FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

GASTeN(time series): Generative Adversarial Stress Test Networks for Time Series

Afonso Duarte de Carvalho Monteiro

Mestrado em Engenharia Informática

Supervisor: Carlos Soares

Co-Supervisor: Yassine Baghoussi

Co-Supervisor: Inês Gomes

November 20, 2024

GASTeN(time series): Generative Adversarial Stress Test Networks for Time Series

Afonso Duarte de Carvalho Monteiro

Mestrado em Engenharia Informática

November 20, 2024

Resumo

Atualmente, a aprendizagem automática (ML) e a aprendizagem profunda (DL) são conceitos predominantes e fundamentais no nosso quotidiano. O aumento do poder computacional também tem contribuído para modelos de DL cada vez mais complexos. As questões de Previsão de Séries Temporais (TSF) também beneficiaram destes avanços, como a aplicação de algoritmos de Análise de Expansão de Base Neural (N-BEATS) [21]. Estes e outros algoritmos de previsão são complexos e difíceis de analisar e descrever. Como tal, é rara uma descrição aprofundada desses algoritmos, quando implementados noutros produtos. Este facto faz com que os consideremos quase como caixas negras, difíceis de compreender na íntegra. No entanto, como Mitchell *et al.* salientou, é necessário que haja mais transparência na construção dos modelos de aprendizagem automática [19] para que sejam mais fáceis de compreender.

Para atingir o nosso objetivo, vamos adaptar a estrutura GASTeN [4] para TSD. Esta estrutura, baseada no trabalho de Cunha *et al.*, utiliza os princípios da geração de dados sintéticos para testar modelos de classificadores de imagens, fornecendo imagens que são propositadamente difíceis de classificar.

Propomos uma abordagem baseada em Redes Adversárias Generativas (GAN) [17, 7, 6], mais especificamente GASTeN [4] que, dado um modelo aprendido numa série temporal específica, gera uma extensão realista dessa série temporal em que o modelo não consegue fazer previsões exactas. Esta abordagem abre caminho para uma compreensão mais profunda dos modelos TSF, porque estamos a descobrir os limites das suas capacidades.

Planeamos validar empiricamente a nossa abordagem com séries temporais de referência, utilizando um quadro de avaliação desenvolvido explicitamente para este fim. Isto implica a utilização de métodos de previsão de referência, como o ARIMA, treinados em dados reais antes de a arquitetura baseada em GAN gerar segmentos de dados de séries temporais (TSD). Exploraremos as configurações dos hiperparâmetros para analisar a confiança na previsão do método Time Series Forecasting (TSF). As nossas experiências visam demonstrar a viabilidade de gerar segmentos de TSD consistentes com dados anteriores. Para além disso, procuramos identificar situações em que a confiança do modelo TSF nas suas previsões começa a diminuir.

Concluímos que, neste caso, conseguimos gerar Extensões de Séries Temporais realistas, mas que essas mesmas extensões não foram capazes de tornar os dados resultantes desafiantes para os previsores. No entanto, existe potencial neste quadro e podem ser efectuados mais testes.

Abstract

Nowadays, Machine Learning (ML) and Deep Learning (DL) are predominant and fundamental concepts in our daily lives. The increase in computing power has also contributed to increasingly complex DL models. Time Series Forecasting (TSF) issues have also benefited from these advances, such as the application of Neural Basis Expansion Analysis (N-BEATS) algorithms [21]. These and other forecasters' algorithms are complex and difficult to analyse and describe. As such, an in-depth description of said algorithms, when implemented in other products, is rare. This makes us perceive them almost as black boxes, difficult to fully understand. However, as Mitchell *et al.* pointed out, there needs to be more transparency in the construction of machine learning models [19] so that they are easier to understand.

To achieve our goal, we will adapt the GASTeN [4] framework for TSD. This framework, based on the work of Cunha *et al.*, uses the principles of synthetic data generation to test image classifier models, providing images that are purposely difficult to classify.

We propose an approach based on Generative Adversarial Networks (GAN) [17, 7, 6], more specifically GASTeN [4] that, given a model learned on a specific time series, generates a realistic extension to that time series where the model fails to make accurate predictions. This approach paves the way for a deeper understanding of TSF models, because we are discovering the limits on its capabilities.

We plan to empirically validate our approach with benchmark time series, using an evaluation framework developed explicitly for this purpose. This involves using benchmark forecasting methods, such as ARIMA, trained on real data before the GAN-based architecture generates time series data (TSD) segments. We will explore hyperparameter configurations to analyse the forecasting confidence of the Time Series Forecasting (TSF) method. Our experiments aim to demonstrate the feasibility of generating TSD segments consistent with previous data. In addition, we seek to identify situations in which the confidence of the TSF model in its forecasts begins to diminish.

We conclude that in this case we could generate realistic Time Series Extensions, but these same extensions were not able to make the resulting data challenging for the predictors. However, there is potential in this framework, and further testing can be done.

Acknowledgments

First, I would like to thank my supervisors, Carlos Soares, Inês Gomes, and Yassine Baghoussi, for guiding me throughout this process.

A thanks to my family, especially my mother and sister, for their support through this journey.

Then, to my (discord) friends, who have been decisive and a pleasant surprise these years have given me. Bruno (Forever), João Gil (Monkin), Leonel (Progress), Alexandre Esteves (Yuri), Filipe Fonseca (Discretive), Henrique (forever DM), Hugo (Frix) and also Ariana. Your support was and still is fateful. There is nothing I can do to express my gratitude for what each one has done for me.

Last, but not least, my father, who, despite not being with us, was the inspiration for the beginning of my professional choice and still is a leading figure in my life.

Dad, wherever you are, your son made it. I hope I made you proud.

Afonso Monteiro

"It is hard to fail, but it is worse never to have tried to succeed."

Theodore Roosevelt

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ACD	Average Confusion Distance
ACF	Auto-Correlation Function
ACIW	Average Confidence Interval Width
ADF	Augmented Dickey-Fuller
AI	Artificial Intelligence
AR	Auto-Regressive
ARIMA	Auto-Regressive Integrated Moving Average
ADT	Abstract Data Type
CD	Confusion Distance
CLS	Confined Latent Space
CNN	Convolutional Neural Network
DL	Deep Learning
DCGAN	Deep Convolutional Generative Adversarial Network
DCT	Dilated Convolutional Transformer
DTW	Dynamic Time Warping
DNN	Deep Neural Networks
ES-RNN	Exponential Smoothing - Recurrent Neural Network
EDA	Exploratory Data Analysis
FFORMA	Feature-based FORecast Model Averaging
FFORMA-N	Feature-based FORecast Model Averaging using Neural Networks
FFORMS-G	Feature-based FORecast Model Selection using Gradient Boosting
FFORMS-R	Feature-based FORecast Model Selection using Random Forest
FID	Frechét Inception Distance
GAN	Generational Adversarial Networks
GASTeN	Generative Adversarial Stress Test Networks
GASTeN-TS	Generative Adversarial Stress Test Networks - Time Series
ISA	Instance Space Analysis
LCIR	Logarithmic Confidence Interval Ratio
LSTM	Long-Short Term Memory
MA	Moving Average
ML	Machine Learning
N-BEATS	Neural Basis Expansion Analysis
NN	Neural Network
OOD	out-of-distribution
PACF	Partial Auto-Correlation Function
rAAE	Regularised Adversarial Autoencoder
RNN	Recurrent Neural Network
TSAD	Time Series Anomaly Detection
TAD-GAN	Time series Anomaly Detection GAN
TSF	Time Series Forecasting
TSD	Time Series Data

Chapter 1

Introduction

In recent years, the integration of machine learning and Artificial Intelligence (AI) models into daily life has increased significantly. These models have impacted personal and business activities, changing how people interact with technology and how businesses operate digitally. From personalised recommendations on streaming platforms to predictive analytics in business decisions, the influence of these models is clear [11]. This rapid growth of machine learning and AI technologies is driven by advancements in computational power and the availability of large datasets, allowing for the training of more complex models across various domains [18]. As these technologies become more widespread, the models used are becoming more complex, requiring effective strategies to address challenges such as natural language processing and image recognition [23].

1.1 Motivation

Many published machine learning models are increasingly complex and challenging to understand, earning them the moniker "black box" [20]. This lack of transparency poses significant challenges for researchers, practitioners, and end-users alike, hindering their ability to interpret model outputs effectively. Furthermore, the opacity of these models raises ethical concerns, as the growing complexity increases the risk of unintended biases and errors with unforeseen consequences [19].

Many of these models, especially in the context of time series, make assumptions about future data trends. Yet, deviations from these assumptions are common, highlighting the need for a distinct and context-specific understanding [2]. This reliance on assumptions raises questions about the reliability and robustness of model predictions, particularly in dynamic and evolving environments. Although the separation of available data into training and testing enables us to estimate how the model is going to behave on data unseen in the process of learning that model, these estimates are still based on the same assumption.

With these risks in perspective, there is a critical need for research efforts to enhance the transparency, interpretability, and accountability of machine learning and AI models. A method

proposed for stress testing image classification models, GASTeN serves as a testament to the importance of systematic evaluation and validation in AI model development [4]. This framework tests image classification models by incorporating them into the Generator's Loss function calculation, which is modified. While GASTeN focuses on image classification, its rigorous testing and evaluation principles can inspire similar approaches tailored to the unique challenges of time series forecasting. The quest for robust and reliable models has become increasingly paramount in the rapidly evolving landscape of artificial intelligence (AI).

Recently, more effort has been directed towards developing techniques to stress-test AI applications to uncover vulnerabilities and enhance performance [26], therefore enhancing transparency. However, despite these advancements, a notable gap exists in time series forecasting, as there is a lack of work in this area. It is important that we understand the limits where the models start to decline in forecasting effectiveness, since it is an important step to make them reliable. Failures or inaccuracies in forecasting can have far-reaching consequences, impacting decision-making processes and potentially leading to substantial financial losses or operational disruptions.

1.2 Problem definition

Another popular application of TSF models is in industries where accurate forecasts are a necessity like energy, and healthcare. Still, these models may even have problems in the face of uncertainty and forecasting in ambiguous scenarios. Forecasts and predictions are commonly associated with confidence intervals, however it is not unheard of for the confidence intervals to become too narrow, which can result in little indication of how certain (or uncertain) those predictions really were.

One of the key challenges is having a process that systematically tests these models for unpredicted conditions in order to ensure that they are robust and anti-fragile which can handle ambiguity. Most existing methods for stress-testing focus on domains like image classification where the stress test is mainly done with synthetic data. You can say that there is no such testing framework for time series data.

The issue we address is how to synthetically create time series extensions that are realistic, yet ambiguous as possible in order to test forecasting models and force them out of their comfort zone. This will allow us to see how far a model can be made useful in predicting reality-induced limitations, and we can make it more accurate.

1.3 Objectives

The primary objective of this research is to develop methodologies explicitly tailored for stress-testing time series forecasting models is imperative. By subjecting these models to testing under various conditions and scenarios, researchers and practitioners can identify weaknesses, improve performance, and enhance the overall reliability of forecasting systems.

This work has a few challenges. First, we need to find a way to adjust the data generation. This will allow us to synthesize data that can be used as an extension of the real data, within but independent of any process occurring during the GAN's training. Then, we need to reliably adapt a relevant loss function by adding a new metric resulting from the output of a predictor model.

Furthermore, another key objective is empirically validating the proposed adaptation of GAS-TeN for TSF. Through experimentation and evaluation, the efficacy and performance of the adapted methodology will be assessed across a diverse range of time series forecasting scenarios. By empirically validating the proposed method, this research seeks to provide empirical evidence of its effectiveness and suitability for stress-testing TSF models.

1.4 Document structure

This dissertation is structured as follows:

- Chapter 2 explains the main concepts and background that will serve as the basis for the work.
- Chapter 3 states the solution perspective, detailing the methodology and validation strategies.
- Chapter 4 exposes the experimentation setup used to validate the proposal.
- Chapter 5 explores the study's results and presents an analysis of them.
- Chapter 6 draws conclusions from the previous chapters.

Chapter 2

Background and related work

This chapter introduces key concepts for understanding this study. Section 2.1 is an overview of Time Series Forecasting (TSF), while section 2.2 will present Generational Adversarial Networks (GAN) and their variants, particularly those relevant to time series. Afterwards, in section 2.3, we will focus on work related to stress-testing machine learning applications. Finally, section 2.3.3 will expose the GASTeN framework.

2.1 Time Series Forecasting

Forecasting is a common task in data analysis across various contexts. It involves predicting future values based on historical data. To be able to be forecast, the data must be recorded chronologically, forming what is known as a time series. This is relevant in industries like finance [24], where accurate predictions can provide a competitive advantage.

Before diving into the core aspects of forecasting, it is essential to address the concept of predictability, which plays a crucial role in determining the feasibility and effectiveness of any forecast. Hyndman [13] identifies several key factors that influence the predictability of an event:

- The degree of understanding of the relevant features;
- The amount of available data;
- How similar future scenarios are likely to be to past events;
- The potential influence the forecast may have on the outcome being predicted.

Once the predictability of the event is assessed, the next step is to examine the available data. Forecasting requires formulating the problem with respect to the data we have and understanding its structure. Each time series is composed of observed values at different points in time, and these values depend on various factors or features related to the subject being forecasted.

After understanding the data, we can develop a forecasting model. A simple model might predict future values based on a combination of features. This could be represented as:

$$\hat{Y} = f(x_1, x_2, ..., x_n, \varepsilon) \tag{2.1}$$

where $\hat{Y}$ is the predicted value, $x_1, x_2, \ldots, x_n$ are the relevant features, and ε is the error term accounting for the model's uncertainty.

For time series data, the model typically uses past observations to forecast future values. In this case, the prediction for the next time step can be expressed as:

$$\hat{Y}_{t+1} = f(Y_t, Y_{t-1}, Y_{t-2}, ..., \varepsilon) \tag{2.2}$$

where Y_t represents the observed value at time t, and $\hat{Y}_{t+1}$ is the forecast for the next period.

After formulating the model, several steps are required before fitting it to the data. These include:

- **Understanding the Problem and Data**: Clearly define the objective (e.g., forecasting horizon) and explore the dataset.
- **Exploratory Data Analysis (EDA)**: Visualize the data, check for missing values or anomalies, and perform descriptive statistics.
- **Stationarity Analysis**: Assess whether the data is stationary, using visual inspections and tests such as the Augmented Dickey-Fuller (ADF) test.
- **Trend and Seasonality Analysis**: Determine whether there are trends or repeating seasonal patterns in the data, for instance, by using the Auto-Correlation Function (ACF) or Partial Auto-Correlation Function (PACF).

2.1.1 Forecasting algorithms

One of the most recognised series of models for forecasting problems is the Exponential Smoothing family of models. There are nine main sub-models, each depending on 1 of 3 seasonal and trending components as seen in Table 2.1.

Trend Component	Seasonal Component		
	N (None)	**A (Additive)**	**M (Multiplicative)**
N (None)	(N, N)	(N, A)	(N, M)
A (Additive)	(A, N)	(A, A)	(A, M)
A_d (Additive Dampened)	(A_d, N)	(A_d, A)	(A_d, M)

Table 2.1: Classification of exponential smoothing methods. Source: Adapted from [13]

Table 2.2 has the formulas associated with each method. In each formula:

- l_t denotes the series level at time t. Level reflects the long-term trend or mean of the Time-Series, without short-term noise.

- b_t denotes the slope at time t
- s_t denotes the seasonal component of the series at time t
- m is the number of seasons in a year

Additionally, $\alpha, \beta, \gamma, \phi$ are smoothing parameters. These parameters are weights that determine the influence of previous observations on the value we are forecasting, with more recent ones having a higher weight. They decrease exponentially the further back in time.

Trend	**N**	**A**	**M**
N	$\hat{y}_{t+h\|t} = \ell_t$ $\ell_t = \alpha y_t + (1-\alpha)\ell_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t - \ell_t) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = \ell_t s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)\ell_{t-1}$ $s_t = \gamma(y_t/\ell_t) + (1-\gamma)s_{t-m}$
A	$\hat{y}_{t+h\|t} = \ell_t + hb_t$ $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + hb_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}$ $s_t = \gamma(y_t - \ell_t) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = (\ell_t + hb_t)s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$ $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}$ $s_t = \gamma(y_t/\ell_t) + (1-\gamma)s_{t-m}$
A_d	$\hat{y}_{t+h\|t} = \ell_t + \phi b_t$ $\ell_t = \alpha y_t + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)\phi b_{t-1}$	$\hat{y}_{t+h\|t} = \ell_t + \phi b_t + s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)\phi b_{t-1}$ $s_t = \gamma(y_t - \ell_t) + (1-\gamma)s_{t-m}$	$\hat{y}_{t+h\|t} = (\ell_t + \phi b_t)s_{t+h-m(k+1)}$ $\ell_t = \alpha(y_t/s_{t-m}) + (1-\alpha)(\ell_{t-1} + \phi b_{t-1})$ $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)\phi b_{t-1}$ $s_t = \gamma(y_t/\ell_t) + (1-\gamma)s_{t-m}$

Table 2.2: Comparison of trend and seasonal components in exponential smoothing models.

Another popular model is the Auto-Regressive Integrated Moving Average (ARIMA) model. This combines Auto-Regressive (AR) and Moving Average (MA) models. Firstly, the AR model is based on the idea that the future values of a time series can be predicted using a linear combination of past observations. This can be represented using the following equation:

$$FT(S)_t = c + \phi_1 FT(S)_{t-1} + \phi_2 FT(S)_{t-2} + ... + \phi_p FT(S)_{t-p} + \varepsilon_t \tag{2.3}$$

where ϕ_i are the auto-regressive coefficients, p is the order of auto-regression, c is a constant and ε_t is the white noise error term.

Secondly, the main theory behind MA models is that the current value of a time series is a linear combination of past white noise error terms. It can be represented as:

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon t - 1 + \theta_2 \varepsilon t - 2 + ... + \theta_q \varepsilon t - q$$

where μ is the mean of the series, θ_i are the moving average coefficients, q is the order of the moving average.

Finally, the combination of both models plus the inclusion of *differencing* gives us the ARIMA model:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} + \varepsilon_t + \varepsilon_t + \theta_1 \varepsilon t - 1 + \theta_2 \varepsilon t - 2 + ... + \theta_q \varepsilon t - q \tag{2.4}$$

2.1.2 Other approaches

More recently, Deep Learning (DL) has also contributed to better results in this problem, nominally implemented in Long-Short Term Memory (LSTM), a type of Recurrent Neural Network (RNN). This type of RNN is designed to tackle some issues that normal RNNs have in capturing long-term dependencies in sequential data, the vanishing gradient problem [10].

Its key features include:

- **Memory cells**: self-contained unit that can store information over long periods, allowing the network to remember and access information from distant time steps.
- **Gating Mechanisms**: LSTMs use gating mechanisms, including input, forget, and output gates, to regulate the flow of information through the network. These gates enable LSTMs to remember or omit information, effectively handling long sequences selectively.
- **Long-Term and Short-Term Memory:** The architecture allows the network to distinguish between short-term and long-term memory, capturing immediate and enduring input data dependencies.

Even though statistical models like ARIMA or Exponential Smoothing-based and DL-based models are the most recognisable in Time Series Forecasting, other models are being formulated and studied more recently. This is the case for ensemble learning methods. Cawood [3] conducted research comparing some proposed ensemble models, namely Feature-based FORecast Model Selection using Gradient Boosting (FFORMS-G) and Feature-based FORecast Model Averaging using Neural Networks (FFORMA-N).

Using the M-competition time series test set (a dataset comprising numerous time series with varying frequencies, including yearly, quarterly, monthly, weekly, daily, and hourly intervals, used in the M4 competition (`https://github.com/Mcompetitions/`)), the research concluded that in fact, the proposed methods could boost the predictive power of advanced forecasting methods, like Exponential Smoothing - Recurrent Neural Network (ES-RNN) and Neural Basis Expansion Analysis (N-BEATS) [21].

2.2 Generative Adversarial Networks

As proposed by Goodfellow *et al.* [7, 6], Generative Adversarial Networks is a family of DL models. It generally involves two Neural Networks (NN), a Generator (G) and a Discriminator (D). The first takes random noise as input, generates synthetic data and learns to create data that is increasingly similar to real data. In contrast, the second receives real and artificial data as input, classifies them as real or fake, and learns to improve its ability to distinguish between accurate and generated data. The goal is for G to produce indistinguishable data from actual data, and D's task is to differentiate between accurate and generated data correctly. An example diagram is shown in Figure 2.1:

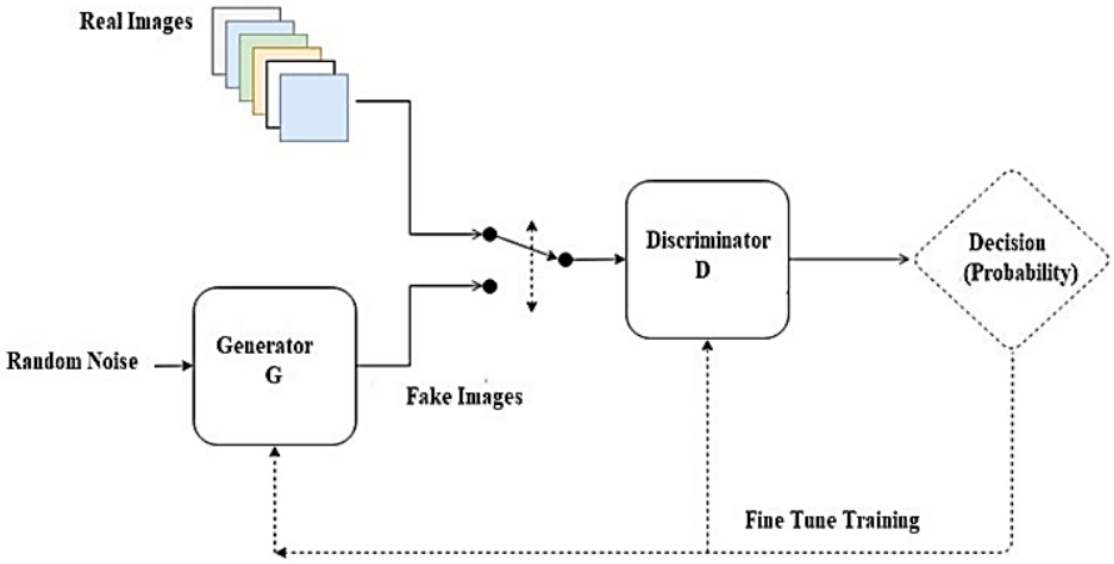

Figure 2.1: Example GAN diagram, taken from [14]

The original equation representing this relation [7] is:

$$\min_G \max_D [E_{x\sim p_d}[\log(D(x_{real}))] + E_{z\sim p_z}[\log(1 - D(G(z))]] \tag{2.5}$$

where $z \sim p_z$ is the random noise sample from the pool mentioned previously, and $x \sim p_d$ is the representation of actual samples.

Both networks' mission is to maximise their efficacy. That is achieved by maximising the loss function. The discriminator function is represented as Equation 2.2.

$$L_D = -E_{x\sim p_d}[\log(D(x))] - E_{z\sim p_z}[\log(1 - D(G(z))] \tag{2.6}$$

Further in the study, Goodwill *et al.* put forward the idea of maximising the odds of $G(z)$ being accurate rather than minimising the chances of being fake ($\log(1 - D(G(z)))$. As such, $\log(1 - D(G(z))$ nears the value of 0 (as $D(G(z)) \to 0$). Hence, the Generator is conditioned to minimise:

$$L_G = -E_{z\sim p_z}[\log(D(G(z))] \tag{2.7}$$

GAN is an unsupervised learning method, generally applied for image generation tasks [7]. However, it has encountered some difficulties [7, 6]. Initially, when two neural networks are updated concurrently, reaching equilibrium may be challenging, leading to potential training non-convergence. Another prevalent issue in GAN training is mode collapse, where the generator struggles to produce varied data, consistently mapping distinct noise samples to the same output [6].

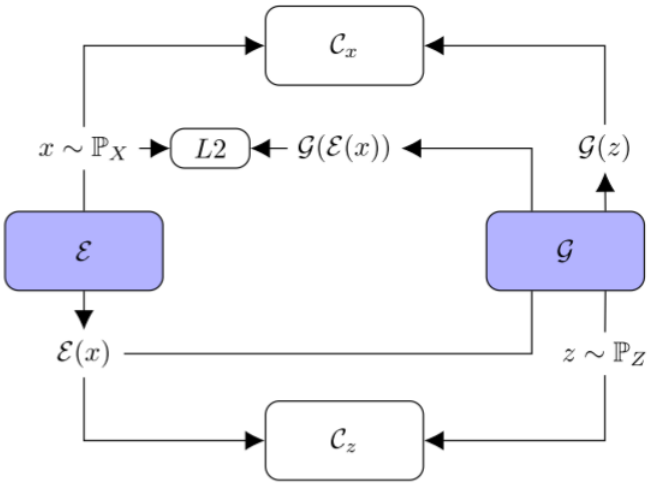

Figure 2.2: TADGAN architecture, adapted from [5]. C is the equivalent of the Discriminator D present in GAN architectures. Generator ε functions as an Encoder, mapping time series sequences, while Generator G acts as a decoder, transforming the latent space into the reconstructed time series. Discriminator C_x is tasked with distinguishing between real-time series sequences from X and the generated time series sequences from $G(z)$. Conversely, Discriminator C_z evaluates the effectiveness of the mapping.

2.2.1 Time Series GAN Variants

As previously mentioned, even though there are examples of GAN system variations built for TSF tasks, the primary purpose is Time Series Anomaly Detection (TSAD). In this section, we address the most relevant ones.

2.2.1.1 Time series Anomaly Detection GAN

The **Time series Anomaly Detection GAN (TADGAN)** [5] introduces an unsupervised anomaly detection approach that utilises Generative Adversarial Networks (GANs) with LSTM Recurrent Neural Networks as base models. The model is trained with cycle consistency loss by encoding and decoding time series sequences and measuring the difference between the original and the decoded sequence. This approach enables effective time-series data reconstruction.

It has been shown to outperform baseline methods in anomaly detection. The test was done across 11 datasets from NASA, Yahoo, Numenta, Amazon, and Twitter sources. The TADGAN's architecture is illustrated in Figure 2.2. The study reveals that multiplying the critic score (anomaly measurement, from component C_x) with Dynamic Time Warping produced the most performant variation of TADGAN. Dynamic Time Warping is a metric that computes the best match between two time sequences and assesses similarity within local regions.

As such, this study improved upon the problem of false positives in time series anomaly detection tasks. Despite this, the accuracy is stated to be a target of improvement. To add to this, the mode collapse issue was a persistent reality of this model.

2.2.1.2 Dilated Convolutional Transformer-based GAN

Recently, Li *et. al* proposed a new GAN variant tailored to improve the generalisation capacity and accuracy. The resulting framework, named **Dilated Convolutional Transformer-based GAN (DCT-GAN)** [16], consists of a single Discriminator and many Generators, each having a dilated Convolutional Neural Network (CNN) and a Transformer block to acquire detailed and broad information from the time series. Furthermore, the model combines two parts: a training stage and an anomaly detection stage.

In the first part of the framework, the generators G_i learn from a different detection window size using Dilated CNN for feature extraction and a Transformer-based Neural Network. The single discriminator receives integrated fake information from generators and corresponding actual sequences, distinguishing between them. They share the same architecture: a linear projection module and a Dilated Convolutional Transformer (DCT) block. This block integrates Dilated CNN and Transformer for feature extraction and sequential data handling.

The DCT aims to generate anomaly scores and identify detected anomalies. It uses a generator for Anomaly Detection *Ga*, which is chosen to reconstruct accurate average data. It has Latent Variable Optimization, which means that it back-propagates the latent variable z to find a proper latent variable (z*) that makes $Ga(z*)$ more similar to actual standard samples.

The framework improves time series anomaly detection with better accuracy and generalisation by using multiple generators to capture different levels of detail. It reduces mode collapse, adapts to various anomalies, and extracts features across different temporal scales. However, it is computationally intensive, requires pre-processing that may cause information loss, and risks overfitting with small datasets.

2.2.2 TimeGAN

Even though the previous examples have shown only GANs designed for anomaly detection tasks, there are examples of GANs used to generate synthetic TSD. Developed by Yoon *et. al* [27], a good example of this concept is the **Time-series Generative Adversarial Networks, or TimeGAN**. Essentially, it is structured with three training phases: the embedding training phase, the supervisor training phase and, finally, the joint training phase.

All components' relations with each other are shown in Figure 2.3.

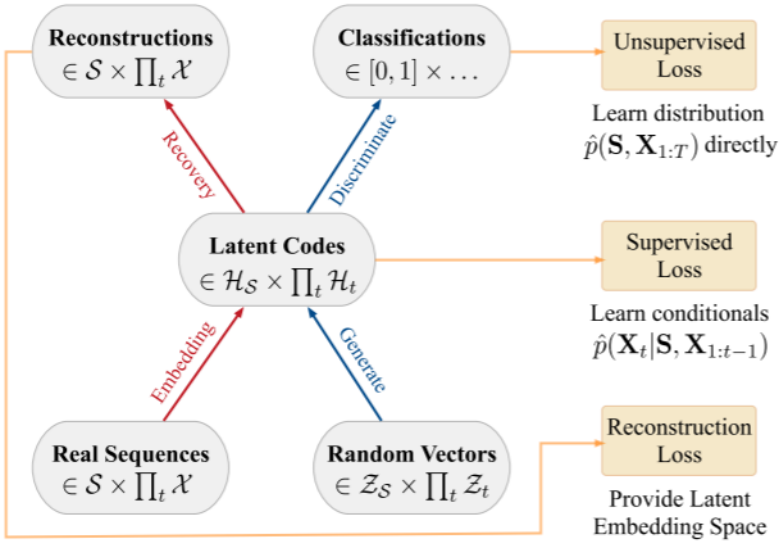

Figure 2.3: Diagram of TimeGAN functions and relations. Taken from [27]

The framework employs four main components: an embedding function, recovery function, sequence generator, and sequence discriminator. The first two, the autoencoding components, are trained jointly with the last two (adversarial components). This way, TimeGAN can generate synthetic data, encode features of the dataset and iterate across time.

Starting with the autoencoding components, they provide a mapping between the feature and its encoded counterpart. For example, for H_s and H_x representing the encoded features S (static) and X (temporal), the embedding function represented in Equation 2.8:

$$e : S \times \Pi_t X \in H_S \times \Pi_t H_X \tag{2.8}$$

takes static and temporal features to their encoded forms: $h_S, h_{1:T} = e(s, x_{1:T})$, where $h_S = e_S(S)$ and $h_t = e_X(h_S, h_{t-1}, x_T)$ (respectively, an embedding network for static and temporal features). It is implemented via a recurrent network. Conversely, the recovery function, represented as Equation 2.9:

$$r : H_S \times \Pi_t H_X \in S \times \Pi_t X \tag{2.9}$$

reverts the codes (temporal and static) back to their feature representations $\tilde{s}, \tilde{x}_{1:T} = r(h_S, h_{1:T})$. This function is implemented via a feed-forward network.

The generator and discriminator are trained within the encoded space for the latter pair of components. Assuming that Z_S and Z_X are vector spaces for static and temporal features, from which random values are extracted to generate the embedded counterparts H_S and H_X, using Equation 2.10:

$$g : Z_S \times \Pi_t Z_X \in H_S \times \Pi_t H_X \tag{2.10}$$

It is implemented through a recurrent network, returning synthetic encoded values from $\hat{h}_S, \hat{H}_{1:T} =$

$g(z_S, z_{1:T})$. On the other end, the discrimination function, represented as Equation 2.11:

$$d : H_S \times \Pi_t H_X \in [0,1] \times \Pi_t [0,1] \tag{2.11}$$

has the encoded features as input and returns the classifications $\tilde{y}_S, \tilde{y}_{1:T} = d(\tilde{h}_S, \tilde{h}_{1:T})$. Note that $\tilde{h}$ can be real or synthetic encoded features.

As for the loss functions implemented(Equations 2.12, 2.13, and 2.14), further illustrated in Figure 2.4:

$$L_R = E_{S,X_{1:T}\sim P}[||s - \tilde{s}||_2 + \sum_t ||x_t - \tilde{x}_t||_2] \tag{2.12}$$

$$L_U = E_{S,X_{1:T}\sim P}[\log yS + \sum_t \log yT] + E_{S,X_{1:T}\sim P}[\log(1 - \hat{y}S) + \sum_t \log(1 - \hat{y}T)] \tag{2.13}$$

$$L_S = E_{S,X_{1:T}\sim P}[\sum_t ||h_t - g_X(h_S, h_{t-1}, z_t)||_2] \tag{2.14}$$

The reconstruction loss (Equation 2.12), represents the encoding and decoding part of the framework. Equation 2.13 illustrates the generator's process: firstly, it receives embeddings and generates the next synthetic vector, with the next step being the computation of gradients. Finally, the supervised loss (Equation 2.14), addresses the limitation of relying solely on the discriminator's binary feedback, which may not sufficiently motivate the generator to capture step-wise conditional distributions in the data.

This concept was then used to test the generated synthetic data and compare it to the original using t-SNE and PCA analysis, revealing that the plots coincided. This means that the synthetic data is realistic, in terms of predictiveness. It also tested their *discriminative score*, where the original data was labelled real, and the synthetic fake and a model were trained to try and distinguish them. Finally, the *predictive score* was also tested by comparing a model's Mean Absolute Error when trained on the real data and the synthetic. Using these metrics, the results show that TimeGAN can generate realistic time series data.

2.3 Stress-Testing

The goal of stress testing in machine learning is to investigate how well a model performs under conditions that are different from the ones represented in the data used for its development. It helps identify where the model performs well and where it struggles, giving a clearer picture of its strengths and limitations. This constitutes a fundamental step towards better understanding and documenting AI models.

Some advances have been made in developing more robust and efficient stress testing techniques. Some examples of research done can be found in the work of Smith-Miles *et al.* [25] by offering new ways of evaluating algorithms, more specifically Instance Space Analysis (ISA). ISA aims to provide insights into algorithm strengths and weaknesses, offering a nuanced assessment beyond average statistical reporting. It employs dimension reduction for visualisation, footprint

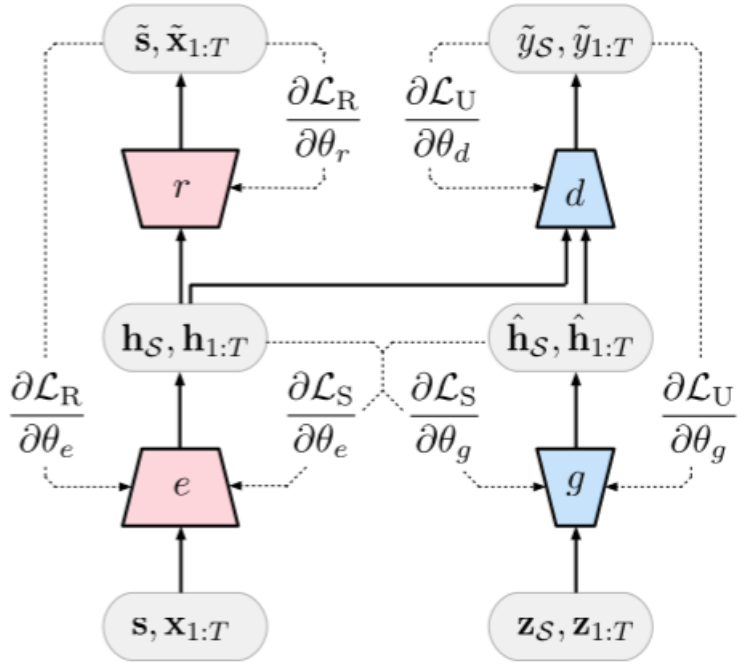

Figure 2.4: Scheme of the TimeGAN (training process). Taken from [27]

estimation for algorithm comparison, and theoretical boundaries to assess the diversity of test instances.

One possibility to further test the limits of predictive models is to provide more data that is capable of inducing stress. A way that we can reliably gather more data that may satisfy that condition is to sufficiently modify the data that we have to get *new* data. One process that does this is called Datamorphing, proposed by Zhu *et al.* [30].

2.3.1 Datamorphing methods

As stated in the research by Zhu *et al.* [30], to apply a metamorphic relation in software testing, we need to:

1. Generate test cases $a_1, \ldots, a_n$ satisfying $V(a_1, \ldots, a_n)$.
2. Execute program P on test cases to obtain outputs $b_1 = P(a_1), \ldots, b_n = P(a_n)$.
3. Check condition R on input and outputs to determine the correctness of the program.

As such, Zhu *et. al* proposed a new method of testing ML applications. It was given the name Datamorphing testing, and it is comprised of three main sets: Base seed test cases, *datamorphisms* and finally, *metamorphisms*. The overall testing process is summarised in Figure 2.5.

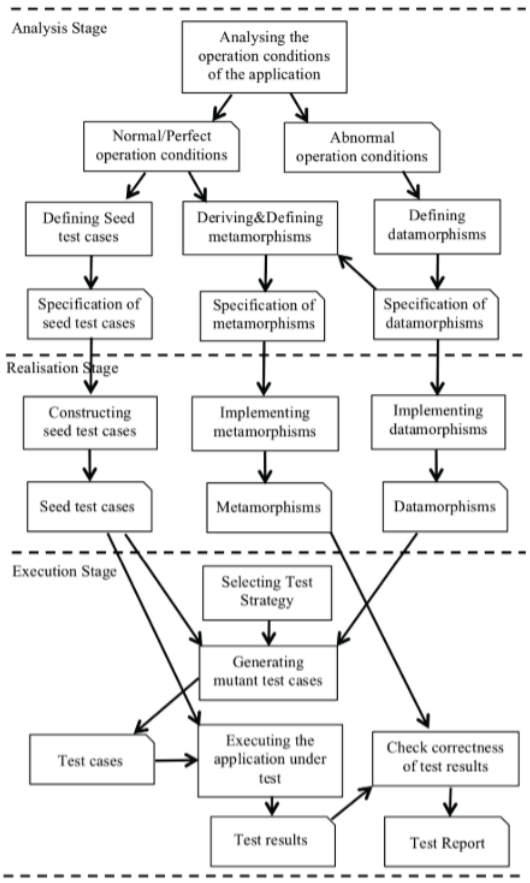

Figure 2.5: Datamorphic testing process. Adapted from [30]

The first concept is the known set of test cases, for example, used as training data. We can select a group of them randomly or with specific criteria as the seeds upon which the *datamorphism(s)* will be applied.

A *datamorphism* is a process that generates new testing data, referred to as mutants, by transforming already existing test data. As an example, for the *sin* function, a binary metamorphic relation can be defined as $(x+y=\pi) \Rightarrow \sin(x)=\sin(y)$. The main difficulties include finding effective metamorphic relations and generating test cases satisfying applicability conditions.

Metamorphisms result from the changes (i.e. *datamorphisms*) applied to the seed test cases. More formally, a metamorphism is a set of *datamorphisms* Ψ on the input domain of program P. A metamorphic relation M is a metamorphism if it can be presented as:

$$R(x_1,\ldots,x_k,P(x_1),\ldots,P(x_k),P(x_1'),\ldots,P(x_m')) \tag{2.15}$$

where $x_i' = \phi_i(\vec{z}_i, l_i)$, $\vec{z}_i$ is a subset of $\{x_1,\ldots,x_k\}$, and $\phi_i \in \Psi$.

When applied to the previous example *sin* function with *datamorphism* $\phi(x)=\pi-x$, it results in the formula 2.16:

$$\sin(x)=\sin(\pi-x) \tag{2.16}$$

Testing can be automated by applying the *datamorphism* to obtain mutant test cases and checking the correctness condition without searching for or solving constraints. Metamorphisms serve as formal specifications, enabling automated testing of AI applications.

The first stage involves analysing the testing problem to design a *datamorphic* test framework. This includes identifying operating conditions and creating seed test cases, *datamorphisms*, and metamorphisms. Normal and abnormal operating conditions are defined, with seed test cases generated from normal conditions. Abnormal conditions serve as candidates for *datamorphisms*, requiring feasible transformations on input data.

In the second stage, Realisation, actual test data is constructed for the seeds, and *datamorphisms* and metamorphisms are implemented. This can be done through software development, existing applications, or manual editing of test cases, making this a more abstract concept. Metamorphisms typically involve invoking the application with seed and mutant test cases and storing or comparing results accordingly.

The final stage, Execution, involves executing the test according to a predefined strategy:

- **Exhaustive:** Generate all possible mutants by applying *datamorphisms* on seeds until no new mutants can be added.
- **Combinatorial:** Use unary *datamorphisms* to create mutants, achieving k-way combinatorial completeness.
- **Optimal:** Treat seed test cases as a population and *datamorphisms* as mutation operators, optimizing the test set based on a fitness function.
- **Random:** Randomly select seed test cases and *datamorphisms* to generate mutants until a specified number or adequacy criterion is met.
- **Exploratory:** Use binary *datamorphisms* to discover class boundaries, such as seeking Pareto fronts for classification results.

Having the framework formulated, Zhu *et al.* proposed test strategies for ML applications [28], which were further revised in 2022 [29]. They presented three primary methods using the Pareto Front problem as an example.

The *random target strategy* ($RT(n)$) explores *datamorphic* testing by randomly selecting two test cases, generating a midpoint if their classifications differ, and repeating until a pair with a small distance is achieved.

A variation involves starting with a single test case and repeatedly applying a unary *datamorphism* until a different classification is found or a predefined limit is reached, resulting in a 'walk' through data points. This is denoted as $DW(m,n) = \langle a,b \rangle$, with m as the walking distance and n as the steps.

The Random Walk strategy ($RW(m,n) = \langle a,b \rangle$) is similar but randomizes the next step when multiple traversal methods are available. Here, m is the walking distance, n is the steps and $\langle a,b \rangle$ is the output.

The studies provide a framework to extend a pool of available data, using applicable general concepts to generate new test data for the training or testing of ML applications.

2.3.2 AmbiGuess

Some research was done towards synthetic data generation independent of the tested model [26].

The study focuses on testing and strengthening the robustness of Deep Neural Networks (DNN) against ambiguous input data. In this context, it means an input that can belong to multiple classes. The new framework, titled AmbiGuess, introduces techniques to achieve those objectives, such as:

- **Regularised latent Space Generation**: Using Regularised Adversarial Autoencoder (rAAE), it creates a latent space that separates two classes to generate ambiguous inputs (in this space). It is divided into three steps: Reconstruction (Autoencoder), Discrimination (differentiate between encoded images from samples taken from a distribution), and Regularisation (adjust encoder so that encoded training data matches the class distribution);
- **Automatic Probabilistic Labeling**: The discriminator assigns probabilistic labels to each generated image. This ensures that the ambiguous examples are marked;
- **Sampling diversity**: There is a weighted sampling in the latent space to have a diversity of the ambiguous generations. This is accomplished by the implementation of Confined Latent Space (CLS) (Definition of the area between class clusters in the latent space where ambiguous samples are likely to be found), the division of CLS into grids, and, finally, a sampling strategy involving the ambiguity (estimated by the discriminator) and the diversity.

Results show that the generated samples induced uncertainty in DNN predictions. However, the best supervisors at detecting ambiguity were worse at detecting out-of-distribution (OOD) inputs. OOD refers to under-represented inputs during training. Conversely, those better at detecting OOD inputs were less effective at detecting ambiguity.

2.3.3 GASTeN

An example of a practical framework that implements stress testing, specifically in the context of image classification, is the Generative Adversarial Stress Test Networks (GASTeN) [4]. GASTeN builds upon the principles of GANs. In GASTeN, these GANs are repurposed to stress test image classifiers by generating challenging examples – images – that are designed to be difficult for the classifier to predict, i.e. choose which class they belong to. By exposing the classifier to these borderline or confusing cases, GASTeN helps evaluate its limits and provides insights into where its performance worsens.

In more technical terms, the following modifications are made to the standard GAN framework in GASTeN:

- the classifier is introduced into the training loop without optimisation, specifically to analyse its behaviour at the frontier. It is the object of evaluation for GASTeN. It is utilised solely for classifying images generated by the generator G(z). The classifier's outputs are then incorporated into the generator's loss function to guide training;
- the original GAN's Generator Loss function is modified by adding a new metric. This is done to influence the learning process to further converge onto the main objective of generating data that induces low confidence in a model;
- introduction of the Confusion Distance (CD) as a function that uses the output of the classifier using the generator's output ($C(G(z))$). By implementing a threshold, a barrier where the image is considered ambiguous, the equation 2.17 for the Confusion Distance (CD) is:

$$cd(C(G(z))) = |C(G(z)) - \Delta| \tag{2.17}$$

  where Δ is the threshold in question.
- addition of the hyperparameter *Weight* α as a factor associated to the CD metric. This is done in order to leverage how much influence the new addition component has, and it influences the balance between generating realistic or ambiguous images.

The metrics used to evaluate this approach were how viable were the generated images, using the Frechét Inception Distance (FID) [8], and a calculation of the mean of all Confusion distance calculations, named Average Confusion Distance (ACD).

To validate this approach, using the MNIST dataset, Cunha *et al.* implemented a Deep Convolutional Generative Adversarial Network (DCGAN), presented in Figure 2.6. The images generated would then be systematically analysed to identify cases where the model's confidence degrades.

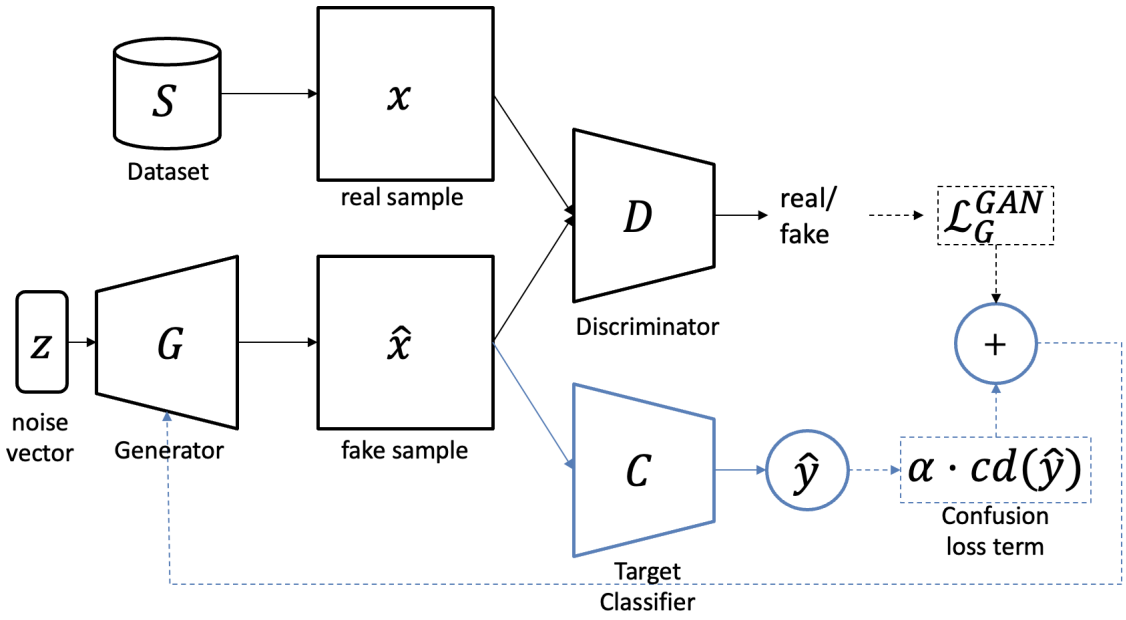

Figure 2.6: GASTeN diagram, adapted from [4]

The study found that optimising for both the objectives of realistic image generation (FID) and the ACD was difficult, with scenarios where one metric dominated the other. The hyperparameter α, if set to high values, made the evolution of the desired metrics unstable.

Despite this, there were scenarios where both objectives were fulfilled. The work demonstrated that stress testing helps understand AI models by defining the boundaries between confident and uncertain predictions. This approach can be applied to test other AI models using the same principles in various contexts.

2.4 Summary

This chapter exposed key concepts for understanding this work. We start by describing core concepts of time series forecasting, including an overview of Exponential Smoothing, ARIMA, and DL models such as LSTMs. We also briefly mentioned more recent advances such as FFORMS-G and FFORMA-N [3].

Following this, we delved into GANs [7, 6], an adversarial framework where a generator trains to synthesise data to deceive a discriminator. Several variants have been developed for more specific contexts. When focusing on time series data, we have mainly GANs concerning the time series anomaly detection task [5, 16]. However, there are also studies covering GANs for time series data generation, namely TimeGAN [27].

Finally, we examine the topic of stress testing. We have studies that go further on this topic [25, 4, 26], close to our main task, and frameworks to generate more data from sample data [30, 28, 29] (concepts useful for stress testing). Even though the prospects of the results of AmbiGuess [26] are promising, the target of that research and the primary state-of-the-art solution derived from it

were image classification problems. Though Weiss *et. al* recognised that it could be extended to other ML models, such as regression models, TSF-oriented work is lacking.

Chapter 3

Time Series GAN for Stress Testing

This chapter presents our approach to generating ambiguous time series data using a GAN-based model. We build on the existing GASTeN framework and adapt TimeGAN to create GASTeN-TS, a new method designed to produce realistic data that poses challenges for forecasting models. The first section outlines the problem (section 3.1), laying out the scope (section 3.1.1) and the hypothesis (3.1.2). The next section explains our proposed solution (section 3.2), describing the main modifications (section 3.2.1) and the validation method for the model using datasets (section 3.2.2). The final section concludes the chapter with a summary of its key points.

3.1 Problem statement

Similarly to the previous GASTeN work [4], we want a GAN-based solution to generate time series data continuations that, when added to the base data, is considered confusing for forecasts. To accomplish this, we want synthetic data that gives the resulting forecast a wide confidence interval. This way, the model responsible for the forecasts is not confident, and therefore, the data is ambiguous. An illustration of that idea can be seen in Figure 3.1.

Besides this objective, the generated data must be realistic enough to be used in real scenarios.

3.1.1 Scope

This study focuses on generating semi-synthetic time series data that will be used in univariate forecasts.

The data should result in forecasts with wide confidence intervals, signalling lower model confidence. Additionally, the generated data must be realistic enough for practical use. Any changes to the base GAN architecture should be minimal to ensure easy integration and future improvements.

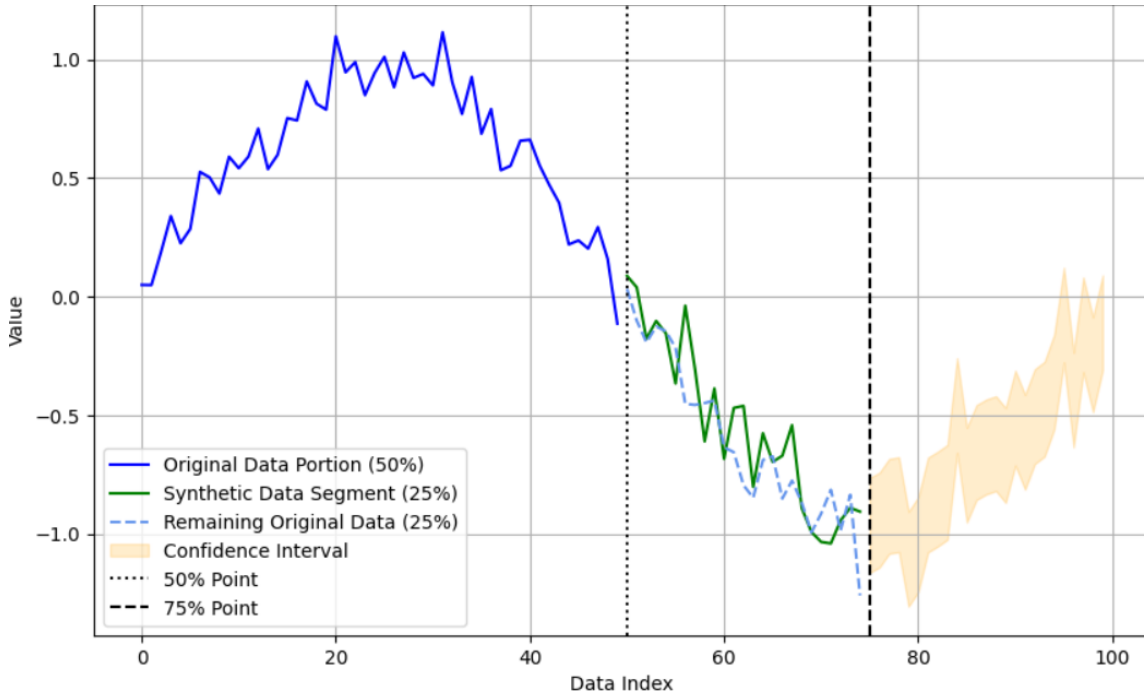

Figure 3.1: Example of mutated data. Blue represents the original data, green is the synthetic data. The mutated data fits in a predictor to generate the orange confidence interval.

3.1.2 Hypothesis

Given the problem presented, we formulate a hypothesis to guide our work. The hypothesis is as follows:

> *"Incorporating a predictor into the GAN's training process and modifying the Generator's loss function will generate time series data that consistently results in challenging examples."*

This ambiguity is reflected in wider confidence intervals. We would also like to know whether adding synthetic data extensions to a dataset will achieve this statement. We hypothesise that this method will consistently produce data that meets these criteria across different datasets, demonstrating the effectiveness of the proposed framework.

3.2 Proposal

Because of its promising results in experiments with various types of time series data (Stock-related data, energy, etc.), we adapt TimeGAN's [27] existing architecture to build a GASTeN framework [4] to achieve this objective, which we name GASTeN-TS.

To do this, we divide the joint training phase of the original TimeGAN into two separate ones. The first phase is the equivalent of the base TimeGAN without any modifications (calculation of normal generator loss function, discriminator (adversarial) loss). In the second phase, we extend

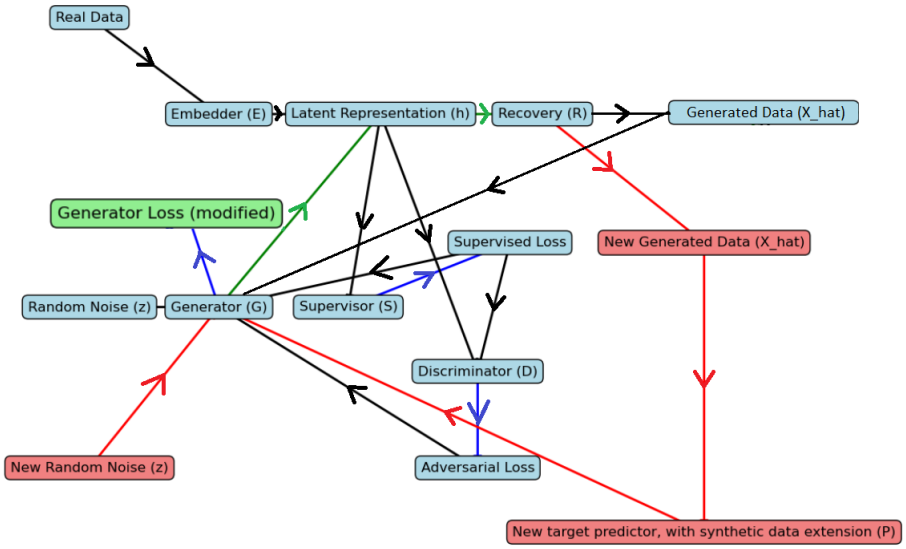

Figure 3.2: GASTeN-TS second joint training phase. Starting from the *Real Data* component, the output of the predictor component is added to the generator's loss calculation. The light-blue components represent the existing TimeGAN components, while the red ones need new additions. Red edges represent added logic flow, while green edges represent flow used by the original TimeGAN framework and the new additions. Black edges represent the pre-existing flow. Blue edges indicate flow to loss score calculations. Finally, the generator loss component appears in green because it suffered modifications.

the generator's loss calculation by adding the predictor to give the new metric to add to the function. The framework for this second training phase is shown in Figure 3.2. In this phase, beyond the original flow of the initial joint training phase (blue components, black, green and blue flow), the original data is extended with a portion of newly synthetic data generated by the GAN (following the red and green flow) and used to fit the predictor model. This will then create the forecast with the corresponding confidence interval needed to formulate the new metric and feed it into the modified generator's loss (in green).

3.2.1 Key implementations

The modifications in GASTeN-TS, which are not present in the original TimeGAN architecture, are the following:

- **Predictor (P)**: The predictor introduced during the GASTeN-TS training loop and the stress-testing target. It can be an ARIMA model, whose order is optimised concerning the original training data fed into the GAN. The output of the predictor is then added to the generator's loss function.

- **Average Confidence Interval Width**: This is a function that picks each limit value from the confidence interval, lower and upper, (fixed at 95%) generated from a probabilistic forecast, calculates the difference between the two, and averages this value throughout the whole forecast. Essentially, we generate synthetic data, take a portion corresponding to the points between the half and three-quarters length (25% of the data), and append it to the input data (50%). We then use this mutated data (75% of the length) to fit the previously defined predictor, generate a forecast for the remaining data length (25%) and calculate its width using the associated confidence interval (refer to Figure 3.1).

  For models that generate point forecasts (e.g. DL), we needed to find a way to have confidence intervals from them. Hyndmann [13] mentions a possible method in his work: generating prediction intervals using bootstrapped residuals. We implemented a residual generation using a one-step prediction with a sliding window method throughout the remaining data. The window has the size of the original data inputted into the GAN (50% of the data), and the residuals are generated from the synthetic extension of the same data (next 25% of the whole data). Then, we implement a one-step ahead prediction for each point of the remaining data (remaining 25%), in which the window is the same size as the whole training data, including the synthetic extension. In each sliding window step, the following point is forecasted, and then we bootstrap possible probabilistic forecasts by randomly selecting 100 times a residual to add to the point forecast. The upper and lower bounds of the simulated confidence interval are the 2.5 and 97.5 quantiles of the resulting list of 100 simulated possible forecasts.

- **Generator Loss Function**: Similarly to the previous GASTeN work done in [4], we add a new objective to the Generator by adding the term into the loss function. This new metric measures the average width of the confidence interval throughout the output's forecast. We name this term *Average Confidence Interval Width (ACIW)*. We associate this with a hyperparameter α to weigh the impact of this new metric (between 0 and 1). This hyperparameter will influence the weight the new additions has to the resulting modified loss function. This results in the following updated Generator loss function 3.1:

$$L_G^{GASTeN-TS} = L_G^{TimeGAN} + \alpha * |LCIR(P(X_i + G(z)_{\frac{n}{2}:\frac{3n}{4}}))| \tag{3.1}$$

where X_i is the GAN input data and $G(z)_{\frac{n}{2}:\frac{3n}{4}}$ is the synthetic extension portion (from $\frac{n}{2} : \frac{3n}{4}$, where n is the total number of points).

As we calculated the ACIW over the original data before the start of the training process, we have a value to compare how ambiguous the synthetic data appended is in comparison to only original data. As such, the Logarithmic Confidence Interval Ratio (LCIR) term used in Equation 3.1 is calculated as:

$$LCIR(P(X_i + G(z)_{\frac{n}{2}:\frac{3n}{4}})) = \ln(\overline{ACIW(P(X_i + G(z)_{\frac{n}{2}:\frac{3n}{4}}))/ACIW_O}) \tag{3.2}$$

where $ACIW_O$ represents the Average Confidence Interval Width calculated on the original data, and $ACIW(P(X_i + G(z)_{\frac{n}{2}:\frac{3n}{4}}))$ is the ACIW calculated using the output of the predictor with mutated data. The metric represents the average fraction between ACIW over mutated and real data throughout the number of datasets used in the GAN input. This process is done each time the generator's forward pass function is called.

One of the hyperparameters used in GASTeN-TS is the number of epochs in the first joint training phase, similar to the pre-training phase of the original GASTeN. The embedding and supervisor training epochs of the TimeGAN are fixed values. We do this because we want the autoencoding feature of GASTeN-TS sufficiently ready for the training phases with the Generator and Discriminator. Otherwise, the framework will not work.

3.2.2 Evaluation strategy

To validate our approach, we run tests on benchmark Time Series datasets, such as the dataset present by default in the original TimeGAN implementation[1]. As previously stated, GASTeN-TS is based on TimeGAN [27]. This GAN variation yielded promising results regarding generating synthetic time series data.

We use different predictors to gauge our proposed method. Since we are using different datasets (the stock CSV(Comma Separated Values) file is structured as seven different datasets), this ARIMA order may differ for each dataset but is the most optimal for each context.

We collect metrics throughout each experiment to see how the model behaves. We save the $ACIW_O$ as the baseline ACIW, calculated using only the original data, with a 75/25 split, which we compare to the successive ACIW calculations saved throughout the training process.

After the training, we also use synthetic time series samples to append to the input data, previously split from the full original data, and calculate confidence intervals using this resulting data (75% of length used to fit, composed of real data (first 50%) and synthetic extension (following 25%)). We use said intervals to contrast with the ones generated using only original data (fitting data comprised of only real data). This will tell us how ambiguous the mutated data is compared to the original.

[1] https://github.com/birdx0810/timegan-pytorch/tree/main

In addition, we use the generated data to calculate the DTW (Dynamic Time Warping) with the original data to compare how similar the generation is to reality. This algorithm is used to measure the similarity between two time series. In this case, we want synthetic data similar to the original counterpart (i.e. a score on the lower spectrum), but we also want a level of dissimilarity (not too close to 0). That way, the data is sufficiently different to possibly induce ambiguity in the predictor's forecast.

The next chapter (Chapter 4) will better describe the experiments, presenting further details on the hyperparameters and target datasets. Additional information regarding the environment of these experiments will be provided.

3.3 Summary

This chapter presents our proposed approach for generating ambiguous time series data using a GAN-based model. Building on the existing GASTeN framework, we introduce GASTeN-TS, an adaptation of TimeGAN designed to produce realistic and challenging data for forecasting models.

We presented the goal to generate time series data that results in forecasts with wide confidence intervals, indicating higher ambiguity. The data must be realistic for practical use, with minimal modifications to the base GAN to facilitate integration and future improvements.

Afterwards, we exposed the adaptation of TimeGAN to create GASTeN-TS. This involves two phases: dividing the existing joint training phase into two (the pre-existing TimeGAN one and a new one), where the second introduces a new predictor and modifies the Generator's loss function to include a term measuring the Average Confidence Interval Width (ACIW). This metric helps ensure the generated data results in forecasts with broad confidence intervals.

Finally, we state that we aim to use benchmark Time Series datasets, with performance measured by comparing ACIW values, Dynamic Time Warping between generated and original data, and confidence intervals. The validation process involves different predictors and metrics to assess the effectiveness and realism of the generated data.

Chapter 4

Experimental Setup

This chapter will describe the experiments used to evaluate our approach. Section 4.1 will describe the selected dataset. Section 4.2 will delve into the architecture of the main modules of the program. After the exposition, section 4.3 and section 4.4 will report what analysis was done to the default TimeGAN and GASTeN-TS, respectively. Appendix A provides details of the implementation.

4.1 Datasets

To test our approach, we used two datasets: a simple sine function and the stock dataset already available in the parent repository [1]. This section describes the stock dataset in more detail.

We used the sine dataset as a basic experiment, for insights on how the proposed method works. But the focus of our analysis was on the stock dataset. The description of this dataset can be found in Appendix D.2.

4.1.1 Stock Dataset

This dataset, as the default one present in the original implementation [2], is formatted to be a multivariate (6 features: Open, High, Low, Close, Adj_Close and Volume) multi-set (with 7 different datasets per index column), each having 3678 entries, making a total of 25745 rows.

To make the ACIW calculation, we focus on a single feature (due to the univariate forecast focus of this work) and use the full number of sub-datasets to train the GAN.

4.2 Framework Architecture

This section will specify the characteristics of the base GAN, the forecasting model used to obtain the ACIW metric, and the evaluation metrics used.

[1] https://github.com/birdx0810/timegan-pytorch/tree/main
[2] https://github.com/birdx0810/timegan-pytorch

Generator
GRU
Linear
Sigmoid

(a) Generator network

Discriminator
GRU
Linear

(b) Discriminator network

Supervisor
GRU
Linear
Sigmoid

(c) Supervisor network

Recovery
GRU
Linear

(d) Recovery network

Embedder
GRU
Linear
Sigmoid

(e) Embedder network

Table 4.1: TimeGAN component networks

4.2.1 TimeGAN

The GASTeN-TS framework is designed to work with most time series data generation GANs by introducing minimum modifications to the base GAN, which, in this case, is the TimeGAN. The implementation was found on GitHub (see footnote 2).

The TimeGANs network specifications are shown in table 4.1. All networks are optimised using Adam Optimiser [15] and have a learning rate of 0.001.

We used a benchmark ARIMA forecasting model, whose order is predetermined to be the best for the original data fed into the GAN for training, using the *AutoArima* package.

4.2.2 Evaluation Metrics

This subsection will expose the main metrics used to evaluate our proposal.

4.2.2.1 Synthetic Data Quality

We use a combination of different factors to measure the quality of the data generated by TimeGAN. One is visualising the resulting data synthesis over the original data, which is not fed into the GAN.

However, this alone might not be sufficient to measure the quality of said data. Therefore, DTW distance calculations are made between a section of exclusively synthetic data and its original counterpart. We are focusing on getting low values, which indicates good realism because it is close to the original data. But, at the same time, these same values must not be too low. Otherwise, it is not different enough to confidently assert that it can induce stress on the predictors.

4.2.2.2 Predictor's Confidence Interval Width

We plot the evolution of the ACIW metric whenever it is calculated throughout the training epochs, saving them into a file and then plotting.

We also decided to make a forecast using the selected model, firstly with only the original data, then real data with synthetic data extension. The resulting upper and lower values for the confidence interval of both cases were then plotted to compare their widths.

4.3 TimeGAN Experiments

This experimentation uses the original TimeGAN to check whether the base implementation is minimally adequate for our task. In summary, we wanted to test whether the base TimeGAN framework was able to achieve the realistic data generation objective.

For this experiment, the changes implemented to the base code were disabling the *shuffle* hyperparameter present in the data split and reducing the number of epochs for each of the 3 parts of TimeGAN's training process. This was done to build a base environment aligned with the work context and premises, and the reduced number of epochs was implemented due to time constraints.

The dataset mainly used in this experiment was the stock dataset. After training, we plotted and contrasted the synthetic and original data. Furthermore, we also did a quick test using the DTW distance calculation routine to have numerical data associated with how realistic the synthetic data is.

This analysis is the equivalent of testing GASTeN-TS, where the weight hyperparameter for the new metric (hyperparameter α) is set to 0. That means nullifying the new additive component of the Generator's Loss Function.

4.4 GASTeN-TS Experiments

After encountering the challenges presented in Appendix C and managing to do the analysis detailed in Section 4.3, we decided to shorten the experimentation to encompass only the following details:

- **Dataset**: matching the default TimeGAN experiments (Section 4.3), we focus on the stock mega-set since it is a multi-set with multiple features that can be focused individually.
- **Hyperparameters**: We have a 10-epoch pre-train phase (with both the embedding and the supervisor parts preceding this being 50 epochs each), and the weight of the new component for the Generator's Loss function was set to 1. This ensured sufficient pre-training epochs to accustom to the input data while not making the process too costly. Therefore, we decided to cap the joint phase (with the new metric incorporated) to 200 epochs.

4.5 Summary

This chapter outlines the experimental setup planned and the one used to evaluate our proposed approach. The chapter began with a description of the datasets planned (4.1), which include a sinusoidal dataset and the default stock dataset, which, in the end, was the main one used. Following that, the architecture of the experimental framework was detailed (4.2), including the base TimeGAN configuration and the predictors used, which were mainly ARIMA-based.

Next, we discussed the proposed experiments. Firtsly, experiments were carried out on the default TimeGAN (4.3) for screening purposes. Finally, we decided to run experiments on GASTeN-TS, using the conditions set in section 4.4. The results that will be presented refer to these last two configurations mentioned (4.3 and 4.4). Preliminary experiments can be foun in Appendix C and the implementation details are described in Appendix A

Chapter 5

Results and Discussion

This chapter focuses on the results obtained in the experiments done with the base TimeGAN and finally with the GASTeN-TS. We will first expose the experiments' results on the original TimeGAN, focusing on the plots of the synthetic data and corresponding original data, complementing this analysis with the DTW distance calculation. Secondly, we will show the outcome of the experimentation done on GASTeN-TS as specified in section 4.4. Most of the plots present in section 5.2 are sections of the complete plots present in Appendix F

5.1 Default TimeGAN

We first analyze the original TimeGAN, as shown in section 4.3. Ideally, one of GASTeN-TS requirements is to generate realistic data that is not too similar to the original data. This condition makes the data viable to constitute extensions that induce stress on a previously established Time Series forecasting model.

After running TimeGAN's training process with the minimal modifications specified, we get the contrast between synthetic and corresponding original data in figure 5.1. To do this, we also ran the DTW distance calculation routine, which was done for a single feature in all subsets, as shown in Figure 5.2.

The visual inspection indicates that the synthetic data might fit the conditions set for this analysis. However, we need more concrete data to support the visual data. Results are illustrated in Figure 5.2

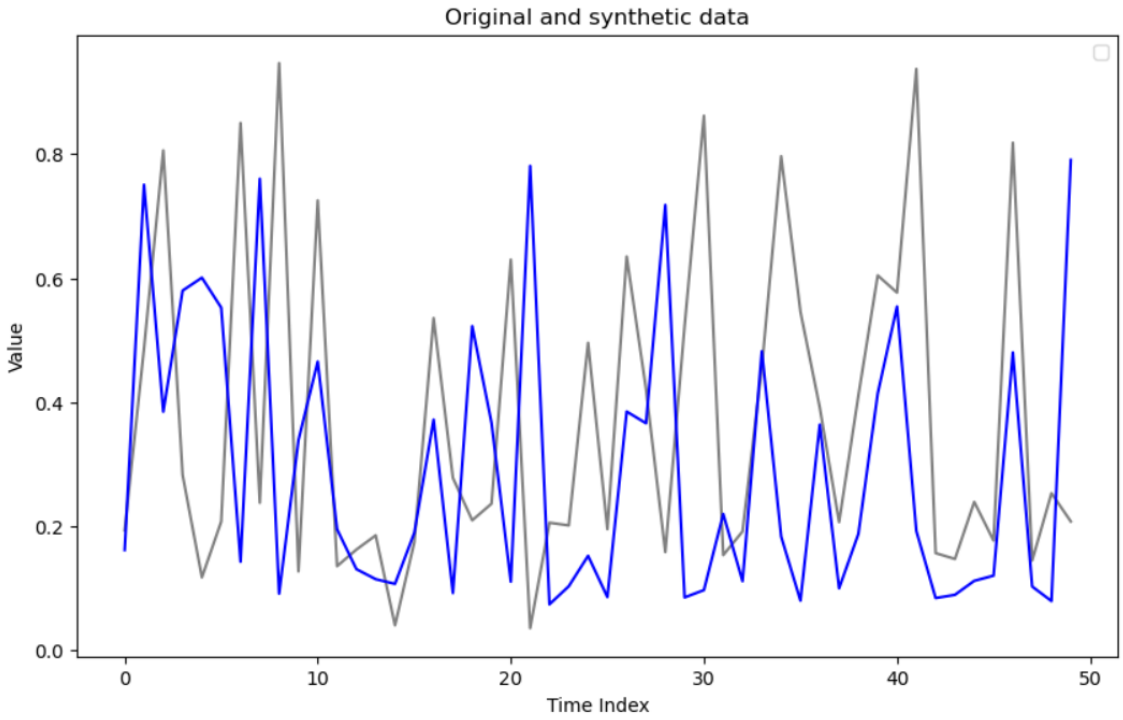

Figure 5.1: Segment of the last 50 points of synthetic (in blue) and original data (in grey), plotted after TimeGAN's training process. This plot corresponds to the 1st feature of the 1st subset of the stock dataset.

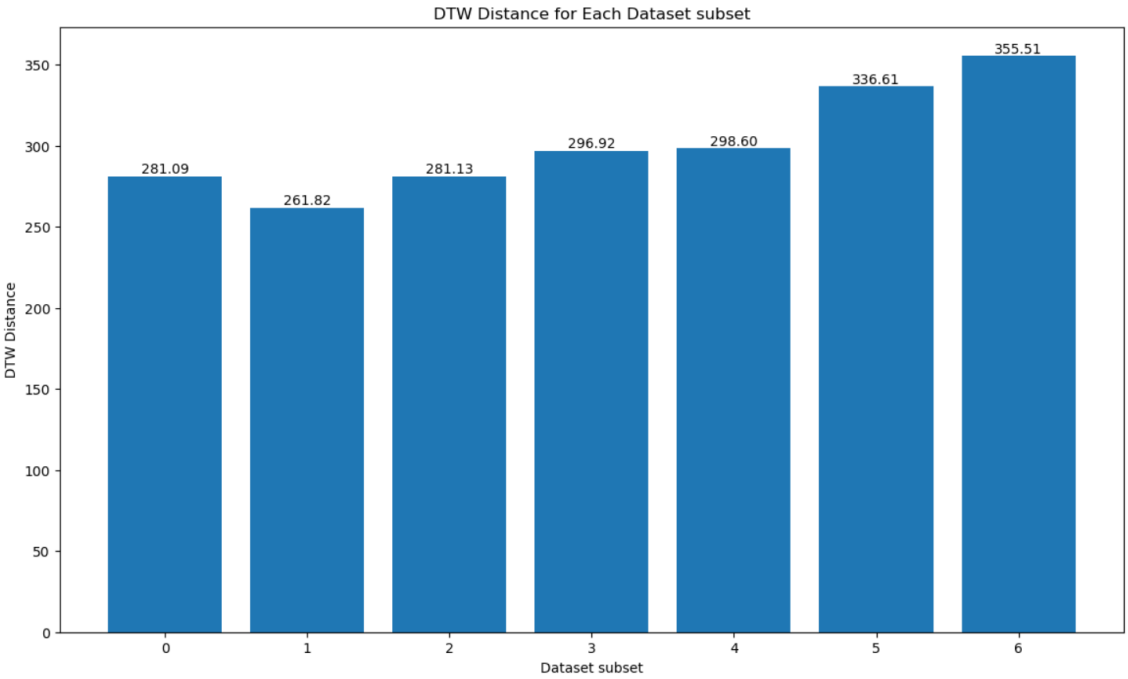

Figure 5.2: DTW distance score for all seven subsets of the stock dataset. This plot refers to the 1st feature.

To make a correspondence between the visual data in Figure 5.1 and Figure 5.2, the synthetic data has a DTW score of 281.09 compared to the original data (the other datasets' scores being respectively, 261.82, 281.13, 296.92, 298.6, 336.61 and 355.51). A score in this dimension indicates some dissimilarity, but the data has a good relevance level. This is because a score too low would indicate that the data is almost equal to its comparison counterpart, while a score too high indicates that there is little to no correlation to the other.

Combining these two results gives the perspective that TimeGAN is a good candidate to be the base GAN needed to make a GASTeN framework proposal around time series data. It fulfils the data generation requirement, and the synthetic data resulting is viable for the proposal because it is similar but with a level of dissimilarity high enough to possibly induce stress on a forecasting model when used as an extension of the dataset previously trained on.

5.2 GASTeN-TS behaviour

Since the analysis done on the validity of the base TimeGAN yielded promising results, the next step was testing the modified algorithm. As previously stated, the goal for GASTeN-TS was to generate data that is sufficiently similar to the real data but sufficiently different to induce low confidence from a time series forecasting predictor. Here, low confidence essentially means a wide confidence interval width, given by the difference between the upper and lower values of said interval.

Since TimeGAN by itself (in the conditions stated in section 4.3) had promising results for the 1st goal, this analysis focuses mainly on whether the ambiguity objective is achieved while also paying attention to see if the realistic data generation condition is also being fulfilled. The conditions of this experiment were exposed in section 4.4.

As such, this section is divided into two main analyses. The discussion will be based on the metrics discussed in section 4.2.2, illlustrated with some plots of the synthetic data extensions generated.

5.2.1 Synthetic data quality

We start by seeing if adding new metrics influences the quality of the synthetic data by visualising the synthetic data against the corresponding real data and analysing the results given by the DTW distance calculation.

The data visualisation can be analysed in figures 5.3 and 5.4. Both refer to the 1st feature of the 1st subset of the stock dataset(*cf.* chapter 4.1).

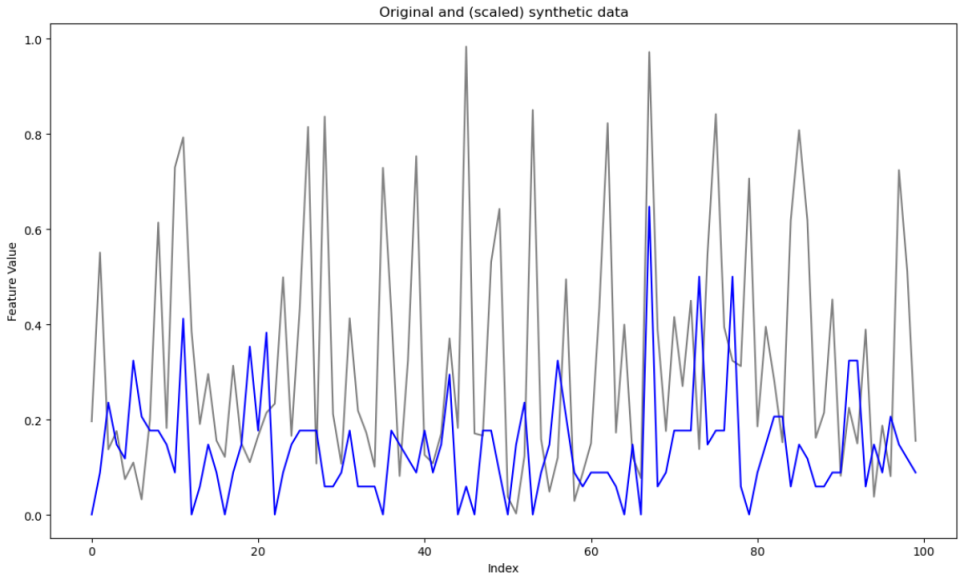

Figure 5.3: 100 points of synthetic data (in blue, scaled) *versus* the corresponding points of the original data (grey, not used as training input).

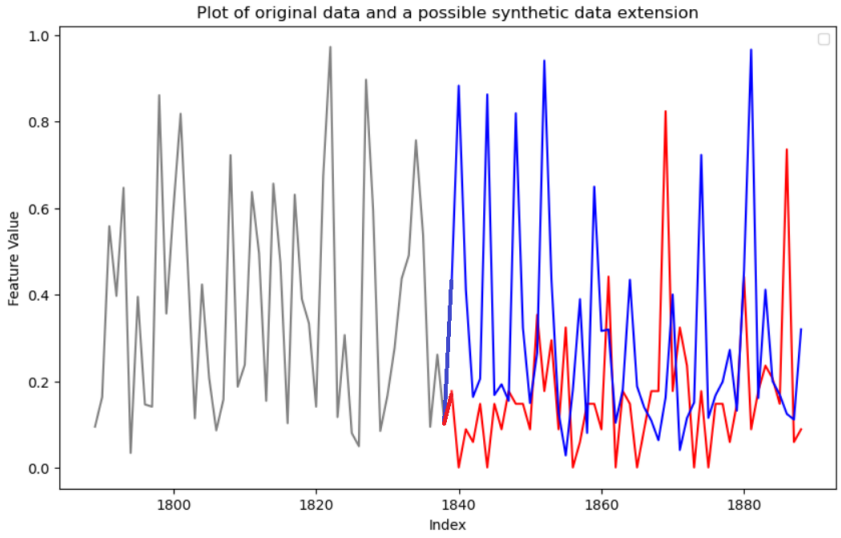

Figure 5.4: In grey, the last 50 points of original data, followed by 50 points of original data (in blue, not used as training input), contrasting the possible 50-point extension of synthetic data (in red, scaled).

These results presented in the first two figures indicate that the synthetic data might follow a realistic pattern while being sufficiently different from the real data. However, to get an objective analysis, we need to analyze the DTW distance between the original and generated data (figure 5.5).

As we can see from the bar plot, the new metric influences the quality of the synthetic data, even if slightly. Even so, when solely analysing this plot, although subjective, it seems to accomplish the objective of achieving realistic data with a degree of dissimilarity since the score is

neither too low (i.e. not too similar) nor too high (i.e., not too dissimilar). This, in turn, makes the synthetic data show good signs of being suitable for being used as an extension of real data to attempt to induce low confidence in a predictor.

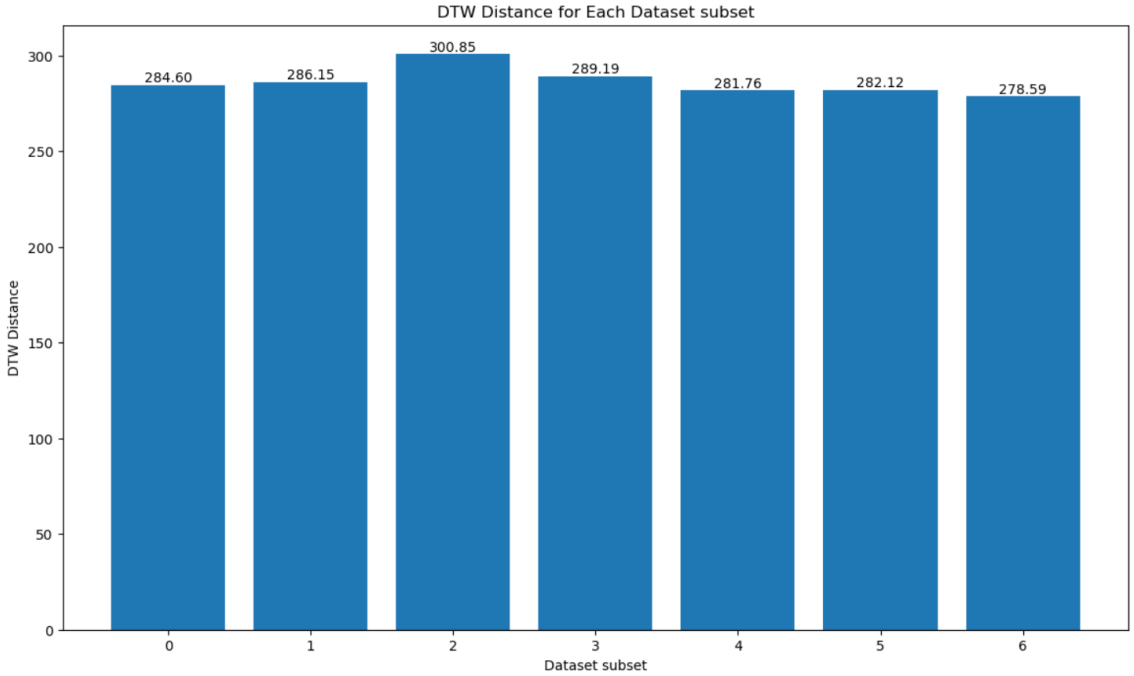

Figure 5.5: DTW distance score for all seven subsets of the stock dataset, after GASTeN-TS's training process. This plot refers to the 1st feature.

5.2.2 Average Confidence Interval Width

First, we decided to plot the progression of the ACIW metric throughout each recording to assess its evolution. The result can be found in figure 5.6. While sometimes the ACIW values is higher than the initial value, the value seems to decrease towards the end. Although the trend is not very clear, further analysis is required to understand the behavior of the method.

To gauge whether the resulting synthetic data, when used as an extension of real data, can induce stress in a predictor by lowering its confidence, we decided in this case to run a forecast, firstly using only original data, then swapping part of said data with synthetic (at the end, to simulate the extension). This way, we could generate a confidence interval for both cases and compare their upper and lower limits. The results can be seen in figure 5.7.

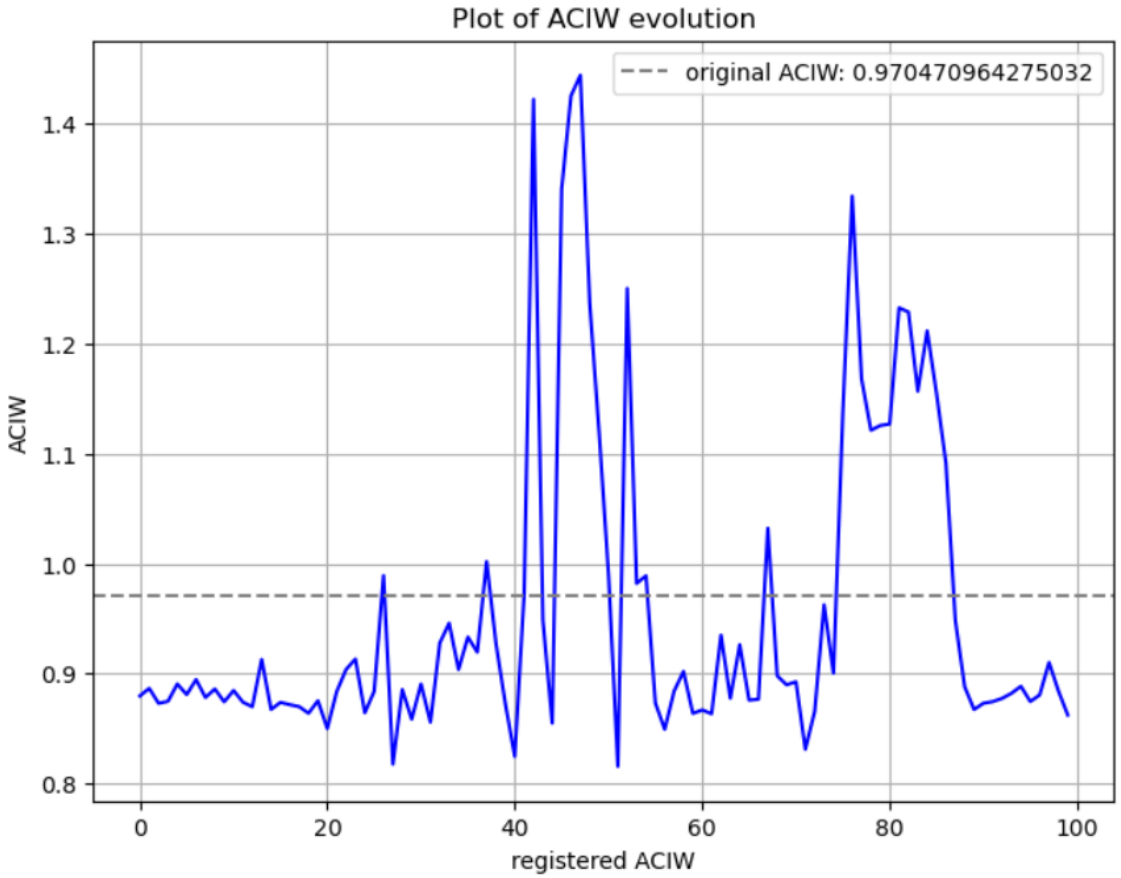

Figure 5.6: Last 100 records of ACIW, for the 1st feature of the 1st subset, taken during GASTeN's training process.

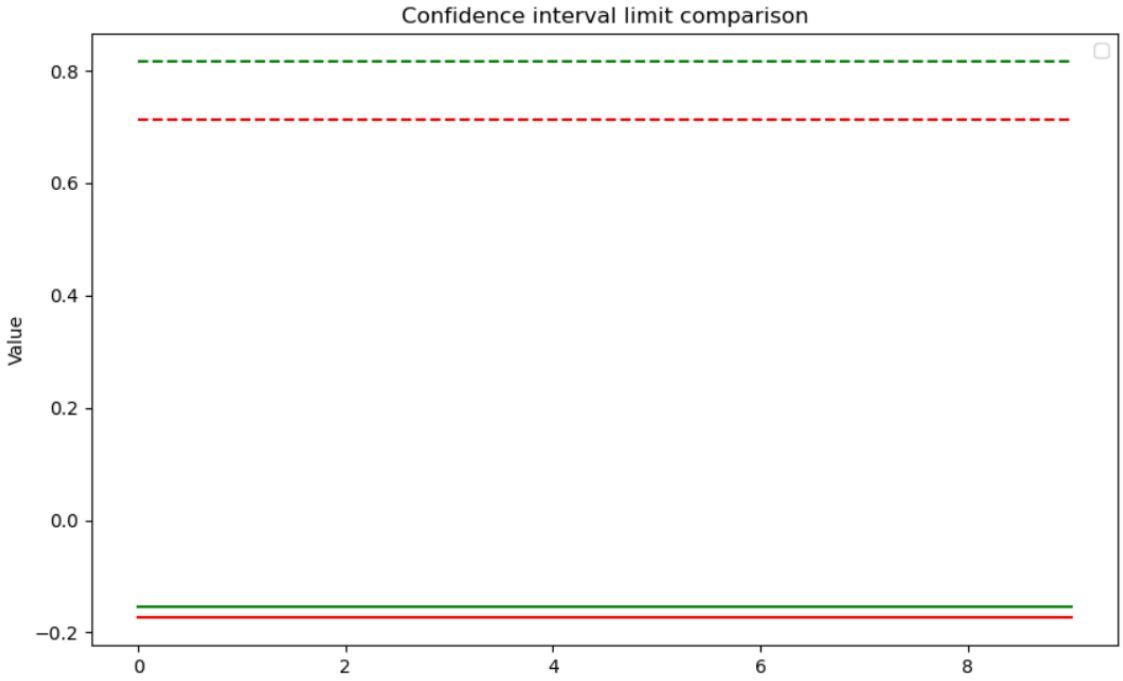

Figure 5.7: Confidence Interval limit (95 % confidence) comparison plot. We can see the mutated data's upper (dotted line) and lower limits in red. Green refers to the limits for the original data.

As we can see, although the synthetic data induces a lesser value for the lower limit, it does not produce a higher value for the upper limit than the original data yields. Furthermore, when combined, the width of the confidence interval is shorter for the synthetic data than for the original data.

5.3 Results Discussion

In section 5.2, we discussed the influence of the new metric on the main objectives of GASTeN-TS. This section will further analyse the results obtained, drawing some conclusions from the evidence.

5.3.1 Data Visualisation

From the data visualisation, even though the synthetic data seems to be following the pattern set by the previous real data points, we notice that said synthetic data looks like it is more compact, with lower high values and higher lower values than the preceding original data. An example can be seen in Figure 5.8. At first sight, the reduced range of values may in fact lower the confidence of the predictions made by the model, which was trained on data with similar patterns but a different range. However, as the analysis of the confidence intervals in the following section indicate, this may be due to the mode collapse problem, which could indicate that the scaling (section **??**) applied to the data is not sufficient [1].

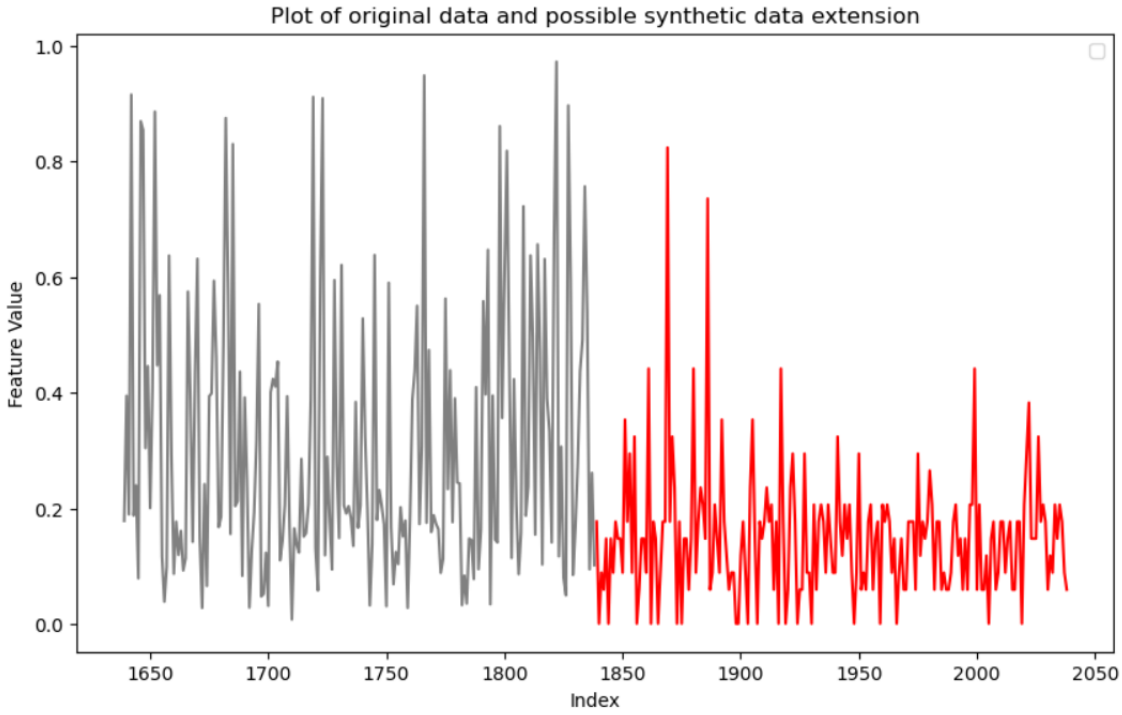

Figure 5.8: Last 200 points of original input data and the first 200 values of a possible synthetic extension (in red). This plot refers to the 1st feature of the 1st subset of the stock dataset

5.3.2 Confidence Interval

When analysing the results and progression of the confidence interval metric, we notice that the resulting forecast (using synthetic data extension) makes the confidence interval width shorter. Unfortunately, this means that the predictor is, in fact, more confident when the synthetic data is used as a partial substitute rather than only the original. This is in line with the progression of the ACIW metric throughout GASTeN-TS's training process, as shown by the segment illustrated in figure 5.6 and further exposed in figure 5.9.

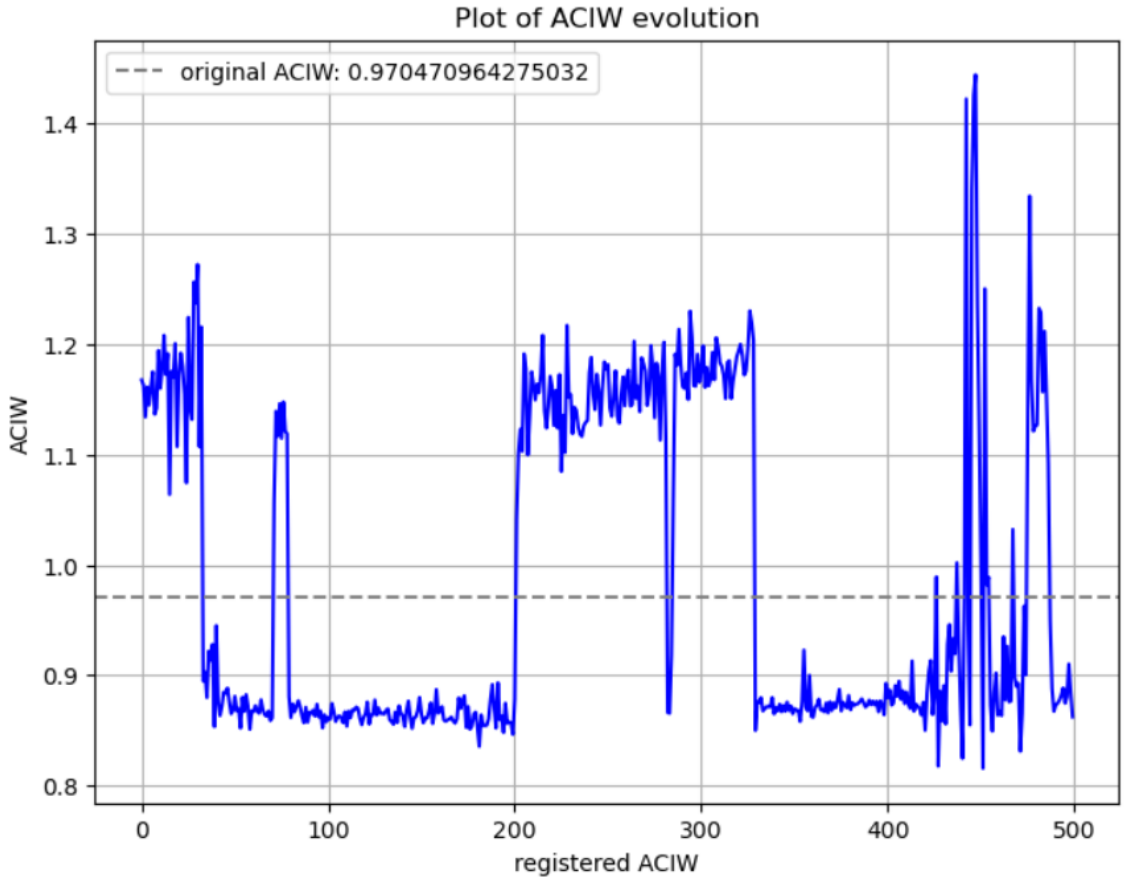

Figure 5.9: More extensive segment of the same ACIW progression shown in figure 5.6

This illustration further exposes the lack of a trend regarding the progression of the ACIW metric. The objective was for the confidence interval width to have an upward trend throughout the training process, but the plot may indicate that randomness is a prevalent influence on its evolution. Thus, this seems to provide further evidence that a mode collapse problem [1] is happening in this model.

5.4 Threats to validity

Despite positive results obtained with GASTeN in image classification, the adaptation of the concept to time series did not yield positive results so far. This may be due to several limitations of the current project, from the limited nature of the experiments run to some technical challenges and solutions implemented, as well as the approach followed.

Firstly, concerning the experiments conducted, we recognise that testing in only one dataset (the stock dataset), a single value for the number of pre-training epochs and a component α weight set to 1 makes the study much limited. However, due to time and computational restrictions, we were unable to extend the experiments.

As we encountered technical difficulties, one important analysis was monitoring the value of the gradients. One example of a gradient recorded can be seen in figure 5.10:

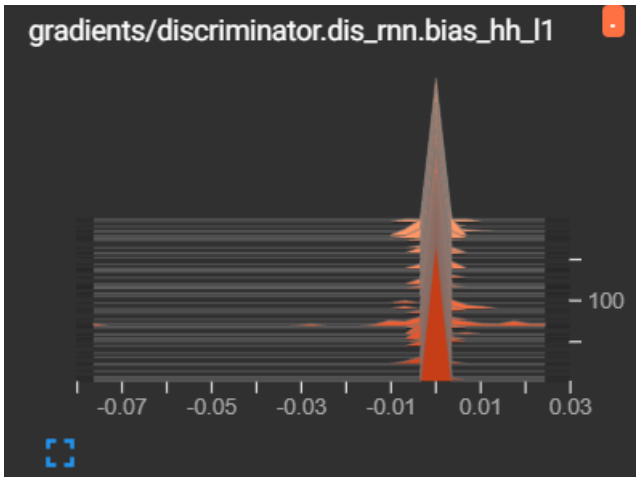

Figure 5.10: One of the 14 gradients recorded after GASTeN-TS's training process

The majority of the gradients recorded (present in appendix E) had a similar histogram, which might be indicative of another issue, possibly a vanishing gradient problem [9].

The difficulty of correctly adapting TimeGAN to a GASTeN framework (thereby accomplishing GASTeN-TS) played a good part in these barriers. The necessary changes were difficult to implement, and those were noticeable and conceivably changed how the base GAN worked.

5.5 Summary

This chapter presented the results of experiments conducted with the default TimeGAN and the modified GASTeN-TS algorithm. The primary objectives were to evaluate the quality of synthetic data generated by these models, assess their similarity and dissimilarity to the original data, and determine their effectiveness in lowering the confidence of a time series forecasting model.

For the default TimeGAN, the results were encouraging. The synthetic data generated demonstrated a realistic pattern while maintaining sufficient differences from the original data. Both the DTW distance calculations and visual inspection supported the conclusion that TimeGAN could serve as a reliable base for the GASTeN framework, meeting the requirements for generating synthetic data that could be used to stress test forecasting models.

However, the results for GASTeN-TS were more mixed. Although the modified algorithm produced realistic data and maintained some dissimilarity, it did not consistently achieve the intended goal of reducing the forecasting model's confidence. The Average Confidence Interval Width (ACIW) metric did not show the expected behaviour; the synthetic data produced a narrower confidence interval than the original data, suggesting higher model confidence rather than lower. Additionally, the progression of the ACIW metric did not indicate a clear trend, raising questions about the effectiveness of the modifications under the current experimental conditions.

Several limitations affected the validity of these findings. These include a restricted experimental scope (testing on a single dataset, fixed pre-training epochs, and specific hyperparameter settings), technical challenges, and other potential issues like vanishing gradients during training. These constraints highlight the need for further experimentation and refinement of the methodology to draw more definitive conclusions.

In summary, while GASTeN-TS shows potential, the observed limitations and unexpected results suggest that further investigation and adjustments are required to achieve its intended objectives fully.

Chapter 6

Conclusion

The integration of stress-testing methodologies tailored to the unique challenges of time series data reflects a commitment to advancing the reliability and robustness of forecasting models.

Using this motivation, we propose a method for generating data to induce low confidence in a time series forecasting predictor when a portion of it is used as an extension of the input train data. In this study, we adapted the GASTeN [4] framework to generate these Time-Series continuations, titled GASTeN-TS. The task was challenging for many reasons. First and foremost, it implied a change in concept. While GANs, and also GASTeN, generally aim to generate the full extension of synthetic data, GASTeN-TS's objective was to generate synthetic continuations of Time-Series data by using a portion of the generated data as an extension of real data, corresponding to realistic estimations of points after the last input data entry.

Secondly, we faced complex challenges. We needed to understand how to include data generation with different circumstances directly into the generator's training phase, namely inside the generator's forward pass function.

We empirically test our proposal using the stock dataset, present by default on the original TimeGAN implementation.[1] The evaluation uses quantitative metrics, one for the realism of the generated data (the DTW distance score) and another for how ambiguous the resulting mutated data is (its ACIW score). We compare results against the DTW score achieved by the base TimeGAN and the Confidence Interval Width realised by the fit of the original data into a predictor model.

GASTeN-TS aims to generate synthetic data that resembles the corresponding real data while maintaining a sufficient degree of difference. The DTW distance scores indicate that since they are not too low, and they are also not too high. Despite this, surprisingly, the confidence interval for mutated data is smaller when compared to the ones calculated using real data only. This may be because of the smaller perceived scale of the synthetic extension, indicative of a possible mode collapse issue [1].

[1] https://github.com/birdx0810/timegan-pytorch/

6.1 Future work

The results of this study highlight several areas for future research to enhance the GASTeN-TS framework and its application to stress testing in time series forecasting. While GASTeN-TS showed potential in generating synthetic data that is both realistic and sufficiently different from the original data, further improvements and explorations are necessary to achieve its objectives fully.

1. **Broader Experiments**: The hyperparameter range of the experimentation should be widened, by including more values for the weight hyperparameter and number of epochs in pre-train phase, as well as incorporating more datasets. This could provide more generalisation and a comprehensive understanding of the algorithm's behaviour under different circumstances. Appendix D details a possible expanded testing environment.

2. **Methodology refinement**: Future research should focus on refining the methodology for achieving GASTeN-TS's objectives. This issue, added to the signs of vanishing gradient problems and mode collapse issues [9, 1], may imply a change in the base GAN used. Another reason is that TimeGAN [27] was formulated with the correlation between multiple features in mind rather than the sequential relation between adjacent points.

3. **Extension to other forecasting methods**: Since ARIMA-based models were the ones fully used in this study, the usage of other models may be useful for gaining more insight into how GASTeN-TS affects different types of predictors and their confidence intervals. We propose using an unoptimised DL-based model, Recursive Neural Network (RNN), LSTM or GRU, which we could not test due to computing restrictions.

By addressing these areas, future work can build upon the findings of this study to advance the GASTeN-TS framework and its potential to improve our understanding and robustness of time series forecasting models through targeted stress testing.

References

[1] Issam Boukhennoufa, Delaram Jarchi, Xiaojun Zhai, Victor Utti, Saeid Sanei, Tracey K.M. Lee, Jo Jackson, and Klaus D. Mcdonald-Maier. A novel model to generate heterogeneous and realistic time-series data for post-stroke rehabilitation assessment. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 31:2676–2687, 2023.

[2] Jenna Burrell. How the machine 'thinks': Understanding opacity in machine learning algorithms. *Big Data and Society*, 3, 1 2016.

[3] Pieter Cawood and Terence van Zyl. Evaluating state of the art, forecasting ensembles- and meta-learning strategies for model fusion. *Preprints*, 1, 3 2022.

[4] Luís Cunha, Carlos Soares, André Restivo, and Teixeira Luís. *GASTeN: Generative Adversarial Stress Test Networks*, pages 91–102. 04 2023.

[5] Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Tadgan: Time series anomaly detection using generative adversarial networks. pages 33–43. Institute of Electrical and Electronics Engineers Inc., 12 2020.

[6] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. 12 2016.

[7] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. 27, 2014.

[8] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[9] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[11] Andreas Holzinger, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. What do we need to build explainable ai systems for the medical domain? 12 2017.

[12] John Hunter. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9:90–95, 06 2007.

[13] R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice, 3rd Edition*. OTexts, Melbourne, Australia, May 2021. https://otexts.com/fpp3/. Accessed on December 7, 2023.

[14] Remya K., Vidya K R, and M. Wilscy. *Detection of Deepfake Images Created Using Generative Adversarial Networks: A Review*, pages 25–35. 02 2021.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 12 2014.

[16] Yifan Li, Xiaoyan Peng, Jia Zhang, Zhiyong Li, and Ming Wen. Dct-gan: Dilated convolutional transformer-based gan for time series anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 35:3632–3644, 4 2023.

[17] Sharon Sone Mamua. Time series forecasting using generative adversarial network. Master's thesis, East Carolina University, Greenville, NC, May 2023.

[18] Gary Marcus. Deep learning: A critical appraisal, 2018.

[19] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. pages 220–229. Association for Computing Machinery, Inc, 1 2019.

[20] Brent Daniel Mittelstadt, Patrick Allo, Mariarosaria Taddeo, Sandra Wachter, and Luciano Floridi. The ethics of algorithms: Mapping the debate. *Big Data and Society*, 3, 12 2016.

[21] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. 5 2019.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. 12 2019.

[23] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. volume 13-17-August-2016, pages 1135–1144. Association for Computing Machinery, 8 2016.

[24] Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing Journal*, 90, 5 2020.

[25] Kate Smith-Miles and Mario Andrés Muñoz. Instance space analysis for algorithm testing: Methodology and software tools. *ACM Computing Surveys*, 55, 3 2023.

[26] Michael Weiss, André García Gómez, and Paolo Tonella. Generating and detecting true ambiguity: a forgotten danger in dnn supervision testing. *Empirical Software Engineering*, 28:146, 11 2023.

[27] Jinsung Yoon, Daniel Jarrett, and Mihaela Van Der Schaar. Time-series generative adversarial networks, 2019.

[28] Hong Zhu and Ian Bayley. Exploratory datamorphic testing of classification applications. pages 51–60. Association for Computing Machinery, 10 2020.

[29] Hong Zhu and Ian Bayley. Discovering boundary values of feature-based machine learning classifiers through exploratory datamorphic testing. *Journal of Systems and Software*, 187, 5 2022.

[30] Hong Zhu, Dongmei Liu, Ian Bayley, Rachel Harrison, and Fabio Cuzzolin. Datamorphic testing: A method for testing intelligent applications. pages 149–156. Institute of Electrical and Electronics Engineers Inc., 5 2019.

Appendix A

Implementation details

The experiments were executed using the Python 3 programming language. TimeGAN and GASTeN-TS are based on the Pytorch framework [22]. The library has a variety of neural network implementations, optimisers, and utility helper functions that ease the tasks needed in these development and research tasks. These details are crucial for reproducing the results and understanding the overall methodology.

For visualisation and result analysis, we use the Matplotlib library [12], as well as Pytorch [22], more specifically the SummaryWriter [1] from Tensorboard [2], to monitor the gradients.

The files generated in the program's execution are stored in the system for later plotting purposes. This is done using the Pickle module from Python. These files include:

- The arguments used in the particular experiment
- The full original data is partitioned into train and test sets after the data pre-processing routine.
- The Synthetic Time Series extension, resulting from the trained model, can be compared with the test section of the real data.
- The ACIW calculated using only the original processed data for comparison purposes.
- The ACIW calculated during the second training phase of GASTeN-TS, as explained in section 3.2, each time the Generator's loss function is called and calculated.
- The final model's state dictionary. That way, the final model can be loaded to generate data based on its training.

Finally, the source code for this project is saved on GitHub [3]

[1] https://www.tensorflow.org/api_docs/python/tf/summary/SummaryWriter
[2] https://www.tensorflow.org/tensorboard
[3] https://github.com/H0wl3r2001/timegan-pytorch/

Appendix B

Program's arguments

This appendix details the arguments and other functions used in the programs of the default TimeGAN and GASTeN-TS. It provides information regarding the conditions on which the GAN ran in the experiments detailed in sections 4.3 and 4.4, as well as code snippets of interest. This information is useful if a third party wants to replicate the experiments.

Listing B.1: Sinusoidal Dataset generator function

```
import numpy as np
import pandas as pd

def sine_data_generation(no, seq_len, dim, min_waves=5, max_waves=10):
    """Sine data generation with varying frequencies ensuring a
    minimum and maximum number of complete waves.

    Args:
        - no: the number of samples
        - seq_len: sequence length of the time-series
        - dim: feature dimensions
        - min_waves: minimum number of complete sine waves per sequence
        - max_waves: maximum number of complete sine waves per sequence

    Returns:
        - data: generated data
    """
    data = list()

    # Calculate the frequency range to achieve between
    min_waves and max_waves
    f_min = min_waves / seq_len
    f_max = max_waves / seq_len
```

```
    for i in range(no):
        temp = list()
        for k in range(dim):
            freq = np.random.uniform(f_min, f_max)
            phase = np.random.uniform(0, np.pi * 2)
            temp_data = [np.sin(freq * j + phase) for j in range(seq_len)]
            temp.append(temp_data)

        temp = np.transpose(np.asarray(temp))
        temp = (temp + 1) * 0.5
        data.append(temp)

    return data

# Example usage:
no = 10  # number of samples (datasets)
seq_len = 500  # sequence length (time points)
dim = 1  # feature dimensions
min_waves = 15  # minimum number of complete waves
max_waves = 30  # maximum number of complete waves

data = sine_data_generation(no, seq_len, dim, min_waves, max_waves)

# Prepare the data for the CSV
csv_data = []

for idx in range(seq_len):
    for dataset_idx, sample in enumerate(data):
        csv_data.append([dataset_idx, idx, sample[idx][0]])

# Convert the list to a DataFrame
df = pd.DataFrame(csv_data, columns=['', 'Idx', 'sin_val'])

# Save the DataFrame to a CSV file
csv_file_path = 'data/sine_wave_data_formatted_min_max_waves.csv'
df.to_csv(csv_file_path, index=False, header=False)
```

Listing B.2: GASTeN-TS arguments

```
parser = argparse.ArgumentParser()
parser.add_argument(
    '--device',
    choices=['cuda', 'cpu'],
    default='cpu',
    type=str)
parser.add_argument(
    '--exp',
    default='test',
    type=str) #directory of the results
parser.add_argument(
    "--is_train",
    type=str2bool,
    default=True)
parser.add_argument(
    '--seed',
    default=0,
    type=int)
# Data Arguments
parser.add_argument(
    '--max_seq_len',
    default=100,
    type=int)
parser.add_argument(
    '--train_rate',
    default=0.5,
    type=float) #train_test_split
# Model Arguments
parser.add_argument(
    '--emb_epochs',
    default=50,
    type=int)
parser.add_argument(
    '--sup_epochs',
    default=50,
    type=int)
parser.add_argument(
    '--gan_epochs',
    default=200,
```

```
        type=int)
    parser.add_argument(
        '--batch_size',
        default=128,
        type=int)
    parser.add_argument(
        '--hidden_dim',
        default=20,
        type=int)
    parser.add_argument(
        '--num_layers',
        default=3,
        type=int)
    parser.add_argument(
        '--dis_thresh',
        default=0.15,
        type=float)
    parser.add_argument(
        '--optimizer',
        choices=['adam'],
        default='adam',
        type=str)
    parser.add_argument(
        '--learning_rate',
        default=1e-3,
        type=float)
```

Listing B.3: Default TimeGAN arguments used

```
parser = argparse.ArgumentParser()
# Experiment Arguments
parser.add_argument(
    '--device',
    choices=['cuda', 'cpu'],
    default='cpu',
    type=str)
parser.add_argument(
    '--exp',
    default='test',
    type=str)
parser.add_argument(
    "--is_train",
    type=str2bool,
    default=True)
parser.add_argument(
    '--seed',
    default=42,
    type=int)
# Data Arguments
parser.add_argument(
    '--max_seq_len',
    default=100,
    type=int)
parser.add_argument(
    '--train_rate',
    default=0.5,
    type=float)
# Model Arguments
parser.add_argument(
    '--emb_epochs',
    default=60,
    type=int)
parser.add_argument(
    '--sup_epochs',
    default=60,
    type=int)
parser.add_argument(
    '--gan_epochs',
```

```
        default=60,
        type=int)
    parser.add_argument(
        '--batch_size',
        default=128,
        type=int)
    parser.add_argument(
        '--hidden_dim',
        default=20,
        type=int)
    parser.add_argument(
        '--num_layers',
        default=3,
        type=int)
    parser.add_argument(
        '--dis_thresh',
        default=0.15,
        type=float)
    parser.add_argument(
        '--optimizer',
        choices=['adam'],
        default='adam',
        type=str)
    parser.add_argument(
        '--learning_rate',
        default=1e-3,
        type=float)
    args = parser.parse_args()
```

Appendix C

Preliminary screening experiments

At the start of the proposed experiments, we noticed some issues regarding achieving the objectives. For example, the graph plotting the ACIW did not seem to change in each experiment, and originally, the resulting synthetic data had almost no similarity to the corresponding original data. This meant that either there were some issues with the modifications implemented, or new modifications were needed, or that the GAN was not learning to optimise around this new metric for then unknown reasons. To verify that the issue was not a technical one and, at the same time, try to find possible explanations for what was occurring, the following verifications were done:

- **Generator loss monitoring**: we saved the evolution of the generator's loss function to monitor its progression throughout each calculation during the training phase.
- **Tensor type verification and normalisation**: one of the hypotheses was that the tensor would ignore the new addition if its type did not correspond to the type seen in the other components of the Generator's Loss function.
- **Monitoring the value of the gradients**: a recurrent problem with Neural Networks is the Vanishing Gradient phenomena [9]. Although TimeGAN's architecture includes Networks of the GRU variety, which supposedly is a way to prevent this situation, we decided to monitor the gradients.

As these verifications were being done, we noticed some key details:

- The data partition between train and test sets was done with a hidden data shuffling component. This, in turn, was a font of major issues, as the sequential relation between consecutive Time Series points was being destroyed, and the correlation between features was more emphasised. This detail compelled us to experiment on the original TimeGAN in Section 4.3.
- Secondly, after solving the previous point, we noticed that the synthetic data that resulted from the GASTeN-TS training routine was scaled shorter than the original data. We think this stems from a mode collapse of the model, an issue where the generated data fails to capture all the relevant information from the original dataset. This subject is a persistent

problem of GANs, particularly in time series related ones [1]. To counter this, a scaling routine was implemented using a basis formula that is also used in Min-Max Scaling:

$$X\_scaled = \frac{X_s - min_s}{max_s - min_s} * (max_o - min_o) + min_o \tag{C.1}$$

Where X denominates the whole set of values, s refers to the synthetic, o refers to the original, max is the maximum value and min is the minimum value. We observed that scaling said synthetic data into the scaling range of the original solved this issue. This was implemented on GASTeN-TS.

Appendix D

Future experiments

This appendix will further explore the possible future experiments, focusing on the hyperparameters and further dataset prepared. These experiments were not carried out due to time constraints and technical challenges.

D.1 New hyperparameter range

Similar to the original GASTeN iteration [4], we were also interested in exploring the algorithm with different values for the weight of the new ACIW metric in the generator's loss function and the number of epochs in the pre-training phase. In this context, this phase corresponds to where the unaltered GAN trains all the networks together after the training process for both the embedding and supervisor networks is carried out.

Analysing the source code and the coefficients multiplied for each of the generator's loss function components, we defined possible values for the weight. Setting this value to 0 makes the GAN behave like it was originally.

The different values proposed for the different hyperparameter values are further defined in the following table D.1.

Hyperparameter	**Values**
Weight (α)	0, 1, 25, 50, 75, 100
Num. epochs 1st joint-train phase	0, 2, 5, 10

Table D.1: Values proposed for each GASTeN-TS hyperparameter.

However, these experiments could not be carried out due to technical challenges. After an extensive verification and implementation of possible solutions for these obstacles, more defined in Appendix C, we decided first to run some experiments on the base TimeGAN, which will be described ahead. Afterwards, we run simpler experiments on the GASTeN-TS to obtain minimally viable results for evaluating the proposed method.

D.2 Sinusoidal Dataset

This dataset represents 10 sinusoidal functions with different phase and wave frequencies to get different curves, each with approximately 3700 points. These sinusoidal functions are generated with varying parameters to capture a diverse range of wave patterns. The points are sampled regularly to ensure consistency and comprehensively represent each function's behaviour.

This dataset was then saved and was expected to be used to test our model. Specifically, the idea was to assess the model's ability to generalise from the training data to unseen data, ensuring that the predictions remain accurate and reliable across different scenarios.

The function used to generate this file is shown in Appendix B. It incorporates parameters such as frequency and phase shift, which are essential for defining the characteristics of each sinusoidal wave. The detailed implementation in the Appendix is a reference for reproducing the dataset, allowing others to verify and build upon the work.

Also, the way this sinusoidal dataset was formed follows the principles of Datamorphing (*cf* chapter 2.3.1). The sinusoidal function is the seed, and the parameters used in the generator are the *datamorphisms*, which, when added to the base seed, generate new, different, but also similar and strongly related data. This method expands the data that can be used to test.

We chose not to use this sinusoidal function because of the technical challenges and timing restrictions we encountered (*cf* chapter **??**). As the experiments were done on the base TimeGAN (*cf* chapter 4.3), we decided to use the stock dataset (*cf* chapter 4.1.1) to test the new algorithm.

Appendix E

GASTeN-TS recorded gradients

This appendix provides more information regarding the gradients recorded and observed after GASTeN-TS's experiment using Tensorboard. The command used to show the information is shown in this chapter as well. The file used as the basis to showcase the info is an automatic product of each program run.

Listing E.1: Command line used to get the information about the gradients

```
tensorboard --logdir=tensorboard/test
```

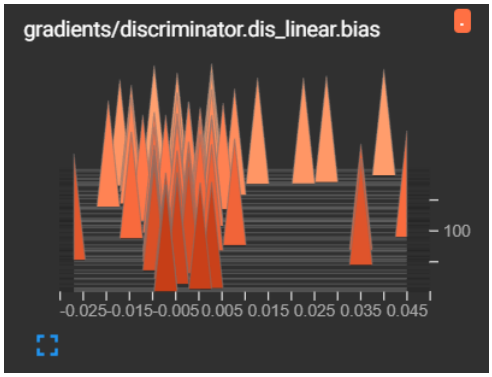

Figure E.1: 1st gradient recorded

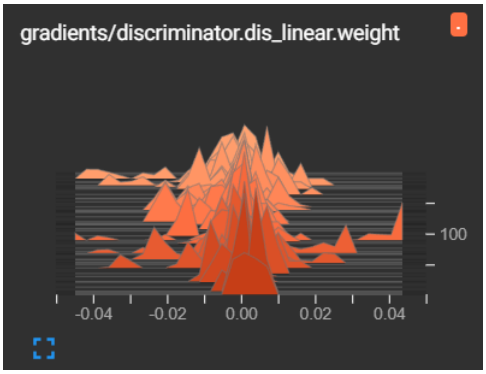

Figure E.2: 2nd gradient recorded

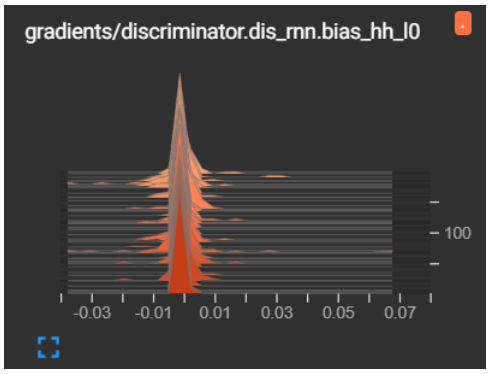

Figure E.3: 3rd gradient recorded

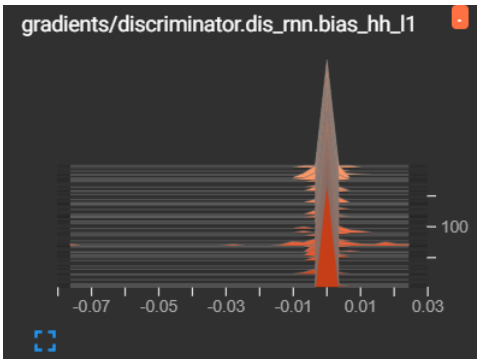

Figure E.4: 4th gradient recorded

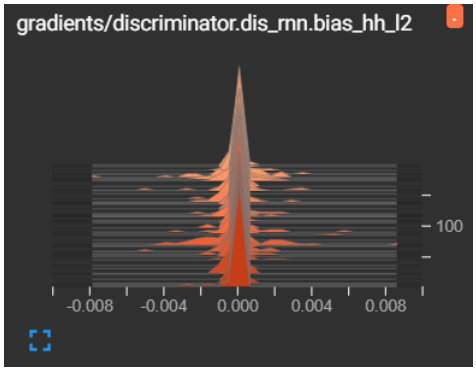

Figure E.5: 5th gradient recorded

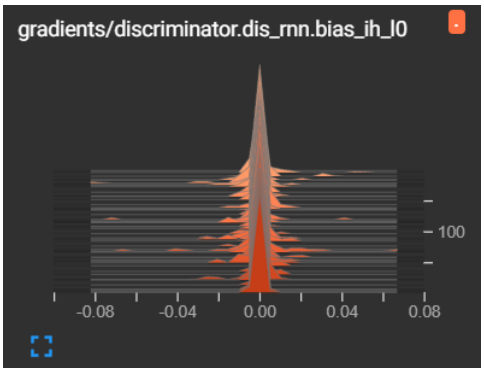

Figure E.6: 6th gradient recorded

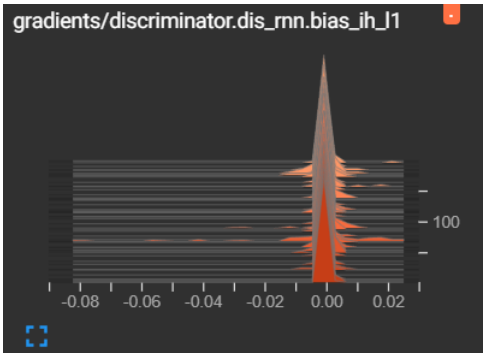

Figure E.7: 7th gradient recorded

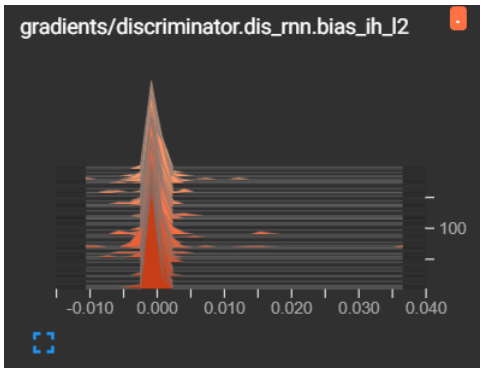

Figure E.8: 8th gradient recorded

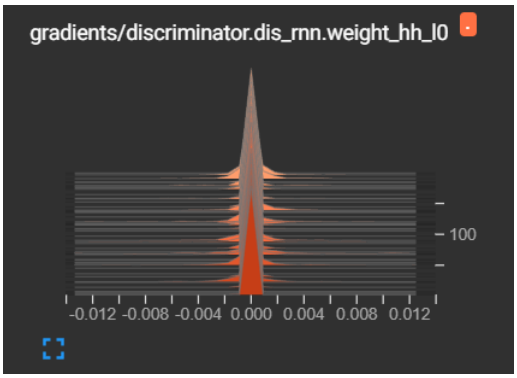

Figure E.9: 9th gradient recorded

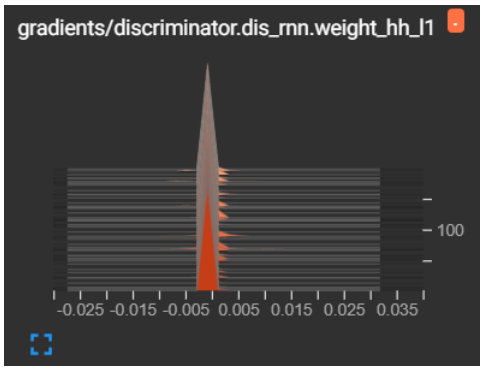

Figure E.10: 10th gradient recorded

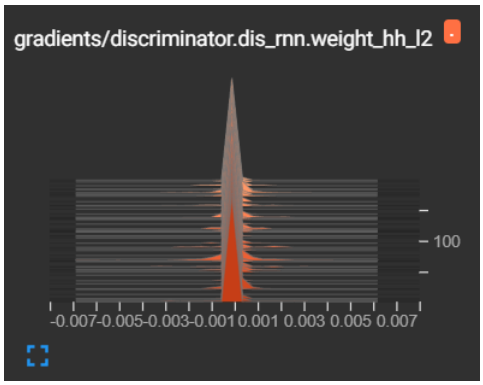

Figure E.11: 11th gradient recorded

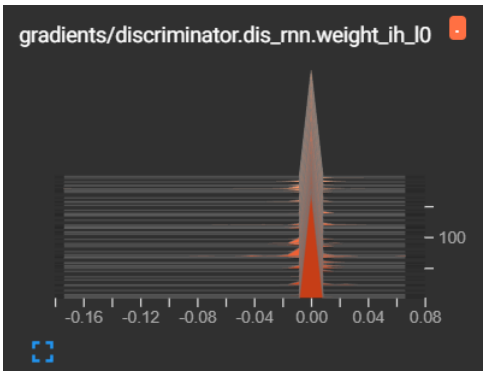

Figure E.12: 12th gradient recorded

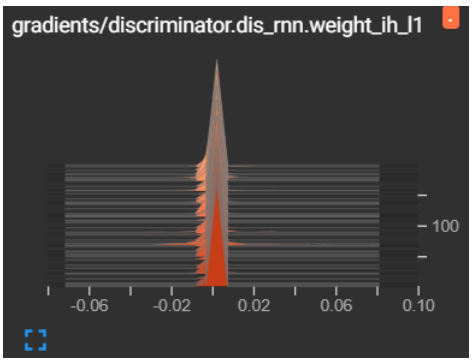

Figure E.13: 13th gradient recorded

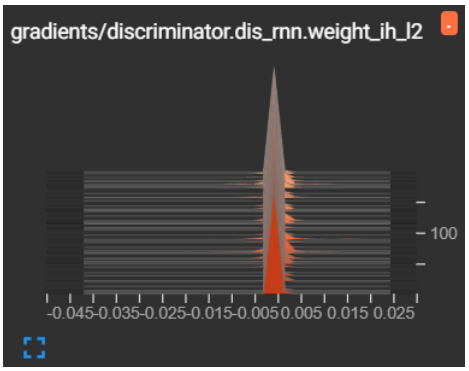

Figure E.14: 14th gradient recorded

Appendix F

Plot full extension

This appendix provides the full extension of some of the plots presented in chapter 5. The images in that chapter were a shortened version of the following for ease of understanding and interpretation.

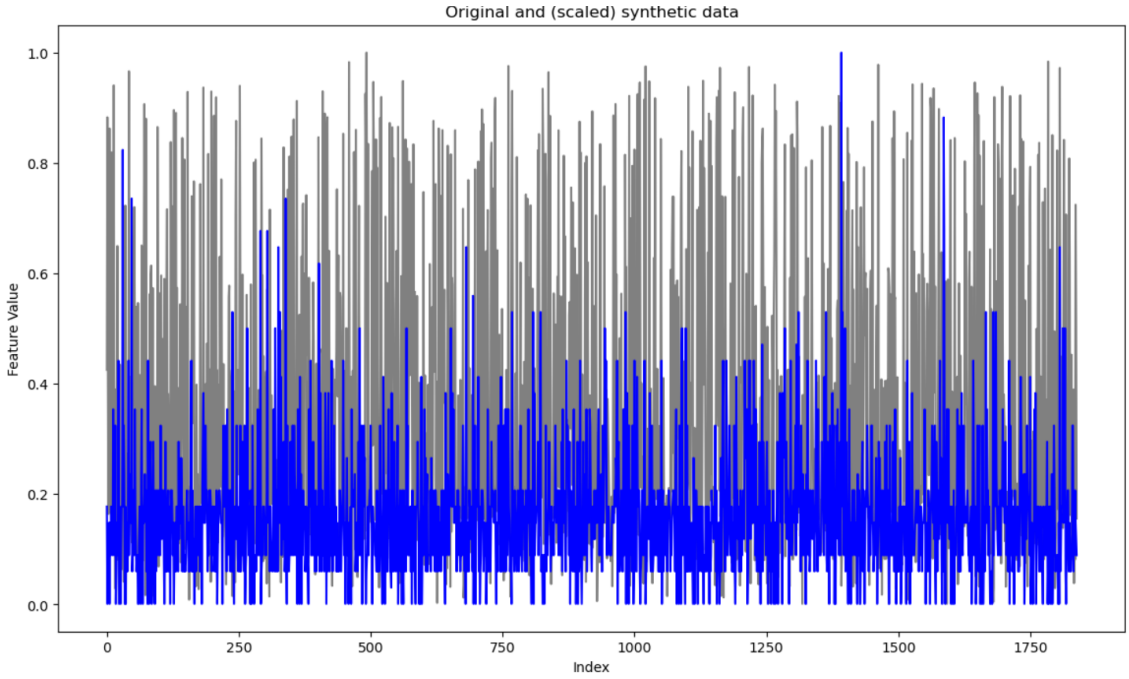

Figure F.1: Full extension of the plot shown in figure 5.3

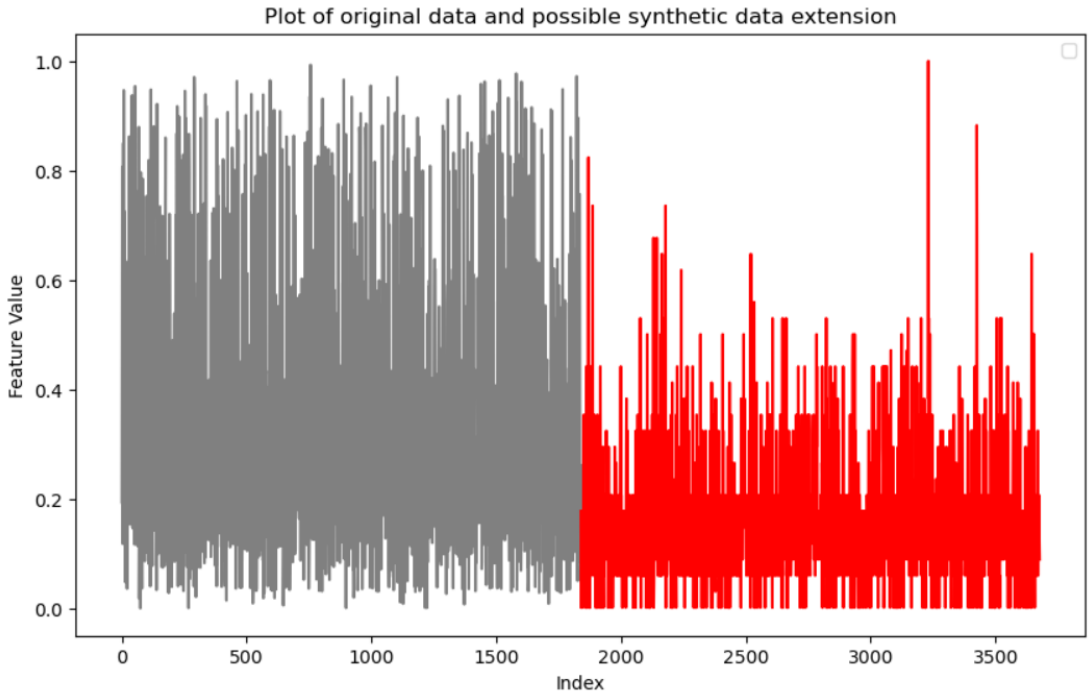

Figure F.2: Full extension of the plot shown in figure 5.8

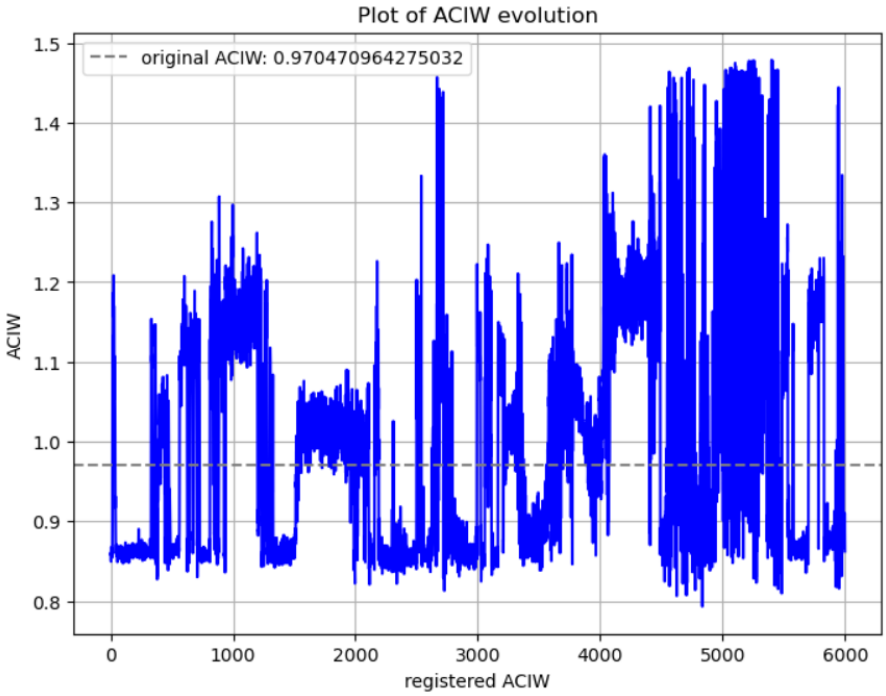

Figure F.3: Full extension of the plot shown in figure 5.6 and 5.9

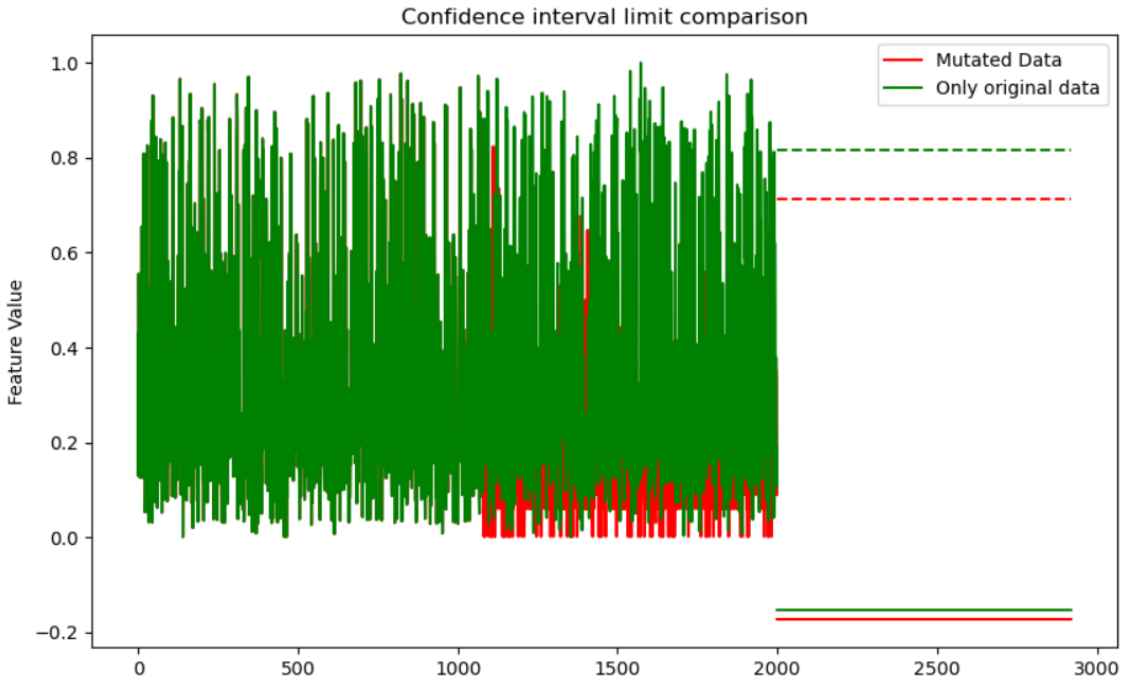

Figure F.4: Confidence interval of the mutated and original data. An extension of the figure 5.7