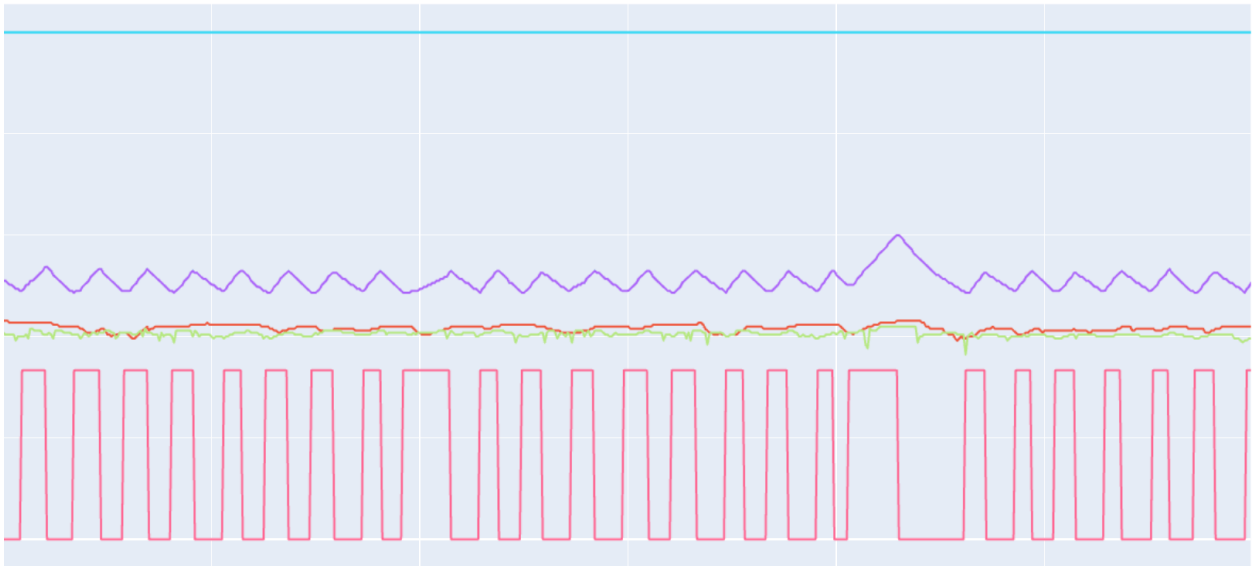

Chalmers University of Technology

Synthetic time-series data generation using Generative Adversarial Networks

Master's thesis in Complex Adaptive Systems

DEEPAK GURU GANESAN

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Synthetic time-series data generation using Generative Adversarial Networks

Development of a deep neural network model for the synthetic time-series data generation

DEEPAK GURU GANESAN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Division of Division name
Chalmers University of Technology
Gothenburg, Sweden 2021

Vehicle Sensor Time Series Data Generation using Generative Adversarial Networks
Development of a deep neural network model for the synthetic time-series data generation
DEEPAK GURU GANESAN

Supervisor: Kuo-Yun Liang, Scania CV AB
Supervisor/Examiner: Mats Granath, Department of Physics

Master's Thesis 2021
Department of Physics
Division of Complex Adaptive Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A 2-D graph showing the normalized multivariate time-series data.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Abstract

Data, in this new hybrid era, is the driving force for many industries. The automotive industry is no exception to this, and the industry has seen an increasing reliance on data for its operations. The automotive sector, nowadays, offers more services than just selling a vehicle. It now provides complete mobility solutions like connected, shared, and autonomous electric vehicles with personalized options. Such services are possible only with the use of high quality data. Manual data collection from automobiles is an expensive and laborious task, due to which only a sparse amount of high quality data is collected. Reduced data means that the operations and analysis performed are also limited, which makes the company able to offer less services to the customers. One solution to increase the data is to generate it synthetically using deep neural networks.

Though there are many methods to generate data synthetically, most of them have limitations on developing diverse data and preserving the temporal dynamics of the original data. This thesis focuses on those issues and studies the possibility of designing a neural network model to generate varied time-series data which has the characteristics of the original data. In this thesis, a Generative Adversarial Network (GAN) is implemented to synthetically generate time-series data.

A conventional method of building a machine learning model from scratch is followed in this thesis, after weighing several factors. The relevant training data is collected from the vehicle and then pre-processed to improve the quality of the data. Following this, an initial GAN model is developed that contains the generator and discriminator structure. Then, an enhanced model with supervised learning mechanism called timeGAN model is developed for achieving more realistic synthetic data. This model is then evaluated with suitable metrics, both in a qualitative and quantitative manner. The thesis aims to resolve the issue of scarce data and thus, paves the way for effective predictive maintenance of vehicles and better services to the customers.

Keywords: Time-series, GAN, Deep learning, Generator, Discriminator, Temporal dynamics, TimeGAN, Data distribution.

Acknowledgements

I would like to express my sincere appreciation to my thesis supervisor Kuo-Yun Liang for his guidance throughout the thesis. His insightful ideas and thoughts are greatly appreciated in this project work. And without his constant support and encouragement, the thesis goals would not have been accomplished. I am equally indebted to my supervisor and examiner Mats Granath, who helped me with his academic excellency and constant assistance during tough times.

I wish to express my deepest gratitude to my thesis partner Sofia Nord who has supported me in every step of the thesis through her constructive opinions and fascinating ideas. The technical and physical contribution by the Edge team members - Abishek Srinivasan, Juan Carlos, and Pär Sundback is highly important and I am greatly thankful to each one of them. I also wish to pay my special regards to Frida Nellros who has helped me during the interview and on-boarding process, and Katarina Prytz who constantly monitored the progress of the thesis, and the thesis students.

I wish to thank my friend Aravind Inbasekaran and Prema Manickavasagam for their technical inputs and mental support during my whole masters. I also wish to acknowledge the support and love of my friends and family, and this project would not have been possible without their support.

Deepak Guru Ganesan,
Gothenburg, June 2021.

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Company Background . . . 1
1.2 Thesis Background . . . 1
1.2.1 Aim of the thesis . . . 2
1.2.1.1 Research Questions . . . 2
1.2.2 Scope of the thesis . . . 3
1.2.3 Goals . . . 3
1.3 Related Work . . . 3

2 Theory **5**
2.1 Time-series Data . . . 5
2.1.1 The Trend . . . 5
2.1.2 Seasonal time-series data . . . 6
2.1.3 Irregular time-series Data . . . 6
2.1.4 Univariate and multivariate Data . . . 7
2.2 Artificial Neural Networks . . . 7
2.3 Feed Forward Neural Networks . . . 7
2.4 Convolutional Neural Networks . . . 8
2.5 Recurrent Neural Networks . . . 9
2.5.1 Many-to-Many RNN . . . 10
2.5.1.1 Tx = Ty . . . 10
2.5.1.2 Tx != Ty . . . 10
2.5.2 Many-to-one RNN . . . 11
2.5.3 Long Short Term Memory . . . 11
2.6 Generative Modeling . . . 13
2.6.1 Generative Adversarial Networks . . . 13
2.6.1.1 Theoretical Loss Functions of GAN . . . 14
2.6.2 TimeGAN . . . 15
2.6.2.1 Embedding and Recovery Functions . . . 15
2.6.2.2 Generator and Discriminator . . . 16

3 Methodology **17**
3.1 Literature Review . . . 17

3.2 Data set . . . 17
3.3 Data Visualisation . . . 18
3.3.1 Graphs using Plotly . . . 18
3.3.2 3D Plot using Principal Component Analysis . . . 19
3.4 Data Pre-processing . . . 20
3.4.1 Data Loading . . . 20
3.4.2 Column Dropping . . . 20
3.4.3 Conditional Shifting . . . 20
3.4.4 Normalization . . . 21
3.5 Standard GAN model . . . 21
3.5.1 CNN based GAN . . . 21
3.5.1.1 Generator architecture . . . 22
3.5.1.2 Discriminator architecture . . . 22
3.5.1.3 Experiments . . . 22
3.5.2 CNN based GAN algorithm . . . 23
3.5.3 CNN based GAN network training . . . 23
3.6 Enhanced GAN model . . . 24
3.6.1 Architecture of the timeGAN network . . . 24
3.6.2 TimeGAN algorithm . . . 25
3.6.3 Randomized Sliding Window Technique . . . 26
3.6.4 TimeGAN training . . . 26
3.6.5 Experiments . . . 27
3.7 Evaluation methods . . . 28
3.7.1 Visual evaluation . . . 28
3.7.2 Quantitative evaluation . . . 28
3.7.2.1 Regression score . . . 29
3.7.2.2 Mahalanobis score . . . 30

4 Results **31**
4.1 CNN based GAN . . . 31
4.2 TimeGAN . . . 32
4.2.1 Pre-training Loss . . . 32
4.2.2 Joint training losses . . . 33
4.2.3 Visual results . . . 35
4.2.3.1 Plotly graphs showing synthetic and original time-series data . . . 35
4.2.3.2 PCA plot showing distributions of original and generated data . . . 38
4.2.4 Quantitative Evaluation of timeGAN . . . 40
4.2.4.1 Regression score for forecasting using RNN . . . 40
4.2.4.2 Mahalanobis score for anomaly detection using GMM 42

5 Conclusion **47**

Bibliography **49**

A Appendix 1 **I**

A.1 CNN Based GAN Network summary I
A.1.1 Generator architecture I
A.1.2 Discriminator architecture II
A.2 TimeGAN Network summary III
A.2.1 Generator architecture III
A.2.2 Discriminator architecture III
A.2.3 Embedding function architecture IV
A.2.4 Recovery function architecture IV
A.3 Visual representation V
A.3.1 Feature 1 V
A.3.2 Feature 2 V
A.3.3 Feature 3 VI
A.3.4 Feature 4 VI
A.3.5 Feature 5 VII
A.3.6 Feature 6 VII
A.3.7 Feature 7 VIII
A.3.8 Feature 8 VIII
A.4 Contribution of thesis students IX
A.4.1 Deepak's contributions: IX
A.4.2 Sofia's contributions: IX

List of Figures

2.1 A sample non-linear downward trend time series graph [16][5] 6
2.2 A sample seasonal time-series graph with cyclic variations [34] 6
2.3 A sample irregular time-series graph without any definite patterns [23] 6
2.4 A simple feed forward neural network with one hidden Layer. 7
2.5 Simplified 1 dimensional CNN architecture [22] 8
2.6 A folded and unfolded RNN architecture [10][9] 9
2.7 Many-to-many RNN architecture [33] 10
2.8 Many-to-one RNN architecture [33] 11
2.9 A detailed view of one LSTM repeating module [32] 11
2.10 The notations for the LSTM module present in figure 2.9 [32] 12
2.11 A simple architecture of GAN model [12] 14
2.12 TimeGAN Network Block Diagram [35] 15

3.1 The overall work flow of the thesis Project 17
3.2 A sample dataframe of the time-series data used in this thesis 17
3.3 Unnormalized time-series data features plotted using plotly 18
3.4 3-dimensional graph plotted using PCA for training data. 19
3.5 Normalized training time-series data plotted using plotly. 21

4.1 Original and generated time-series data for 2 minutes time duration . 31
4.2 Auto-encoder network diagram 32
4.3 Auto-encoder pre-training loss 33
4.4 Graph showing generator and discriminator losses for 2 settings . . . 34
4.5 Graph showing generator losses for 2 settings 35
4.6 Original and synthetic data trained with 30 second sequence length . 36
4.7 Original and synthetic data trained with 60 second sequence length . 37
4.8 Distribution of original and fake data features for the model trained with (30 samples) 38
4.9 Distribution of original and fake data features for the model trained with (60 samples) 39
4.10 Two graphs showing TRTS technique for GMM model 42
4.11 Box pot showing mahalanobis scores of original and synthetic data . . 43
4.12 Two graphs showing TSTR technique for GMM model 43
4.13 Two graphs showing TRTS technique for GMM model 44
4.14 Box pot showing mahalanobis scores of original and synthetic data . . 45
4.15 Two graphs showing TSTR technique for GMM model 45

A.1 CNN based GAN generator summary I
A.2 CNN based GAN discriminator summary II
A.3 TimeGAN generator summary III
A.4 TimeGAN discriminator summary III
A.5 TimeGAN embedding function summary IV
A.6 TimeGAN recovery function summary IV
A.7 Original and synthetic data's Feature 1 plotted in graph V
A.8 Original and synthetic data's Feature 2 plotted in graph V
A.9 Original and synthetic data's Feature 3 plotted in graph VI
A.10 Original and synthetic data's Feature 4 plotted in graph VI
A.11 Original and synthetic data's Feature 5 plotted in graph VII
A.12 Original and synthetic data's Feature 6 plotted in graph VII
A.13 Original and synthetic data's Feature 7 plotted in graph VIII
A.14 Original and synthetic data's Feature 8 plotted in graph VIII

List of Tables

3.1 Table showing various settings used in the CNN based GAN model . 22
3.2 Table showing various settings used in the timeGAN model 28

4.1 Table showing regression scores for setting 1 40
4.2 Table showing percentage difference for different evaluation techniques 40
4.3 Table showing regression scores for setting 2 41
4.4 Table showing percentage difference for different evaluation techniques 41

List of abbreviations

APS	Air Pressure System
CAN	Control Area Network
DTW	Dynamic Time Warping
ECU	Electronic Control Unit
GAN	Generative Adversarial Networks
GMM	Gaussian Mixture Model
IID	Independent and Identically Distributed
JS	Jensen-Shannon divergence
KL	Kullback-Leibler divergence
LSTM	Long Short Term Memory
MAE	Mean Absolute Error
MSLE	Mean Squared Log Error
PCA	Principal Component Analysis
RNN	Recurrent Neural Networks
TRTR	Train on Real Test on Real
TRTS	Train on Real Test on Synthetic
TSTR	Train on Synthetic Test on Real
TSTS	Train on Synthetic Test on Synthetic
VAE	Variational Auto Encoders

1
Introduction

Scania CV AB, a world-leading provider of mobility solutions in the commercial vehicle segment is known for its flexible modular product system offering tailor-made solutions to its customers. The company is actively involved in research and development providing high-quality connected solutions to its clients. Various research activities in the company concentrate on anomaly detection applications to detect the faults beforehand leading to successful predictive maintenance of the vehicle. One key issue in detecting the faults effectively is the non-availability of a large amount of data for analysis, which is addressed in this report. This issue is rectified by synthetically generating the time-series data using Generative Adversarial Networks (GAN).

Knowing about the company, and the research activities carried out there are necessary to get hold of the complexities in the thesis, and this chapter will explain the background, the purpose, and the problems associated with the thesis.

1.1 Company Background

Scania CV AB is the leading manufacturer of heavy trucks, lorries, busses, and diesel engines for marine and other industrial applications, having its headquarters in Södertälje, Sweden. The company is a subsidiary of Traton Group - a holding company for the heavy commercial vehicle division of the Volkswagen group. Scania has research centers in Sweden, Brazil, and India, with its production facilities concentrated in various countries from Europe, Latin America, Africa, and Asia.

Scania's commitment towards changing society into a fossil-free environment and sustainable life can be seen from its introduction of a wide range of electric mobility solutions. The company has set a goal to achieve its science-based target of reducing 50% of CO_2 from its operations by 2025 when compared to 2015 [14] and aims towards becoming a carbon-neutral company.

1.2 Thesis Background

The Air Pressure System (APS) is one of the vital modules in a heavy commercial vehicle. The system maintains optimal pressure values in different sub-systems and pumps the pressure to the sub-systems whenever there is a pressure drop. The sensor sends analog pressure values of various components to Electronic Control

Unit (ECU) periodically, and it is processed further for the functioning of the vehicle. The readings obtained from the sensor contain multiple variables and it is time-series readings of sensor values. The time-series data obtained is systematic data that is multivariate and each of its variables' range is defined. The data is dependent on many external factors like speed, braking, and road conditions as well, which make the data more complicated and not follow the seasonal trends. This time-series data contains readings that depict whether the APS works normally or it is in a fault state. Currently, the available data resources are unbalanced and create a bias, containing more data with the normal functioning of the system and less data with faulty systems. The balanced datasets are very important for further proceeding with the predictive maintenance of the vehicle. To develop balance in the datasets, more fault data is required, and collecting this data manually from the vehicle requires hours of driving and the process is very expensive.

There are many methods available to increase the balance in the data set, and only some methods prove to be efficient and provide promising results. This thesis proposes a novel solution to increase the balance in the data by generating the data artificially using deep neural networks. GAN is a part of broad spectra of machine learning algorithms first proposed by Ian Goodfellow et al., in 2014 [13]. This paper describes about two neural networks which are trained simultaneously and produce generative models. One neural network, the generator, that can learn the distribution of training data and generate data x with inputs as prior noise variables and another neural network, the discriminator, which outputs a scalar which represents the likelihood that data x came from the generator rather than from original training data. Thus the generator tries to cheat the discriminator by producing more and more realistic data while the discriminator tries to identify the fake data as much as possible. Thus the two networks train in an adversarial manner and generate data with similar data distribution as training data.

1.2.1 Aim of the thesis

The primary aim of this project is to investigate different methods used to generate fake time-series data and develop a reliable neural network that can generate realistic and diverse data. The important questions that can be answered at the end of this research work are mentioned below.

1.2.1.1 Research Questions

1. Which network models can generate synthetic data that has characteristics similar to the original data distribution?
2. Whether the generated data is unique, and diverse, and can the model produce long sequences of synthetic data with small quantity of training data ?
3. What are the suitable evaluation metrics available to estimate the performance of the model?

1.2.2 Scope of the thesis

One of the prevalent methods of increasing the data repository these days is by employing methods to artificially generate 'fake data', especially where the datasets are skewed, and contain large quantities of minority classes. Collecting time-series data from the vehicles is a traditional task followed for many years in the automobile companies, but they prove to be a time-consuming and expensive task. To overcome these issues, this thesis aims to artificially synthesize data that closely resembles the original data.

The primary aim is to build a generalized neural network model that can be used to generate any kind of time-series data with minimal changes and thus the generated data can later be used to train anomaly detection problems used in different projects in the company. The generated data should also be compared against the distribution of the original data by suitable evaluation metrics.

1.2.3 Goals

1. The source of data must be studied thoroughly to understand the meaning of the data. Various data representation methods are to be analyzed to get a better realization of the importance of each data.
2. Finding proper evaluation metrics to assess the reliability of the model should be decided as a part of the pre-study and proper justification has to be made for choosing the metrics.
3. Data pre-processing should be done to make the data feasible for use in the project. The goal is to preserve the properties of the present data by studying different pre-processing techniques and extract useful information by the means of visual representations of the data.
4. The creation of an unsophisticated data generation model is the next goal. The model should focus on fulfilling the objectives and a suitable model should be chosen with the help of research made during the literature study.
5. The next goal is to think of a better model with enhanced capabilities and parameters which can overcome the limitations of the naive model presented before.
6. The final goal is to present the generated data in a 2-dimensional view, compare the correlations of different features of original and synthetic data and assess the quality of generated data using suitable evaluation metrics. The final result must satisfy the proposed objective from a technical point of view.

1.3 Related Work

In the field of synthetic time-series data generation, many approaches have been proposed by different researchers over the period of time. Almost all the approaches are based on three main generative modeling approaches. The first approach is auto-regressive models put forward by Hugo Larochelle and Iain Murray in 2011 through their neural auto-regressive distribution estimator [21]. This paper captures the

distribution of the given data and later uses it for prediction and forecasting. But the paper lacked the problem of capturing multidimensional data and the use of unsupervised learning models for stochastic estimation of the future data.

The second approach is the Variational Auto Encoders (VAE) introduced by Kingma et al., through the article Auto-Encoding Variational Bayes[20] in 2013, which works with encoder, decoder, and a loss function. The VAE network estimates the observation in the latent space in a probabilistic manner. Thus the encoder is designed in such a way as to estimate the probability distribution of the latent attribute rather than estimating a single output of the latent attribute. The third approach of generative modeling is the one this thesis is based upon and it is proposed by Ian Goodfellow et al. in 2014 [13]. Different kind of GANs emerged since 2014 based on different field of applications like deep convolutional GAN [28] that uses convolutional layers in its model for a generation of images and videos, conditional GANs [26] that gives a specific condition as input in addition to the noise, cyclic GANs [37] which works with an image to image translation by learning features from source variable and generating target variable, and recurrent GANs using Long Short Term Memory (LSTM) architecture [6] for generating sequence-based inputs.

When investigating more adversarial networks, the following research works were analyzed for time-series data generation. Research in [2] explores the use of sophisticated classifier models that differentiates between real and fake data. The paper investigates the traditional Recurrent Neural Network (RNN) architecture which produces deterministic data and provides a solution of using mixture density networks with a gaussian mixture model to generate realistic data with a good variance. However the papers fail to produce more stochastic realistic data due to the absence of an adversarial training feedback loop from the discriminator to the generator. A related article [15] proposes a method of using SeqGAN in a reinforcement learning setting and they are also trained with domain-specific objectives in mind together with the discriminator rewards. The rewards are reduced for less diverse and non-uniquely generated datasets thus avoiding the mode-collapse which is a very common issue in handling GANs. The problem of the discriminator being 'perfect' is also addressed by the use of Wasserstein-1 distance as a loss function. The experiments are run with different algorithms - seqGAN, naive reinforcement learning, object reinforced GAN, and object reinforced Wasserstein GAN comparing each with maximum likelihood estimation with different domain-specific objectives in place. Experimental works in [8] move away from using the GAN models and instead approaches the problem with the Dynamic Time Warping (DTW) technique. The data is generated by averaging a set of time series and use that average as new synthetic data. The averaging technique followed is the DTW barycentric averaging. By this method, the paper avoids the consequence of averaging all the time-series equally, and instead the averaging is made by assigning weighted averages to each time-series data and thus an infinite number of synthetic data can be generated from a relatively small number of available datasets.

2
Theory

This chapter provides an explanation of the technical content that is needed to apprehend the thesis work in detail.

Section 2.1 gives an description of the data that is being handled in this thesis and section 2.2 refreshes readers with the working of artificial neural networks.

2.1 Time-series Data

Time-series data consists of a sequence of data that is collected over a time interval, thus recording the changes happened over that time interval. Independent and Identically Distributed (IID) data which is used in most of the machine learning applications collect information over a single point of time and this is the main difference between the IID and the time-series data. The time-series data can handle data of multiple time instances such as milliseconds, seconds, hours, or even years. A time-series data can contain either numbers or other characters as well, since the data depends upon the source.

Time-series data can be categorized into multiple types based on different criteria. Firstly, they can be divided according to the pattern they follow over a period of time.

1. The trend
2. Seasonal time-series data
3. Irregular time-series data

Secondly, they can also be divided according to the number of features present in the data. The data can be a combination of many dependent or independent features collected at a certain point of time and accumulated at that instance of time. The types are

1. Univariate Data
2. Multivariate Data

2.1.1 The Trend

This type of data either shows a long-term upward movement or a long-term downward movement of signals without being impacted by seasonal changes or irregularities in the noise pattern. The data analyzing part for this trend data becomes

relatively easy. Figure 2.1 shows the sample image of a time series graph [16][5] which shows a non-linear downward trend graph without any repetitions seasonally and without any irregularities over the period of time.

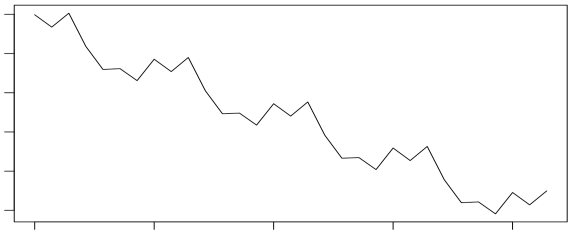

Figure 2.1: A sample non-linear downward trend time series graph [16][5]

2.1.2 Seasonal time-series data

A seasonal effect in the time series data is observed whenever the data inherits various seasonal factors like weekly, quarterly, or annual repetitions in its data. The seasonal variations can be due to natural forces like different seasons, or weather conditions or it can also be of a man-made convention like fashion or habits. Figure 2.2 shows a sample graph that is seasonally repetitive where a cyclic pattern is observed.

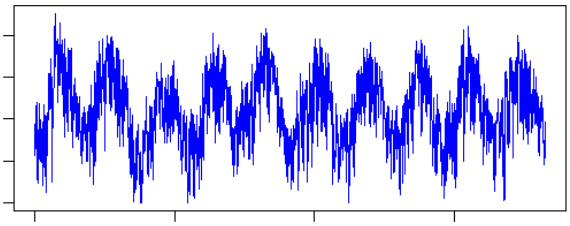

Figure 2.2: A sample seasonal time-series graph with cyclic variations [34]

2.1.3 Irregular time-series Data

This type of time-series graph as shown in figure 2.3 does not follow any pattern and shows a lot of noise in the data. The variables that are present in the data in this thesis usually are this type of irregular data and it, therefore, becomes difficult to understand their distribution and analyze them.

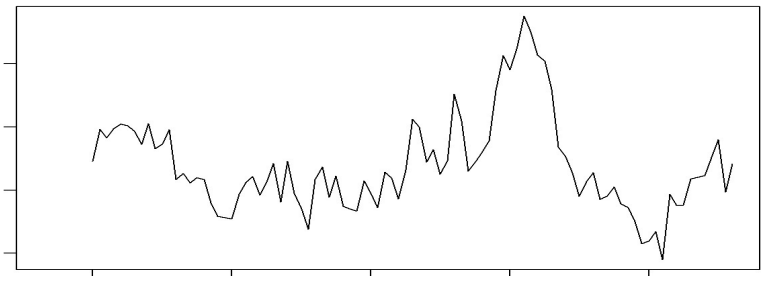

Figure 2.3: A sample irregular time-series graph without any definite patterns [23]

2.1.4 Univariate and multivariate Data

Time-series data is also divided into two types based on the number of features it contains. There can be more than one variable in the time-series data that can be received at each time instance and this type of data is called multivariate data. If only one variable is present in the data for each time instance, then it is called, univariate time-series data. The data that this deals with, is the multivariate data consisting of multiple columns of features.

2.2 Artificial Neural Networks

This section helps readers refresh the primary concept of what are artificial neural networks and how do they work. Neural networks are greatly inspired by the functioning of the neural system inside the animal's brain. The neural network consists of multiple neurons and edges connecting the neurons. These edges have weights which denotes the strength of the edges between two neurons. The learning happens by increasing and decreasing weights over the whole training process. More on artificial neural networks can be referred from this article [25] by Bernard Mehlig.

2.3 Feed Forward Neural Networks

Feed forward neural networks are the simplest form of neural networks in which a machine learning task can be made possible. Figure 2.4 represents a simple neural network model with one hidden layer and two output layers.

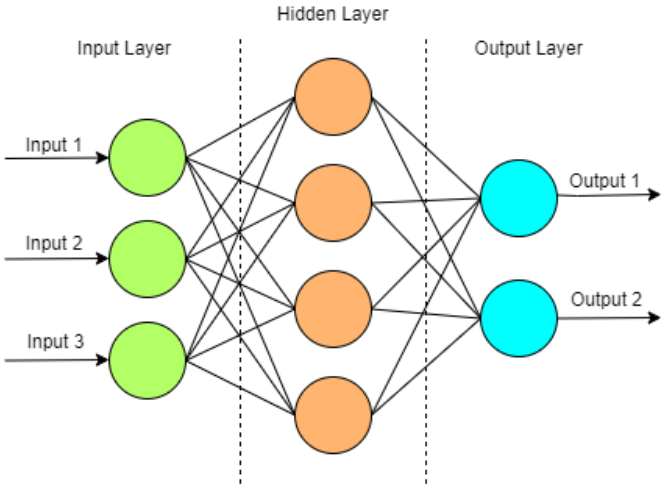

Figure 2.4: A simple feed forward neural network with one hidden Layer.

The objective of any feed forward neural network is to approximate the function f(x) and perform any machine learning tasks such as classification, regression or prediction. The training of a simple multi-layer perceptron with hidden layers involves forward propagation and backward propagation. Considering an n-dimensional input of $\vec{x} = \{x_1, \ldots, x_n\}$, and n- dimensional outputs of $\{y_1, \ldots, y_n\}$, a forward

propagation is iteratively performed to calculate each neurons in the next layer using equation $\boldsymbol{y}^{(l)} = \sigma\left(\boldsymbol{W}\boldsymbol{y}^{l-1} + \boldsymbol{b}\right)$, where $\boldsymbol{W}$ is the weights and $\boldsymbol{b}$ is the bias of the neurons present in the nodes of layer l. σ is the activation function used.

Cost function is calculated between the desired output y and the obtained output $\boldsymbol{y}_{(out)}$. With this cost function, the weights and the biases are updated during back propagation to minimize the loss function. Thus the model is trained and can be implemented in various machine learning applications.

2.4 Convolutional Neural Networks

The Convolutional Neural Networks (CNN) are a type of neural networks which are specialized in handling grid like structures like images. They are built up of multiple layers consisting of convolution, pooling and fully connected layers. There are different type of CNNs present. The most common type of CNN is 2 dimensional CNN which can handle images. But there are also 1 dimensional CNNs available, where the forward pass happens along single direction. Time-series data is a particular kind of data where the kernel slides in one direction.

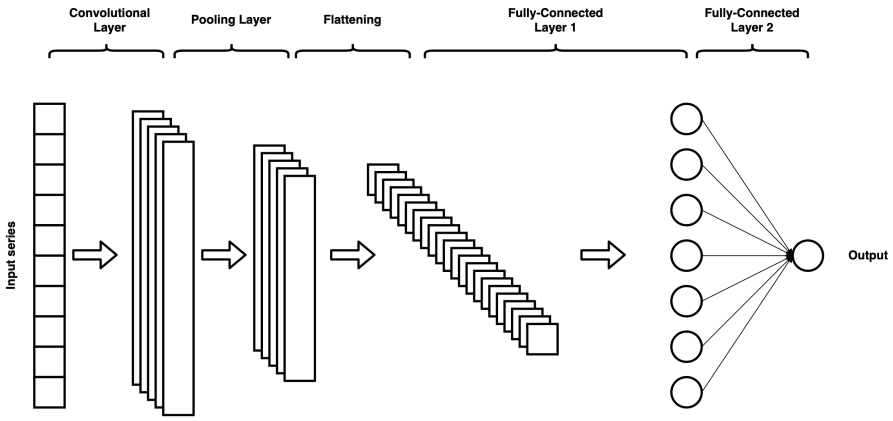

Figure 2.5: Simplified 1 dimensional CNN architecture [22]

Figure 2.5 explains the various layers present in the CNN architecture, where the input time-series is sent into the convolution layer that performs dot product between two matrices - one being the sliding kernel and the other matrix is the restricted receptive field portion. Then the output from the convolution layer is passed into the pooling layer, where the outputs at some positions are replaced by the summary statistics of one of the nearby outputs [27], thus effectively reducing the spatial size of the output. Later this output is flattened and sent to the multi-layer perceptron layer and the final output is derived out of that dense layer.

Yi Zheng et al, through the research paper [36] modified the traditional CNN and performed classification on multivariate time-series data. In the paper, multivariate time-series data is divided into univariate features and then perform feature

extraction on each of the univariate data. Later, all the extracted features from the univariate data are concatenated and sent it to dense layers for classification. However, because of the absence of correlations exhibited between different univariate features, this method cannot be followed completely in this thesis. Nevertheless, an initial GAN model based on CNN is tried out to examine the possible result.

2.5 Recurrent Neural Networks

Unlike the traditional feed forward neural networks which perform well on simple tabular data and data without time dependencies, the Recurrent Neural Network (RNN) is a type of artificial neural network that performs well on sequential data like time-series data, text data, and audio data. These RNNs are used in applications in Natural Language Processing (NLP), machine translation, weather forecasting, image captioning, and many more.

When compared with the simple neural networks, the output of the RNNs will allow previous outputs as its inputs while being in the hidden layer. Figure 2.6 shows the unfolded and folded architecture of a sample RNN [17]. This first half of the image explains the rolled version of RNN which shows the weight $\boldsymbol{W_{xh}}$ between the input x and the RNN Cell h. The weight between the output sequence y and the RNN cell h is $\boldsymbol{W_{hy}}$.

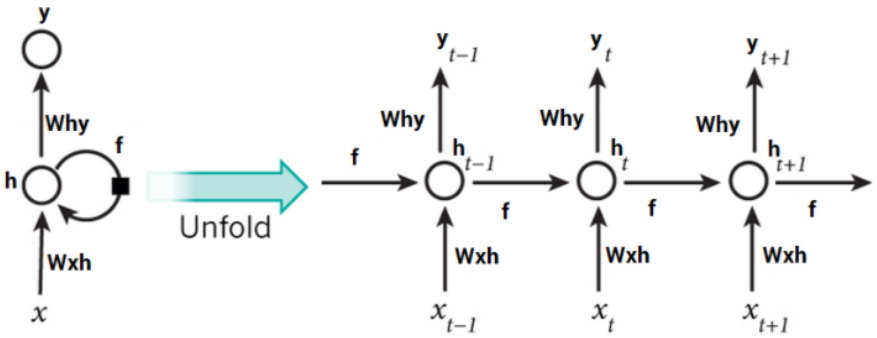

Figure 2.6: A folded and unfolded RNN architecture [10][9]

When the network is unfolded, a clear view of the sequential flow of data is viewed. It can be seen that during each time step t, the RNN cell gets the input x_t and return y_t which is an output term, and also h_t which is an RNN term that is sent as an input to the next neuron. Thus the sequential information is carried on to the subsequent neurons. Equation (2.1) and (2.2) shows how the h_t and the output term y_t are calculated.

$$h_t = g_1 \left(W_{hh} h_{t-1} + W_{xh} x_t + b_h \right) \tag{2.1}$$

$$y_t = g_2 \left(W_{hy} h_t + b_y \right) \tag{2.2}$$

where W_{xh}, W_{hh}, W_{hy}, b_h, b_y are temporarily shared coefficients, and g_1, g_2 are

activation functions.

One of the main characteristics why RNNs operate very well is, that the model architecture need not be increased with the increase in the length of sequence inputs. Additionally, they can also process input sizes of any lengths. Since the weights are shared across time, the same weights get updated at the end, and there is a fewer number of trainable parameters than anticipated. But the network contains a fair amount of disadvantages as well. As the network becomes complicated, the training becomes slow, and thus it becomes, inefficient in complicated tasks. There are many types of RNNs.

2.5.1 Many-to-Many RNN

Figure 2.7 shows the many-to-many recurrent neural architecture, where inputs are multiple sequences of information and output gives a fixed length of sequence output. There are two types of many-to-many RNNs, depending on the thickness of the input and output sequences.

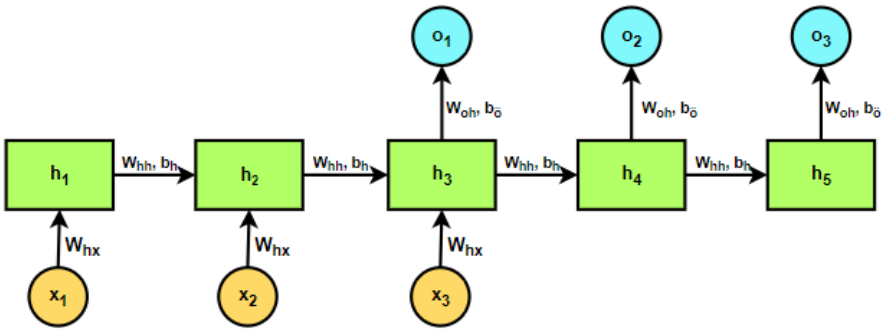

Figure 2.7: Many-to-many RNN architecture [33]

2.5.1.1 Tx = Ty

This type of neural networks have an equal number of input (Tx) and output (Ty) sequence lengths. This type of neural networks are used in CNN based GAN in the generator part, where the input noise has exactly the same dimensions as the output synthetic time-series data.

2.5.1.2 Tx != Ty

This type of neural networks have an unequal number of input(Tx) and output (Ty) sequence lengths. These many-to-many neural networks are used in the timeGAN model in this thesis because of the mapping between the feature and latent space. The generator and embedding function receive the input in feature space, and output in the latent space. The recovery function in other hand gets the input from latent space and output the data in feature space.

2.5.2 Many-to-one RNN

Figure 2.8 shows the many-to-many recurrent neural architecture [24], where inputs contain multiple feature dimensions but the output is just single dimensional. This type of neural networks are used in this thesis in the discriminator part of the CNN based GAN and timeGAN which gives the classifications as a binary value at the output.

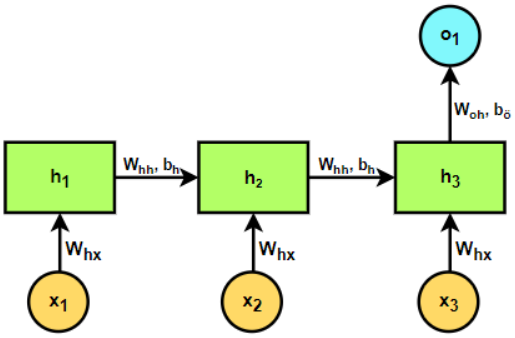

Figure 2.8: Many-to-one RNN architecture [33]

2.5.3 Long Short Term Memory

The traditional RNNs suffer from two key problems which are vanishing and exploding gradients. The RNNs generally have activation functions in each of its layers, and thus the gradients of the loss function of these RNNs approach zero during the course of training, and then the model no longer understands the long-term dependencies of the sequential data. This short-term memory problem is called the vanishing gradient issue, and the Long Short Term Memory (LSTM) is a special kind of RNN Network that is capable of handling this problem.

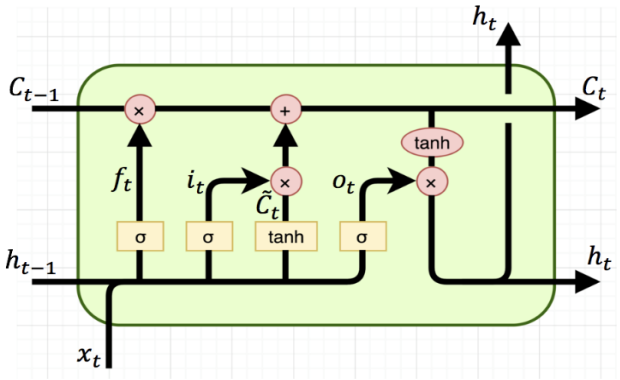

Figure 2.9: A detailed view of one LSTM repeating module [32]

The first research studies about LSTM took place in the year 1997 by Sepp Hochreiter et al., through his article neural computation [18]. An LSTM network has a different repeating module from a traditional RNN network which makes it possible

to remember the long-term dependencies of the data. A normal RNN has a simple repeating module architecture consisting of usually, a single activation layer while the LSTM has around four different neural network layers in its repeating module. These layers can also interact with each other. Figure 2.9 gives an overview of the type of layers present in the LSTM network [32], and figure 2.10 gives the notations for understanding the various components present in the repeating module.

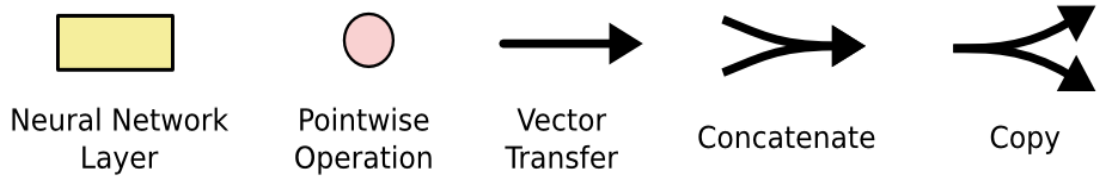

Figure 2.10: The notations for the LSTM module present in figure 2.9 [32]

The first layer from left in figure 2.9 is the forget layer which takes in $\boldsymbol{h_{t-1}}$ and $\boldsymbol{x_t}$ as input and gives a value between 0 and 1 as output implying the model to forget everything or keep everything respectively. The formula for this forget gate is given by equation (2.3), where W_f and b_f are weights and biases of the forget gate, and [.,.] is a concatenation.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.3}$$

The second and third layer from the left in figure 2.9 is used to update the current cell state C_t. The sigmoid layer which is called the 'input gate layer' will choose which values to update, and the *tanh* layer will create a vector containing new values and then these two processes are combined and the cell state is updated. The updating and the addition equations are given in (2.4) and (2.5) respectively.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{2.4}$$

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{2.5}$$

Here W_i and b_i are the weights and biases of the input function while W_C and b_C are the weights and biases of the update layer and $\tilde{C}_t$ is the new candidate vector. The three equations (2.3), (2.4), (2.5) interact using the point-wise multiplicative and addition operations thereby forgetting unnecessary information in the old state and adding the required information to remember in the current state thus creating an overall formula (2.6).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.6}$$

The next layer shown in figure 2.9 will be the output layer O_t where a sigmoid function is used on the new candidate layer $\tilde{C}_t$ to show which information the model is going to let out through output, and then later multiplied by cell state C_t put through *tanh* function as shown in equation (2.7) and (2.8).

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{2.7}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{2.8}$$

This LSTM Network is used in this thesis inside generator and discriminator Model which is briefly explained in the upcoming section 2.6.1

2.6 Generative Modeling

Generative modeling is a branch of deep learning techniques that can capture and learn the trends and patterns followed in the data, and starts generating similar data. Using the knowledge from the original distribution of the data, the generative models must be able to recreate new outputs which could replicate the properties of the original data as well as producing unique data each time thereby making the data diverse.

Generally speaking, there are two ways of generative modeling algorithm namely,

1. Variational Auto Encoders (VAE)
2. Generative Adversarial Networks (GAN)

Both VAE and GAN can be termed generative modeling methods because both models can be used to generate synthetic data. Generally, a model is called as a generative model when the latent input component has probability distribution connected with it.

VAE consists of two components called encoder and decoder which are used to produce synthetic data. The encoder gets the original data as input into the encoder and collapses that data into smaller dimensions and creates a latent representation out of it. Later this representation is passed through the decoder to reconstruct and decode the original data that is sent into the encoder. The training dynamics in the VAE are considered to be a bit weird because of the reason, it picks out the probability masses from the latent distribution sometimes where it does not makes sense. This could impact the performance of the VAE sometimes, and this is one of the reasons to choose GAN for the synthetic generation of time-series data.

2.6.1 Generative Adversarial Networks

The Generative Adversarial Networks (GAN) can be termed as unsupervised learning models that use supervised loss as a part of its training. This network contains two blocks in it. One is the generator which is trained to generate new samples, and the second is the discriminator that classifies the input sample as either real or fake.

In detail, the generator $G(.)$ in figure 2.11 is typically a kind of neural network that is based on a transformation function. The input for this generator is a random noise z with density taken from p_z and once the generator is trained, should output a random sample $x_g = G(z)$ that follows the distribution of the original sample.

The discriminator $D(.)$ as shown in figure 2.11 on the other hand is based upon a different neural architecture and it acts as a discriminative function. It gets the original sample (x_t with density p_t) or the generated sample (x_g with density p_g) as input and outputs the probability $D(x)$ of this sample to be a 'real' one.

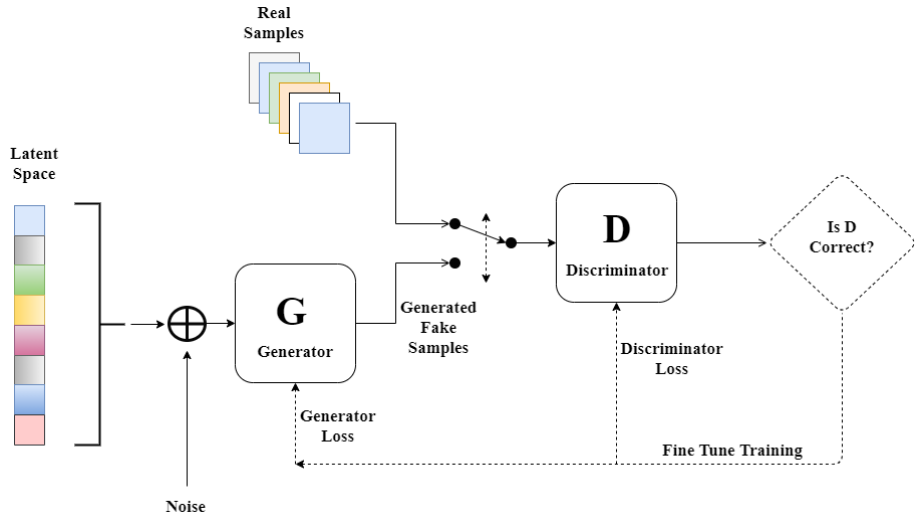

Figure 2.11: A simple architecture of GAN model [12]

2.6.1.1 Theoretical Loss Functions of GAN

Rocca has written an article [29] describing the theoretical loss functions present in GAN models. Consider that equal amounts of real and generated data are sent for classification into the discriminator, then the expected absolute error E caused by the discriminator is given by equation:

$$\begin{aligned} E(G,D) &= \frac{1}{2}\mathbb{E}_{x\sim p_t}[1-D(x)] + \frac{1}{2}\mathbb{E}_{z\sim p_z}[D(G(z))] \\ &= \frac{1}{2}\left(\mathbb{E}_{x\sim p_t}[1-D(x)] + \mathbb{E}_{x\sim p_g}[D(x)]\right) \end{aligned} \tag{2.9}$$

This absolute error must be minimized in case of the discriminator since the discriminator must classify between the samples as 'real' and 'fake' whereas the main goal of the generator is to cheat the discriminator into producing realistic looking samples, and thus the generator will try to maximize this error as shown in equation:

$$\max_G \left(\min_D E(G,D)\right) \tag{2.10}$$

The best discriminator is the network which can successfully minimize the absolute error for any generator of probability density p_g with equation:

$$\mathbb{E}_{x\sim p_t}[1-D(x)] + \mathbb{E}_{x\sim p_g}[D(x)] = \int_{\mathbb{R}} (1-D(x))p_t(x) + D(x)p_g(x)dx \tag{2.11}$$

To minimize this integral function, the function inside this integral can be minimized for every x value. It then gives the best discriminator that can be designed for any generator. Similarly, the best generator has to be defined for the given discriminator which tends to maximize the following integral equation:

$$\int_{\mathbb{R}} \left(1-D_G^*(x)\right) p_t(x) + D_G^*(x)p_g(x)dx = \int_{\mathbb{R}} \min\left(p_t(x), p_g(x)\right) dx \tag{2.12}$$

In this way, the generator tries to fool the discriminator. The generator network is trained in such a way that it looks to maximize the classification error, and the

discriminator tries to find out fake samples as much as possible and minimizes the classification error. Thus, the GAN can be explained as a min-max game between two neural networks. This adversarial setting between the network is the base for this thesis and the synthetic time series data are successfully generated using this concept.

2.6.2 TimeGAN

TimeGAN is a modified version of the normal GAN which is suitable for time-series data generation, and also consists of some extra blocks along with the traditional generator and the discriminator.

The timeGAN is proposed by Jinsung Yoon et al., through the research article [35], discusses about the model that can preserve the correlations and temporal dynamics of the original distribution when generating data. In this paper, the flexibility offered by the unsupervised model is combined with the control provided by the supervised learning, and so the network model will stick to the training data dynamics during sampling. The timeGAN architecture consists of two extra blocks namely Embedding and Recovery functions that give a supervised paradigm to the conventional adversarial setting of GAN.

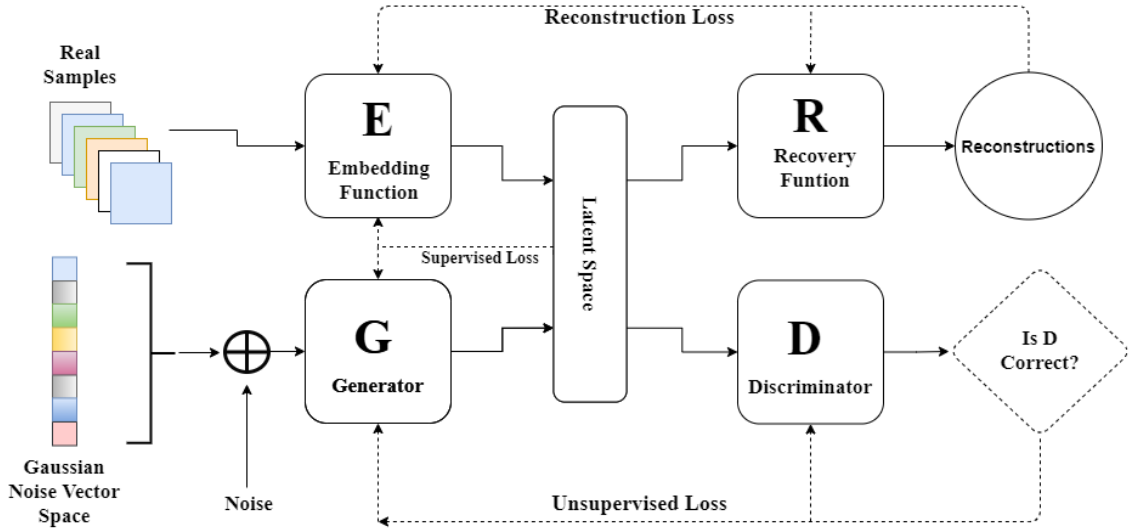

Figure 2.12: TimeGAN Network Block Diagram [35]

2.6.2.1 Embedding and Recovery Functions

The embedding function acts as an encoder and it converts the real data into its latent representation and then the reconstruction function, converts the latent representation present in the latent space back into reconstructed data. Through this conversion between latent and feature spaces, the embedding and the recovery functions allow the adversarial network to understand the characteristics of the original data.

The latent space is represented by $\mathcal{H}$ for the corresponding feature space $\mathcal{X}$, where the $\mathcal{X}$ denote the features of the original data. Equation $e : \prod_t \mathcal{X} \rightarrow \prod_t \mathcal{H}$ shows how features are converted into latent spaces and e represents the embedding function and it is implemented through an LSTM network. The $\mathbf{h}_t$ in equation (2.13) represents the latent code that is obtained from the feature space $\mathbf{x}_t$ and the previous input $\mathbf{h}_{t-1}$.

$$\mathbf{h}_t = e\left(\mathbf{h}_{t-1}, \mathbf{x}_t\right) \tag{2.13}$$

Similarly equation $r : \prod_t \mathcal{H} \rightarrow \prod_t \mathcal{X}$ brings the latent codes back into the corresponding feature representations, and the r here is the reconstruction function, which is implemented through an LSTM model. Equation (2.14) shows the conversion from latent codes into features through reconstructions.

$$\tilde{\mathbf{x}}_t = r\left(\mathbf{h}_t\right) \tag{2.14}$$

2.6.2.2 Generator and Discriminator

The generated output from the generator goes into the latent space rather than producing the output directly into the feature space. This can be shown from $g : \prod_t \mathcal{Z} \rightarrow \prod_t \mathcal{H}$, where $\mathcal{Z}$ is defined in a Gaussian vector space, from which the random input is drawn from. In the equation (2.15), $\mathbf{z}_t$ denotes the random noise vector, while the $\hat{\mathbf{h}}_t$ refers to the generated output in the latent space.

$$\hat{\mathbf{h}}_t = g\left(\hat{\mathbf{h}}_{t-1}, \mathbf{z}_t\right) \tag{2.15}$$

Similarly, the discriminator also picks up its input from the latent space, and outputs a classification of whether the data is real or fake. The equation is given by $d : \prod_t \mathcal{H} \rightarrow [0,1] \times \prod_t [0,1]$, where the input is latent representation that produces classification $\tilde{y}_{1:T} = d\left(\mathbf{h}_{1:T}\right)$ where $\tilde{\mathbf{h}}_{1:T}$ represents the latent representation from either the embedding function or the generator, and $\tilde{y}_{1:T}$ represents the classifications for real or synthetic data.

3
Methodology

In this chapter, various synthetic data generation methodologies are discussed. Also, the steps followed during the project period is shown in figure 3.1, starting from the literature pre-study to the evaluation of the final results.

Figure 3.1: The overall work flow of the thesis Project

3.1 Literature Review

A detailed literature study is done before starting the thesis, on possible solutions to accomplish the project. Many online resources such as Scopus, Google scholar, and Chalmers open repository are utilized during the literature study. Various research articles and their codes are studied in-depth, and their use-cases and limitations are considered while developing this project. Data visualization techniques for multivariate time-series data, benchmark model development, enhanced model development, qualitative and quantitative evaluation methods are finalized during this period. The literature pre-study made is shown in section 1.3.

3.2 Data set

	1	2	3	4	5	6	7	8
1	0.490196	0.443137	0.490196	0.490196	1.0	1.0	0.333333	0.400000
2	0.490196	0.443137	0.490196	0.490196	1.0	1.0	0.333333	0.407843
3	0.494118	0.443137	0.494118	0.494118	1.0	1.0	0.333333	0.415686
4	0.494118	0.439216	0.494118	0.494118	1.0	1.0	0.333333	0.415686

Figure 3.2: A sample dataframe of the time-series data used in this thesis

The air compressor fills different pressure systems present in the vehicle, whenever needed in a certain priority, and are monitored and controlled by the Air Pressure System (APS). This APS system receives sensor readings indicating the pressure maintained in different circuits in each time step. And the APS controls the air compressor status based on the threshold values of the pressure systems. The sensor readings received are in the form of a Control Area Network (CAN) message, and these readings consist of time-dependent multivariate data. Figure 3.2 shows the sample dataframe used in this project. The values (sensor readings) in figure 3.2 are not the original values but are the normalised values. The features are named differently so that the sensitivity of the data is preserved.

3.3 Data Visualisation

Data visualisation is a crucial step before proceeding with the actual machine learning problem. Visualising the data gives insights into the distribution of different features present in the data. Data can be visualised in many ways based upon the type of data available, and these representations can be a map, a graph or a plot. Long term trends can be analysed, and the outliers in the data can easily be identified using the data visualisation techniques.

3.3.1 Graphs using Plotly

Different features in the data are plotted as pulse graphs using a library called plotly in python. The pulse graphs differentiate various features using unique colours and plot each feature across the time in x *axis*. The y *axis* denotes the distribution range of each attribute. Users can select a particular feature in the data and only concentrate on one specific feature for analysis in the real-time graphical dashboard.

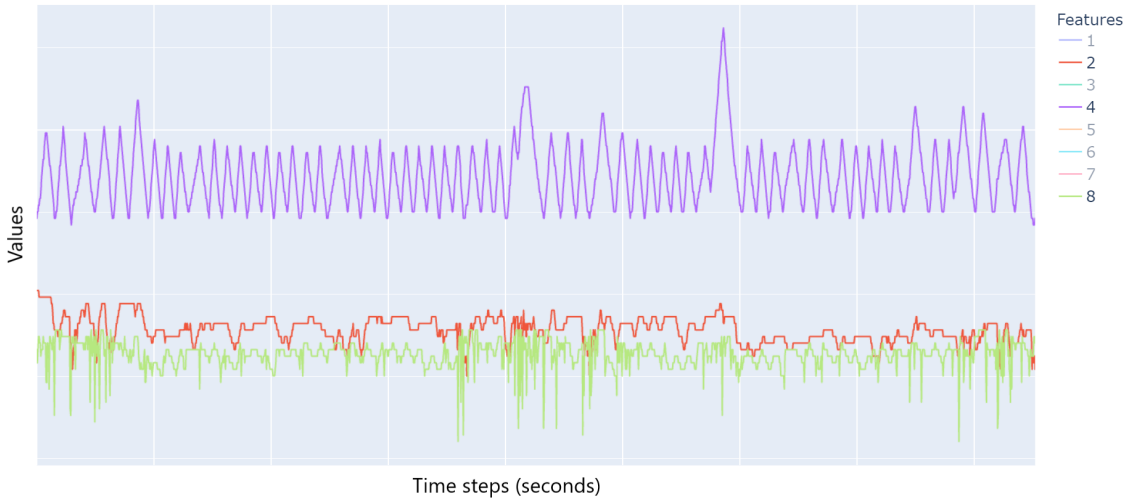

Figure 3.3: Unnormalized time-series data features plotted using plotly

Figure 3.3 shows how different time-series data features can be plotted and visually analysed using the legend.

3.3.2 3D Plot using Principal Component Analysis

One way of finding the correlation between different features is by performing Principal Component Analysis (PCA) on the available data, and then later present the data in a 2-dimensional or 3-dimensional Scatter plot. This type of visualisation is done to analyse the prominent features present in the data, and later compare the distributions of original and synthetic data. The PCA in this thesis is executed for the sole purpose of visual evaluation only and not for later use in the machine learning task.

Since the data contains eight features, it becomes difficult to view the data in 8-dimensional space. To simplify this problem, the dimensions are reduced, and brought down to 3 so that the data points in each time frame can be viewed in a 3D Plot. Figure 3.4 shows the PCA 3-dimensional plot,that shows the 3 most prominent features of healthy and faulty data. The separation of these two groups of data are assumed to be a characteristic executed by feature 7 in the training data where there are frequent fluctuations in faulty data and less fluctuations in healthy data.

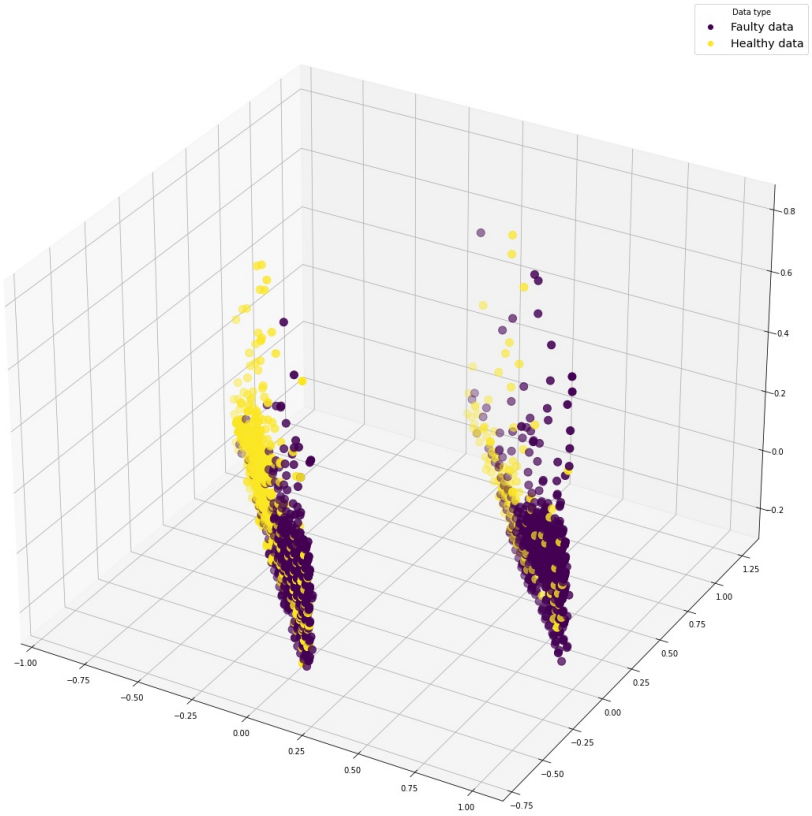

Figure 3.4: 3-dimensional graph plotted using PCA for training data.

While performing dimensionality reduction, it is possible to lose some information. To assess the percentage of original data maintained, a metric called cumulative explained variance ratio is calculated. The number of principal components in this

thesis is three (since reducing the data to 3-dimensions). Firstly, the ratio of the variance exhibited by each principal component to the total variance is calculated. Later, the ratio of each principal component is added cumulatively, and thus the metric is computed.

The cumulative explained variance varies between 0 and 1, with zero being no information is preserved and one meaning all the information is maintained. When the PCA is performed on the training data for visual insights, the cumulative explained variance ratio is 0.988, which means 98.8 percentage of the original data is preserved during dimensionality reduction.

3.4 Data Pre-processing

Data pre-processing is an important step in the machine learning process. Data pre-processing is a combination of multiple tasks that are performed to clean the data, remove missing values from the data, encoding and modifying the data according to the use case. These processes are performed because the raw data obtained from the sensors usually are inconsistent, incomplete, and might have some errors.

3.4.1 Data Loading

There are many data files available in Scania repository with different driving times which can be used as the training data. For example, one data file contains time-series data worth 30 minutes, while another data file contains 20 minutes of driving data. All these data are input as dataframes, and later, some columns are removed as mentioned in section 3.4.2, and then appended in the form of dataframes inside a list. This list contains multiple time-series data sets in the form of dataframes.

3.4.2 Column Dropping

The data sets available for this project are in the Comma Separated Value (CSV) format, with the first column of the data-set being the 'serial number' and the second column being the 'Time' in seconds for each data point. These two columns are dropped using the 'drop' function because these features are inappropriate for the time-series data generation process and they affect the quick convergence during the training.

3.4.3 Conditional Shifting

Conditional shifting is a method used to replace a data point if it is wrong or if the data point is missing. Each feature in the dataframe is bound to a specific distribution range, and the whole data should not contain any 'NaN' values. To remove such inappropriate and missing values, each data point (each row) of the feature is checked. If a wrong data point is found, it is replaced with the next closest appropriate value in that particular feature.

3.4.4 Normalization

Normalization is the process of organizing the data set to make it look similar across all the fields. Normalization process usually improve the quality of the data. In order to bring such characteristics into the training data, the normalization is done across each feature in the data set. The mathematical formula for the normalization is given by the equation (3.1).

$$X_{\text{Normalized}} = (X - X_{\text{min}}) / (X_{\text{max}} - X_{\text{min}}) \tag{3.1}$$

Here the X_{min} is the data point with the least value and X_{max} is the data point with the highest value of each particular feature across the whole dataset. Figure 3.5 shows the data after normalization process, where all the features of the data lie in range of (0,1).

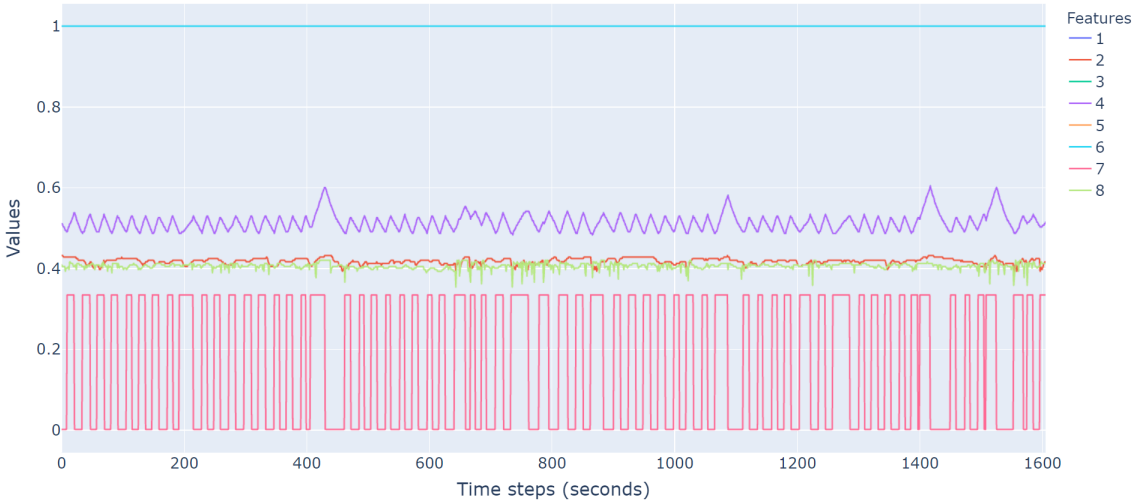

Figure 3.5: Normalized training time-series data plotted using plotly.

3.5 Standard GAN model

Now, as the data is loaded and pre-processed, the next step is to create a standard benchmark model to generate the time-series data. The standard GAN architecture followed is explained in section 2.6.1. The initial GAN model is tried out with CNN based GAN network where the generator and the discriminator part consists of convolutional layers and some fully connected layers in it.

3.5.1 CNN based GAN

Convolutional 1-dimensional layers are used in the CNN based GANs, in both the generator and the discriminators [4]. Section 3.5.1.1 and 3.5.1.2 depicts in detail about the CNN architecture for the generator and the discriminator blocks. The pictorial representation of the networks is given in the appendix section A.1.

3.5.1.1 Generator architecture

The generator of the CNN based GAN contains multiple layers. The model consists of input layer to begin with. Later, the the input goes into series of convolutional layers with leaky relu as its activation function. Zero padding is followed on the first convolutional layer, and a padding value of two is followed on other convolutional layers. Later the output from the convolutional layers are flattened and sent into fully connected dense layer of node size = $s \times n$, where s is the output sequence length of the generated sequence required by the end user, and n is the number of features present in the data. The input size of the generator is a random Gaussian noise vector with a lesser dimension than that of the generated data.

3.5.1.2 Discriminator architecture

The discriminator receives two types of inputs into its CNN architecture. The first input will be the generated data from the generator produced from random Gaussian noise vector, and the second input is the original normalized training data. After going through the convolutional layers, these inputs sequentially passes through dense layers, in which the final dense layer consists of only a single node with sigmoid activation function. This layer performs a classification task and gives out a validation score whether the data received is real or generated data. This loops back to update the weights and biases of generator and discriminator.

3.5.1.3 Experiments

CNN based GAN training parameters	
Optimizer	Adam
Learning rate	0.0005
Number of epochs	15000
LeakyRelu parameter α	0.2
Conv1D layer hidden units	64
kernel size	1 and 5

Table 3.1: Table showing various settings used in the CNN based GAN model

Different settings are tried out in the CNN based GAN with dimension size 32, 64, and 128 in the convolutional layers, kernel sizes of 1, 5, and 10, number of convolution layers, and also different padding sizes. The RMSProp optimizer is tested along with the Adam optimizer. Different learning rates - 0.001, 0.005, 0.0001 and 0.0005 are tested and different activation functions between each convolution layers are also tested. The generated time-series data for all these settings are visually compared with the original data. The results are restricted to visual comparisons and further evaluations are not made for the CNN based GAN since the CNN layers lack correlation among different features. The best results are obtained for the settings shown in table 3.1. The results for those settings are shown in section 4.1.

3.5.2 CNN based GAN algorithm

The GAN model is capable of generating realistic and diverse synthetic time-series data by training both generator and discriminator in a sequential order for epochs e and update their gradients. The whole process of the working of the GAN is explained in the algorithm 1 [3].

Algorithm 1: CNN based GAN algorithm

Input: γ = learning rate, e = epochs, n = number of features present in the data. mb = batch size, n_{gen} = the number of iterations of generator per discriminator iteration.
Initialize: θ_d= initial discriminator parameters, θ_g= initial generator parameters.
for t= *0,....,e* **do**
    The sequence lengths of data changes during each epoch in random.
    Sample $\left(\mathbf{z}_{1,1:T_n}\right), \ldots, \left(\mathbf{z}_{n_{mb},1:T_{n_{mb}}}\right) \overset{\text{i.i.d.}}{\sim} p_z$
    Sample $\left(\mathbf{x}_{1,1:T_n}\right), \ldots, \left(\mathbf{x}_{n_{mb},1:T_{n_{mb}}}\right) \overset{\text{i.i.d.}}{\sim} \mathcal{D}$
    Train and Update the gradients of the discriminator by ascending its gradient descent.
    $\nabla_{\theta_d} = \frac{1}{mb} \sum_{i=1}^{mb} \left[\log d\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - d\left(g\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$
    $\theta_d = \theta_d - \gamma \nabla_{\theta_d}$
    for y = *0,....,*n_{gen} **do**
        Train and Update the generator by descending its gradient.
        $\nabla_{\theta_g} = \frac{1}{mb} \sum_{i=1}^{mb} \log\left(1 - d\left(g\left(\boldsymbol{z}^{(i)}\right)\right)\right)$;
        $\theta_g = \theta_g - \gamma \nabla_{\theta_g}$
    end
end

In this algorithms, it can be seen that the generator and the discriminator are not trained simultaneously, but they are instead trained sequentially, and that the generator is trained twice more than the discriminator. The generator is trained more than discriminator to compensate for the number of training batches trained by the discriminator. Since, the discriminator trains with the equal number of fake data and real data, but generator trains only with the random noise, the generator is trained two times more than discriminator. Algorithm 1 explains the flow of the standard GAN where many parameters are declared initially and the hyper parameter settings tried are tuned constantly to achieve better results.

3.5.3 CNN based GAN network training

Let us take a look at the outer most for loop in algorithm 1. This loop denotes the total number of epochs for which the GAN is trained. For the GAN to have the diversity in the generation of new data, and also to learn the complete characteristic distribution of the training data, each epoch is run on training data of different sequence lengths. For example, on first epoch, the model is trained on time-series

data of length 1 minute, and in next epoch, the model is trained on five minutes time-series data, and so on. However, because of the fixed architectural design of CNN, the dimensions in CNN layers could not be varied for each epoch. So, the training data was split into same sequence lengths for all the epochs and the GAN is trained. The number of data samples extracted from the training data sets during each epoch for training are 60.

The discriminator is trained on both original data as well as the synthetic data generated by the generator. After the discriminator's gradients are updated, the generator is trained to produce realistic sequences and fake the discriminator. During the training of generator, only the generator's gradients are updated and the discriminator's learnable parameters remain unchanged.

3.6 Enhanced GAN model

After the results of CNN based GAN model is analysed and visually evaluated, a more enhanced complex GAN model is developed. This model leveraged the advantages of auto-regressive models and adversarial networks to build a new model called timeGAN [35]. The four blocks that form the timeGAN are explained in sections 2.6.2.1 and 2.6.2.2. The timeGAN pseudo code is explained by the algorithm 2.

This enhanced GAN model is inspired from the research article on timeGAN[35], but differs from data pre-processing, random sliding window method, and also in hyperparameter settings. There are also changes in the architecture of different blocks in the model, and also in the evaluation techniques to assess the quality of generated data. The algorithm for this enhanced GAN model is inspired from `https://www.vanderschaar-lab.com/papers/NIPS2019_TGAN_Supplementary.pdf`

3.6.1 Architecture of the timeGAN network

The architectures of different blocks of the timeGAN almost resemble the same. Except the input dimensions and the output dimensions, the number of layers present in the embedding function, recovery function, generator and discriminator are same. The blocks are made up of three LSTM layers with return sequences being True. The three LSTM layers are followed by a Dense layer with sigmoid activation function.

The dimensions of the dense layer differs with respect to the blocks. The generator and embedding function has the dimensions in dense layer equal to the number of features in the latent space, while the dense layer of the recovery function has dimensions equal to number of features in feature space. The discriminator dense layer dimension is just 1, indicating that the block acts as a classification layer. The detailed representation of each block are depicted in the appendix section A.2.

3.6.2 TimeGAN algorithm

Algorithm 2: TimeGAN Algorithm []

Input: $\lambda = 1$, $\eta = 10$, mb = batch size, γ = learning rate, e = epochs, $\mathcal{D}$
Initialize: θ_e, θ_r, θ_g, θ_d
for t= *0,....,e* **do**
  (1) Map between feature space and latent space
  Sample $\left(\mathbf{x}_{1,1:T_n}\right), \ldots, \left(\mathbf{x}_{n_{mb},1:T_{n_{mb}}}\right) \overset{\text{i.i.d.}}{\sim} \mathcal{D}$
  for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
    $\mathbf{h}_{n,t} = e\left(\mathbf{h}_{n,t-1}, \mathbf{x}_{n,t}\right)$
    $\tilde{\mathbf{x}}_{n,t} = r\left(\mathbf{h}_{n,t}\right)$
  end
  (2) Generate Synthetic Latent Codes
  Sample $\left(\mathbf{z}_{1,1:T_n}\right), \ldots, \left(\mathbf{z}_{n_{mb},1:T_{n_{mb}}}\right) \overset{\text{i.i.d.}}{\sim} p_z$
  for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
    $\hat{\mathbf{h}}_{n,t} = g\left(\hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t}\right)$
  end
  (3) Distinguish between Real and Synthetic Codes
  for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
    $y_{n,t} = d\left(\mathbf{h}_{n,t}\right)$
    $\hat{y}_{n,t} = d\left(\hat{\mathbf{h}}_{n,t}\right)$
  end
  (4) Compute Reconstruction $\left(\hat{\mathcal{L}}_R\right)$, Unsupervised $\left(\hat{\mathcal{L}}_U\right)$, and Supervised $\left(\hat{\mathcal{L}}_S\right)$ Losses
  $\hat{\mathcal{L}}_R = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\sum_t \left\|\mathbf{x}_{n,t} - \tilde{\mathbf{x}}_{n,t}\right\|_2\right]$
  $\hat{\mathcal{L}}_U = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\sum_t \log y_{n,t}\right] + \sum_t \log\left(1 - \hat{y}_{n,t}\right)$
  $\hat{\mathcal{L}}_S = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\sum_t \left\|\mathbf{h}_{n,t} - g\left(\mathbf{h}_{n,t-1}, \mathbf{z}_{n,t}\right)\right\|_2\right]$
  (5) Update $\theta_e, \theta_r, \theta_g, \theta_d$ via Stochastic Gradient Descent (SGD)
  $\theta_e = \theta_e - \gamma \nabla_{\theta_e} - \left[\lambda \hat{\mathcal{L}}_S + \hat{\mathcal{L}}_R\right]$
  $\theta_r = \theta_r - \gamma \nabla_{\theta_r} - \left[\lambda \hat{\mathcal{L}}_S + \hat{\mathcal{L}}_R\right]$
  $\theta_g = \theta_g - \gamma \nabla_{\theta_g} - \left[\eta \hat{\mathcal{L}}_S + \hat{\mathcal{L}}_U\right]$
  $\theta_d = \theta_d + \gamma \nabla_{\theta_d} - \hat{\mathcal{L}}_U$
end
(6) Synthetic Data Generation *(6-1) Sample* $\left(\mathbf{z}_{1,1:T_n}\right), \ldots, \left(\mathbf{z}_{n_{mb},1:T_{n_{mb}}}\right) \overset{\text{i.i.d.}}{\sim} p_{\mathcal{Z}}$
(6-2) Generate artificial latent codes
for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
  $\hat{\mathbf{h}}_{n,t} = g\left(\hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t}\right)$
end
(6-3) Converting to the feature space
for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
  $\tilde{\mathbf{x}}_{n,t} = r\left(\mathbf{h}_{n,t}\right)$
end
Output: $\hat{\mathcal{D}} = \left\{\hat{\mathbf{x}}_{1:T_n}\right\}_{n=1}^{N}$

In this algorithm λ is a hyperparameter which balances the losses generated by the embedding and the recovery functions, while η is a hyperparameter that balances the losses produced by generator and the discriminator, $\mathcal{D}$ is the training data, p_z is the random vector distribution and $\hat{\mathcal{D}}$ is the generated data. θ_e, θ_r, θ_g and θ_d are initial parameters for embedding function, recovery function, generator and discriminator respectively.

3.6.3 Randomized Sliding Window Technique

Listing 3.1: Python Code for extracting the data samples on each epoch

```
#One Sequence length is chosen before each epoch
seq_length = 30

temp_data=[]
for j in data:
    #20 starting points are randomly generated for each of
       the three data files
    r= [random.randrange(1, len(j)-seq_length, 1) for i in
       range(20)]

    for k in r:
        _x = j[k:k + seq_length]
        temp_data.append(_x)
train_data = np.array(temp_data)
```

The python script 3.1 explains how the training data is split and sent into different mini-batches containing number of samples for training. The random sliding window technique uses a specified sequence length during each epoch. With this input, the starting point of each data sequence is chosen in random from a long time-series training data file and a sequence of specified length is cut down and stored as a single training sample. Similarly 60 data samples are extracted randomly from the three-training data files available.

3.6.4 TimeGAN training

The joint training on how to encode, generate and decode is shown in this section [35]. The embedding and recovery functions should enable complete reversible mapping between latent space and feature space, and convert the latent representations $h_{1:T}$ into precise representations $\tilde{\mathbf{x}}_{1:T}$ in feature space. The reconstructed loss $\mathcal{L}_{\mathrm{R}}$ is calculated by equation (3.2).

$$\mathcal{L}_{\mathrm{R}} = \mathbb{E}_{\mathbf{x}_{1:T}\sim p}\left[\sum_t \left\|\mathbf{x}_t - \tilde{\mathbf{x}}_t\right\|_2\right] \tag{3.2}$$

During training, the generator produces output for two different types of inputs. In open-loop mode, the generator behaves in an auto-regressive fashion by using its synthetic embeddings $\hat{h}_{1:t-1}$ produced during the previous time steps, and generates next vector $\hat{h}_t$. This adversarial behaviour where the generator looks to maximise and the discriminator looks to minimize the likelihood of precise classifications $\hat{y}_t$ for both the outputs $h_{1:t}$ from the embedding function, and $\hat{h}_{1:t}$ from the generator. This loss is called unsupervised loss and it is shown by equation (3.3).

$$\mathcal{L}_{\mathrm{U}} = \mathbb{E}_{\mathbf{x}_{1:T}\sim p}\left[\sum_t \log y_t\right] + \mathbb{E}_{\mathbf{x}_{1:T}\sim \hat{p}}\left[\sum_t \log\left(1-\hat{y}_t\right)\right] \tag{3.3}$$

To capture the distributions of the original data additionally to just the discriminator's adversarial feedback , another loss called supervised loss is introduced. The loss is computed by training the generator in a closed-loop system, where the generator calculates the next step input by comparing the latent representations of original data received by the generator from the embedding function $p\left(\mathbf{H}_t \mid \mathbf{H}_{1:t-1}\right)$ and the output produced by the generator $\hat{p}\left(\mathbf{H}_t \mid \mathbf{H}_{1:t-1}\right)$ during its previous step input. Then maximum likelihood is obtained from this comparison and supervised loss is computed by equation (3.4)

$$\mathcal{L}_{\mathrm{S}} = \mathbb{E}_{\mathbf{x}_{1:T}\sim p}\left[\sum_t \left\|\mathbf{h}_t - g\left(\mathbf{h}_{t-1}, \mathbf{z}_t\right)\right\|_2\right] \tag{3.4}$$

Therefore , in total, at any time step in training process, the discrepancy between the synthetic next step latent vector produced by the generator and the actual next step latent vector produced by the embedding functions are evaluated, and thus the losses are computed thereby giving an advantage of adversarial learning with auto-regressive learning. The unsupervised loss $\mathcal{L}_{\mathrm{U}}$ focuses more on the generator to produce realistic time-series sequences, while the supervised loss $\mathcal{L}_{\mathrm{S}}$ focuses that it gives similar step-wise transformations.

To improve the variances during the generation of synthetic time-series data, an additional loss factor is added to the generator which is known as moment loss (g_m). The moment loss calculates the difference in the mean and variances of the original and the generated data, and this difference is added to the generator loss during training.

3.6.5 Experiments

The experiments are tried for different LSTM dimensions, number of LSTM layers, activation layers, learning rates, number of epochs, ans sequence lengths of training data each epoch. The settings set during the generation of realistic synthetic time-series data are shown in table 3.2. The generated time-series data and the quantitative results are displayed in section 4.2.

TimeGAN training parameters	
Optimizer	Adam
Learning rate	0.0005
Number of epochs	30000
Number of nodes in all blocks	32
Activation function in LSTM layers	Tanh
Activation function in Dense layers	Sigmoid
Loss function	Binary Cross Entropy Loss
λ	1
η	10

Table 3.2: Table showing various settings used in the timeGAN model

3.7 Evaluation methods

There are various evaluation methods available to assess the quality of the synthetic time-series data. The generated data must be compared with original time-series data to check whether the machine learning model has learnt the data distributions of the original data correctly. These evaluation techniques should also compute the percentage of deviation of synthetic data from the real data. This section explains various metrics that are used in this thesis to evaluate the performance of the standard GAN and enhanced GAN models.

3.7.1 Visual evaluation

Visual evaluation is one of the easiest and the most convenient evaluation techniques used for the evaluation of time-series data.

Plotly: The original and the generated data are plotted in the time-series graph, and the distribution range of the two data are compared. The graphs are usually plotted using plotly which is 2-dimensional time-series graph where each feature can be separately viewed and compared.

PCA Evaluation: The original and the synthetic data undergo dimensionality reduction, and are plotted in 3-d space. When plotted, the original and synthetic data are differentiated by unique colouring scheme, and thus it is easy to compare the generated results. Also PCA is performed without dimensionality reduction, and features are separated in pairs. These paired features are plotted in 2-d space with different colour schemes for synthetic and original data.

3.7.2 Quantitative evaluation

Even though visual evaluation is an easy way of evaluating the synthetic data, a validity score is required to exactly measure the closeness of synthetic data distribution to the real data distribution. Such a validity score can be obtained by various quantitative methods.

3.7.2.1 Regression score

An external ad-hoc RNN sequence prediction model is used here to evaluate the generated time-series data. This sequence prediction model predicts time-series data one-step ahead, by giving the previous n steps as input, where n varies according to the user's input. This regression model is first trained on real data set and tested on real data set. This type of training and testing on real data set is called TRTR [7].

TRTS Method: The TRTR method does not directly test the goodness of synthetic data but it acts as a comparison metric for the model trained on real data and tested on synthetic data (TRTS). TRTS technique is usually used to evaluate the capability of the model to generate realistic data.

TSTR Method: Train on synthetic and test on real (TSTR) on the other hand is a similar evaluation technique like TRTS, but instead the roles of the original and generated data are reversed in training and testing. The TSTR method is used to check if there is a diversity in the synthetic data and if the artificially generated data covers corner cases on the original data distribution. So here, the sequence prediction model is developed and it is trained on synthetic data and tested on the real data. Unlike the TRTS technique, the TSTR can suffer from the mode-collapse problem. If the trained GAN model is suffering from mode-collapse issue, then the generated synthetic data might not capture the diversity or the original data's distribution. So, it is important to check whether the GAN model suffers from mode-collapse, before proceeding with the TSTR method.

In conclusion, TRTS and TSTR methods are compared with TRTR based on metrics like Mean Absolute Error (MAE) and Mean Square Log Error (MSLE).

MAE: MAE is the measure of average magnitude of error between the actual prediction and the obtained prediction. MAE is generally used to measure the accuracy in continuous data. MAE calculates the absolute differences, and so the MAE score always lies between 0 and ∞ [19]. The formula for MAE is shown in equation (3.5).

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{3.5}$$

MSLE: MSLE is similar to the traditional Root Mean Squared Error (RMSE) with the main difference in taking logarithms for both actual and predicted values. MSLE evaluates the model better than RMSE in case of presence of outliers and it also does not explode suddenly for small changes in prediction values like RMSE [30]. the formula for MSLE is shown in equation (3.6).

$$\sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(\log\left(y_j+1\right)-\log\left(\hat{y}_j+1\right)\right)^2} \tag{3.6}$$

In the equations (3.5) and (3.6), y_j is the actual value, $\hat{y}_j$ is the predicted value, and n is the total number of prediction. Both MAE and MSLE are negatively inclined scores, meaning lower the better.

3.7.2.2 Mahalanobis score

An external Gaussian Mixture Model (GMM) developed for anomaly detection [1] is used in this thesis to evaluate the original and the synthetic data. At first, the data is pre-processed using Gaussian filter and mean filter in a moving window fashion with time periods of 60 and 120 seconds. Then, the model is built and trained with these pre-processed data where each data set is considered as a cluster. Then, the mahalanobis score [31] that gives the distance between two different distributions is calculated. This distance is calculated for all the data sets with one data set as a reference cluster.

A graph is plotted with this mahalanobis score for all the clusters present, and a horizontal line is drawn between the clusters to separate them if they have different distributions. The TRTR, TSTR, and TRTS techniques are used here to calculate the mahalanobis score for original and fake data.

TRTR Method: During this method, the original data containing normal and faulty data are used for training the GMM model. These data are then plotted as clusters with one of the normal data as the reference cluster. This way, a horizontal line is plotted separating the original normal and faulty data. This method is not used directly for evaluation but instead used for comparing the scores from TSTR and TRTS methods.

TRTS Method: During this TRTS method, the already trained GMM model in TRTR method is used for testing the generated fault data. The synthetic fault data is plotted as a cluster in the graph along with the original data and see whether the synthetic data falls in the same distribution as the original fault data. Also the mean and standard deviation of the synthetic data generated for different sequence lengths is compared with the original fault data using a scatter plot.

TSTR Method: During the TSTR method, the generated data by timeGAN model is used for training the GMM model. And one of the fake data is used as a reference cluster. Later the trained GMM model is tested with original normal data to see if the distribution of the normal data differs from the fault data and a graph is plotted with the original normal data and synthetic fault data.

4

Results

This section consists of results and the corresponding discussions in the same order as described in chapter 3.

4.1 CNN based GAN

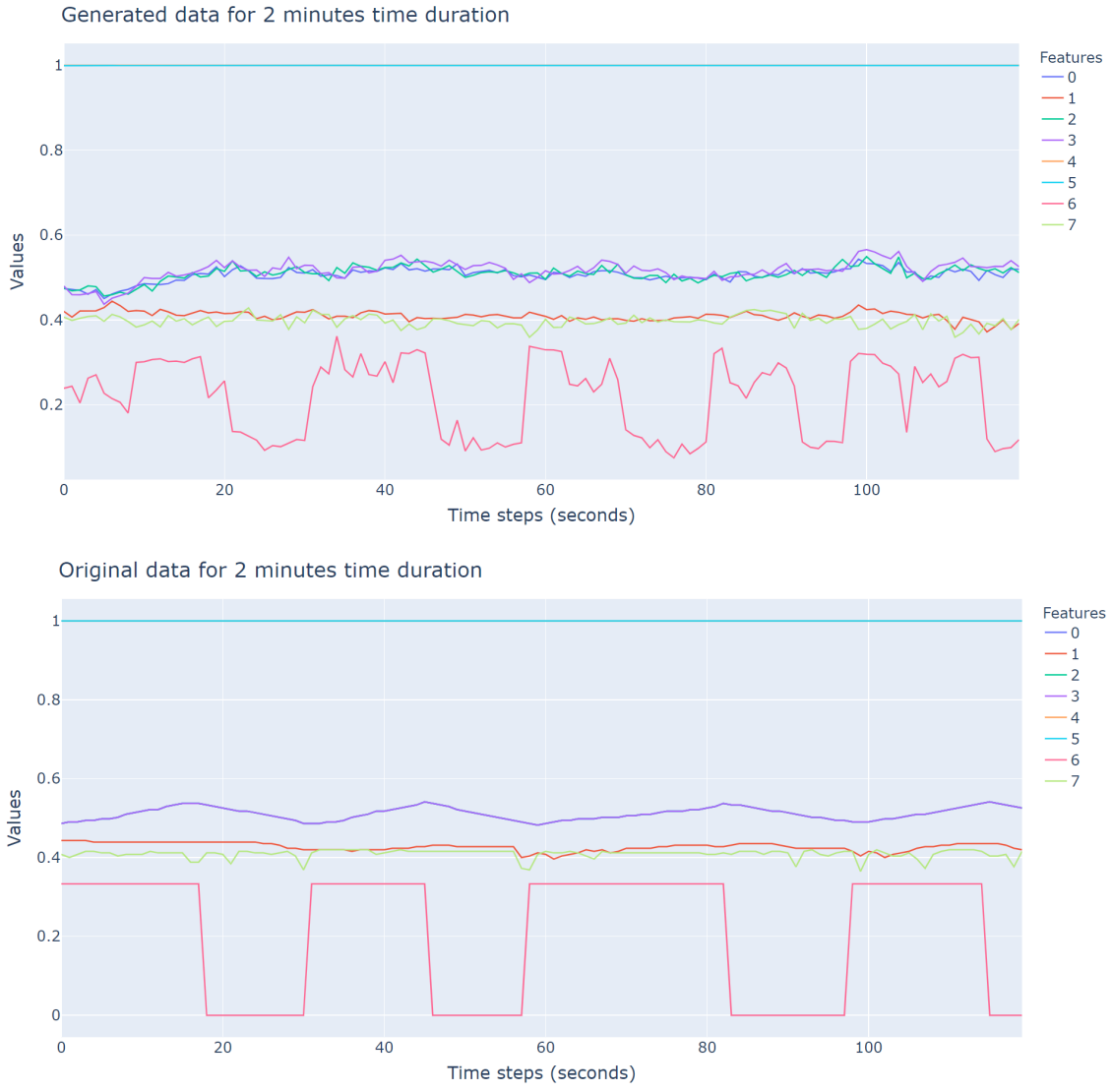

Figure 4.1: Original and generated time-series data for 2 minutes time duration

The GAN designed initially to generate synthetic data, contains CNN architectures in both generator and discriminator. The synthetic and the original time-series data are shown in figure 4.1. Figure 4.1 implies that the GAN based on CNN was able to learn the dynamics of the original data, but the result generated is very noisy and contains many fluctuations. Each feature of the generated data is independent, and so, the GAN did not capture the dependency between the features.

A notable drawback in using this GAN is the inability to generate synthetic data of variable sequence lengths. For example, if the model is trained on 5 minutes of original data, then the GAN can produce synthetic data of only 5 minutes. This is due to the fact, that the CNN dimensions are fixed during the network initialization, and they cannot be changed according to the variable sequence lengths during each epoch. Another downside, is that this model was unable to produce realistic looking data when using random sliding window technique, and because of this. When the random sliding window technique is not utilized in the GAN, then the model is unable to learn the complete distribution of the training data and thus produce poor results.

4.2 TimeGAN

The timeGAN model overcomes the disadvantages of traditional GAN model based on CNNs and learns the temporal dynamics and characteristics of the original data. The original data is normalized and then sent into the timeGAN for training. The timeGAN model combines the benefits of supervised learning objectives with unsupervised learning models to obtain better accuracy. The timeGAN model in this thesis is trained and evaluated with two settings.

- timeGAN trained with original data split into 30 second sequence samples.
- timeGAN trained with original data split into 60 second sequence samples.

4.2.1 Pre-training Loss

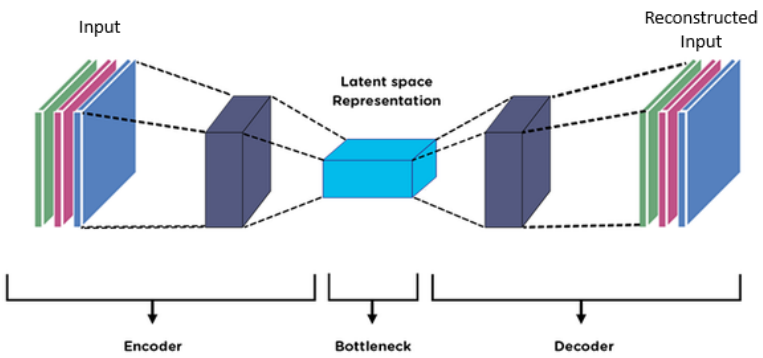

Figure 4.2: Auto-encoder network diagram

Figure 4.2 shows how the embedding and recovery function perform during the pre-training mode as an auto-encoder. As mentioned in section 2.6.2, the auto-encoder part (Embedding and recovery functions) is pre-trained with the original normalized data. The pre-training was made for 200 epochs and the embedding and recovery function has already trained well, mapping the time-series data between latent space and feature space. The pre-training results are almost the same for both the settings, and thus the loss graph for only one settings is shown in figure 4.3.

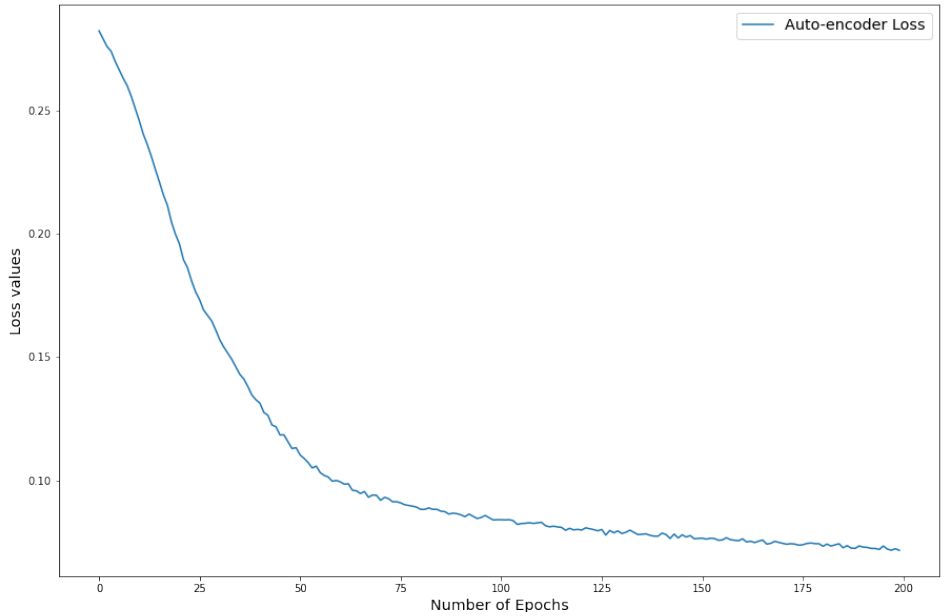

Figure 4.3: Auto-encoder pre-training loss

Figure 4.3 shows how the auto-encoder loss decreases from 0.30 to almost 0 during pre-training of the timeGAN model. This shows that the model learns the supervised objectives during the pre-training process of timeGAN well. After the pre-training, the model undergoes joint training which is described in section 3.6.4. The parameters with which the model is trained is given in table 3.2.

4.2.2 Joint training losses

The complete joint training process is explained in section 3.6.4. Figure 4.4 consists of the generator and discriminator losses exhibited by models trained in two different settings. Both sub figures in the figure 4.4 indicates that the generator and discriminator loss reduces exponentially over 30000 epochs. Initially, the losses were fluctuating till 5000 epochs, but later the model starts learning the original data characteristics, and then converges. When looking at the sub figure (a), the model converges when it reaches 30000 epochs. But in sub figure(b), it can be seen that the generator and discriminator losses still did not converge completely, and the generator is still in the process of learning the characteristics of original data.

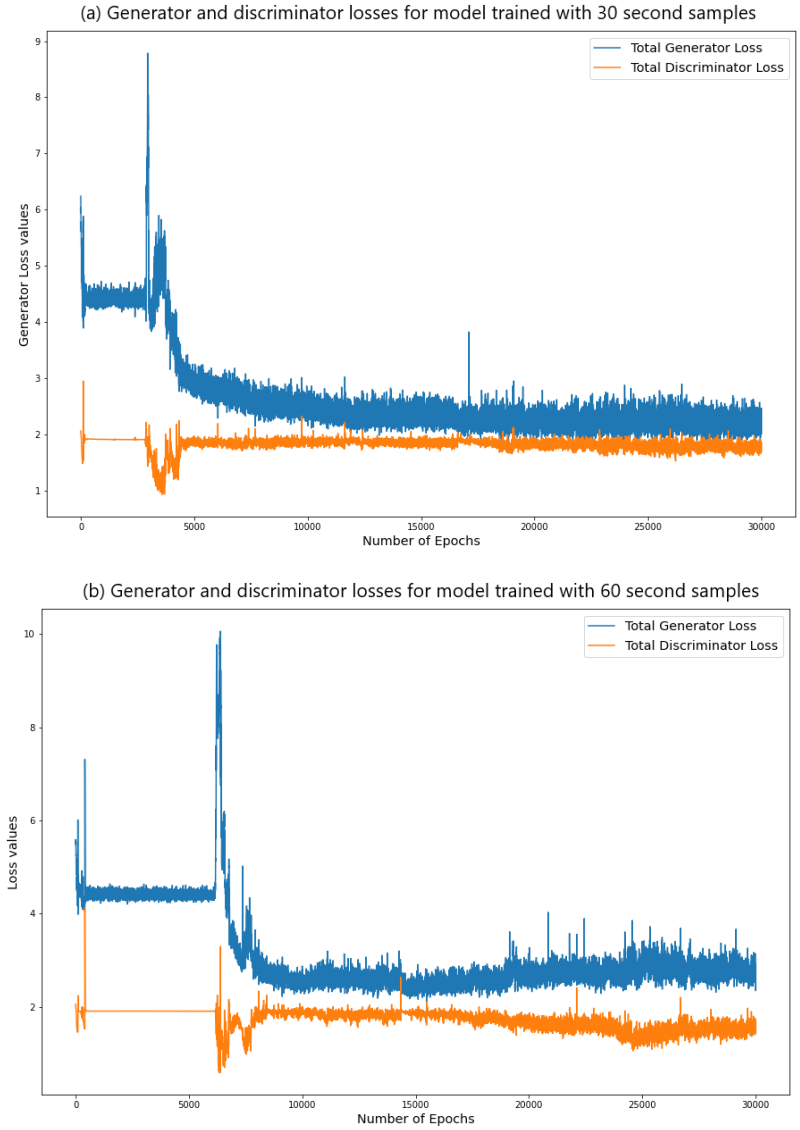

Figure 4.4: Graph showing generator and discriminator losses for 2 settings

Figure 4.5 shows the different generator losses the model contains, and also the sum of all the generator losses for the model during two settings. The blue line in the figure shows the total generator loss, and the operands that are subjected to the summation are generator moment loss, generator supervised and unsupervised loss. The generator unsupervised loss is obtained as the result of adversarial training between the generator and the discriminator. The generator supervised loss helps generator to learn from the embeddings of the original data through embedding function. The generator moment loss is obtained through computing the mean and variance between the generated synthetic data and the original data. As it is shown in figure 4.5, the generator learns constantly to produce more realistic data and tries fake the discriminator during classification.

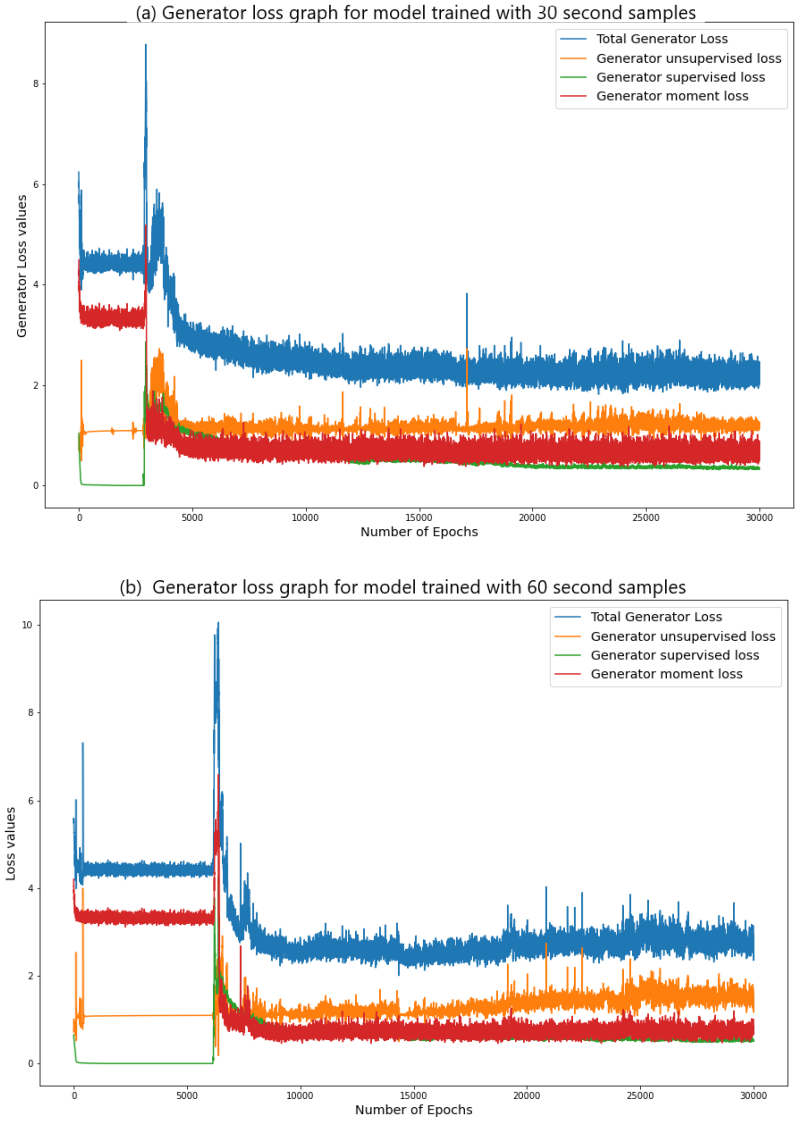

Figure 4.5: Graph showing generator losses for 2 settings

4.2.3 Visual results

4.2.3.1 Plotly graphs showing synthetic and original time-series data

The original and the generated data shown in the upcoming figures 4.6 and 4.7 are normalized to preserve the sensitivity of the data. From the figures 4.6 and 4.7, it can also be observed that each of the generated time-series data is unique and diverse, and produces realistic looking data. The detailed representation of each of the original and generated data's features for both the settings are shown in appendix A.3.

Setting 1: timeGAN trained with data split into 30 second samples
Figure 4.6 shows the original and the generated synthetic time-series data for the time period of 600 seconds (10 minutes). This data is obtained as a result of the timeGAN model trained with original data sequence length of 30 seconds.The legend

shows the different features present in the data.

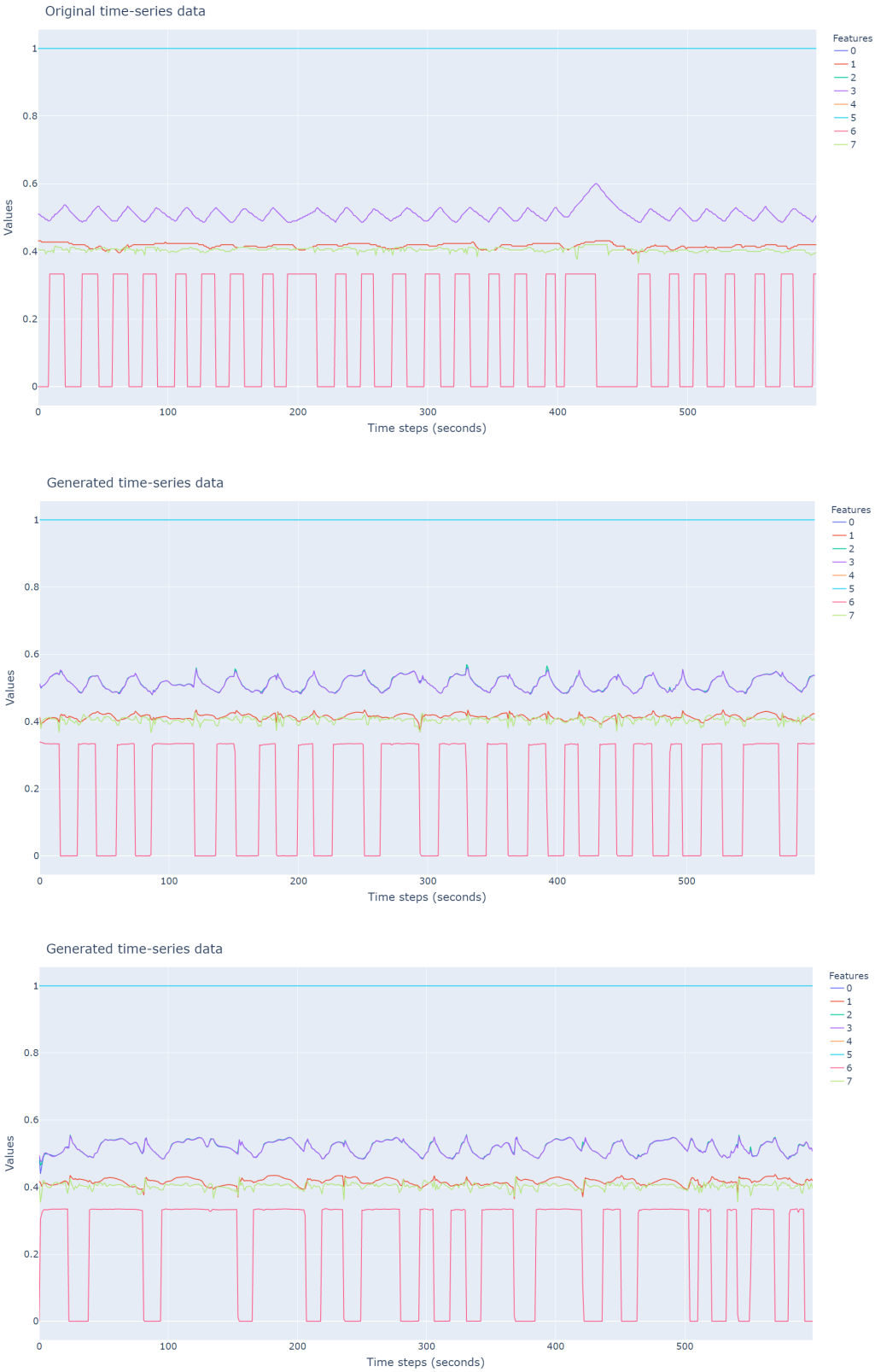

Figure 4.6: Original and synthetic data trained with 30 second sequence length

Setting 2: timeGAN trained with data split into 60 second samples

Figure 4.7: Original and synthetic data trained with 60 second sequence length

Similarly, figure 4.7 represents the original data and the generated synthetic data trained using 60 seconds sequence lengths (sequence lengths indicate each sample in a batch of 60 samples) of original time-series data.

4.2.3.2 PCA plot showing distributions of original and generated data

The PCA is performed for the synthetic and original time-series data. Both original and synthetic data has same principal components as a result of PCA. The dimensions are not reduced during the PCA evaluation. The features are plotted in pairs, and the features 1 to 8 are ordered in terms of their importance. The PCA graphs are plotted for 2 settings, exactly like the previous section.

Setting 1: timeGAN trained with data split into 30 second samples

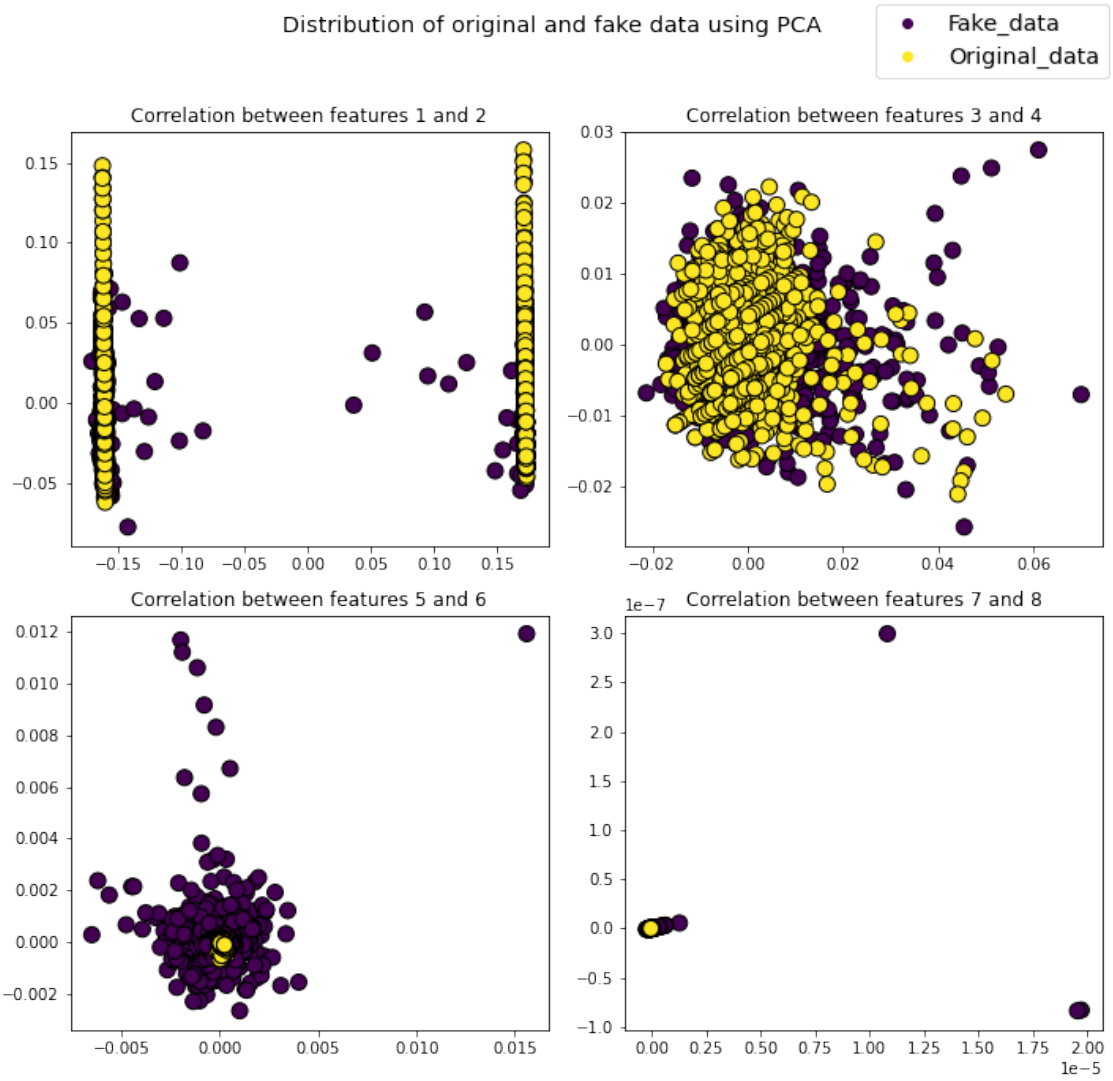

Figure 4.8: Distribution of original and fake data features for the model trained with (30 samples)

Figure 4.8 shows the distribution of the original and generated data with their features plotted in pairs. This PCA plot is plotted for the model trained with original data of 30 seconds sequence length. The subplot in the top left corner explains, that the most of the generated fake data follows the distribution of the original data, while the second subplot in the top right also shows that the fake data is exactly

distributed in the same fashion as the original data. The graph between feature 5 and 6 shows that the fake data is scattered more than the original data, but the change is very small in the order of 0.01 which can be seen in the x and y axis of the graph.

Setting 2: timeGAN trained with data split into 60 second samples

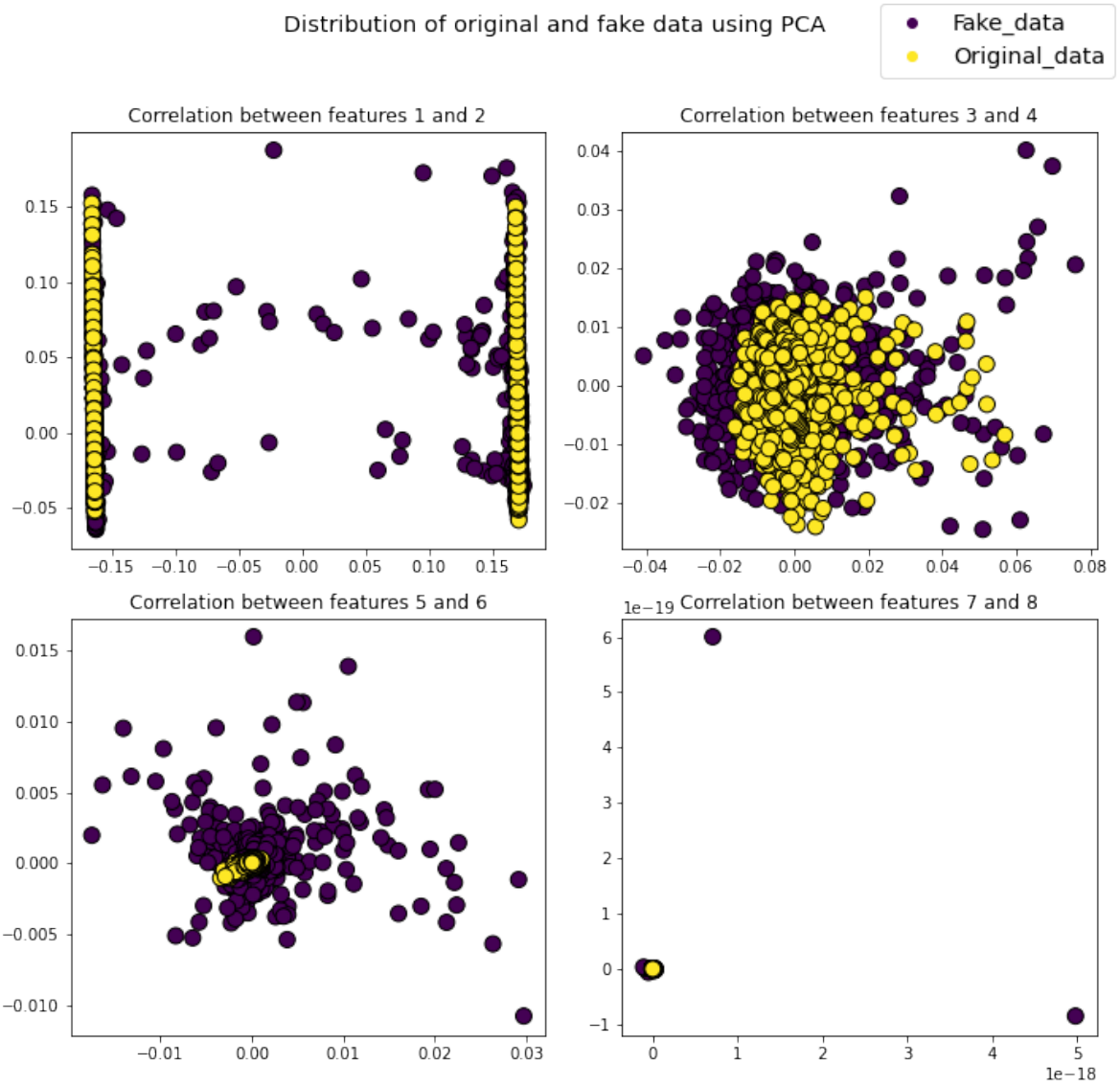

Figure 4.9: Distribution of original and fake data features for the model trained with (60 samples)

Figure 4.9 shows the PCA performed for the timeGAN model trained with 60 second sequence samples. This correlation graph between different features based on their order of importance convey many useful information. The graph between features 1 and 2, is more scattered here than the same graph for model trained with 30 seconds sequence lengths. This shows that the model needs to be trained for more epochs. But an important thing to note from the correlation between feature 1 and 2 is that

the fake data here captures the distribution better than its counterpart in setting 1. Same is the case for other graphs and their counterparts as well. This shows that if the model is trained for more epochs, better results can be obtained.

Conclusion from the visual evaluation: The timeGAN is trained for many different settings to identify the best performing model, and the model particularly performed well on two settings with which the visual evaluation is carried on so far. As it can be seen from the PCA evaluation, and Plotly representation, both the settings performed equally well, and it is hard for the viewer to easily recognize the correct settings. To give a clearer understanding on this, next section gives the quantitative evaluation for the model trained with the two settings.

4.2.4 Quantitative Evaluation of timeGAN

4.2.4.1 Regression score for forecasting using RNN

An external RNN model to forecast and predict the next step of the sequence given the previous sequence is developed as explained in section 3.7.2.1. The network is tested for two metrics - MAE and MSLE with three techniques. The testing done for two settings are shown in next section.

Setting 1: timeGAN trained with data split into 30 second samples

The scores for different evaluation methods		
Type	**MAE**	**MSLE**
TRTR	0.004111	0.000516
TSTR	0.007655	0.000596
TRTS	0.005961	0.000721

Table 4.1: Table showing regression scores for setting 1

Percentage of difference between TRTR vs TRTS and TSTR		
Type	**MAE**	**MSLE**
TRTR vs TSTR	86.21 %	15.5 %
TRTR vs TRTS	45 %	39.73 %

Table 4.2: Table showing percentage difference for different evaluation techniques

Table 4.1 shows the evaluation scores for MAE, and MSLE for three techniques which are trained on tested using the timeGAN model on setting 1. The error values of TRTS and TSTR techniques are relatively close to the reference TRTR value in case of both the MAE and MSLE. However table 4.2 shows a clear percentage of

difference in scores between TRTR vs TRTS, and TRTR vs TSTR.

Setting 2: timeGAN trained with data split into 60 second samples

The scores for different evaluation methods		
Type	**MAE**	**MSLE**
TRTR	0.006526	0.000789
TSTR	0.006732	0.000834
TRTS	0.008306	0.000848

Table 4.3: Table showing regression scores for setting 2

Percentage of difference between TRTR vs TRTS and TSTR		
Type	**MAE**	**MSLE**
TRTR vs TSTR	3.16 %	5.7 %
TRTR vs TRTS	27.28 %	7.48 %

Table 4.4: Table showing percentage difference for different evaluation techniques

Table 4.3 shows the evaluation scores for MAE, and MSLE for three techniques which are trained on tested using the timeGAN model on setting 2. The error values of TRTS and TSTR techniques are relatively close to the reference TRTR value in case of both the MAE and MSLE. However table 4.2 shows a clear percentage of difference in scores between TRTR vs TRTS, and TRTR vs TSTR.

Conclusion for regression scores: When comparing the tables 4.2 and 4.4, it can be clearly seen that the percentage difference is lesser in setting 2 than the setting 1 for both MAE ans MSLE values. This quantitative score thus proves that the timeGAN model trained with original data split into 60 second samples generate more realistic looking and diverse time-series data close to the original distribution.

4.2.4.2 Mahalanobis score for anomaly detection using GMM

The mahalanobis score is calculated for TRTR, TRTS, and TSTR techniques as described in section 3.7.2.2. This score is calculated for both the settings, and are explained in the upcoming sections.

Setting 1: timeGAN trained with data split into 30 second samples

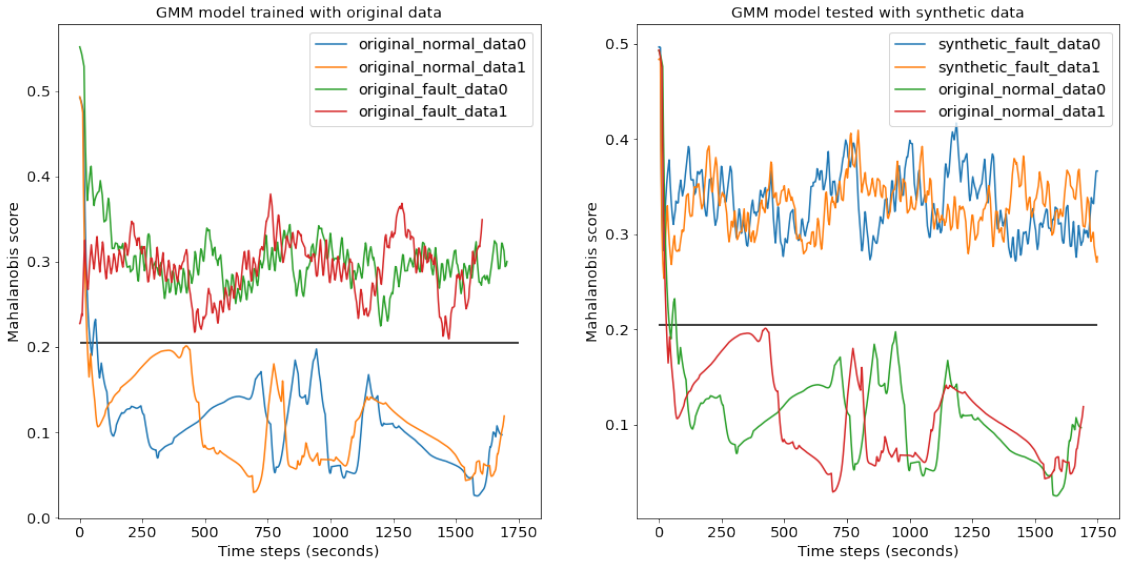

Figure 4.10: Two graphs showing TRTS technique for GMM model

TRTS: Figure 4.10 shows two graphs, where in the left graph, the model trained with original normal and fault data are plotted as clusters and a horizontal line is drawn at the mahalanobis score of 0.201 to separate the normal and fault data. Whereas in the right graph, when tested the same GMM model with synthetic fault data, the same horizontal line at 0.205 score is able to separate the synthetic fault data and the original normal data. This shows that the timeGAN model has produced realistic fault data similar to the original fault data distribution.

Boxplot: For further inference, figure 4.11 shows the box plot containing many synthetic fault data and the original fault data. Over 100 synthetic data sequences are generated on different sequence lengths, and the box plot is plotted. The mean mahalanobis score of the synthetic fault data is around 0.3254, while the mean score for the original synthetic data was 0.2975 which shows a difference of around 0.0279.

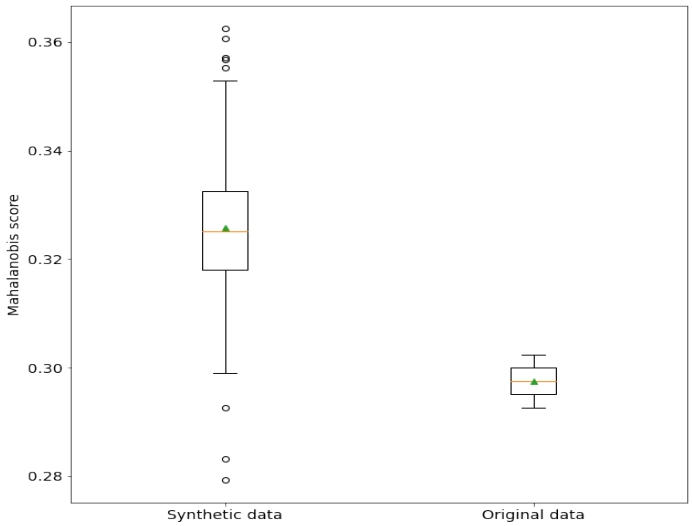

Figure 4.11: Box pot showing mahalanobis scores of original and synthetic data

TSTR: Figure 4.12 shows two graphs where, the GMM model is trained with different synthetic data and and a horizontal line is plotted in reference to one of the synthetic cluster to show that any distribution above that horizontal line is a different distribution. In the right figure, it can be seen that original normal data and original fault are tested with this GMM model and they clearly are differentiated using the same horizontal line from the left figure of score 0.14. From this it can be concluded that the synthetic fault data lies in the distribution curve of the original fault data.

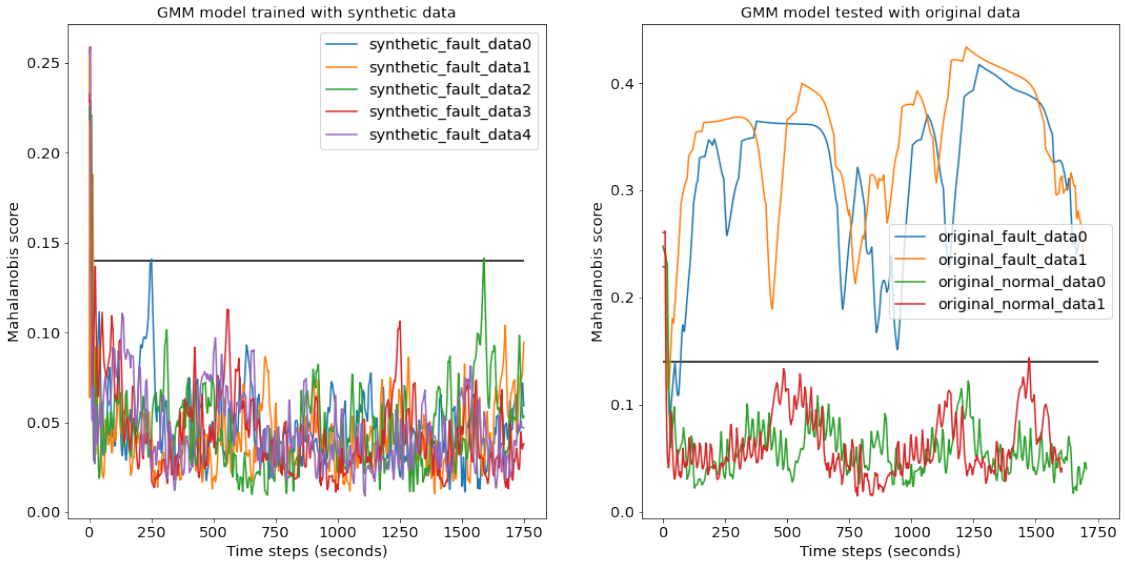

Figure 4.12: Two graphs showing TSTR technique for GMM model

Setting 2: timeGAN trained with data split into 60 second samples

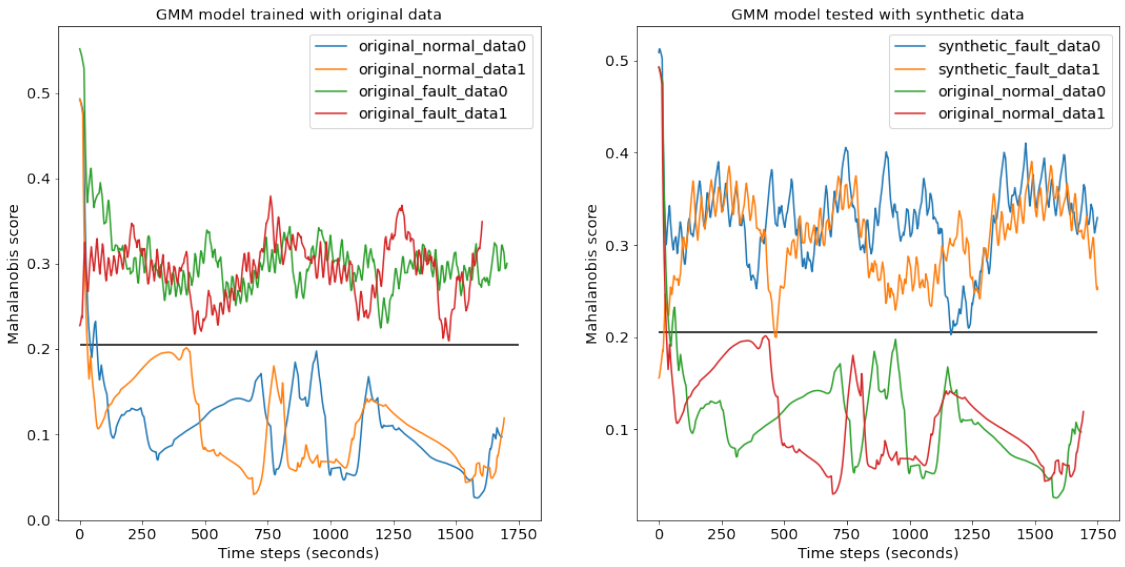

Figure 4.13: Two graphs showing TRTS technique for GMM model

TRTS: Figure 4.13 shows two graphs plotted for TRTS method. In the left graph, the model trained with original normal and fault data are plotted as clusters and a horizontal line is drawn at the mahalanobis score of 0.201 to separate the normal and fault data. Whereas in the right graph, when tested the same GMM model with synthetic fault data, the same horizontal line at 0.205 score is able to separate the synthetic fault data and the original normal data. When comparing the second subplot in figure 4.13 with the second subplot in figure 4.10, it can be seen clearly that the distribution of the synthetic fault data in figure 4.13 resembles more closely to the distribution of the original fault data.

Boxplot: For further inference, figure 4.14 shows the box plot containing many synthetic fault data and the original fault data. Over 100 synthetic data sequences are generated on different sequence lengths, and the box plot is plotted. The mean mahalanobis score of the synthetic fault data is around 0.3092, while the mean score for the original synthetic data was 0.2975 which shows a difference of around 0.0117. This difference is less when compared to the difference present in the setting 1, which shows that the timeGAN model trained with setting 2 produces more diverse and realistic looking data than setting 1.

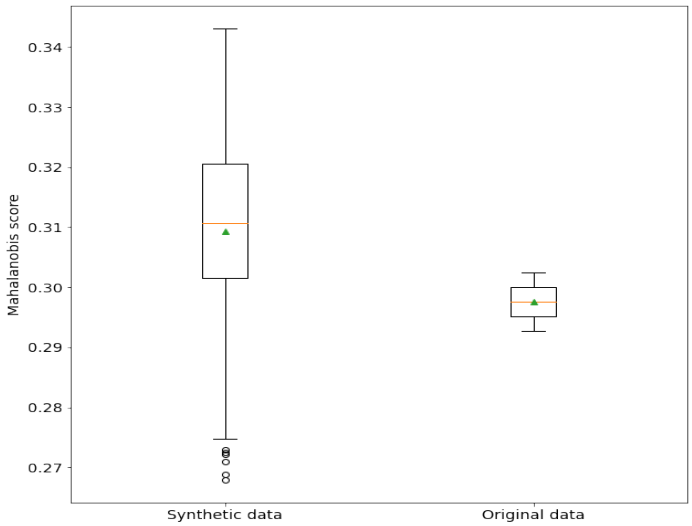

Figure 4.14: Box pot showing mahalanobis scores of original and synthetic data

TSTR: Figure 4.15 shows two graphs where, the GMM model is trained with different synthetic data and and a horizontal line is plotted in reference to one of the synthetic cluster to show that any distribution above that horizontal line is a different distribution. In the right figure, it can be seen that original normal data and original fault are tested with this GMM model and they clearly are differentiated using the same horizontal line from the left figure of score 0.14. From this it can be concluded that the synthetic fault data lies in the distribution curve of the original fault data.

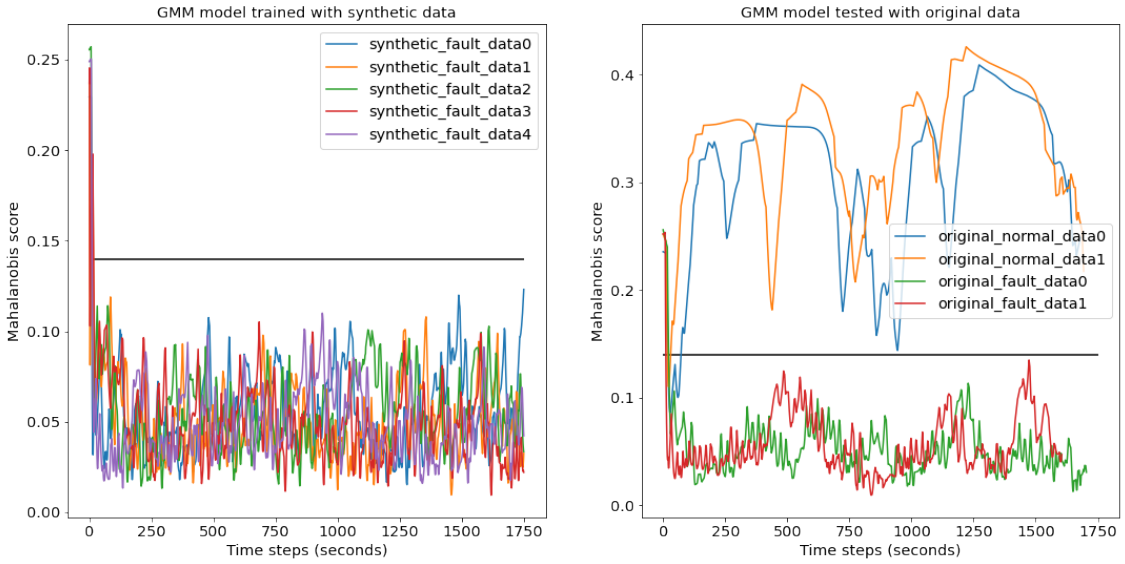

Figure 4.15: Two graphs showing TSTR technique for GMM model

Conclusion for mahalanobis score: The GMM model is developed for anomaly detection and it is evaluated with TRTS, and TSTR techniques for timeGAN model trained with both settings. As the difference in the mean mahalanobis scores between original and synthetic fault data for 2 settings clearly imply, the timeGAN model trained with data split into 60 seconds produce more realistic time-series data than the timeGAN model trained with data split into 30 second samples.

5
Conclusion

The automotive industries are increasingly in need of a large amount of data for improving their services to customers in various aspects. The time-series data play a crucial role in many of Scania's research activities. Collecting this time-series data from the sensors in the vehicle whenever required becomes a time-consuming and expensive process. This thesis focuses on the generation of synthetic time-series data by using less amount of training data. The proposed machine learning model in this thesis can be modified with changes in data pr-processing, and hyper-parameter settings to produce any kind of multivariate time-series data. This makes the timeGAN model more robust.

In this thesis, we studied and explored various machine learning models for the generation of synthetic data. We began our project in the literature review phase, where we reviewed different methods to pre-process the data [11], develop the model [35][13][21][28], and identified suitable quantitative evaluation metrics [31][1]. We looked into many similar research articles, and projects and got insight from those resources. At the end of the literature study, we came up with the idea of trying out two techniques.

Initially, the properties of various features present in the data, how each feature is dependent on each other, and the boundary conditions of each feature are studied. Later, data cleaning and pre-processing on the available training data is conducted. This pre-processed data is used as the training data to develop a GAN model based on CNN architecture. The synthetic time-series data generated using the CNN based GANs were present in the same boundary range as the original data, but the data was noisy and consisted of lots of outlier points.

Since the CNN layers did not capture the dependencies across different features, we switched our focus to working with LSTM based GANs. The LSTM models consume more computational power and time to train because of their complex structure. The GANs built using LSTMs require training for thousands of epochs to get an acceptable result. Considering these factors, we added the auto-encoder part (embedding and recovery functions) with the traditional GAN part (generator and discriminator) and incorporated supervised learning into the unsupervised machine learning model. In this way, timeGAN was developed, and trained for around 30000 epochs for different settings.

Out of different settings, the timeGAN model trained with the two most prominent

settings is selected, and compared. The generated synthetic data by the timeGAN network is shown in section 4.2. Visual evaluation is made by comparing the original and synthetic data using plotly, and PCA. Quantitative evaluation for the synthetic time-series data is done by calculating the regression, and Mahalanobis scores with TRTR, TSTR and TRTS methods. These visual and quantitative results prove that the timeGAN model with a particular setting can generate realistic looking time-series data similar to the original data.

The synthetic data generated using the timeGAN model produce some points away from the original data distribution. To be able to overcome this issue, the model requires more optimization and tuning in future work. Due to the time constraint, the GMM model designed for anomaly detection application is specific for this particular time-series data, and it must be tweaked to be used as an evaluation method to assess all kinds of synthetic time-series data. More rigorous evaluation metrics are required to assess the quality of the synthetic data so that it can be utilised in real-time services provided to the customers.

The thesis concludes by saying that this research work will push Scania one more step towards becoming a customer-centric company by providing better services and drive the automotive industry with its AI-focused innovative products.

Bibliography

[1] Scania CV AB. *LOBSTR - Learning On-Board Signals for Timely Reaction*. 2020. URL: https://www.vinnova.se/en/p/lobstr---learning-on-board-signals-for-timely-reaction/.

[2] M. Alzantot, S. Chakraborty, and M. Srivastava. "SenseGen: A deep learning architecture for synthetic sensor data generation". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2017, pp. 188–193. DOI: 10.1109/PERCOMW.2017.7917555.

[3] Jason Brownlee. *How to Code the GAN Training Algorithm and Loss Functions*. Jan. 2020. URL: https://machinelearningmastery.com/how-to-code-the-generative-adversarial-network-training-algorithm-and-loss-functions/.

[4] *Deep Convolutional Generative Adversarial Network nbsp;: nbsp; TensorFlow Core*. URL: https://www.tensorflow.org/tutorials/generative/dcgan.

[5] EricEric. *Flatten or detrend a seasonal time series*. May 1966. URL: https://stackoverflow.com/questions/46508919/flatten-or-detrend-a-seasonal-time-series.

[6] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*. 2017. arXiv: 1706.02633 [stat.ML].

[7] Mohammad Fekri, Ananda Mohon Ghosh, and Katarina Grolinger. "Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks". In: *Energies* 13 (Dec. 2019). DOI: 10.3390/en13010130.

[8] Germain Forestier et al. "Generating Synthetic Time Series to Augment Sparse Datasets". In: Nov. 2017, pp. 865–870. DOI: 10.1109/ICDM.2017.106.

[9] OpenGenus Foundation. *When to use Recurrent Neural Networks (RNN)?* Nov. 2018. URL: https://iq.opengenus.org/when-to-use-recurrent-neural-networks-rnn/.

[10] Raouf Ganda and Ausif Mahmood. "Deep Learning approach for sentiment analysis of short texts". In: Apr. 2017, pp. 705–710. DOI: 10.1109/ICCAR.2017.7942788.

[11] Shay Geller. *Normalization vs Standardization-Quantitative analysis*. Apr. 2019. URL: https://towardsdatascience.com/normalization-vs-standardization-quantitative-analysis-a91e8a79cebf.

[12] Al Gharakhanian. URL: https://www.linkedin.com/pulse/gans-one-hottest-topics-machine-learning-al-gharakhanian/?trk=pulse_spock-articles.

[13] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[14] Scania Group. *Sustainability ambitions and targets*. May 2021. URL: https://www.scania.com/group/en/targets.html.

[15] Gabriel Lima Guimaraes et al. *Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models*. 2018. arXiv: 1705.10843 [stat.ML].

[16] Aishwarya Gulve. *Everything about Components of Time Series: Part-1*. Apr. 2020. URL: https://aishwaryagulve97.medium.com/everything-about-components-of-time-series-part-1-7476fb521477.

[17] Dishashree Gupta. *Recurrent Neural Network: Fundamentals Of Deep Learning*. Nov. 2020. URL: https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/.

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

[19] Jj. *MAE and RMSE - Which Metric is Better?* Mar. 2016. URL: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d.

[20] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[21] Hugo Larochelle and Iain Murray. "The Neural Autoregressive Distribution Estimator". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 29–37. URL: http://proceedings.mlr.press/v15/larochelle11a.html.

[22] Eryk Lewinson. *Python For Finance Cookbook*. 1st ed. Birmingham, UK: Packt Publishing, 2020. ISBN: 9781789618518.

[23] Lizzie. URL: https://fukamilab.github.io/BIO202/09-A-time-series.html.

[24] $Many_to_One_with_Variable_Sequence_Length$. URL: https://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/many-to-one-with-variable-sequence-length.

[25] B. Mehlig. *Machine learning with neural networks*. 2021. arXiv: 1901.05639 [cs.LG].

[26] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[27] Mayank Mishra. *Convolutional Neural Networks, Explained*. Sept. 2020. URL: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[28] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[29] Joseph Rocca. *Understanding Generative Adversarial Networks (GANs)*. Mar. 2021. URL: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29.

[30] Sharoon Saxena. *RMSE vs RMLSE-What's the Difference? When Should You use Them?* Feb. 2020. URL: https://medium.com/analytics-vidhya/root-mean-square-log-error-rmse-vs-rmlse-935c6cc1802a.

[31] Stephanie. *Mahalanobis Distance: Simple Definition, Examples*. Sept. 2020. URL: https://www.statisticshowto.com/mahalanobis-distance/.

[32] *Understanding LSTM Networks*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[33] Super User. $Many_to_One_with_Fixed_Sequence_Length$. URL: https://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/many-to-one-with-fixed-sequence-length.

[34] user3623888. 2016. URL: https://stats.stackexchange.com/questions/209247/more-effective-seasonal-adjustment-to-time-series-data.

[35] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: www.shorturl.at/evGT5.

[36] Yi Zheng et al. "Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks". In: *Web-Age Information Management*. Ed. by Feifei Li et al. Cham: Springer International Publishing, 2014, pp. 298–310. ISBN: 978-3-319-08010-9.

[37] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. arXiv: 1703.10593 [cs.CV].

A

Appendix 1

A.1 CNN Based GAN Network summary

A.1.1 Generator architecture

```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 120, 8)]          0
_________________________________________________________________
conv1d (Conv1D)              (None, 120, 64)           576
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 120, 64)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 120, 64)           20544
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 120, 64)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 120, 64)           20544
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 120, 64)           0
_________________________________________________________________
flatten (Flatten)            (None, 7680)              0
_________________________________________________________________
dense (Dense)                (None, 960)               7373760
_________________________________________________________________
reshape (Reshape)            (None, 120, 8)            0
=================================================================
Total params: 7,415,424
Trainable params: 7,415,424
Non-trainable params: 0
```

Figure A.1: CNN based GAN generator summary

A.1.2 Discriminator architecture

```
Model: "Discrminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 120, 8)]          0
_________________________________________________________________
conv1d (Conv1D)              (None, 120, 64)           576
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 120, 64)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 120, 64)           20544
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 120, 64)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 120, 64)           20544
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 120, 64)           0
_________________________________________________________________
flatten (Flatten)            (None, 7680)              0
_________________________________________________________________
dense (Dense)                (None, 8)                 61448
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 103,121
Trainable params: 103,121
Non-trainable params: 0
```

Figure A.2: CNN based GAN discriminator summary

A.2 TimeGAN Network summary

A.2.1 Generator architecture

```
Model: "Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, None, 32)          5248
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
dense (Dense)                (None, None, 4)           132
=================================================================
Total params: 22,020
Trainable params: 22,020
Non-trainable params: 0
```

Figure A.3: TimeGAN generator summary

A.2.2 Discriminator architecture

```
Model: "Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, None, 32)          4736
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
dense (Dense)                (None, None, 1)           33
=================================================================
Total params: 21,409
Trainable params: 21,409
Non-trainable params: 0
```

Figure A.4: TimeGAN discriminator summary

A.2.3 Embedding function architecture

```
Model: "Embedding function"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, None, 32)          5248
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
dense (Dense)                (None, None, 4)           132
=================================================================
Total params: 22,020
Trainable params: 22,020
Non-trainable params: 0
```

Figure A.5: TimeGAN embedding function summary

A.2.4 Recovery function architecture

```
Model: "Recovery function"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, None, 32)          4736
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
lstm_2 (LSTM)                (None, None, 32)          8320
_________________________________________________________________
dense (Dense)                (None, None, 8)           264
=================================================================
Total params: 21,640
Trainable params: 21,640
Non-trainable params: 0
```

Figure A.6: TimeGAN recovery function summary

A.3 Visual representation

This section contains the feature-wise visual comparison of the original and synthetic data for the most prominent setting which is timeGAN trained with original data split into 60 second sequence samples (setting 2).

A.3.1 Feature 1

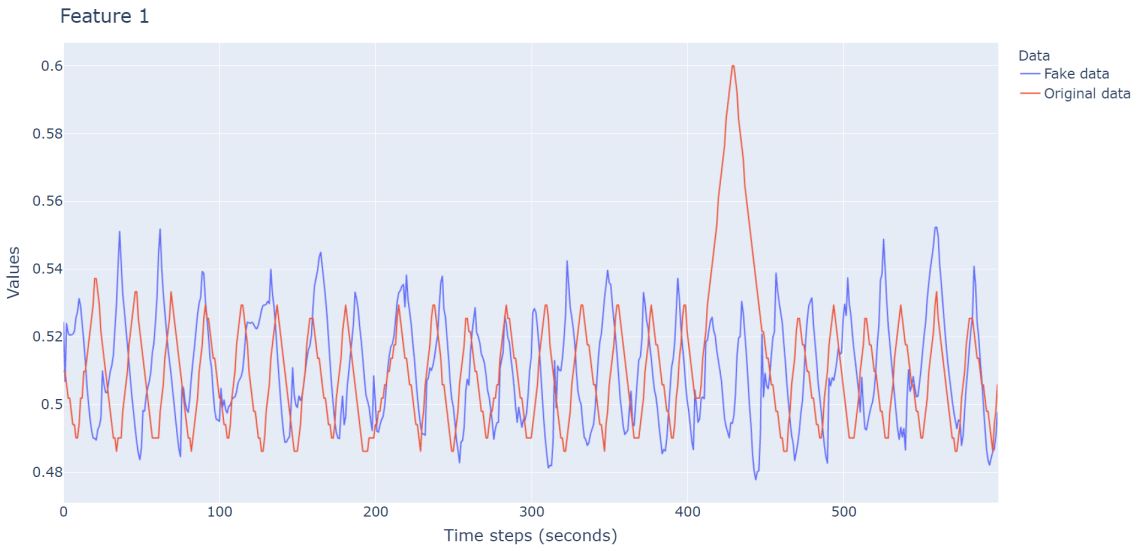

Figure A.7: Original and synthetic data's Feature 1 plotted in graph

A.3.2 Feature 2

Figure A.8: Original and synthetic data's Feature 2 plotted in graph

A.3.3 Feature 3

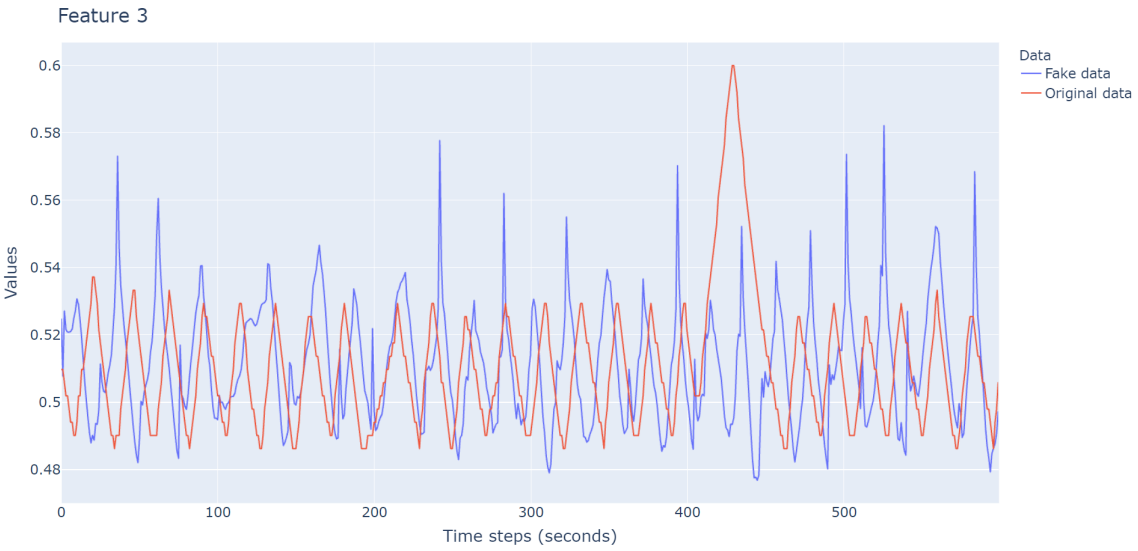

Figure A.9: Original and synthetic data's Feature 3 plotted in graph

A.3.4 Feature 4

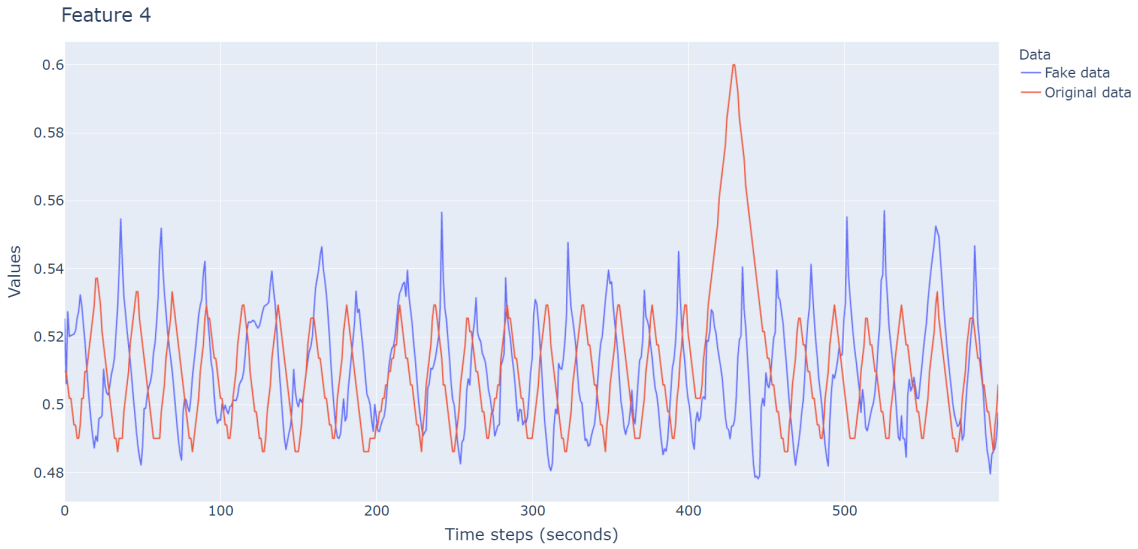

Figure A.10: Original and synthetic data's Feature 4 plotted in graph

A.3.5 Feature 5

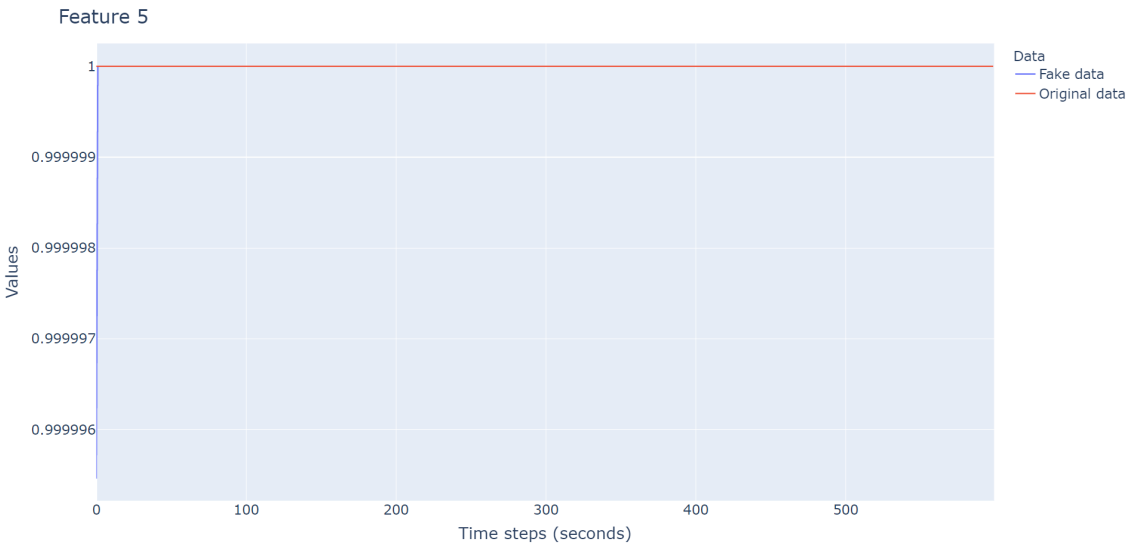

Figure A.11: Original and synthetic data's Feature 5 plotted in graph

A.3.6 Feature 6

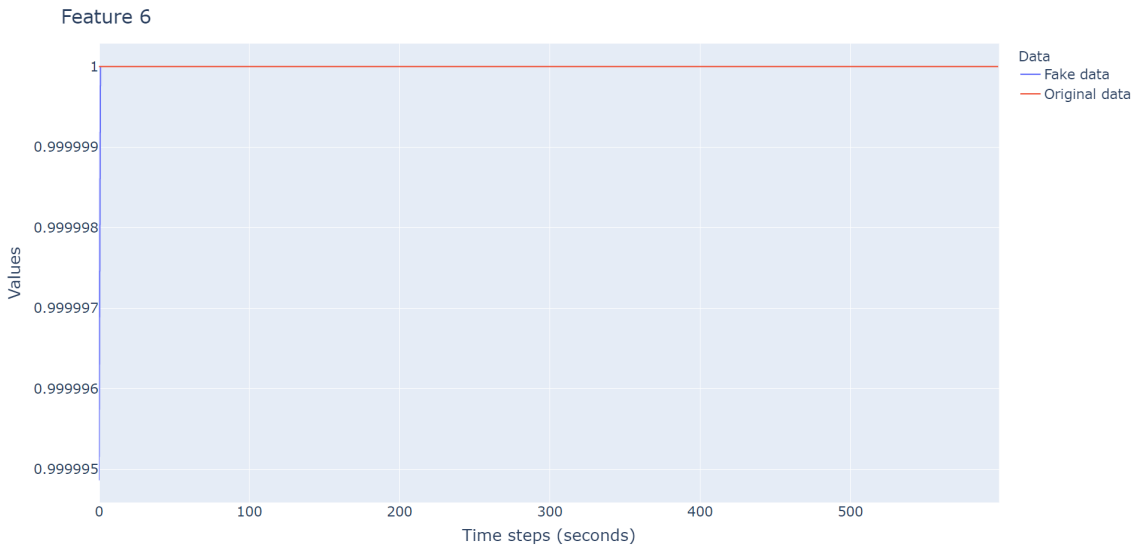

Figure A.12: Original and synthetic data's Feature 6 plotted in graph

A.3.7 Feature 7

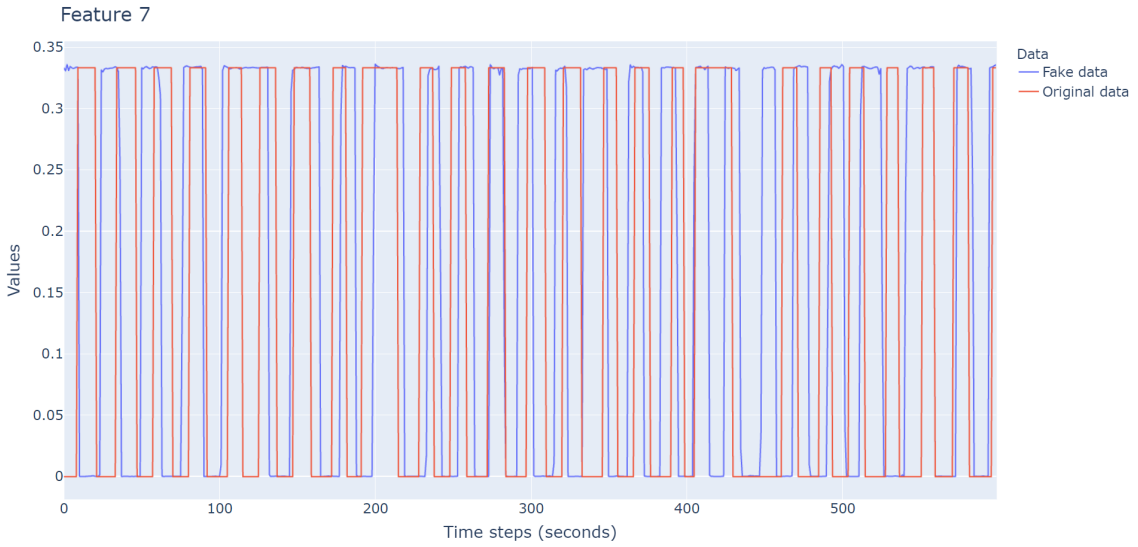

Figure A.13: Original and synthetic data's Feature 7 plotted in graph

A.3.8 Feature 8

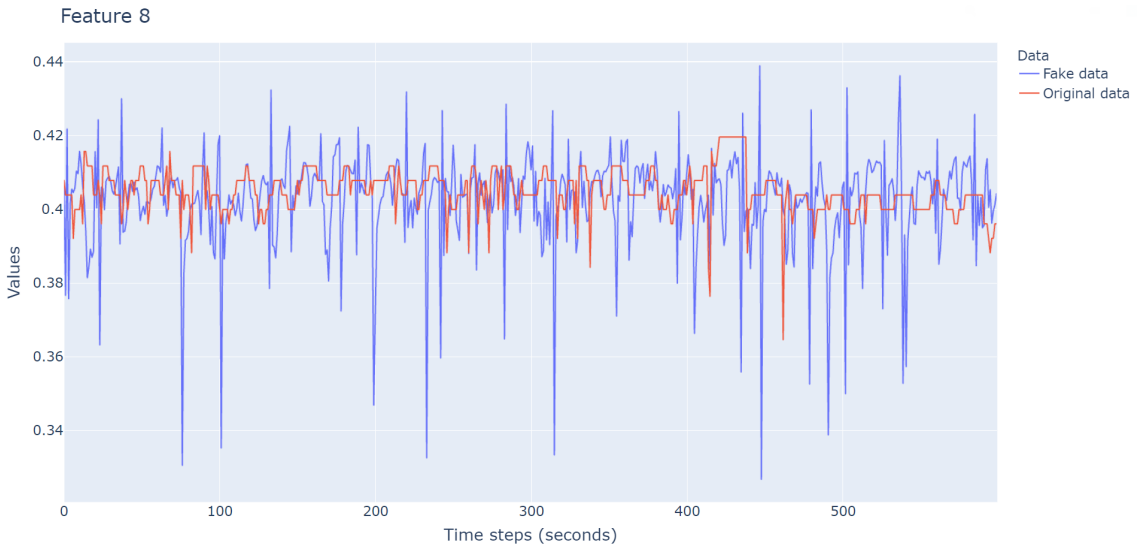

Figure A.14: Original and synthetic data's Feature 8 plotted in graph

A.4 Contribution of thesis students

My partner Sofia Nord from KTH University and I have made equal contribution for this thesis. We have split our work equally, and our individual contribution is shown below.

A.4.1 Deepak's contributions:

- PCA Visualization
- Data pre-processing – combined all data files, column dropping
- Build Discriminator (CNN based GAN)
- Model training (CNN based GAN)
- Training Function(enabled training from several data files)
- Enabled variable sequence length data generation

TimeGAN part:

- Developed build recovery function
- Developed train discriminator function
- Developed get samples function
- Developed train supervisor function
- Developed train embedding function
- Developed train function
- Developed results function
- Generated graphs for reports
- PCA Evaluation
- Developed Mahalanobis Score from external reference(Evaluation Method)

A.4.2 Sofia's contributions:

- Data pre-processing - Normalization
- class TimeGAN/GAN shell
- Build Generator (CNN based GAN)
- Model training (CNN based GAN)
- Training Function(enabled training several batches)
- Loss Plot

TimeGAN part:

- TimeGAN init update function
- Developed build embedder function
- Developed train generator function
- Developed train autoencoder function
- Developed train function
- Converted .ipynb files to .py files
- Developed train function
- Developed Regression Score from external reference (Evaluation method)
- Generated graphs for reports

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY