olsson, T. Darrell, and I. Goodfellow, "Discriminator rejection sampling," *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–16, 2019. arXiv: 1810.06758.

[65] S. Sinha, Z. Zhao, A. Goyal, C. Raffel, and A. Odena, "Top-k Training of GANs: Improving GAN Performance by Throwing Away Bad Samples," no. NeurIPS, 2020, ISSN: 10495258. arXiv: 2002.06224.

[66] Y. LeCun, L. Bottou, G. B. Orr, and K. .-. Müller, "Efficient BackProp," in 1998, pp. 9–50. DOI: 10.1007/3-540-49430-8_2.

[67] G. V. Steeg. "Gaussianzie." version latest. (2015), [Online]. Available: `https://github.com/gregversteeg/gaussianize`.

[68] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2623–2631, 2019. DOI: `10.1145/3292500.3330701`.

[69] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019, ISSN: 1573756X. DOI: `10.1007/s10618-019-00619-1`.

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[71] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: `10.1038/s41586-020-2649-2`.

[72] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: `10.1038/s41592-019-0686-2`.

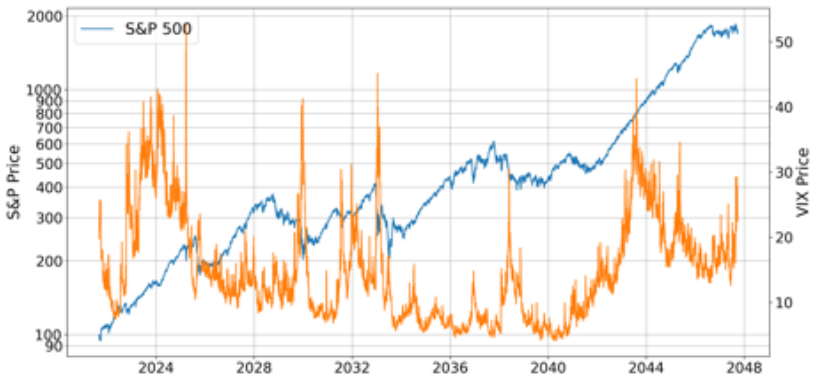
[73] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: `10.5281/zenodo.3509134`.

[74] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56–61. DOI: `10.25080/Majora-92bf1922-00a`.

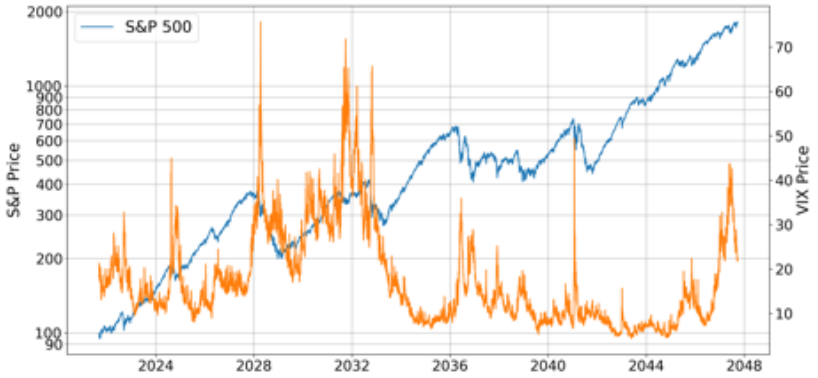
[75] D. Snow, "MTSS-GAN: Multivariate Time Series Simulation Generative Adversarial Networks," *SSRN Electronic Journal*, 2020, ISSN: 1556-5068. DOI: `10.2139/ssrn.3616557`.

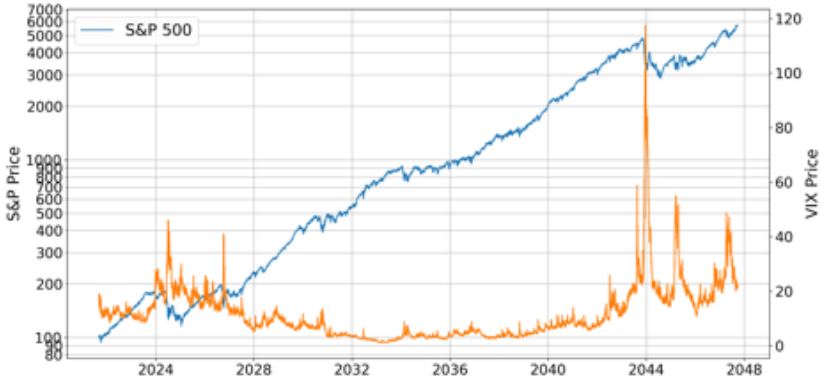
Appendix A

A.1 S&P and VIX



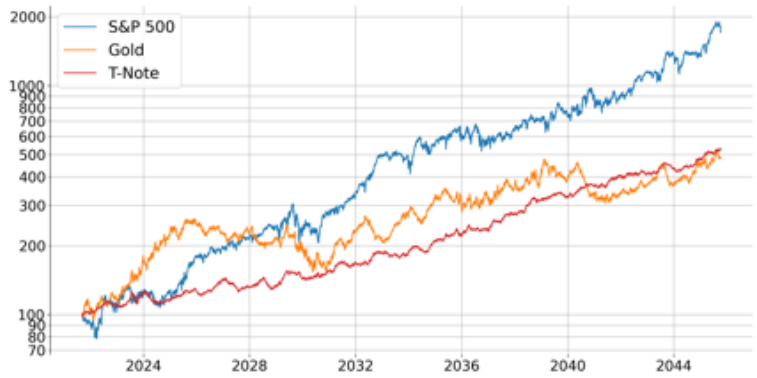

S&P 500
S&P Price
VIX Price
1000
900
800
700
600
500
400
300
200
100
90
200
175
150
125
100
75
50
25
0
2024
2028
2032
2036
2040
2044
2048


S&P 500
S&P Price
VIX Price
2000
1000
900
800
700
600
500
400
300
200
100
90
70
60
50
40
30
20
10
2024
2028
2032
2036
2040
2044
2048


S&P 500
S&P Price
VIX Price
7000
6000
5000
4000
3000
2000
1000
900
800
700
600
500
400
300
200
100
80
120
100
80
60
40
20
0
2024
2028
2032
2036
2040
2044
2048

A.2 S&P 500, Gold and T-Note


S&P 500
Gold
T-Note
2000
1000
900
800
700
600
500
400
300
200
100
90
80
70
2024
2028
2032
2036
2040
2044


S&P 500
Gold
T-Note
2000
1000
900
800
700
600
500
400
300
200
100
90
80
70
2024
2028
2032
2036
2040
2044


S&P 500
Gold
T-Note
3000
2000
1000
900
800
700
600
500
400
300
200
100
90
80
70
2024
2028
2032
2036
2040
2044

A.3 Project Summary

Project Title	Time Series GANs for Asset Management
Company	XAI - Asset Management
Industrial Supervisor	Federico Fontana
Supervisor Email	federico.fontana@xai-am.com
Description of Work	Review the current literature and then reproduce some of the current frameworks in Python. Identify and improve on weaknesses of them.
Own Contribution	I have produced a novel GAN framework that builds upon a TCN based network architecture that allows the ability to generate multidimensional financial market data.

Table A.1: Project Summary