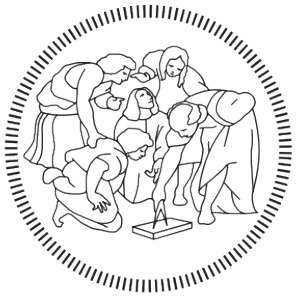
Time-Conditional Generative Adversarial Networks for Augmentation of Irregularly Sampled Time Series



POLITECNICO
MILANO 1863

Lorenzo Corti

Student Id: 893845

Nicolò Oppido

Student Id: 899078

Supervisor: Prof. Marco Brambilla

Co-Advisor: Prof. Pavlos Protopapas

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

This thesis is submitted for the
Master of Science in Computer Science and Engineering
December 2019

If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them.

Henry David Thoreau

Ringraziamenti

I nostri primi ringraziamenti vanno alle nostre famiglie le quali ci hanno supportato, tra alti e bassi, durante i nostri percorsi universitari.

Ringraziamo i nostri amici e compagni di università. Grazie a tutti per esserci stati durante questi anni e per averli resi così speciali.

Ringraziamenti particolari vanno anche al professor Marco Brambilla per averci guidato durante il lavoro di tesi.

Grazie, infine, a tutti i professori incontrati al Politecnico di Milano i quali ci hanno arricchito con i loro insegnamenti.

Abstract

In the field of time-series analysis, one of the most common problem is handling of missing values. These values can go missing either for unforeseeable causes during the measurement process or even because this is expected to happen. In the former case, removing the records with incomplete data may well be an available option while, in the latter, it may hinder the construction of predictive models. Therefore, over the years, several techniques for imputing missing data, and a variety of their implementations, have been proposed. These techniques include interpolation and statistical models, such as ARIMA, and their variants. However, there are not many methods that deal with irregularly sampled time-series in their unaltered form. In fact, the previously mentioned strategies can't be successfully applied to this type of data 'as-is'.

In this thesis, three generative approaches based on recent developments in the Deep Learning field will be proposed and compared. The goal is to reconstruct incomplete and irregularly sampled time-series. The introduced methods used throughout the experiments are based on Convolutional Neural Networks, CNN, and Recurrent Neural Networks, RNN. These models have been trained with the framework called GAN which stands for Generative Adversarial Networks. Specifically, the conditioned variants of the networks have been used to generate the missing values. The outcome of this work shows, both qualitatively and quantitatively, how these techniques can capture the dynamics of the time-series and reproduce them correctly.

Abstract

Nel campo dell'analisi di serie temporali uno dei problemi più comuni è la gestione dei dati mancanti. Questi valori possono venire meno per cause imprevedibili nel processo di misurazione oppure perché la natura del fenomeno in analisi li prevede. Nel primo caso, rimuovere i record che presentano dati incompleti può non impattare sui modelli basati su tali dati. Nel secondo caso, rimuovere queste osservazioni influenzerebbe negativamente la realizzazione di modelli predittivi. Per questo motivo, molte tecniche di imputazione dei dati mancanti sono state proposte negli anni, così come le altrettante loro implementazioni. Queste tecniche includono interpolazione e modelli statistici, come ARIMA, nelle loro declinazioni. Poche sono però le metodologie che trattano serie temporali campionate irregolarmente nella loro forma originale. Di fatto, le strategie riportate prima non possono essere applicate con successo a questo tipo di dati.

In questa tesi proponiamo e confrontiamo tre approcci generativi basati sui recenti sviluppi nell'ambito del Deep Learning per ricostruire serie temporali irregolari incomplete. I metodi che andremo a descrivere ed usare nei nostri esperimenti si basano su Reti Neurali Convoluzionali, CNN, e Reti Neurali Ricorrenti, RNN. Questi modelli sono allenati sfruttando il framework chiamato GAN, ovvero, Reti Generative Avversarie. Più precisamente andremo a sfruttare le varianti condizionate di queste reti per produrre i valori persi. I risultati del nostro lavoro mostrano, sia qualitativemente sia quantitativamente, come queste tecniche siano in grado di catturare le dinamiche delle serie temporali e riprodurle correttamente.

Contents

List of Figures **ix**

List of Tables **xiii**

1 Introduction **1**
1.1 Context and problem statement 1
1.2 Proposed solution 2
1.3 Structure of the thesis 3

2 Background **4**
2.1 Neural Networks 4
2.2 Recurrent Neural Network 6
2.2.1 Long-Term Memory Problem 6
2.2.2 Long Short-Term Memory Networks 7
2.3 Convolutional Neural Networks 8
2.4 Generative Adversarial Networks 10
2.4.1 Conditioned Generative Adversarial Networks 11
2.5 Irregular Time Series 12
2.6 AUROC 12

3 Related Work **13**
3.1 Missing values in time series data 13
3.2 Dealing with irregularly sampled time series 14
3.3 Applications of Generative Adversarial Networks 14

4 Methodology **15**
4.1 Missing data and proposed solutions 15
4.2 Baseline Definition 16
4.3 Conditioning the Generator 17

4.4 Learning the relationship between values and time . . . 20
4.5 Recurrent Neural Networks . . . 23
4.6 Evaluating the generated examples . . . 25

5 Implementation **27**
5.1 Development and Deployment . . . 27
5.2 Data Preparation . . . 28
5.3 Baseline . . . 28
5.4 CNN Based GANs . . . 28
5.5 LSTM Based GAN . . . 28

6 Experiments and results **30**
6.1 Datasets . . . 30
6.1.1 Synthetic Periodic Time Series . . . 30
6.1.2 Synthetic Not-Periodic Time Series . . . 31
6.1.3 Real Time Series . . . 32
6.2 Experimental Strategy . . . 34
6.3 Baseline . . . 36
6.3.1 Experiments on Periodic Synthetic Time Series . . . 36
6.3.2 Experiments on Real Time Series . . . 39
6.4 Time Conditional GAN . . . 42
6.4.1 Experiments on Periodic Synthetic Time Series . . . 42
6.4.2 Experiments on Non Periodic Synthetic Time Series . . . 45
6.4.3 Experiments on Real Time Series . . . 48
6.5 TA-CGAN . . . 52
6.5.1 Experiments on Periodic Synthetic Time Series . . . 53
6.5.2 Experiments on Non Periodic Synthetic Time Series . . . 56
6.5.3 Experiments on Real Time Series . . . 59
6.6 LSTM GAN . . . 64
6.6.1 Experiments on Periodic Synthetic Time Series . . . 64
6.6.2 Experiments on Non Periodic Synthetic Time Series . . . 69
6.6.3 Experiments on Real Time Series . . . 72
6.7 Summary of results . . . 78

7 Conclusion **80**
7.1 Summary . . . 80
7.2 Threats to validity . . . 80

7.3 Future works 81

Bibliography **82**

List of Figures

2.1 Feed Forward Neural Network example 5
2.2 Rolled Recurrent Neural Network [1] 6
2.3 LSTM Neural Network [2] 8
2.4 Example of Convolutional Neural Network 9
2.5 Generative Adversarial Networks [3] 11
2.6 Conditional Generative Adversarial Networks 11

4.1 T-CGAN method 18
4.2 T-CGAN architecture 19
4.3 TA-CGAN method 20
4.4 TA-CGAN architecture 22
4.5 LSTM GAN method 24
4.6 LSTM GAN architecture 25

6.1 Examples of Sine wave signals 31
6.2 Examples of Sawtooth signals 31
6.3 Examples of Sine wave signals with increasing amplitude over time 32
6.4 Examples of Parabolae 32
6.5 Examples from SonyAIBORobotSurface dataset 33
6.6 Examples from Wafer dataset 34
6.7 Baseline - Examples of Sine wave signals completed (second half) 37
6.8 Baseline - Examples of Sawtooth signals completed (second half) 37
6.9 Baseline - Examples of Sine wave signals completed (initial and final ends) 37
6.10 Baseline - Examples of Sawtooth signals completed (initial and final ends) . 38
6.11 Baseline - Examples of Sine wave signals completed (first half) 38
6.12 Baseline - Examples of Sawtooth signals completed (first half) 38
6.13 Baseline - Examples of Sony 'Class 1' signals completed (second half) . . . 39
6.14 Baseline - Examples of Sony 'Class 2' signals completed (second half) . . . 40

6.15 Baseline - Examples of Sony 'Class 1' signals completed (initial and final ends) 40
6.16 Baseline - Examples of Sony 'Class 2' signals completed (initial and final ends) 40
6.17 Baseline - Examples of Sony 'Class 1' signals completed (first half) 41
6.18 Baseline - Examples of Sony 'Class 2' signals completed (first half) 41
6.19 T-CGAN - Examples of Sine wave signals completed (second half) 42
6.20 T-CGAN - Examples of Sawtooth signals completed (second half) 42
6.21 T-CGAN - Examples of Sine wave signals completed (initial and final ends) 43
6.22 T-CGAN - Examples of Sawtooth signals completed (initial and final ends) 43
6.23 T-CGAN - Examples of Sine wave signals completed (first half) 43
6.24 T-CGAN - Examples of Sawtooth signals completed (first half) 44
6.25 T-CGAN - Examples of Parabolae completed (second half) 45
6.26 T-CGAN - Examples of Sine wave signals with increasing amplitude completed (second half) 45
6.27 T-CGAN - Examples of Parabolae completed (initial and final ends) 46
6.28 T-CGAN - Examples of Sine wave signals with increasing amplitude completed (initial and final ends) 46
6.29 T-CGAN - Examples of paraboale completed (first half) 46
6.30 T-CGAN - Examples of Sine wave signals with increasing amplitude completed (first half) 47
6.31 T-CGAN - Examples of Sony 'Class 1' signals completed (second half) . . 48
6.32 T-CGAN - Examples of Sony 'Class 2' signals completed (second half) . . 48
6.33 T-CGAN - Examples of Wafer 'Positives' signals completed (second half) . 49
6.34 T-CGAN - Examples of Wafer 'Negatives' signals completed (second half) 49
6.35 T-CGAN - Examples of Sony 'Class 1' signals completed (initial and final ends) 49
6.36 T-CGAN - Examples of Sony 'Class 2' signals completed (initial and final ends) 50
6.37 T-CGAN - Examples of Wafer 'Positives' signals completed (initial and final ends) 50
6.38 T-CGAN - Examples of Wafer 'Negatives' signals completed (initial and final ends) 50
6.39 T-CGAN - Examples of Sony 'Class 1' signals completed (first half) 51
6.40 T-CGAN - Examples of Sony 'Class 2' signals completed (first half) 51
6.41 T-CGAN - Examples of Wafer 'Positives' signals completed (first half) . . . 51
6.42 T-CGAN - Examples of Wafer 'Negatives' signals completed (first half) . . 52
6.43 TA-CGAN - Examples of Sine wave signals completed (second half) 53

6.44 TA-CGAN - Examples of Sawtooth signals completed (second half) 53
6.45 TA-CGAN - Examples of Sine wave signals completed (initial and final ends) 54
6.46 TA-CGAN - Examples of Sawtooth signals completed (initial and final ends) 54
6.47 TA-CGAN - Examples of Sine wave signals completed (first half) 54
6.48 TA-CGAN - Examples of Sawtooth signals completed (first half) 55
6.49 TA-CGAN - Examples of Parabolae completed (second half) 56
6.50 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (second half) 56
6.51 TA-CGAN - Examples of Parabolae completed (initial and final ends) . . . 57
6.52 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (initial and final ends) 57
6.53 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (first half) 58
6.54 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (first half) 58
6.55 TA-CGAN - Examples of Sony 'Class 1' signals completed (second half) . 60
6.56 TA-CGAN - Examples of Sony 'Class 2' signals completed (second half) . 60
6.57 TA-CGAN - Examples of Wafer 'Positives' signals completed (second half) 60
6.58 TA-CGAN - Examples of Wafer 'Negatives' signals completed (second half) 61
6.59 TA-CGAN - Examples of Sony 'Class 1' signals completed (initial and final ends) 61
6.60 TA-CGAN - Examples of Sony 'Class 2' signals completed (initial and final ends) 61
6.61 TA-CGAN - Examples of Wafer 'Positives' signals completed (initial and final ends) 62
6.62 TA-CGAN - Examples of Wafer 'Negatives' signals completed (initial and final ends) 62
6.63 TA-CGAN - Examples of Sony 'Class 1' signals completed (first half) . . . 62
6.64 TA-CGAN - Examples of Sony 'Class 2' signals completed (first half) . . . 63
6.65 TA-CGAN - Examples of Wafer 'Positives' signals completed (first half) . . 63
6.66 TA-CGAN - Examples of Wafer 'Negatives' signals completed (first half) . 63
6.67 LSTM GAN - Examples of completely generated Sine wave signals 65
6.68 LSTM GAN - Examples of completely generated sawtooth signals 65
6.69 LSTM GAN - Examples of Sine wave signals completed (second half) . . . 66
6.70 LSTM GAN - Examples of Sawtooth signals completed (second half) . . . 66
6.71 LSTM GAN - Examples of Sine wave signals completed (initial and final ends) 67

6.72 LSTM GAN - Examples of Sawtooth signals completed (initial and final ends) 67
6.73 LSTM GAN - Examples of Sine wave signals completed (first half) 67
6.74 LSTM GAN - Examples of Sawtooth signals completed (first half) 68
6.75 LSTM GAN - Examples of Parabolae completed (second half) 69
6.76 LSTM GAN - Examples of Sine wave signals with increasing amplitude completed (second half) 69
6.77 LSTM GAN - Examples of Parabolae completed (initial and final ends) . . 70
6.78 LSTM GAN - Examples of Sine wave signals with increasing amplitude completed (initial and final ends) 70
6.79 LSTM GAN - Examples of paraboale completed (first half) 71
6.80 Examples of Sine wave signals with increasing amplitude completed (first half) by the LSTM GAN 71
6.81 LSTM GAN - Examples of Sony 'Class 1' signals completed (second half) 73
6.82 LSTM GAN - Examples of Sony 'Class 2' signals completed (second half) 73
6.83 LSTM GAN - Examples of Wafer 'Positives' signals completed (second half) 73
6.84 LSTM GAN - Examples of Wafer 'Negatives' signals completed (second half) 74
6.85 LSTM GAN - Examples of Sony 'Class 1' signals completed (initial and final ends) 74
6.86 LSTM GAN - Examples of Sony 'Class 2' signals completed (initial and final ends) 74
6.87 LSTM GAN - Examples of Wafer 'Positives' signals completed (initial and final ends) 75
6.88 LSTM GAN - Examples of Wafer 'Negatives' signals completed (initial and final ends) 75
6.89 LSTM GAN - Examples of Sony 'Class 1' signals completed (first half) . . 75
6.90 LSTM GAN - Examples of Sony 'Class 2' signals completed (first half) . . 76
6.91 LSTM GAN - Examples of Wafer 'Positives' signals completed (first half) . 76
6.92 LSTM GAN - Examples of Wafer 'Negatives' signals completed (first half) 76
6.93 Comparison between T-CGAN and LSTM GAN for the complete generation of synthetic, periodic, time series 78
6.94 Average AUROC measurements for T-CGAN (a), TA-CGAN(b) and LSTM GAN (c) on sythetic data. The three experimental settings Initial (solid, blue), Central (dotted, orange) and Final (dashed, grey) are reported separately. . . 79

List of Tables

6.1 Summary of experimental settings on synthetic time series 35
6.2 Summary of experimental settings on real world time series 36
6.3 AUROC - Baseline - Periodic Synthetic Time Series 39
6.4 AUROC - Baseline - Sony Time Series 41
6.5 AUROC - T-CGAN - Periodic Synthetic Time Series - Initial Conditioning . 44
6.6 AUROC - T-CGAN - Periodic Synthetic Time Series - Central Conditioning 44
6.7 AUROC - T-CGAN - Periodic Synthetic Time Series - Final Conditioning . 45
6.8 AUROC - T-CGAN - Non Periodic Synthetic Time Series - Initial Conditioning 47
6.9 AUROC - T-CGAN - Non Periodic Synthetic Time Series - Central Conditioning 47
6.10 AUROC - T-CGAN - Non Periodic Synthetic Time Series - Final Conditioning 48
6.11 AUROC - T-CGAN - Real Time Series - Sony 52
6.12 AUROC - T-CGAN - Real Time Series - Wafer 52
6.13 AUROC - TA-CGAN - Periodic Synthetic Time Series - Initial Conditioning 55
6.14 AUROC - TA-CGAN - Periodic Synthetic Time Series - Central Conditioning 55
6.15 AUROC - TA-CGAN - Periodic Synthetic Time Series - Final Conditioning 56
6.16 AUROC - TA-CGAN - Non Periodic Synthetic Time Series - Initial Conditioning 59
6.17 AUROC - TA-CGAN - Non Periodic Synthetic Time Series - Central Conditioning 59
6.18 AUROC - TA-CGAN - Non Periodic Synthetic Time Series - Final Conditioning 59
6.19 AUROC - TA-CGAN - Real Time Series - Sony 64
6.20 AUROC - TA-CGAN - Real Time Series - Wafer 64
6.21 AUROC - LSTM GAN - Periodic Synthetic Time Series - Complete Generation 65
6.22 AUROC - LSTM GAN - Periodic Synthetic Time Series - Initial Conditioning 68
6.23 AUROC - LSTM GAN - Periodic Synthetic Time Series - Central Conditioning 68
6.24 AUROC - LSTM GAN - Periodic Synthetic Time Series - Final Conditioning 69

6.25 AUROC - LSTM GAN - Non Periodic Synthetic Time Series - Initial Conditioning 71
6.26 AUROC - LSTM GAN - Non Periodic Synthetic Time Series - Central Conditioning 72
6.27 AUROC - LSTM GAN - Non Periodic Synthetic Time Series - Final Conditioning 72
6.28 AUROC - LSTM GAN - Real Time Series - Sony 77
6.29 AUROC - LSTM GAN - Real Time Series - Wafer (halved length) 77

Chapter 1

Introduction

1.1 Context and problem statement

Almost everything that can be measured can be considered as a time series. Therefore, whenever data is measured and stored, issues with missing values [23] can occur. These values can go missing for unforeseeable causes during the measurement process or because the nature of the phenomenon at hand allows for them to happen. Having complete time series data is a crucial requirement for the effectiveness of the analyses. Conventional methods such as mean or mode imputation, deletion, and other methods are sometimes not good enough to handle missing values. At least not enough to avoid creating data bias. The management of missing values is generally included in the pre-processing phase, precisely to avoid that the models built or trained on the data may present unexpected behaviors in the future. If the data is randomly lost, removing or replacing records with incomplete data may be an option and may not impact the analysis. However if the data is not lost at random, for example due to prolonged breakage of measurement systems or because that data is supposed to be missing, removing or replacing it with non-advanced techniques could cause inconsistencies in future analyses made with predictive models. For this reason, many techniques [21] for imputing missing data have been proposed over the years, as well as their many implementations. Since imputation is a huge area, many studies revolving this problem can be found. These techniques include interpolation and statistical models, such as ARIMA, and their variants. There is also an additional problem to consider: what happens if data with missing values is also sampled irregularly? A common approach to analyzing irregularly spaced time series is to transform the data into equally spaced observations using some form of interpolation, often linear, in order to apply existing methods for equally spaced data. However, transforming data in such a way can introduce a number of significant and hard to quantify biases, especially if the spacing of observations is highly irregular. So ideally,

irregular sampled time series should be analyzed in their unaltered form. However, few are the methods that deal with irregularly sampled time series in their unaltered form. In fact, the previously mentioned strategies can't be successfully applied to this type of data 'as-is', and there are just few methods in literature that handle this kind of data, like [9].

1.2 Proposed solution

In this study, three generative approaches based on recent developments in the Deep Learning field are proposed and compared to address the problem described in the previous section. All of the proposed methodologies are based on the notion of Generative Adversarial Networks, GAN, first introduced by Goodfellow et al. [7]. This framework allows to estimate a generative model capable of producing new data like images, text and audio. Two out of the three techniques make use of Convolutional Neural Networks [15] to define the generative model G and the discriminative model D. Differently, the third approach is based on Recurrent Neural Networks, more precisely on one of their variants: Long-Short Term Memory Networks [12].

In each of the three implementations, the two networks competing in the adversarial process are conditioned with auxiliary information regarding time or the time series that need to be completed. Conditioning the networks has proven a successful strategy when applied to GAN as in [18] to guide the Generator G both during training and model deployment.

Finally, we evaluate the quality of the completed time series via a complementary model entirely trained on these reconstructed data. The proposed methods and evaluation process are detailed in their entirety in Chapter 4.

1.3 Structure of the thesis

The structure of the thesis is as follows:

- Chapter 2 defines and explains the background knowledge and concepts that are related to the work presented in this thesis.
- Chapter 3 presents the past works that are related to this thesis in the problem they try to answer and the solution they propose.
- Chapter 4 contains a high level description of the employed methods that are used in this thesis.
- Chapter 5 describes the technologies and tools used to implement the proposed methods.
- Chapter 6 presents and discusses the results of the performed experiments.
- Chapter 7 concludes this thesis by summarizing the work, doing a critical discussion and highlighting possible future works.

Chapter 2

Background

This chapter briefly introduces the theoretical topics that form the foundations of the thesis work we are presenting.

2.1 Neural Networks

Artificial Neural Networks are processing devices with an architecture that follows the brain computational model. Like the brain, their processing units are distributed among simple nonlinear units, redundant and thus fault tolerant and intrinsically parallel. Large network might have hundreds or thousands of this units, the neurons. For this reason, they have the capability to interact with noisy data and can adapt themselves to various circumstances, guaranteeing excellent pattern-recognition skills. Artificial Neural Networks, also called Feed Forward Neural Networks, are typically organized in layers which are made up of several interconnected neurons containing an activation function.
The layers can be:

- Input: brings the data into the architecture for processing them by other layers of artificial neurons, finding patterns
- Hidden: a layer in between the input layer and the output layer, where artificial neurons take in a set of weighted inputs and produce an output, the pattern
- Output: the last layer of neurons that produces given outputs

The connections between layers are called weighted connections, and they are adjusted with a real-valued number attached to them. To each neuron is also associated a parameter called bias which acts as a threshold for its activation function.

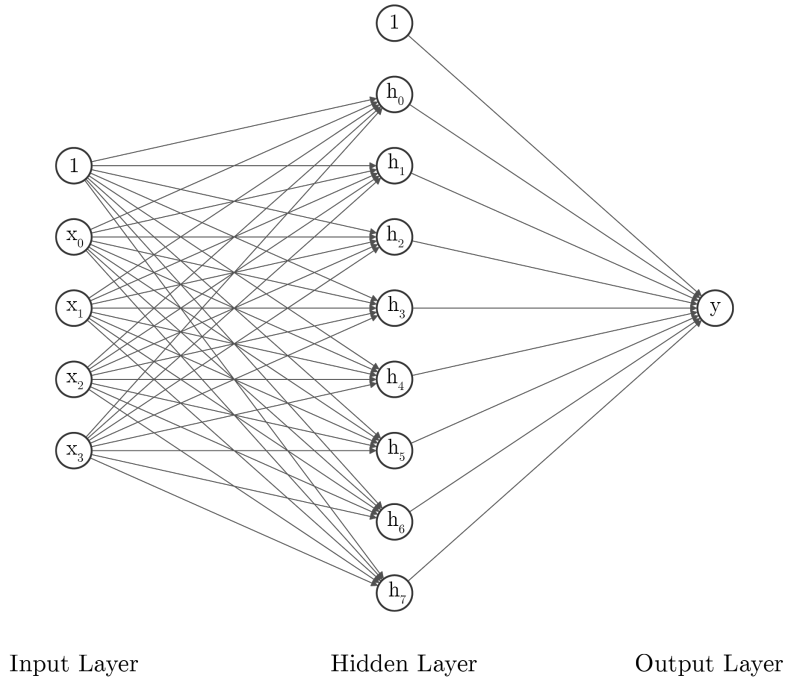


Figure 2.1 Feed Forward Neural Network example

Example of a simple Feed Forward Neural Network with a single Hidden layer denoted by h_i. x_i represent the inputs and y the output.

These model parameters are learnt via an iterative algorithm introduced by Rumelhart et al. called Backpropagation [27]. Backpropagation is a supervised learning algorithm that implements a Gradient Descent strategy in two steps that allows to adjust weights and biases as the network faces different data examples. The first step consists of feeding the network with training examples, letting the information flow forward from one layer to another, and collecting the produced outputs. During the second step the loss function is computed and propagated backwards, fine-tuning all the weights and biases. This is done at each iteration, or epoch, with the goal of reducing the previous loss. This can be seen as a minimization problem from the point of view of the loss function in which the right parameters for the model are the ones that minimize such function over the training data. Through the years different optimizations for Backpropagation have been presented like the use of Stochastic Gradient Descent (SGD) or Mini-Batch Gradient Descent with the latter being the most commonly adopted.

Neural Networks, in their different connotations created in recent years, have found both personal and commercial applications like in handwriting recognition, image recognition, time series analysis, financial operations and enterprise planning.

2.2 Recurrent Neural Networks

Feed Forward Neural Networks don't have the ability to use prior knowledge to inform new decisions. To make up for this major shortcoming Recurrent Neural Networks were introduced. This is a family of networks that present feedback connections to the hidden layers in their architecture in order for the information to persist. These additional connections allow a Recurrent Network to produce outputs that, besides weights and biases, are also influenced by past information the network is able to remember. Thus the output function depends not only by the input but also by the hidden internal state of the model. A basic RNN structure is made of a single neural network layer that is repeated over time, as shown in Figure 2.2.

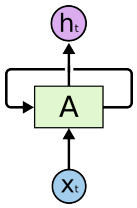


Figure 2.2 Rolled Recurrent Neural Network [1]

X_t is the input at timestep t, A represents the Neural Network layer and h_t is the output at timestep t.

Since these networks operate over their internal time representation, the training algorithm is an adaptation of the Backpropagation: Backpropagation Through Time. This algorithm is applied on the Unrolled Recurrent Network which is equivalent to a Feed Forward Network where each layer corresponds to one timestep of the RNN.

However, these networks present the so called Long-Term Memory Problem which makes them quite hard to train.

2.2.1 Long-Term Memory Problem

There are cases in which the input sequences are too long and we need to have more context, not just searching through recent information. As the distance between the relevant information increases, the RNNs aren't able to keep up with the growing gap. This is caused by what is called the Vanishing Gradient problem: in deep networks we are unable to update, thus learn effectively, the weights as the gradients of the loss function approaches zero for the initial layers. When unrolling through time a RNN, we are faced with this exact issue and the network is incapable of learning to correlate information that is temporally too far distant.

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs

2.2.2 Long Short-Term Memory Networks

LSTM Networks [12] are a special kind of Recurrent Networks that are capable of avoiding the problem of long-term dependencies since they are specifically designed to learn them. They process data by passing on information as usual RNNs but the differences are in how they operate within the repeating modules. Instead of a single neural network layer, LSTMs cells are composed by four layers which constitute the *cell gates*. These gates are able to learn which data in an input sequence is important, therefore to be retained, or not important, therefore to be eliminated.

The other important component of this type of architectures is the *cell state* representing the memory of the network. It transfers the relevant information from previous temporal steps to successive ones through the chain of unrolled recurring cells. Information is added or removed from the state by the gates. The three gates use Sigmoid and Hyperbolic Tangent (Tanh) as their activation functions, either by themselves or combined. The different gates in an LSTM cell are:

- Forget Gate: it decides which information should be thrown away from the previous state C_{t-1}. The activation function used here is the Sigmoid. If the output value is close to 0 it means that the information is to be forgotten. The same reasoning holds for the opposite: output values close to 1 mean the information has to be retained.
  The functionality of this gate is summarized by the equation:

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

  To actually forget information we just need to multiply C_{t-1} by f_t.

- Input Gate: it is the one who determines which information will be added to the cell state. The previous hidden state h_{t-1} and the current input x_t are passed to a Sigmoid layer, to decide which values will be updated, and to a Tanh layer that creates a new vector of candidate values $\widetilde{C}_t$ that could be added to the cell state. The outputs from these two layers are then multiplied. This final vector is then added to C_{t-1}, after already being altered by the Forget Gate, to produce the new state C_t.
  The functionality of this gate is summarized by the equations:

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

  $$\widetilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- Output gate: the final gate decides what is going to be the output by filtering the cell state C_t. First, the previous hidden state h_{t-1} and the current input x_t are passed into a Sigmoid layer to decide which parts will be in the output. Then the cell state C_t passes through a Tanh layer. Finally we multiply the outputs of the Tanh layer and the Sigmoid layer in order to only output the decided parts. The new cell state C_t and the new hidden state h_t are then provided to the next time step.
  The functionality of this gate is summarized by the equations:

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

$$h_t = o_t * C_t$$

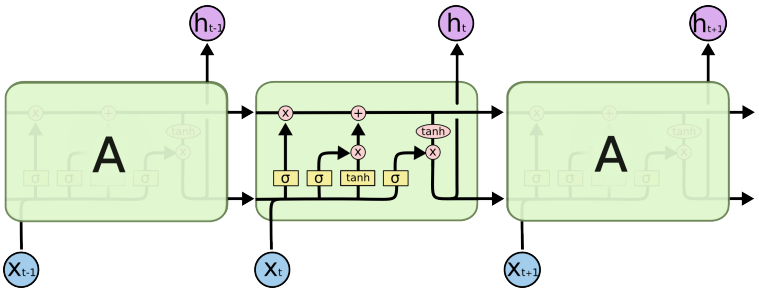


Figure 2.3 LSTM Neural Network [2]

Illustration of a LSTM cell showing the 4 interacting layers constituting the Forget, Input and Output gates as described in this section.

2.3 Convolutional Neural Networks

A Convolutional Neural Network, introduced by LeCun et al. [15], is a kind of network which can take the input, in general an image, and assign importance, through learnable weights and biases, to its various aspects to be able to tell them apart. This kind of network generally requires less data pre-processing when compared to other classification algorithms. Images are only seen as matrices of the pixels' values. There are two main components that by working in tandem contribute to the functionalities and success of Convolutional Neural Networks: Convolutional Layers and Pooling Layers.

Convolutional Layers group together what are called Convolutional Filters, or Kernels. These particular objects move horizontally and vertically, with a given Stride, on the entirety

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs

of the size of a data example performing element-wise multiplications with the part it is currently hovering. The resulting values are finally summed together in order to obtain a single entry of the filter responses, called Feature Map. Filters are capable of successfully capturing spatial dependencies in an image. Moreover their use makes this kind of networks spatial invariant and less prone to overfitting due to the fewer model parameters, especially when compared to FFNN. The input of a generic Convolutional Layer is a $h \times w \times c$ image where h is the height, w is the width and c is the number of channels of the image. On the other hand, Filters are $m \times n$ sized where m is the height and n the width. However, more often than not square filters are used. Their depth always coincides with the one of the input. The other fundamental part in CNNs are the Pooling Layers. They are responsible for shrinking in size the Feature Maps, effectively performing downsampling. This operation is needed in order to decrease the computational power required to process the data. It can be performed in two ways: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the map it covers in a given moment while Average Pooling returns the average of all values from the area of the map currently covered by it. Max Pooling turned out to be more effective than Average Pooling especially in cases with noisy data, acting as a Noise Suppressant.

Several Convolutional Layers and Pooling Layers can be stacked to build a trainable feature extractor capable of recognizing increasingly complex features. This is done depending on the complexities within the data examples. The output of this initial part of a CNN is generally fed into Fully Connected Layers to perform machine learning tasks like classification on the newly extracted features. In Figure 2.4 an example of a CNN is reported.

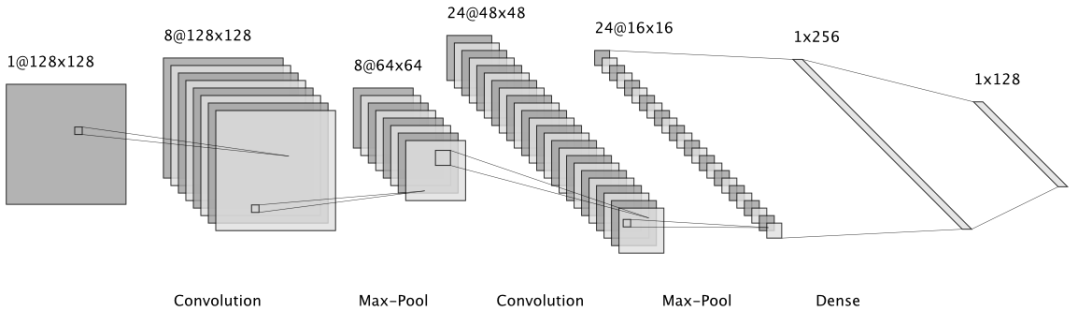


Figure 2.4 Example of Convolutional Neural Network
An example of a CNN with two Convolutional layers, each followed by a Max-Pooling layer and finally two Fully Connected layers. Filters sizes omitted for simplicity.

The advantages that Convolutional Neural Networks brought to the table, over FFNN and previous algorithms, were the abilities of learning directly from the data itself which were

the important features and of building a feature extractor for those as a Neural Network that can be trained at the same time as the classifier with Backpropagation.

2.4 Generative Adversarial Networks

Generative Adversarial Networks were first introduced as a framework to estimate generative models through an adversarial process by Goodfellow et al. [7] in 2014. This process is based on Game Theory and it constitutes a two player game in which a generative model is pitted against a discriminative one with the end goal of obtaining a model capable of producing new realistic examples. In more detail the two networks present in GAN are:

- Generator $G(\cdot)$: it is a generative model trained in order to create new realistic examples. It starts from an input noise vector z sampled from a probability space with density function p_z and produces a new vector $x_g = G(z)$. These random values are usually sampled from a Gaussian probability space.
- Discriminator $D(\cdot)$: it is a discriminative model tasked with the objective of estimating the probability that its input came from the training data rather than from $G(\cdot)$.

The two networks $G(\cdot)$ and $D(\cdot)$ are trained simultaneously through what is called Adversarial Training. $G(\cdot)$'s objective is to maximize the probability of $D(\cdot)$ making a mistake while $D(\cdot)$ is trained to maximize the probability of assigning the correct label to both fake, produced by the Generator, and real, coming from the training dataset, examples.
This procedure can be seen as a two-player minimax game:

- If the Discriminator successfully identifies real and fake samples it is rewarded, or no change is needed to the model parameters, whereas the Generator is penalized with large updates to model parameters.
- If the Generator succeeds in fooling the Discriminator it is rewarded, or no change is needed to the model parameters, but the Discriminator is penalized and its model parameters are updated.

The game ideally ends when the Generator produces samples that are close enough to the real thing from the input domain every time and the Discriminator cannot reliably tell the fake and real examples apart, predicting randomly each time.
So, Adversarial Training can be summarized as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[log\ D(x)] + \mathbb{E}_{z \sim p_z(z)}[log\ (1\ -\ D(x))] \tag{2.1}$$

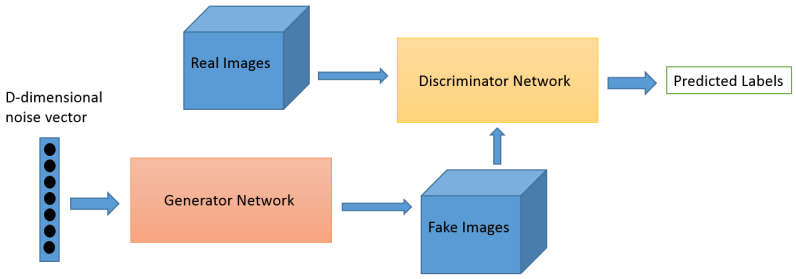


Figure 2.5 Generative Adversarial Networks [3]
High-level depiction of the Generative Adversarial Networks framework.

2.4.1 Conditioned Generative Adversarial Networks

An important extension of GAN are Conditioned GAN. In this setting both the Generator and the Discriminator networks are conditioned with some auxiliary information y. This approach has proven to be successful in guiding the generation of new examples like in [18] by Mirza et al. . Thus, the minimax game becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x|y)}[log\ D(x)] + \mathbb{E}_{z\sim p_z(z)}[log\ (1\ -\ D(x|y))] \tag{2.2}$$

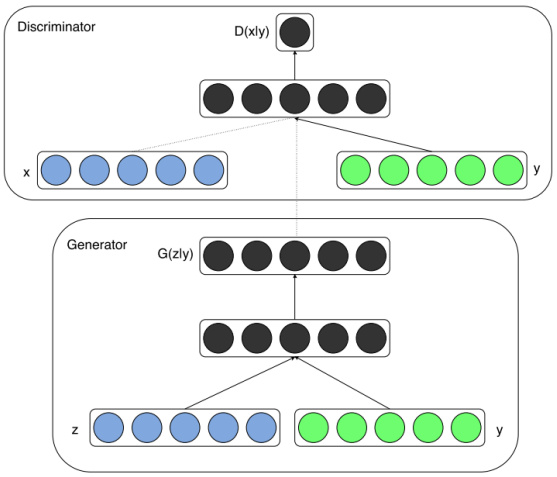


Figure 2.6 Conditional Generative Adversarial Networks
High-level depiction of the Conditional GAN framework by Mirza et al. [18].

[3] https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners

2.5 Irregular Time Series

An irregular time series is a sequence of observation in pairs (t_n, x_n) , with t_n being time instant and x_n a value, with a non-constant spacing between observation. This subset of time series is particularly important when modeling unexpected phenomena like equipment malfunctions or more extreme natural events like floods and earthquakes. A common approach to analyzing unevenly spaced time series is to transform this data into equally spaced observation with some interpolation process in order to apply existing algorithms to analyze them. Few methods currently exist designed specifically for analyzing unevenly spaced time series data untouched.

2.6 AUROC

AUROC is a performance index used in classification problems and Information Retrieval systems evaluation. It combines the sensitivity and specificity of a model into a single metric. It tells how much a model is capable of distinguishing between classes, the higher the better. It is computed starting from the Receiver Operating Characteristic curve (ROC) which is plotted with True Positive Rate (TPR) against False Positive Rate (FPR).
In order to do so we first report the building blocks for it and, for the sake of this explanation, the two classes will be '0' and '1':

True Positive: an example is predicted as '1' and the actual class is '1'
False Positive: an example is predicted as '1' and the actual class is '0'
True Negative: an example is predicted as '0' and the actual class is '0'
False Negative: an example is predicted as '0' and the actual class is '1'

$$TPR = Sensitivity = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives} \tag{2.3}$$

$$Specificity = \frac{False\ Positives}{True\ Negatives\ +\ False\ Positives} \tag{2.4}$$

$$FPR = 1 - Specificity = \frac{True\ Negatives}{True\ Negatives\ +\ False\ Positives} \tag{2.5}$$

After computing these indices, AUROC represents the area under the ROC.

Chapter 3

Related Work

In this chapter we'll be going through the existing strategies and methodologies revolving around irregularly sampled time series and existing techniques to fill in missing data for time series in general. Finally, we report some of the recent applications of Generative Adversarial Networks.

3.1 Missing values in time series data

Time series data may well present missing values due to problems in reading or recording data. This largely affects Machine Learning algorithms performance. Depending on whether the time series are multivariate or univariate some existing algorithms can or can't be applied. For multivariate time series we can rely on inter-attribute correlations while in the case of univariate time series the only dependency at our disposal is time. Other factors that can shift the attention from an approach to another one are the presence of trends and seasonalities in the data. Broadly speaking, to fill these gaps in our data different strategies can adopted starting from more general methods like mean, meadian, mode and random sample imputation to time series specific ones. However these ways of doing imputation usually lead to poor results in the case of univariate time series since the information regarding time is not used. Time series specific imputation techniques include Last Observation Carried Forward (LOCF), Next Observation Carried Backward (NOCB), linear interpolation, polynomial interpolation and Spline interpolation. Multivariate algorithms can sometimes be applied to the univariate case via the use of lags and leads which provide the time information as covariates. Lags are variables taking values from other ones from the previous time step. Leads function the same way but take from the next time step. The literature also provides us with more sophisticated techniques like Kalman Smoothing applied to the state space representation of an ARIMA model [20].

3.2 Dealing with irregularly sampled time series

In some scenarios, due to the nature of the phenomenon at hand, it is not feasible to use as a representation of the events a regularly spaced time series. For example, measurements in observational astronomy are subject to weather conditions, time availability and planetary configurations. The problem here is that a 'gold standard' kind of method does not exist. More classical models like ARIMA and GARCH cannot be applied 'as-is' since they operate on regularly sampled data. The most common strategy in this cases is to apply some form of interpolation to transform those particular time series into equally spaced ones and only then make use of the existing analytical methods to analyze them. As in the case of missing data, linear or spline interpolation can be performed between adjacent points in time. Other possibilities involve smoothing techniques, simple ones like linear and quadratic smoothing or more complex ones such as moving averages, running lines and kernel-based approaches. One technique thought specifically for unevenly sampled time series is presented in [4]. In this paper an exponential smoothing approach is proposed to analyze an irregularly observed ARIMA(0,1,1) model.

3.3 Applications of Generative Adversarial Networks

Generative models as a whole have been exploited in many different applications to artificially generate images, videos and audio. They have been used for representation learning [24] [28], text-to-image synthesis [26], super resolution [16], style transfer [6] and video generation [30] just to name a few. Since the introduction of GAN by Goodfellow et al. [7] in 2014, a plethora of papers and variants have surfaced in the recent years. Focusing on time series we can find some applications and proposals. In [5] Hyland et al. sought to generate medical entire records via a GAN implementation based on Recurrent Neural Networks. In [19] a generative adversarial model aimed at the creation of classical music was proposed. Finally in [17] a strategy to impute missing values through a generative adversarial process has been proposed. These approaches are united by the fact that they use regularly sampled time series. On the subject of unevenly sampled time series the only available work, to the best of our knowledge, is T-CGAN by Ramponi et al. [25]. Through their implementation of GAN they succeeded in reliably generating irregularly sampled and noisy time series. The two adversarial networks are conditioned with the timestamps of the different observations, allowing this method to operate without altering the time series.

Chapter 4

Methodology

In this chapter the methods we propose to address the problem of data filling for time series, introduced in the previous chapters, are presented in detail. To be able to do so we will be going through the main ideas, how these methods work in practice and finally how we evaluated the quality of the results.

4.1 Missing data and proposed solutions

Datasets of any kind can present missing values and it is not unusual at all to go through some form of pre-processing before using that data for any analysis. Depending on the chosen algorithm these blank spots in the data are treated differently. They might be ignored even if this can result in a degradation of the analyses results. The records that present them may be removed altogether, shrinking the overall size of the dataset; this also hinders the performance of any model based on such data. Finally, they can be filled with some placeholder value, a constant a-priori value or a statistic of some kind like the mean value for that particular attribute. In the case of time series this means that the data will be discarded, padded or completed with values that may not be representative of the particular phenomenon depicted by the series.

In this thesis, three different generative strategies to undertake data filling tasks through the use of GAN are proposed.

- T-CGAN: Time Conditional GAN, extension of the method proposed in [25] based on CNNs
- TA-CGAN: Time Aligned Conditional GAN, an alternative CNN-based approach aimed at studying the ability to detect the relationship between a value and its observation time instant

- LSTM GAN: the third and final proposed approach is based on Long-Short Term Memory (LSTMs) networks, a popular variant of Recurrent Neural Networks (RNNs)

These approaches are aimed at completing irregularly sampled time series, allowing us to bypass all the problems just described above.

For what concerns the third method we also compare it to the solution proposed in [25] for generating complete synthetic sine and sawtooth time series. Finally we compare its performance with our two previous proposals for data filling both for synthetic and real timeseries.

We will describe in detail how the different architectures have been implemented in Chapter 5.

4.2 Baseline Definition

In Chapter 2 we discussed about how GAN are capable of producing realistic examples by learning the underlying probability distribution of the training data and how the generation process can be guided by feeding the networks with auxiliary information about the expected output. When dealing with irregularly sampled time series, it is important to also provide the time instants in which a particular value has occurred. Related to this topic, in [25] Ramponi et al. illustrate how they were able to generate full-length, irregularly sampled, time series by conditioning both the Generator and the Discriminator networks with timestamps.

However in this case, as well as in their original formulation, GAN can't be applied 'as-is' in a setting where only part of a given example needs to be generated. This is mainly due to the fact that the Generator network has no knowledge regarding what parts of the input should be kept intact and what we want to it create.

To prove this we define as the *baseline model* for our approaches a GAN framework were none of the two networks are conditioned with any kind of extra information. In this model both Generator and Discriminator are CNNs. Then, parts of the generated curves are replaced with the real values we already know. We tested it against both synthetic periodic time series (sine and sawtooth waves) and real world time series from the dataset created by [29]. At the beginning of Chapter 6 a more in depth description of the datasets can be found.

4.3 Conditioning the Generator

In the previous section we stated that the Generator network has no way of knowing which parts of a time series it should keep and which should be created. So, our first attempt focuses on modifying the Generator in order to achieve such behavior.

Given our *baseline* results (reported later on) and past studies, we can see how important it is to feed into the Generator the information regarding time when working with irregular time series. Moreover, since our end goal is to fill time series that present missing values, we condition the Generator network with the parts we know of such series. These values are provided to the Generator with the objective of better letting the network know of the trend and type of signal that it is expected to be generated.

In this setting we can represent a generic input example for the Generator as a vector with the following structure

$$[z_1 \cdots z_n | y_1 \cdots y_n | x_k \cdots x_m]$$

The three main components are

- $z_1 \cdots z_n$: values sampled from the Noise space Z
- $y_1 \cdots y_n$: timestamps from the time window $Y = [y_{start}, y_{end}]$
- $x_k \cdots x_m$: known values from a time series t_i

On the other hand the Discriminator, alongside the generated or real time series, is conditioned only with the timestamps. As a result a single input example for this network is structured like

$$[a_1 \cdots a_n | y_1 \cdots y_n]$$

In which the components are

- $a_1 \cdots a_n$: either the values $[x_1 \cdots x_n]$ from a real time series t_i or the values $[g_1 \cdots g_n]$ from one produced by the Generator
- $y_1 \cdots y_n$: timestamps from the time window $Y = [y_{start}, y_{end}]$

Since the information the network is conditioned with is meant as a help to it during the training phase, when a new example is produced, the values we know of the time series are used to overwrite the ones coming from the Generator. This additional step allows both the Generator and the Discriminator to focus only on the parts that are missing from the series without compromising on the variety or quality of the produced output.

A high level representation of the method workflow discussed in this section can be seen in Figure 4.1.

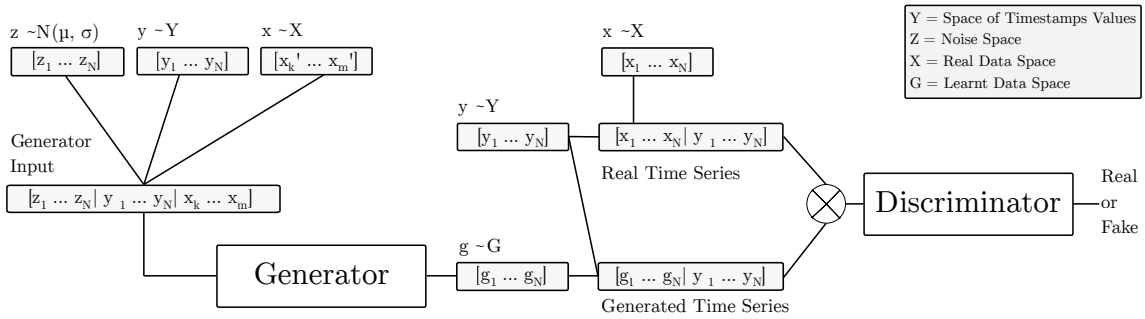


Figure 4.1 T-CGAN method

Graphical representation of the method described in this section. It shows the different input components previously mentioned and how they are combined to create the inputs for the two networks. The size of the batch is not reported for simplicity.

This first method exploits Convolutional Neural Networks and in Figure 4.2 the complete architectures for the Generator and Discriminator models are reported. In this case the two networks are really similar to the T-CGAN one [25] with the exception of the conditioning on the time series values. The Generator model is fully convolutional and performs Transposed Convolutions, or Deconvolutions [31], to be exact. Input tensors are treated as if they were 1×1 images with $s + s + c$ channels where s is the timeseries length and c is the conditioning length. On the other hand, the Discriminator model is a fairly common Convolutional Neural Network that outputs the logits scores for real and generated timeseries. The actual class probabilities are computed when evaluating the losses of the generative and discriminative models, as shown in Chapter 2. In the Generator the number of filters depends on the length of the time series so that each single examples is slowly upscaled to the correct length. Batch Normalization [13] is applied in the Generator network to help with stability during training. Adam [14] is the optimizer of choice for the two networks, used with its default parameters.

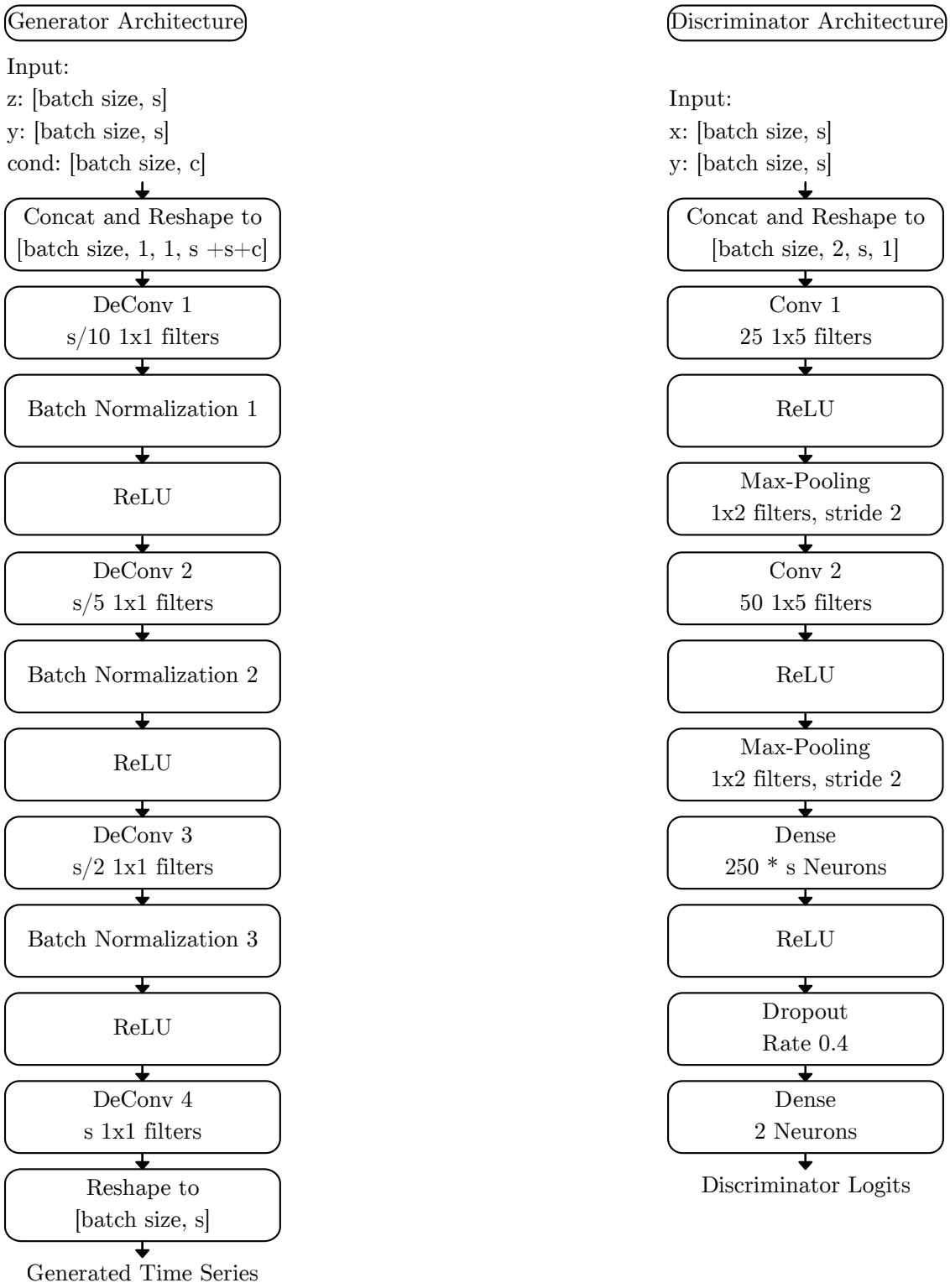


Figure 4.2 T-CGAN architecture

Representation of the generative and discriminative networks structures for the approach described in this section. The inputs, the number and sizes of convolutional and max-pooling filters and the used activation functions are reported.

4.4 Learning the relationship between values and time

One question that can arise given the scenario described in the preceding section is: how the network can understand at which time instant a value occurs? On this matter we'll be now describing a second method aimed at answering this exact question.

In this second approach the Generator still receives as part of the conditioning the timestamps from the whole time window Y but the known values from a real time series t_i are paired together with their respective timestamps. The main goal here is to try to exploit the filers in the convolutional layers in order to capture the relationship that is present between values and timestamps, particularly for the known values we condition the Generator with. This means that the input vector, in the case of a single example, for the network now becomes

$$\begin{bmatrix} z_1 \cdots z_n | x_k \cdots x_m \\ y_1 \cdots y_n | y'_k \cdots y'_m \end{bmatrix}$$

The three vectors $(z_1 \cdots z_n)$, $(y_1 \cdots y_n)$ and $(x_k \cdots x_m)$ have the same meaning as in the description given in Section 4.3 while $(y'_k \cdots y'_m)$ are the timestamps for the real observations $(x_k \cdots x_m)$. As explained in the preceding section, the generated time series are overwritten with the values that are known from the time series in the corresponding time instants. In this variation, the Discriminator structure and its input are unaltered with respect to the definition given in the prior section. A high level representation of the alternative method discussed in this section can be seen in Figure 4.3.

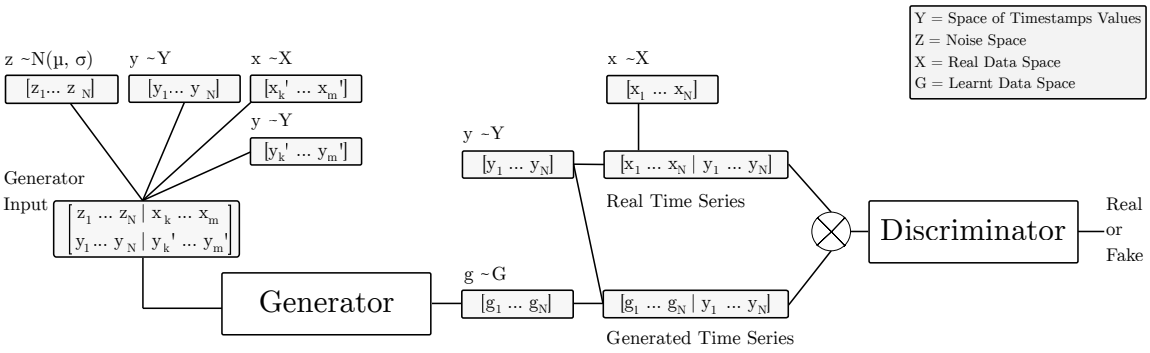


Figure 4.3 TA-CGAN method

Graphical representation of the method described in this section. It shows the different input components previously mentioned and how they are combined to create the inputs for the two networks. In this particular diagram the size of the batch is not reported for simplicity.

The idea here is to treat noise values z_i, timestamps y_i and timeseries values x_i used to condition the generator as if they were part of a single $2 \times (s + c)$ image where s is the

length of the timeseries and c is the length of the conditioning. Differently from the first approach, the hypothetical images have only 1 channel. As before the Generator is fully convolutional, performing Deconvolutions, and the Discriminator is a Convolutional Neural Network.

The architectures for the networks in this scenario are slightly different from the ones of the previous method. Most of the changes have been issued by taking inspiration from the talk *How to train a GAN?* held during the workshop *Adversarial Training* at NIPS 2016 [3] where a series of heuristics to stabilize GAN were illustrated.

Both networks make use of Leaky ReLU instead of ReLU in order to obtain softer gradients. Another change that helps with the stability of our method is the substitution of Max-Pooling layers with Avg-Pooling layers in the Discriminator. The Deconvolutional layers in the Generator, as we move deeper in the network, get fewer and fewer but grow in length so that we slowly upscale to the correct time series length. Similarly to the first method, Batch Normalization is used here too for the Generator and Adam is the optimizer in place for both networks.

The complete architectures are shown in Figure 4.4.

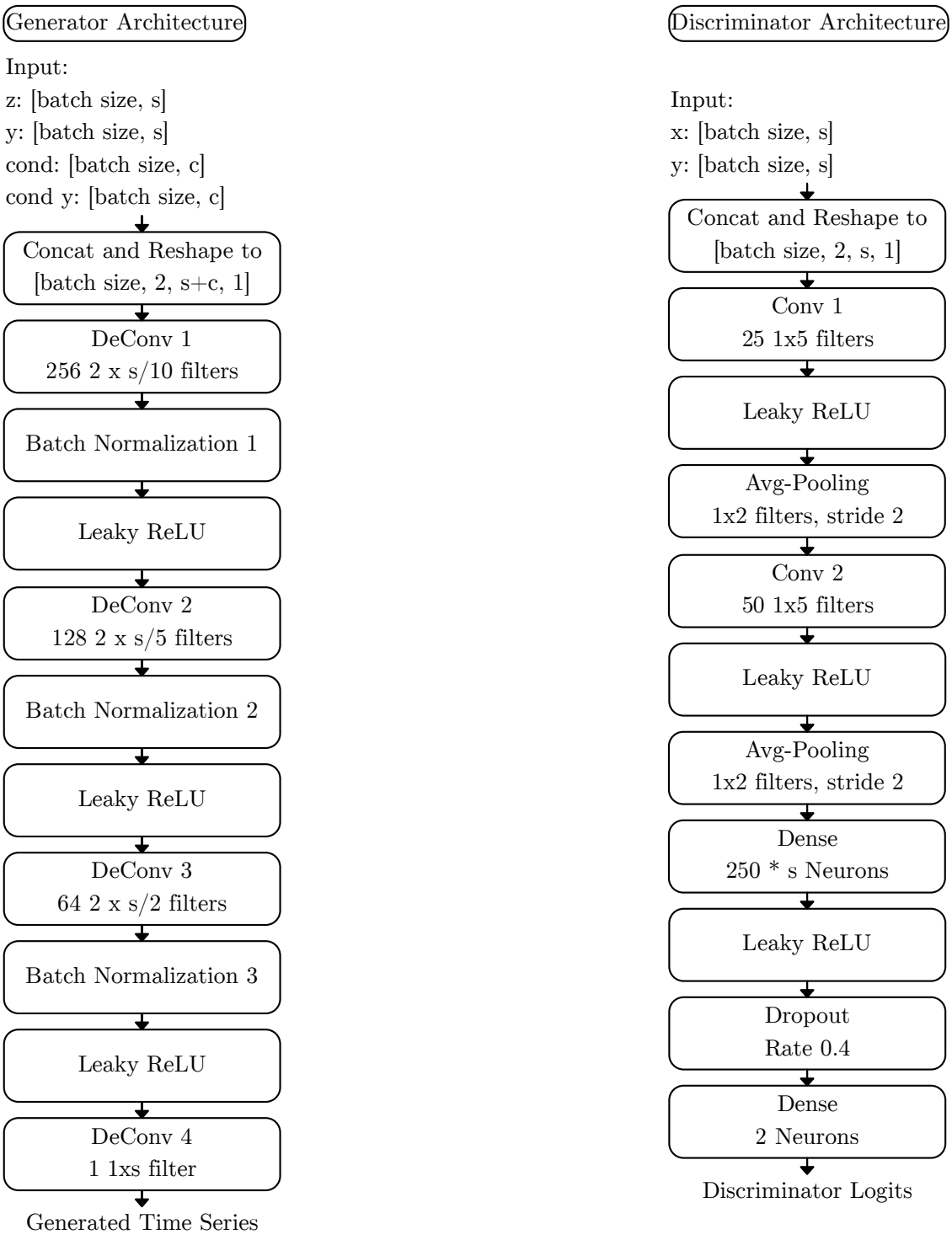


Figure 4.4 TA-CGAN architecture

Representation of the generative and discriminative networks structures for the approach described in this section. The inputs, number and sizes of convolutional and avg-pooling filters and activation functions used are reported.

4.5 Recurrent Neural Networks

The two previous implementations were based on Convolutional Networks. The final implementation that we propose is based on Recurrent Neural Networks (RNNs). This family of neural networks are widely used in scenarios in which the data is sequential: time series, text classification and text generation are just some examples. In our specific implementation we built the Generator and the Discriminator as two LSTM networks.

However the LSTM architecture, in its basic formulation, presents a substantial limitation: the time interval between any two observations of a sequence is assumed to be of constant value. If we were to fed irregular time series inside these networks they would in turn produce sub-par results.

To solve this problem we considered the default structure of a LSTM cell and modified it by adding a new gate dedicated to accepting the information about time. Differently from the techniques in 4.3 and 4.4 we didn't provide as part of the input the timestamps but rather the times elapsed between pairs of timestamps. This was done to allow the networks to correctly discount the effects of values that occurred too far away or too close in time to the last observation. The way the time was incorporated into the architecture is in accordance with the following equations presented in [2]:

- Short-term memory: $C^S_{t-1} = tanh(W_d C_{t-1} + b_d)$
- Discounted short-term memory: $\hat{C}^S_{t-1} = C^S_{t-1} * g(\Delta t)$ where g(·) is a heuristic decaying function
- Long-term memory: $C^T_{t-1} = C_{t-1} - C^S_{t-1}$
- Adjusted previous memory: $C^*_{t-1} = C^T_{t-1} + \hat{C}^S_{t-1}$
- Current memory: $C_t = f_t * C^*_{t-1} + i_t * \tilde{C}$

The final behavior of this particular LSTM unit is obtained by combining these statements with the original formulation exhibited in Chapter 2.

Besides the elapsed times, in the same way as in Sections 4.3 and 4.4, the Generator is conditioned with values from the time series. Before proceeding further we need to define the different data units for this method:

- z_i : value sampled from the Noise space Z
- Δy_i : time delta, defined as $\Delta y_i = y_j - y_k$ with $j < k$; y_j and y_k are two timestamps from the time window $Y = [y_{start}, y_{end}]$

- b_i : conditioning values, they can be either a real value x_i from the time series t_i or a placeholder for a missing value.

Since RNNs as a whole consider the data sequentially it is important that, during each of the timesteps of the Recurrent Network, all the relevant factors illustrated above are provided correctly. Following the *Stack* strategy illustrated in [11], we can represent a generic input example for the Generator as:

$$\begin{bmatrix} \begin{bmatrix} z_1 \\ \Delta y_1 \\ b_1 \end{bmatrix} \cdots \begin{bmatrix} z_n \\ \Delta y_n \\ b_n \end{bmatrix} \end{bmatrix}$$

where the different components are considered as vectors with 3 features.
Likewise, a generic input examples for the Discriminator will look like:

$$\begin{bmatrix} \begin{bmatrix} z_1 \\ \Delta y_1 \end{bmatrix} \cdots \begin{bmatrix} z_n \\ \Delta y_n \end{bmatrix} \end{bmatrix}$$

A high level representation of the method based on LSTM Networks can be seen in Figure 4.5.

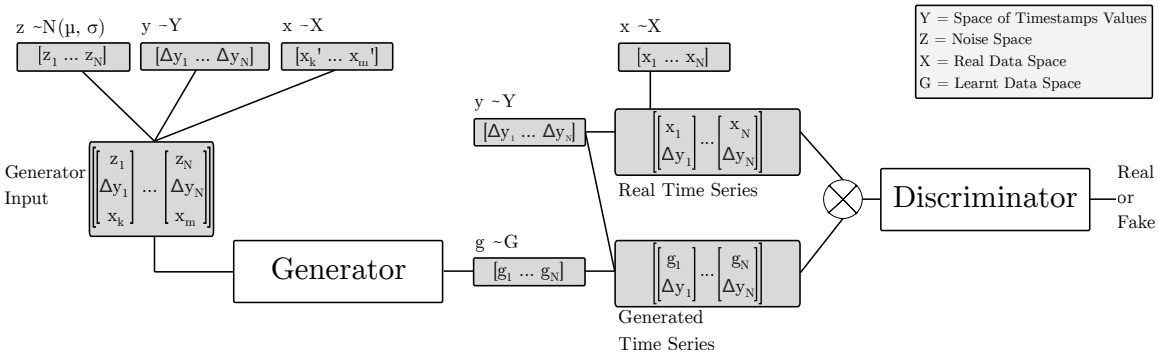


Figure 4.5 LSTM GAN method

Graphical representation of the method described in this section. It shows the different input components previously mentioned and how they are combined to create the inputs for the two networks. The size of the batch is not reported for simplicity.

The network architectures of both models, reported in Figure 4.6, for this third and final method are comprised of a single repeating LSTM cell with 100 hidden units. In the case of the Generator the LSTM cell is followed by a Tanh layer while for the Discriminator we have a Sigmoid layer.

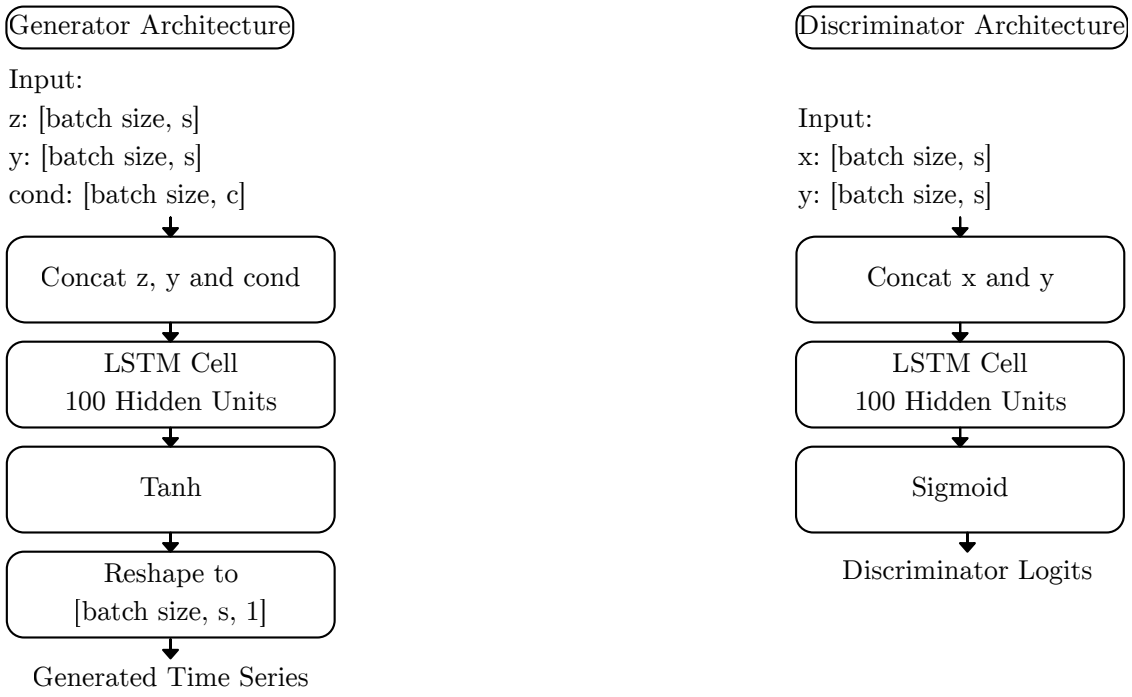


Figure 4.6 LSTM GAN architecture

Representation of the generative and discriminative networks structures for the approach described in this section. The inputs, number and sizes of convolutional and avg-pooling filters and activation functions used are reported.

4.6 Evaluating the generated examples

On the matter of evaluating the performance of GAN there is no agreed upon way to evaluate the Generator model nor to compare two trained models. Applications of indices and algorithms like Inception Score (IS) [28], Frechét Inception Distance (FID) [10] or Maximum Mean Discrepancy (MMD) [8] can be found throughout the literature to estimate the generative capabilities of the models, namely, quality (especially in the case of images) and diversity of the produced samples.

The procedure we use to evaluate the quality of the samples was introduced by [5] and it is called *Train on Synthetic, Test on Real*. This technique is based on the idea that the produced samples by the GAN are considered of good quality if a classifier trained with them is successful when presented with real world data. This validates whether or not the data produced via GAN can be used for real application. This evaluation strategy poses an immediate advantage over other techniques by avoiding the shortcomings of Inception-based methods [1] and the expensive operations required to compute MMD.

The evaluation model we used is a simple CNN and it is tasked with classifying two classes at a time (binary classification) whose training examples come from the Generator network. Its performance has been quantified by the evaluation metric AUROC: Area Under the Receiver Operating Characteristic curve.

We have tested our methods with synthetic time series, periodic and non-periodic, as well as with time series coming from real world datasets SonyAIBORobotSurface [29] and Wafer [22]. In Chapter 6 we will present both quantitative and qualitative results that show that the methods proposed thus far are capable of completing time series regardless of which values are missing.

Chapter 5

Implementation

This chapter is structured along the same lines as Chapter 4 and the implementations of the methods introduced in it are described in detail. A brief explanation regarding development and deployment choices is included.

5.1 Development and Deployment

The entirety of the project was developed in Python, specifically, through the use of Jupyter notebooks. This was mainly due to abundance of deep learning libraries and frameworks implemented in such language. The code was entirely deployed on Google Colaboratory cloud environment. Colaboratory offers free access to NVIDIA Tesla K80 GPUs and Google's own TPUs allowing users to run their own Jupyter notebooks benefitting from processing boosts when compared to CPU computations. Although Colaboratory is meant for interactive use, we were able to build the Jupyter notebooks that were necessary to retrieve and pre-process the time series data, run the models implementing our methods as well as their performance evaluation routines.

The methods we propose required for each one the implementation of the two neural networks that would act as the Generator and the Discriminator and to do so our deep learning framework of choice has been TensorFlow.

TensorFlow, initially developed by Google for its own uses, is now an open source library that allows researchers to build and share models to be used to solve Machine Learning tasks like Classification, Prediction and more.

In the following section we will delve into how each method has been implemented more specifically.

5.2 Data Preparation

In the experiments that we are going to explain and illustrate the results of, we used synthetic datasets of periodic and non-periodic time series as well as real world datasets. In order to prepare the data for the different GAN implementations we relied heavily on the use of the library *NumPy*. *NumPy* is the fundamental package for scientific computing in Python. Through the use of this library we were able to analyze and partition the real world datasets according to our needs.

Another Python package we used is *signal*. Used in conjunction with *NumPy*, we created synthetic datasets of irregularly sampled and noisy time series like sine waves, sawtooth waves, parabolae and sine waves with their amplitude increasing over time.

5.3 Baseline

As a basis for the implementation of the baseline, which makes use of CNNs, we used as a starting point the model presented in [24]. Thanks to TensorFlow we defined the two networks following some of the architectural choices made in [24] but adapting them as necessary in order to work with time series. This let us collect the results that are going to be used as reference in Chapter 6.

5.4 CNN Based GANs

The first two methods, as we discussed in Chapter 4, exploit in both cases CNNs. We started implementing these techniques from the work done in [25]. In this setting too, TensorFlow APIs have proven to be crucial to the task of realizing our own architectures.

5.5 LSTM Based GAN

The final and third method makes use of LSTM Networks. This time what we did was to take the class implementing LSTM cells directly from the TensorFlow library and add our code to it. This allowed us to use the auxiliary information about time and known values of the time series to correctly condition the Generator and Discriminator of our GAN.
In the original implementation of a basic LSTM cell the weights are stored within a single tensor of shape `[input depth + h depth, 4 * num units]` where

- `input depth`: depth of the input tensor

- `h depth`: depth of the hidden state of the LSTM cell
- `num units`: number of hidden units of the LSTM cell

The first thing we did was to extend the 2nd dimension of the matrix to accommodate for the parameters of the neural network layer computing the influence of time. Then, adhering to the equations from [2] reported in Chapter 2 we added the following code:

```
# Short-term memory, t = w_t * C_t-1 + b_t
cell_short_term = tanh(t)

# Discounted short-term memory
cell_short_term_dis = tf.multiply(input_dtimes, cell_short_term)

# Adjusted previous memory
prev_cell = c_prev - cell_short_term + cell_short_term_dis
```

In the original paper [2] a function $g(\cdot)$ was used to alter the time intervals since the observations in the datasets they used were really far apart in time. In our experiments this particular function is not present since the elapsed times are quite small both in the case of synthetic and real time series. Finally the updated cell state and output are computed.

```
# New cell state
c = (sigmoid(f + self._forget_bias) * prev_cell + sigmoid(i) * tanh(j))

# New output
m = sigmoid(o) * tanh(c)
```

Chapter 6

Experiments and results

The goal of this chapter is to show how the methodologies and implementations discussed in the previous sections have been put to the test in various experiments. First of all, we will talk about the datasets used, both synthetic and real, and our baseline. Then, we will focus on the types of experiments undertaken, bringing examples, results and performance for each type. Finally, we will discuss and compare the results, in order to understand which of the methods implemented by us could be the best solution.

6.1 Datasets

Our experiments were carried out on both synthetic datasets and real datasets. As for the synthetic data, we have generated it with specific functions, which produce the signals according to their equations, using time spaces provided as inputs, generating a number of points of choice, adding a portion of noise to the signal and, more importantly, returning irregular sampled signals as outputs, in a time window from 0 to 12. On the other hand we got the real data from the internet [1] as .csv files and we made some pre-processing operations in order to feed it inside the networks.

6.1.1 Synthetic Periodic Time Series

A time series is defined periodic if it completes a pattern within a measurable time frame, called a period and repeats that pattern over identical subsequent periods. The completion of a full pattern is called a cycle. A period is defined as the amount of time required to complete one full cycle.

[1] http://www.timeseriesclassification.com

Sine wave Signals

Sine waves have been generated with an amplitude $A = 1$ and frequency $f = 1\ Hz$. In the examples in 6.1 we can see values exceeding the $[-1, 1]$ range due to the noise added to these time series. Noise values have been samples from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.2$.

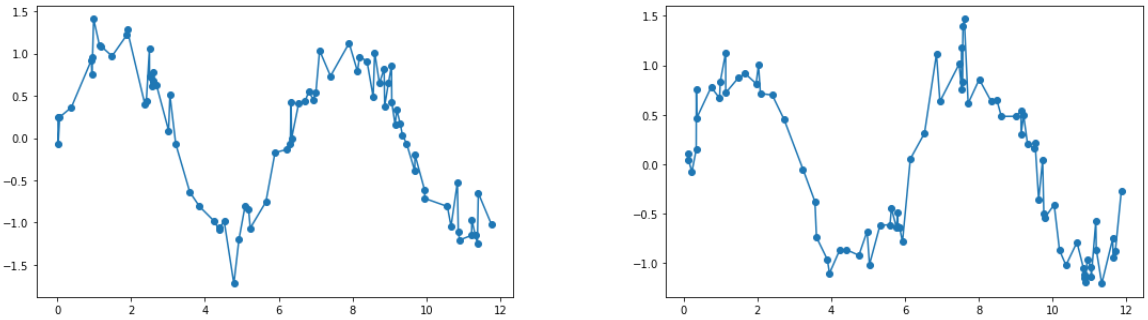


Figure 6.1 Examples of Sine wave signals

Sawtooth Signals

Sawtooth waves, similary to Sine waves, have been generated with an amplitude $A = 1$ and frequency $f = 5\ Hz$. In this case too noise has been added by sampling a Gaussian distribution with $\mu = 0$ and $\sigma = 0.2$.

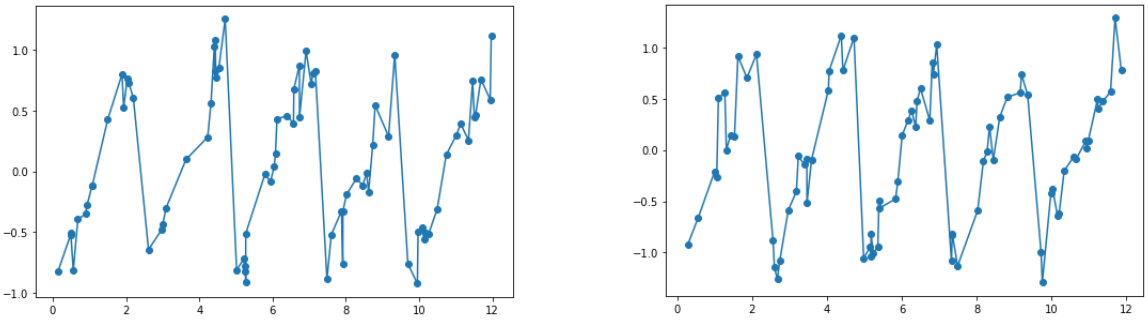


Figure 6.2 Examples of Sawtooth signals

6.1.2 Synthetic Not-Periodic Time Series

A time series which does not repeat itself after a specific interval of time is defined non-periodic. In our experiments we generated Sine waves with their amplitude A that increases over time and Parabolae.

Sine wave Signals With Increasing Amplitude

These signals have been generated starting from Sine waves with frequency $f = 1\ Hz$ and amplitude $A = 1$ which is then scaled proportionally to the time.

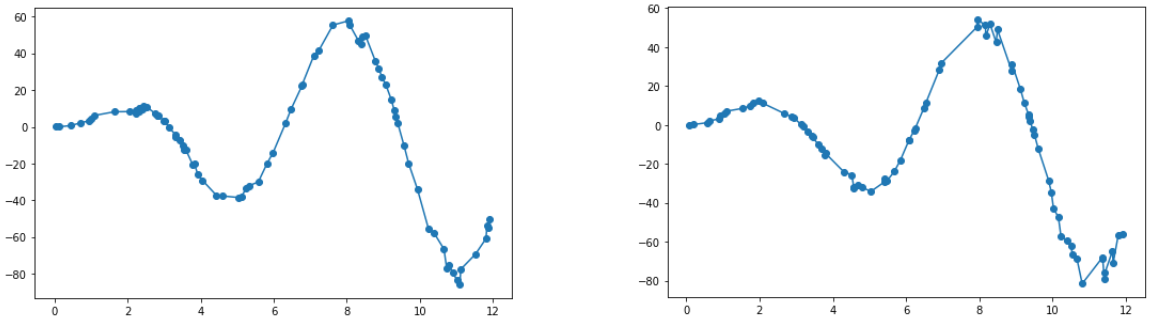


Figure 6.3 Examples of Sine wave signals with increasing amplitude over time

Parabola Signals

These time series have been generated following the canonical equation: $ax^2 + by + c = 0$ where $a \in [1,\ 10]$, $b = 0$ and $c = 0$. We added noise sampled from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.2$.

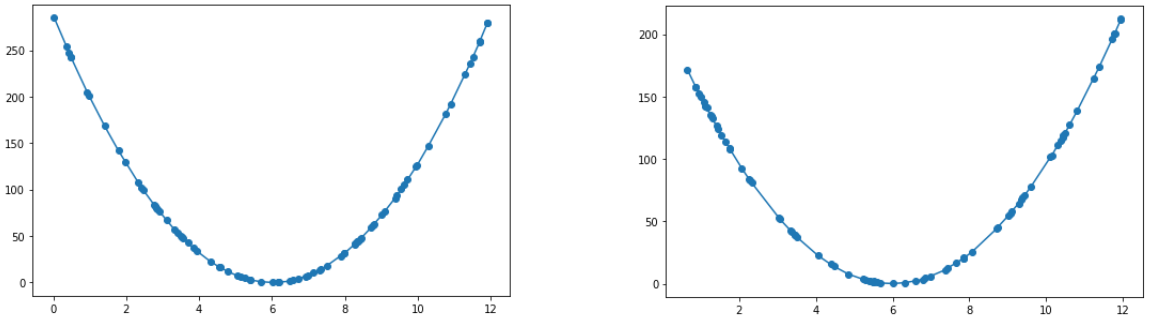


Figure 6.4 Examples of Parabolae

6.1.3 Real Time Series

To test the introduced methods, 2 real datasets have been selected: SonyAIBORobotSurface [29] and Wafer [22]. Before being able to use these time series for our experiments we applied some pre-processing steps. In both cases the datasets were already divided in Training and Test sets. However, what we needed was:

- A subset of time series to train the GAN

- A subset of incomplete time series to fill and use to train the fictitious classifier
- A final subset used as test set for such classifier to measure its performance

So, respecting the separation in classes, we redistributed the time series as follows.

SonyAIBORobotSurface

The SonyAIBORobotSurface dataset (also referred to as Sony from here on out) contains accelerometer measurements, specifically the X-axis readings, from the Sony AIBO robot. Each time series represents one walk cycle of this robot on 2 different surfaces, carpet and cement, denoting the separation in classes.

- Class 1
  - Training GAN: 150 time series
  - Completed by trained GAN: 100 time series
  - Testing Classifier: 126 time series
- Class 2
  - Training GAN: 200 time series
  - Completed by trained GAN: 200 time series
  - Testing Classifier: 204 time series

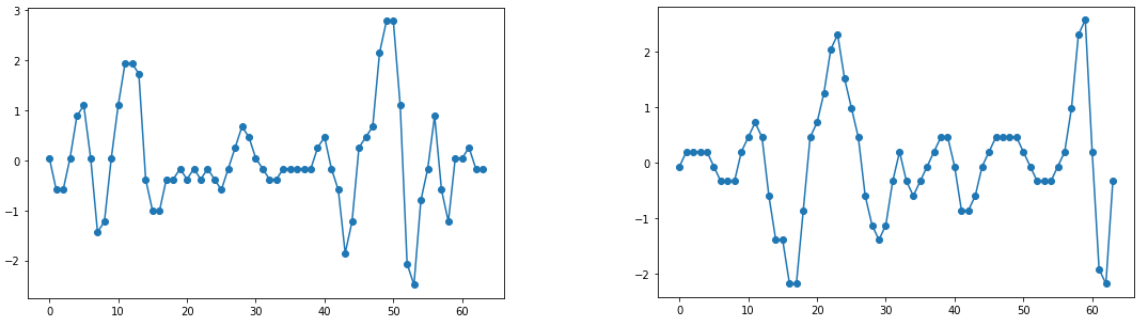


Figure 6.5 Examples from SonyAIBORobotSurface dataset

Time series examples taken from the SonyAIBORobotSurface dataset of both classes: *Class 1* (left) and *Class 2* (right).

Wafer

The Wafer dataset contains inline process control measurements for several sensors during the fabrication of silicon based semiconductors. The two classes are *positives* and *negatives*.

- Positives
  - Training GAN: 360 time series
  - Completed by trained GAN: 300 time series
  - Testing Classifier: 200 time series
- Negatives
  - Training GAN: 360 time series
  - Completed by trained GAN: 240 time series
  - Testing Classifier: 150 time series

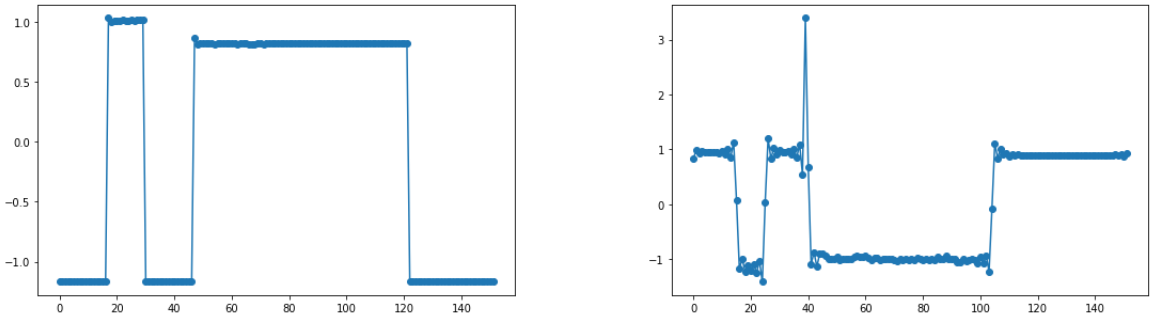


Figure 6.6 Examples from Wafer dataset

Time series examples taken from the Wafer dataset of both classes: *Positives* (left) and *Negatives* (right).

The time series belonging to these two datasets are all regularly sampled. So, before using them to evaluate our methods, we defined an evenly spaced time axis for each series and then added Gaussian noise ($\mu = 0$, $\sigma = 0.3$) to introduce the irregularities in time.

6.2 Experimental Strategy

In all the experiments we have carried out, we have used a common approach to allow a final comparison between the three developed methodologies. In showing data and collected

performance measurements, we will divide the experiments based on the part of input received as conditioning for the Generator network: initial - I, central - M, final - F. In all of these settings, given a time series of length X, we condition with $X/2$ values the generative model. So in the *initial* case the part of the curve given in conditioning will be the first half, in the *central* case it will be the middle part and in the *final* case the final half. For each experiments section we also show results divided by the number of points and number of curves reconstructed to better analyze if there is any kind of correlation as these two parameters vary.

Table 6.1 summarizes, for each method, what kind of experiments have been performed, on which data and under what conditions. In this table the data are precisely listed by row on (number of points, number of curves). These two parameters concern the classifier specifying, for each type of experiment, on how many curves it was trained and how long these were. The same holds for real world time series, reported in Table 6.2. We have divided the representation into another table since in the real data we have kept the number of curves and points unchanged. Each performance value was taken using the *Train on Synthetic, Test on Real* approach as described in Chapter 4.

Train Size,	Baseline		T-CGAN		TA-CGAN		LSTM GAN	
Points	P	NP	P	NP	P	NP	P	NP
40, 40	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
40, 80	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
40, 120	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
80, 40	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
80, 80	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
80, 120	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
120. 40	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
120, 80	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F
120, 120	I - M - F	-	I - M - F	I - M - F	I - M - F	I - M - F	C - I - M - F	I - M - F

Table 6.1 Summary of experimental settings on synthetic time series

- P: Periodic time series
- NP: Non periodic time series
- C: Complete generation, no conditioning on values
- I: Initial conditioning
- M: Central conditioning

- F: Final conditioning

Train Size, Points	SonyAIBORobotSurface Dataset		Wafer Dataset	
	Class 1	Class 2	Positives	Negatives
N. Time Series	100	200	240	300
N. Points	65	65	152	152
Experiments	I - M - F	I - M - F	I - M - F	I - M - F

Table 6.2 Summary of experimental settings on real world time series

For what concerns synthetic time series, all the included screenshots refer to the "80, 80" experimental setting. However regardless of the experiment, with the exception of the baseline, the implemented GAN have been trained with 4 times the number of time series with respect to the ones used to train the auxiliary classifier. For example, in the "80, 80" case this means that the GAN have been trained with 320 time series per class, 80 series per class are then reconstructed and used to train the classifier. Finally the classifier is tested against 80 synthetic series per class. On the other hand, for real time series, the sizes of training and test sets have been reported before in the respective dataset descriptions.

In conclusion, AUROC performance measures, detailed for each experiment in the tables that will follow, are the result of 5 different tests, averaged together.

6.3 Baseline

In this section you will find experimental data related to the baseline framework described in Section 4.2. Just to recall: a very simple implementation of GAN, without any conditioning information fed to the two models, no timestamps or no known values, was used. We performed experiments only on synthetic sine and sawtooth waves, in the "80, 80" scenario, and on a single real dataset, SonyAIBORobotSurface.

6.3.1 Experiments on Periodic Synthetic Time Series

In this section we will show the screenshot and AUROC measures collected during the experiments conducted on periodic synthetic data using the baseline GAN.

Reconstructing the second half

The original values are shown in blue and the generated ones in orange. We can see, at least visually, that the two curves do not join together perfectly. Despite this fact, the generated

values still follow faithfully the trend of the training examples. In the case of sawtooth waves we can see some instances in which the Generator has problems reproducing this simple time series.

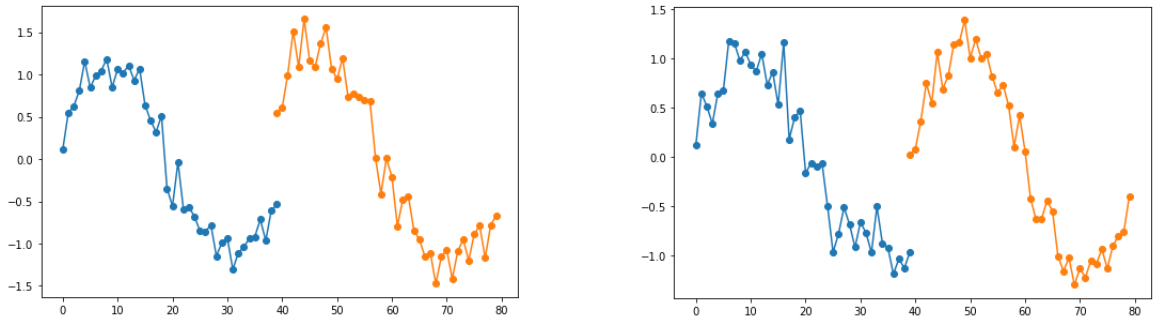


Figure 6.7 Baseline - Examples of Sine wave signals completed (second half)

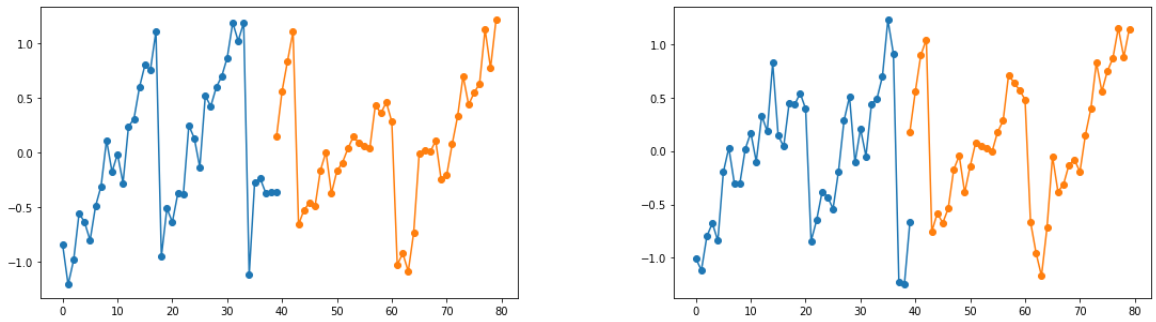


Figure 6.8 Baseline - Examples of Sawtooth signals completed (second half)

Reconstructing the initial and final ends

As in the previous case, the two curves do not come join together perfectly.

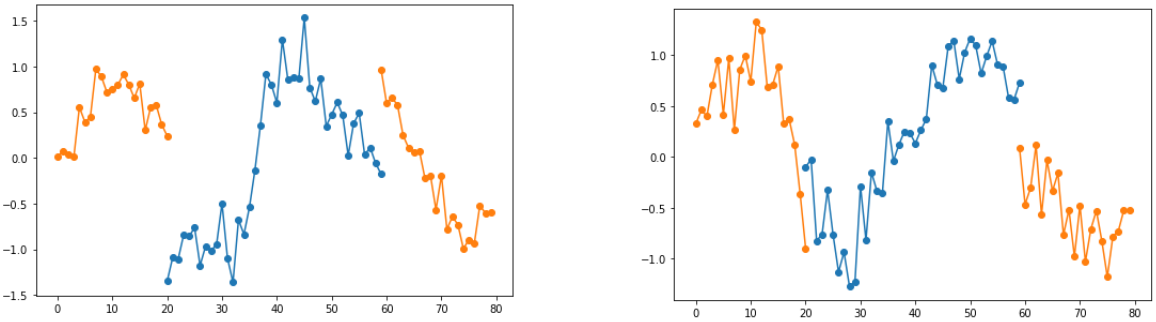


Figure 6.9 Baseline - Examples of Sine wave signals completed (initial and final ends)

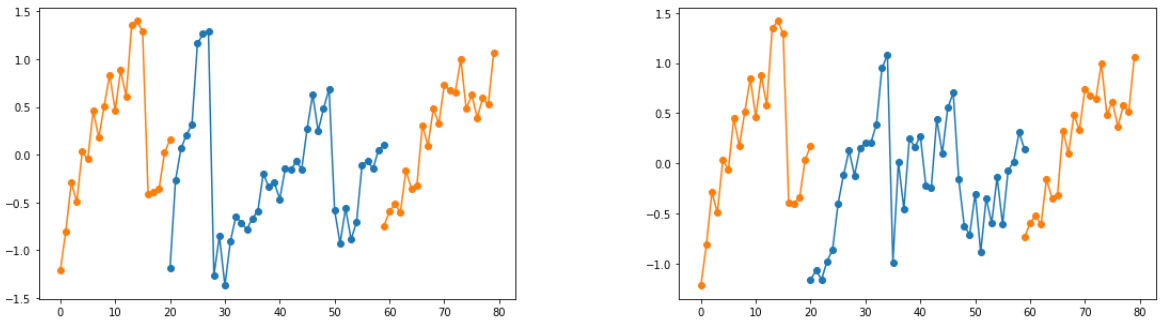


Figure 6.10 Baseline - Examples of Sawtooth signals completed (initial and final ends)

Also from these generated curves, it can be seen that the baseline model still struggles with generating the missing values in a reliably manner.

Reconstructing the first half

To conclude, by conditioning with the final part things do not change.

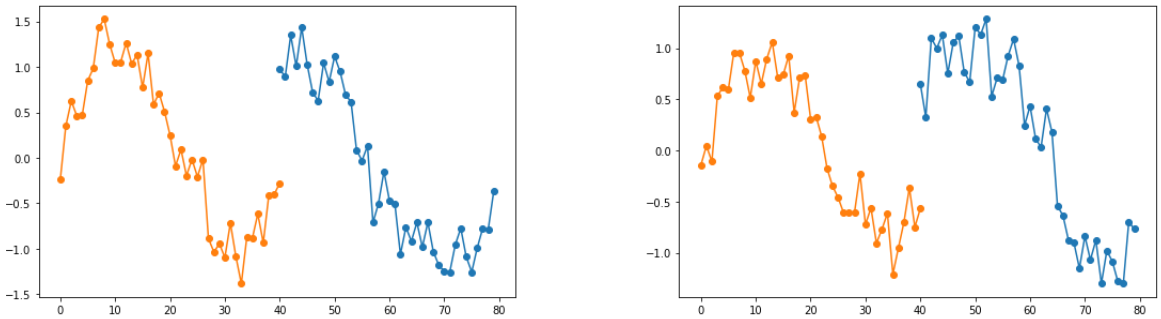


Figure 6.11 Baseline - Examples of Sine wave signals completed (first half)

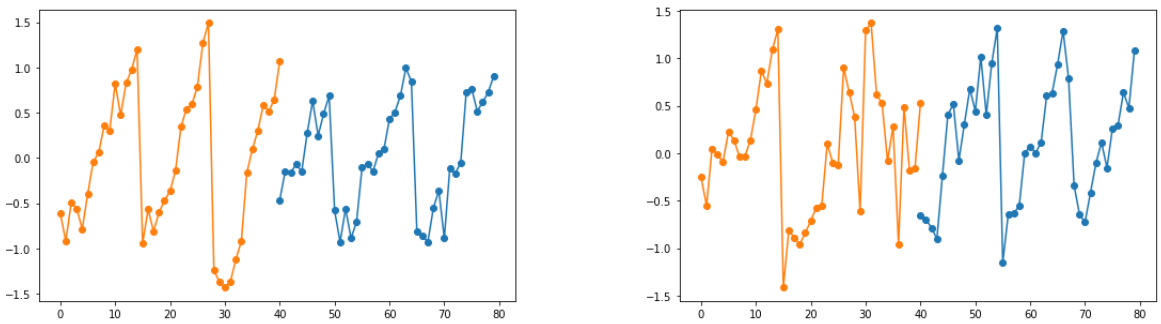


Figure 6.12 Baseline - Examples of Sawtooth signals completed (first half)

The results of this scenario are basically the reflection of what we have seen in the case of Initial Conditioning: the trend may be respected but the joining point cannot be accurately located.

Performance

The performance were obtained from the classifier by comparing Sine wave and sawtooth signals as opposite classes. Since measurements of AUROC between 0.80 - 0.90 are considered good and measures between 0.90 - 1 are considered excellent. In spite of the noticeable gap between the real and generated components of the time series, from the point of view of the classifier we can still obtain acceptable classification capabilities.

Initial	Central	Final
0.92375	0.93	0.925

Table 6.3 AUROC - Baseline - Periodic Synthetic Time Series

6.3.2 Experiments on Real Time Series

For the baseline model experiments on real data were performed only on the SonyAIBORobotSurface dataset.
As before the blue colored values are the real ones while the orange ones are produced by the Generator.

Reconstructing the second half

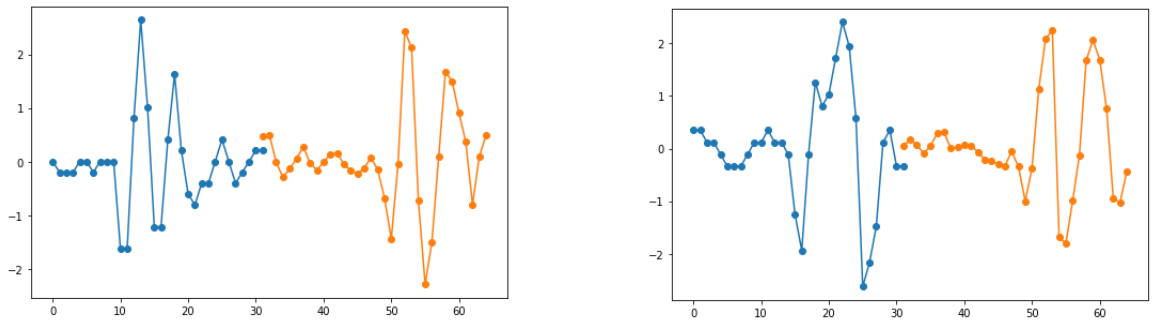


Figure 6.13 Baseline - Examples of Sony 'Class 1' signals completed (second half)

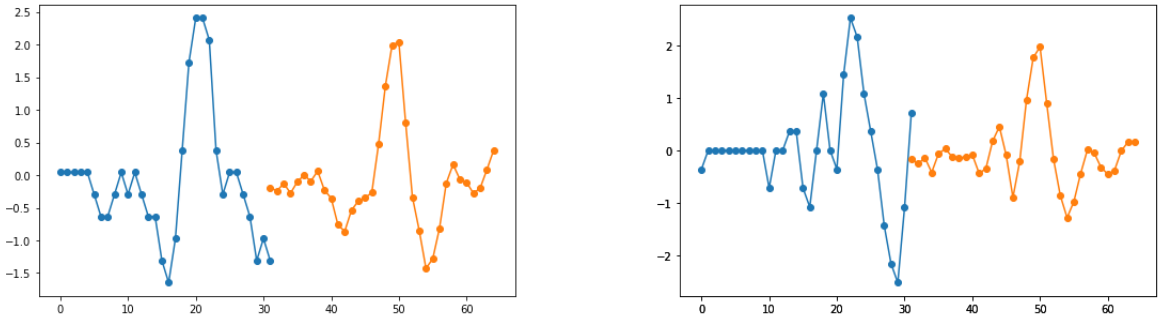


Figure 6.14 Baseline - Examples of Sony 'Class 2' signals completed (second half)

Reconstructing the initial and final ends

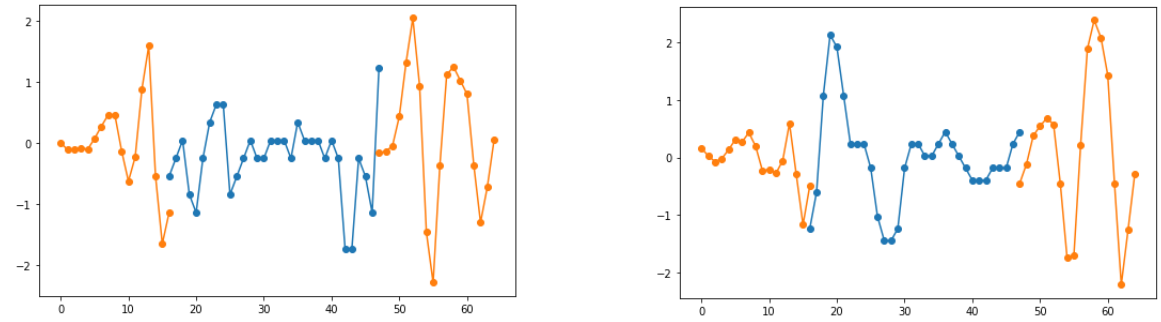


Figure 6.15 Baseline - Examples of Sony 'Class 1' signals completed (initial and final ends)

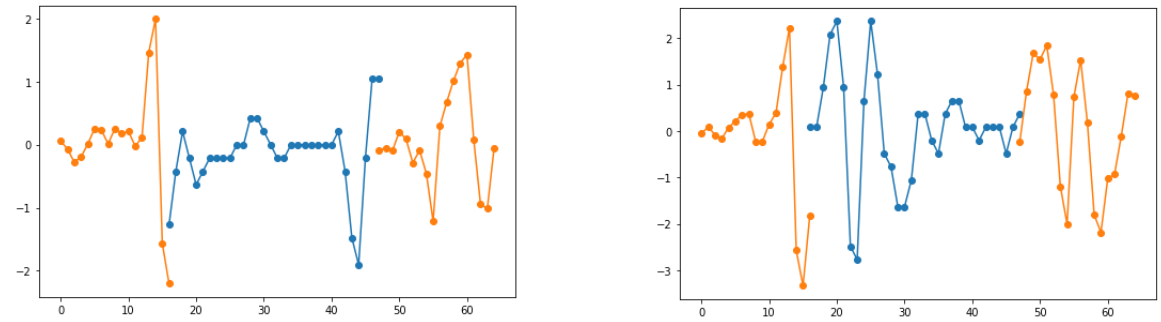


Figure 6.16 Baseline - Examples of Sony 'Class 2' signals completed (initial and final ends)

Reconstructing the first half

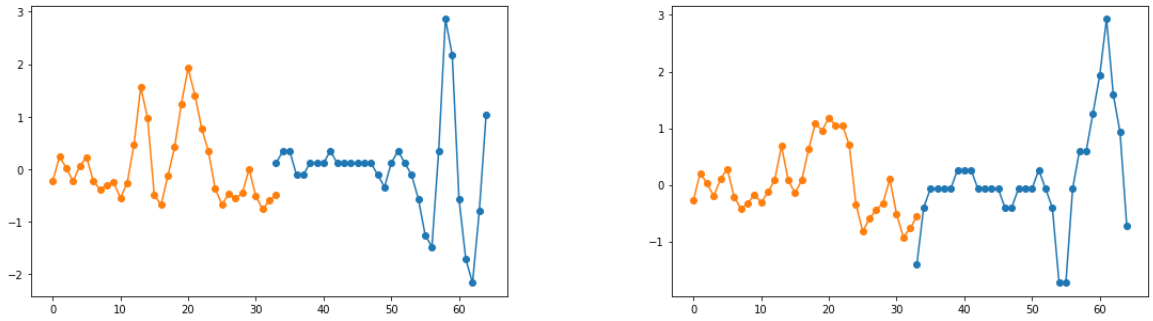


Figure 6.17 Baseline - Examples of Sony 'Class 1' signals completed (first half)

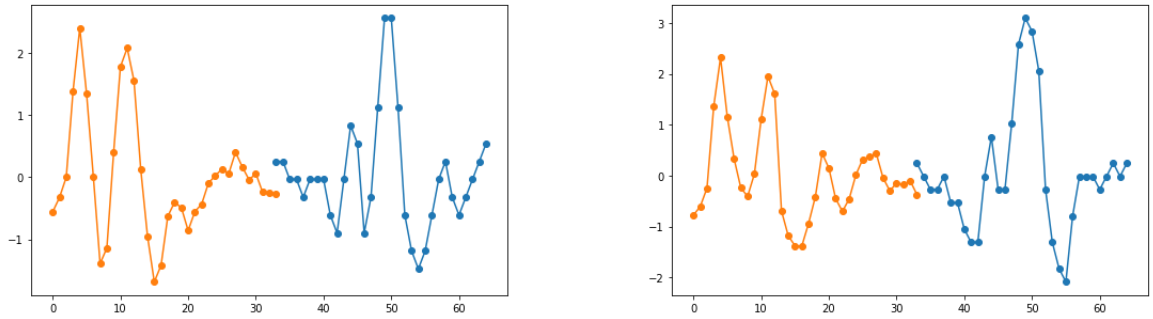


Figure 6.18 Baseline - Examples of Sony 'Class 2' signals completed (first half)

In this series of experiments on real data we can see that, compared to synthetic data, the result does not change. The junction point between the two halves of the curves remains very distant, although still maintaining the trend in the generated part.

Performance

When using real data we still have problems when it comes down to the point of connection between real and generated values. Generally speaking real data are less predictable than the periodic signals we used so, as expected, we are confronted with a degradation in the performance of the classifier.

Initial	Central	Final
0.799394	0.790606	0.8018

Table 6.4 AUROC - Baseline - Sony Time Series

6.4 Time Conditional GAN

This section deals with the experiments conducted using the first method we propose, as explained in detail in Section 4.3. Since the experiments carried out on the baseline result in maintaining the trend in the generated part without correctly joining the half in the conditioned half generated, we decided to implement the first version of GAN, to verify if the junction could be improved.
As before, in this series of experiments too the conditioning part is the blue one while the orange part is generated by the GAN.

6.4.1 Experiments on Periodic Synthetic Time Series

Initial Conditioning

The part given as conditioning to the Generator G is the first half of the time series, asking the network to generate the second half.

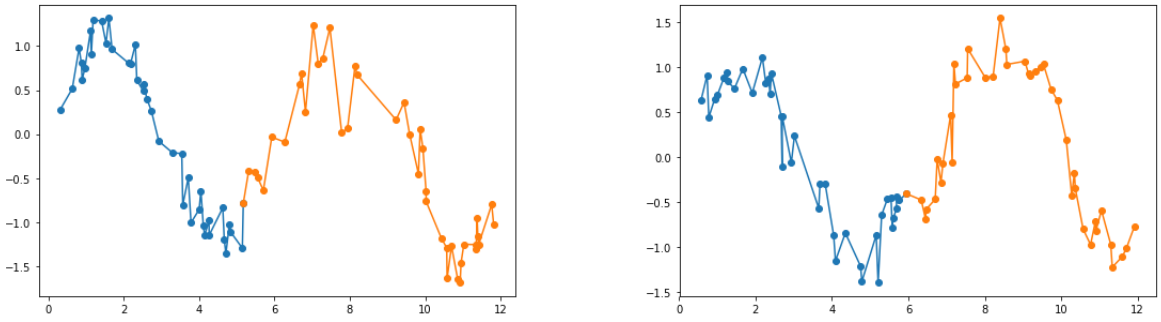


Figure 6.19 T-CGAN - Examples of Sine wave signals completed (second half)

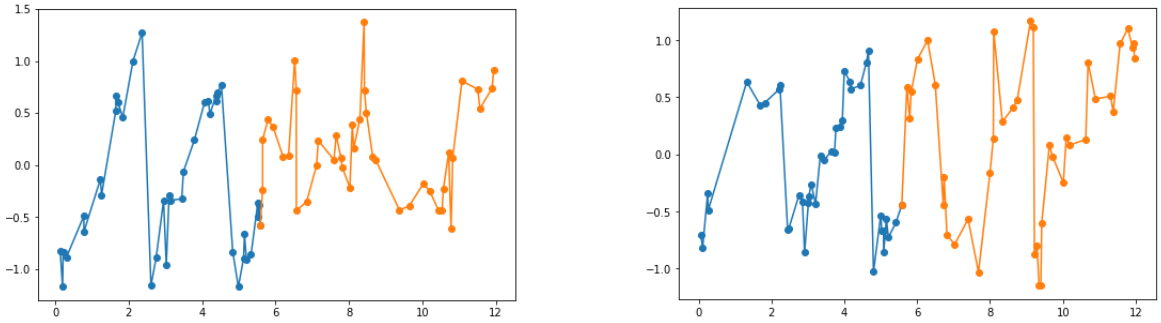


Figure 6.20 T-CGAN - Examples of Sawtooth signals completed (second half)

Central Conditioning

The part given as conditioning was only the central slice of the curve, asking the network to generate the first and the last quarter.

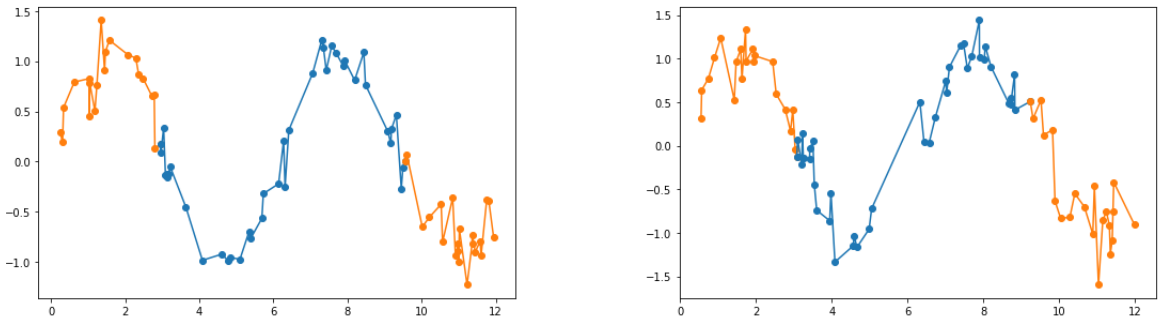


Figure 6.21 T-CGAN - Examples of Sine wave signals completed (initial and final ends)

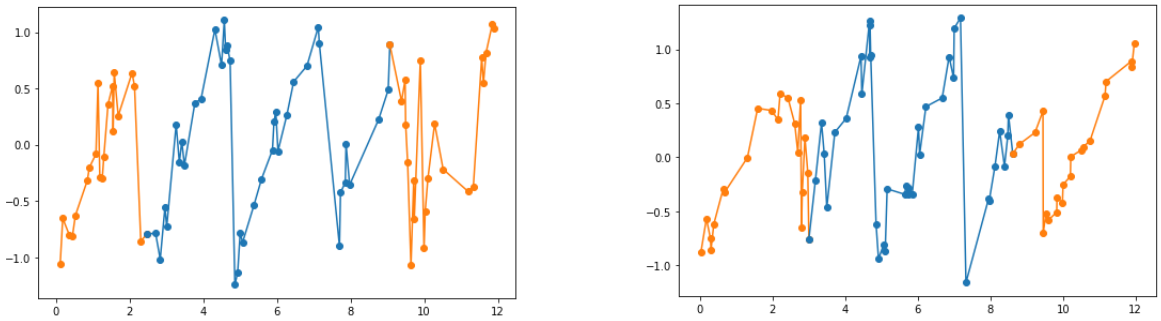


Figure 6.22 T-CGAN - Examples of Sawtooth signals completed (initial and final ends)

Final Conditioning

The part given as conditioning was only the final half of the curve, asking the network to generate the first half.

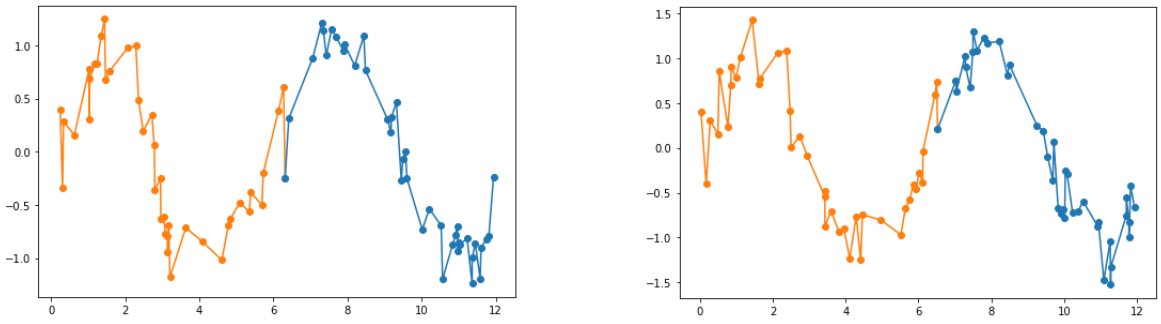


Figure 6.23 T-CGAN - Examples of Sine wave signals completed (first half)

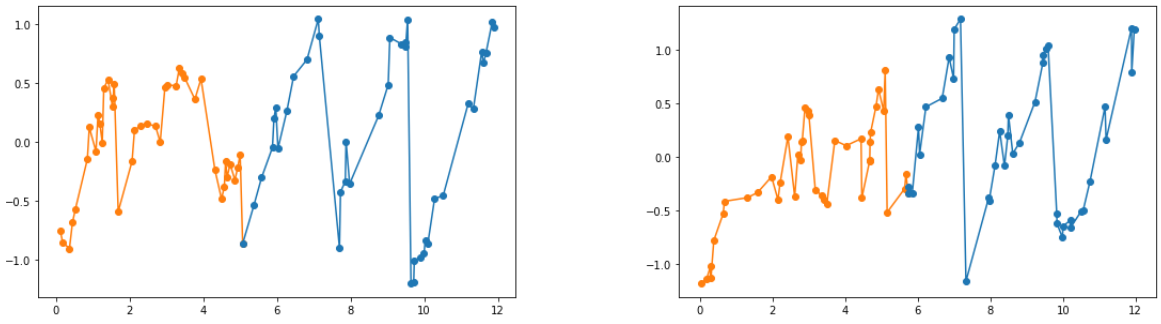


Figure 6.24 T-CGAN - Examples of Sawtooth signals completed (first half)

Performance

Right off the bat, when compared with the baseline GAN, we can see that the generated values start either right where the real values stopped being available or in close proximity. Like in the baseline model the general trend of these signals is captured. When it comes down to reproducing sawtooth waves, although the trend is generally respected, the quality of the time series is visually worse with respect to sine waves. Tables 6.5, 6.6 and 6.7 outline improvements in classification performance, regardless of the number of curves and number of values, when using the completed time series. Having AUROC indices close to 1 means, not only visually but also quantitatively, that this first technique is able to fill the missing values consistently better than the baseline.

	Time Series Length		
Training Set Size	40	80	120
40	0.99500	0.99750	1.00000
80	0.99250	0.99875	1.00000
120	0.99500	1.00000	1.00000

Table 6.5 AUROC - T-CGAN - Periodic Synthetic Time Series - Initial Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0.99500	1.00000	1.00000
80	0.99250	0.99750	1.00000
120	0.98417	1.00000	1.00000

Table 6.6 AUROC - T-CGAN - Periodic Synthetic Time Series - Central Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0.99500	0.99750	1.00000
80	0.99875	0.99750	1.00000
120	0.99083	1.00000	1.00000

Table 6.7 AUROC - T-CGAN - Periodic Synthetic Time Series - Final Conditioning

6.4.2 Experiments on Non Periodic Synthetic Time Series

Given these results, we decided to investigate further, by conducting experiments on non-periodic synthetic signals. Always in blue the conditioned part while in orange the generated part.

Initial Conditioning

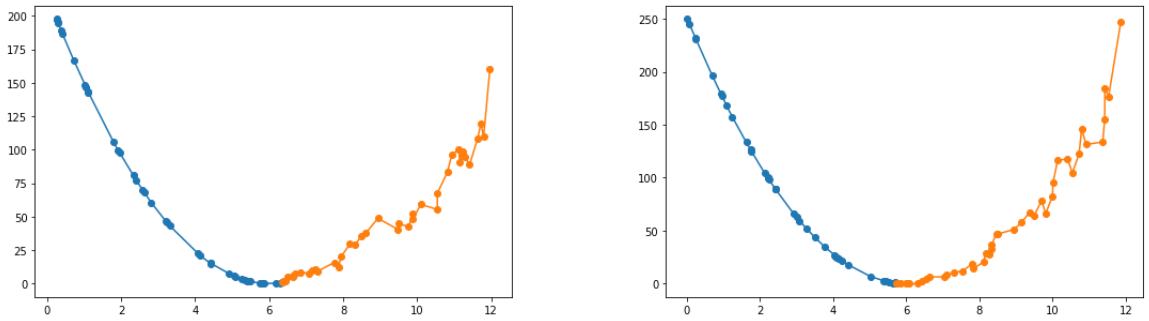


Figure 6.25 T-CGAN - Examples of Parabolae completed (second half)

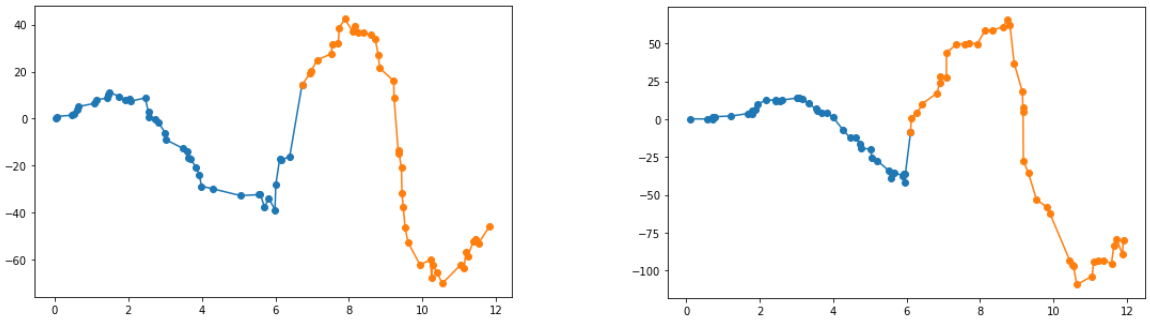


Figure 6.26 T-CGAN - Examples of Sine wave signals with increasing amplitude completed (second half)

Central Conditioning

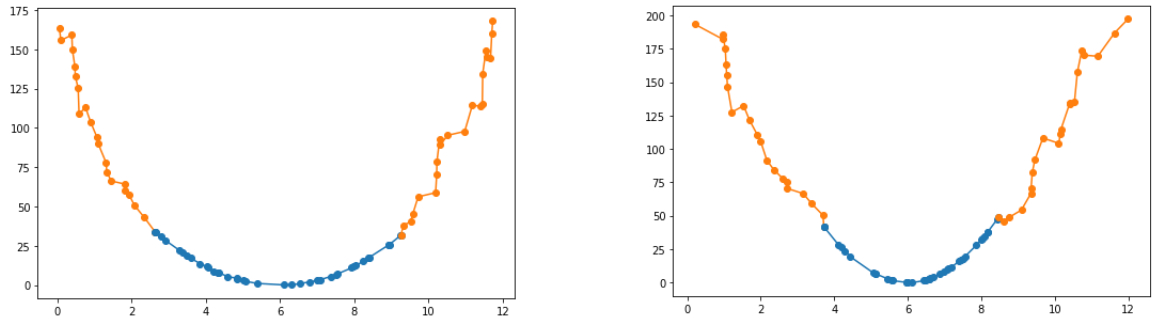


Figure 6.27 T-CGAN - Examples of Parabolae completed (initial and final ends)

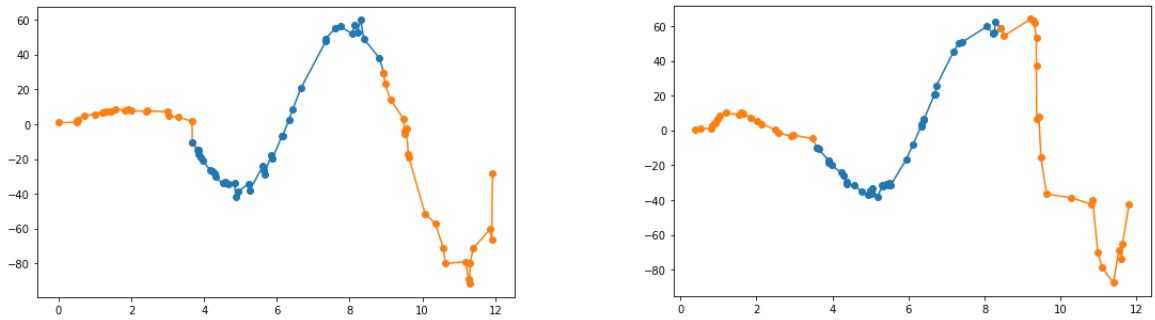


Figure 6.28 T-CGAN - Examples of Sine wave signals with increasing amplitude completed (initial and final ends)

Final Conditioning

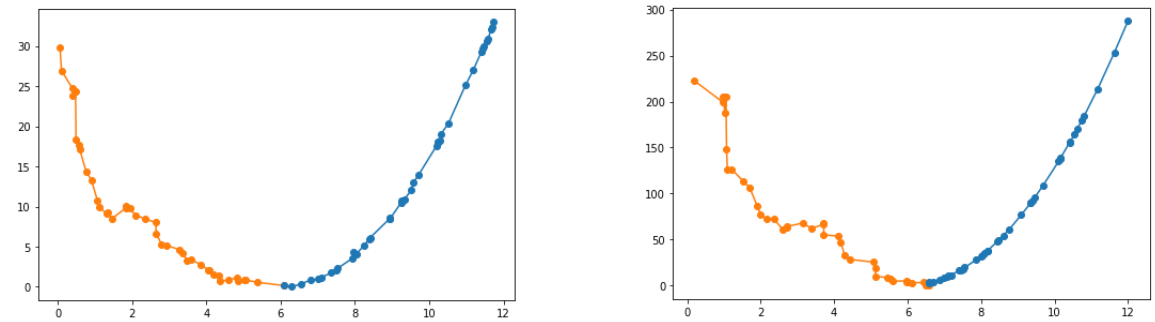


Figure 6.29 T-CGAN - Examples of paraboale completed (first half)

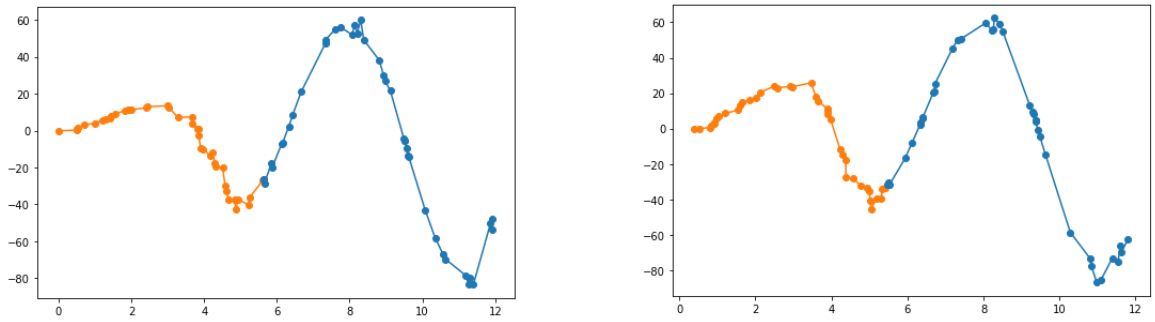


Figure 6.30 T-CGAN - Examples of Sine wave signals with increasing amplitude completed (first half)

Performance

Moving to non-periodic signals which may be a bit more complex to generated due to their non-repetitiveness we can both visually and quantitatively confirm the good quality of the results. In this case the classifier had to distinguish between the two kinds of non periodic time series depicted above.

	Time Series Length		
Training Set Size	40	80	120
40	1.00000	1.00000	1.00000
80	1.00000	1.00000	1.00000
120	1.00000	1.00000	1.00000

Table 6.8 AUROC - T-CGAN - Non Periodic Synthetic Time Series - Initial Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0.99667	1.00000	1.00000
80	1.00000	1.00000	1.00000
120	0.99833	1.00000	1.00000

Table 6.9 AUROC - T-CGAN - Non Periodic Synthetic Time Series - Central Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	1.00000	1.00000	1.00000
80	1.00000	1.00000	1.00000
120	1.00000	1.00000	1.00000

Table 6.10 AUROC - T-CGAN - Non Periodic Synthetic Time Series - Final Conditioning

We can therefore say that our first method is highly reliable even in the presence of non-periodic signals.

6.4.3 Experiments on Real Time Series

Finally, for this first method, we try to verify its capabilities on actual time series datasets coming from real scenarios. Experiments have been performed on SonyAIBORobotSurface and Wafer datasets described at the beginning of this chapter.

Initial Conditioning

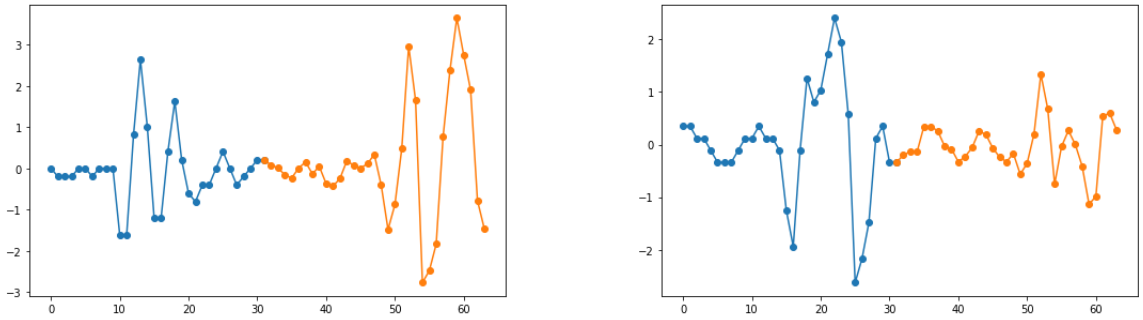


Figure 6.31 T-CGAN - Examples of Sony 'Class 1' signals completed (second half)

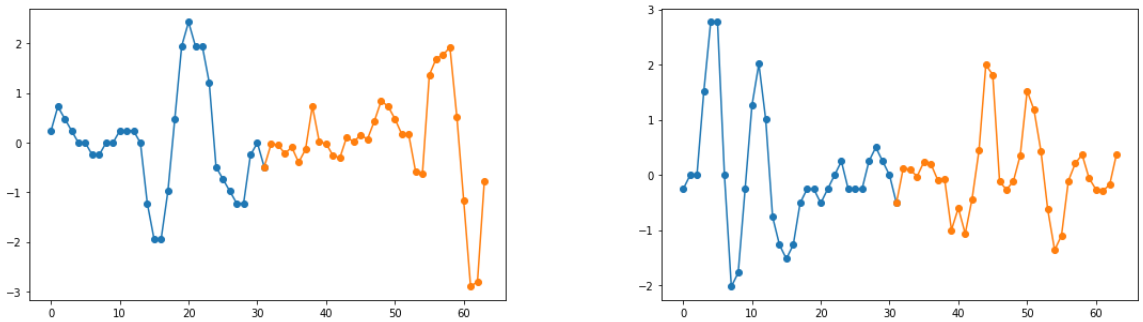


Figure 6.32 T-CGAN - Examples of Sony 'Class 2' signals completed (second half)

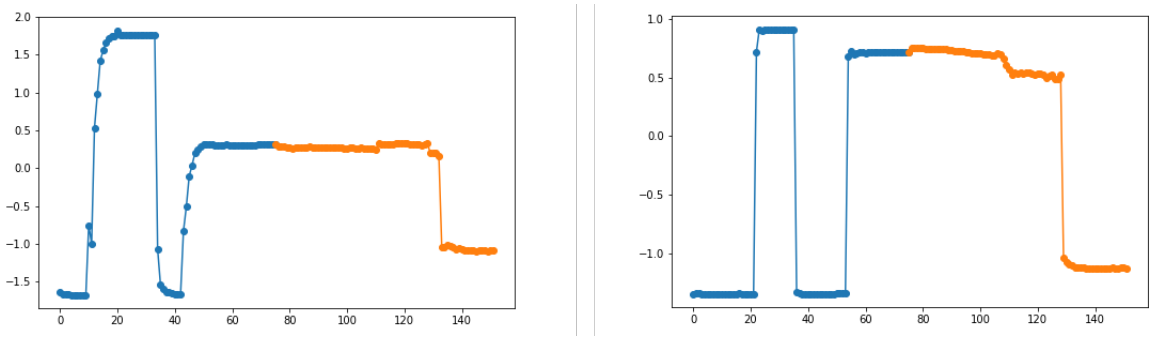


Figure 6.33 T-CGAN - Examples of Wafer 'Positives' signals completed (second half)

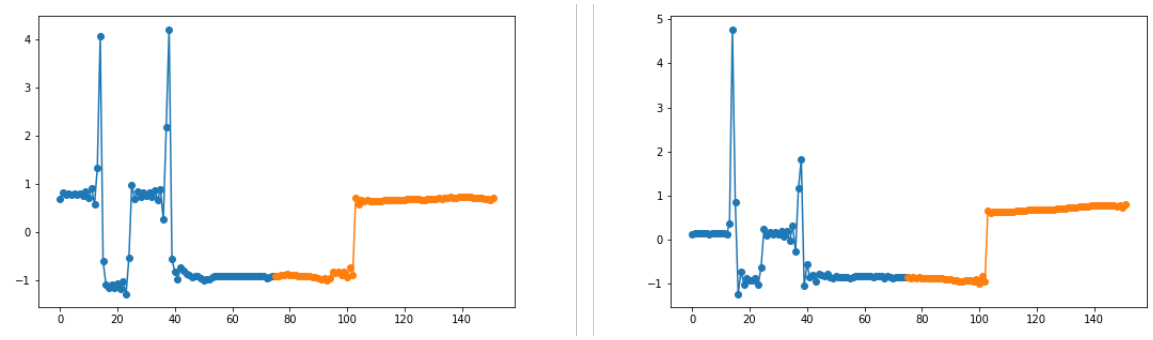


Figure 6.34 T-CGAN - Examples of Wafer 'Negatives' signals completed (second half)

Central Conditioning

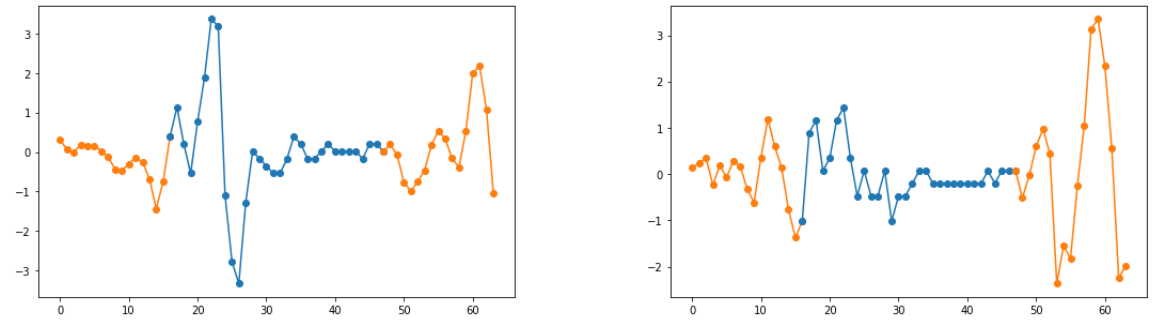


Figure 6.35 T-CGAN - Examples of Sony 'Class 1' signals completed (initial and final ends)

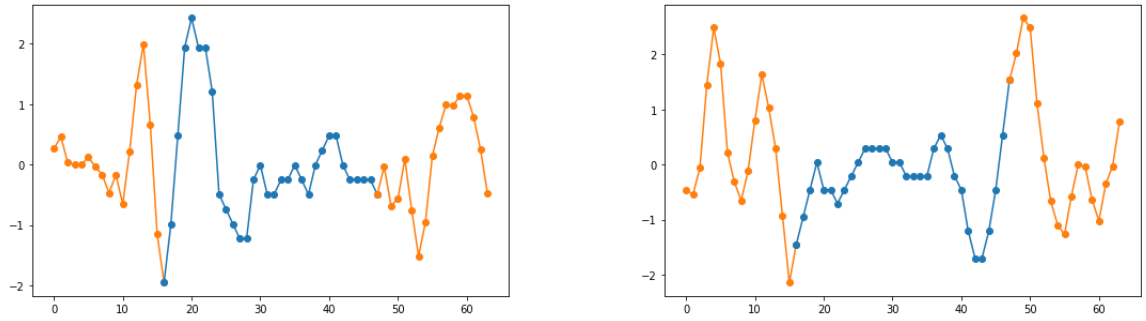


Figure 6.36 T-CGAN - Examples of Sony 'Class 2' signals completed (initial and final ends)

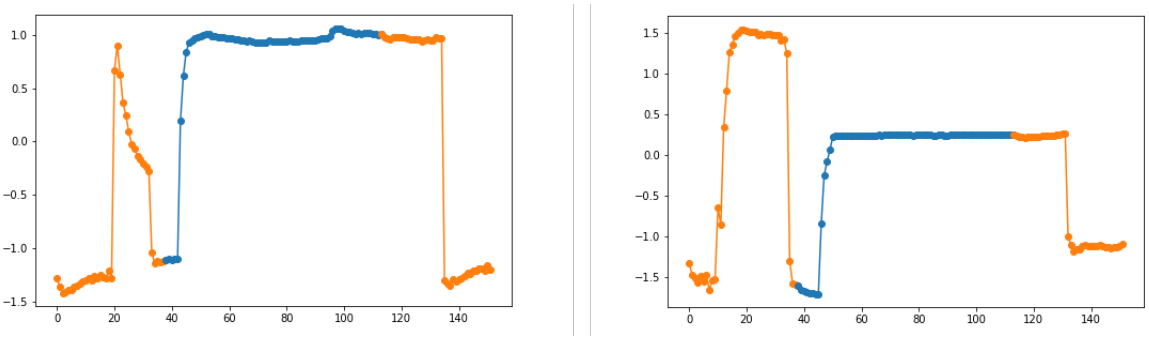


Figure 6.37 T-CGAN - Examples of Wafer 'Positives' signals completed (initial and final ends)

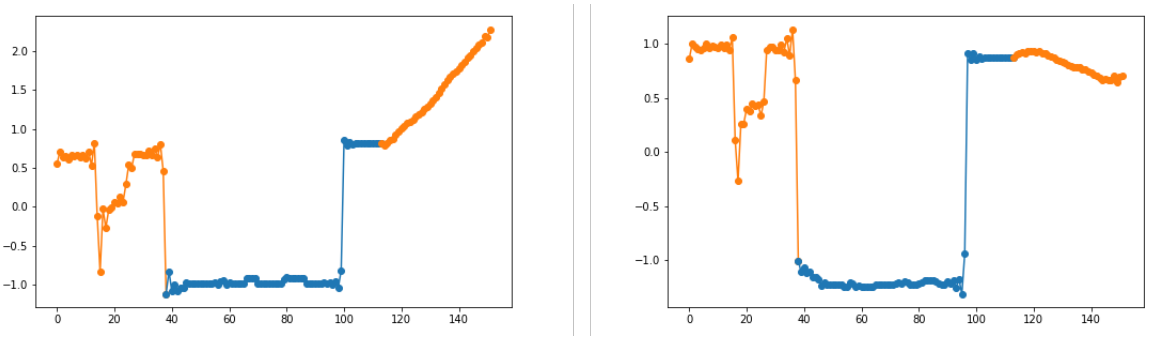


Figure 6.38 T-CGAN - Examples of Wafer 'Negatives' signals completed (initial and final ends)

Final Conditioning

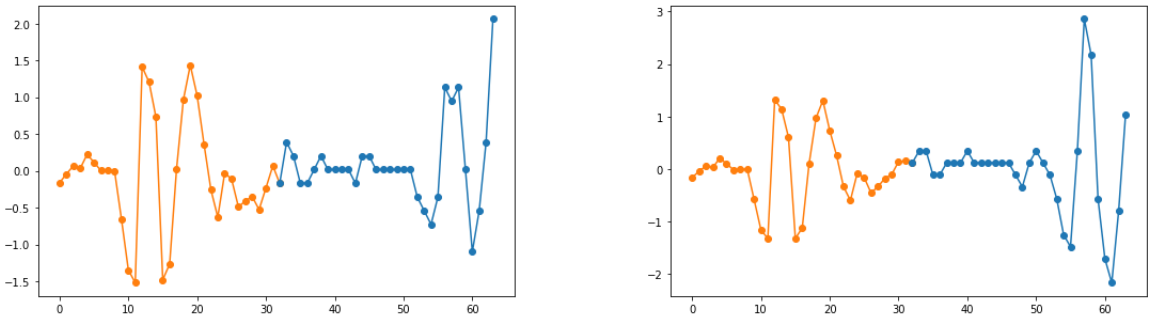


Figure 6.39 T-CGAN - Examples of Sony 'Class 1' signals completed (first half)

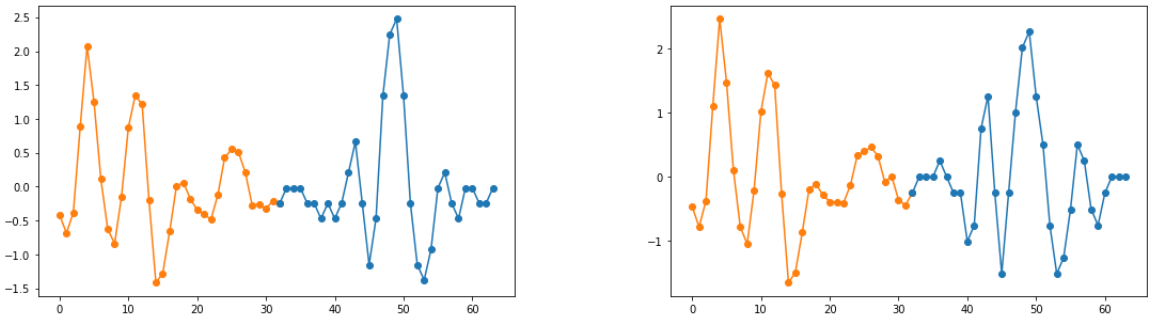


Figure 6.40 T-CGAN - Examples of Sony 'Class 2' signals completed (first half)

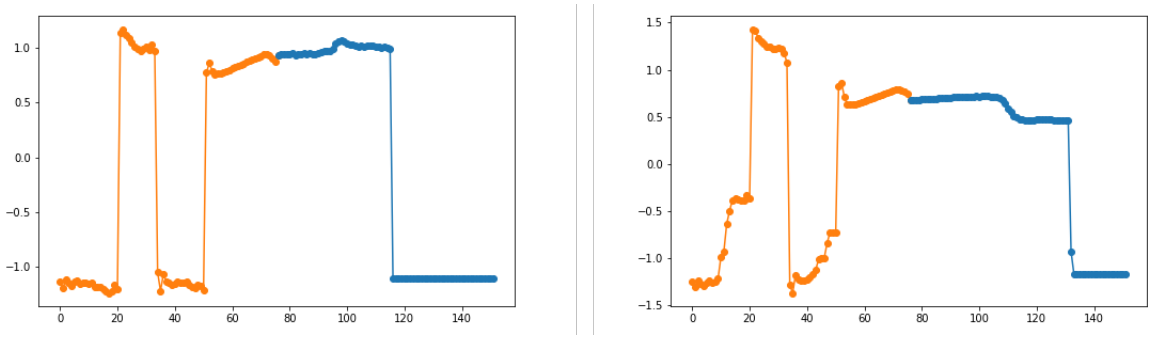


Figure 6.41 T-CGAN - Examples of Wafer 'Positives' signals completed (first half)

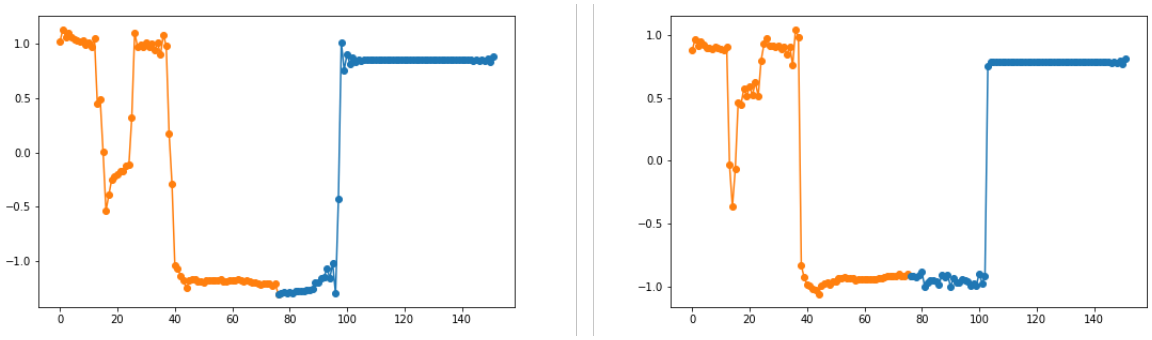


Figure 6.42 T-CGAN - Examples of Wafer 'Negatives' signals completed (first half)

Performance

From a visual standpoint the time series belonging to the different classes of the two datasets have been faithfully rebuilt in each of the three different conditioning scenarios. Newly completed series from the Wafer dataset can however present some disparities in the neighborhood of the connection point. Despite these visual imperfections the classifier was able to perform rather well on real data, even more so on SonyAIBORobotSurface time series compared to the baseline framework.

Initial Conditioning	Central Conditioning	Final Conditioning
0.8824243	0.88303032	0.86727274

Table 6.11 AUROC - T-CGAN - Real Time Series - Sony

Initial Conditioning	Central Conditioning	Final Conditioning
0.9302857	0.85733992	0.810857142

Table 6.12 AUROC - T-CGAN - Real Time Series - Wafer

6.5 TA-CGAN

In this section we will show the results and examples obtained through the experiments done using the second version of our method, explained in Section 4.4. The section is structured the same way as the two previous ones. The main objective here is to understand if this second framework, which is more complex than the first one, is able to learn the actual relationship between values and timestamps and if the increase in model complexity brings any qualitative or quantitative upsides to the first approach.

6.5.1 Experiments on Periodic Synthetic Time Series

Here we will show the experiments conducted on periodic synthetic data, on the same kinds of signals seen before. The blue half is the part given in conditioning, to be completed. The second half, in orange, is the generated part.

Initial Conditioning

The part given as conditioning was only the first half of the curve, asking the network to generate the second half.

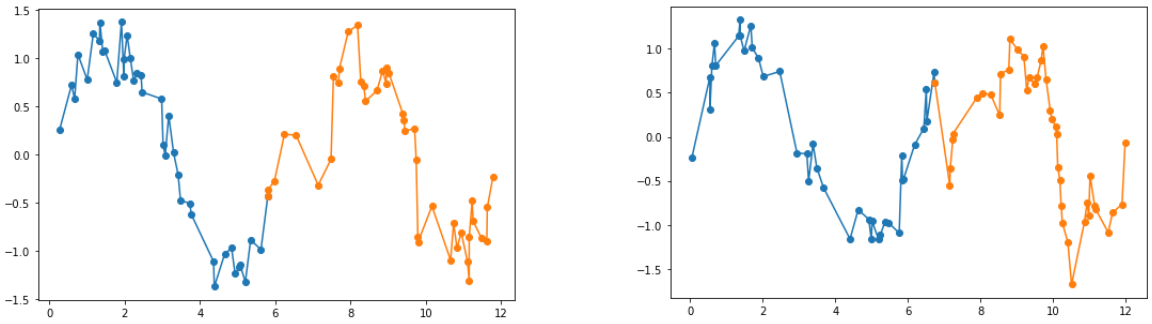


Figure 6.43 TA-CGAN - Examples of Sine wave signals completed (second half)

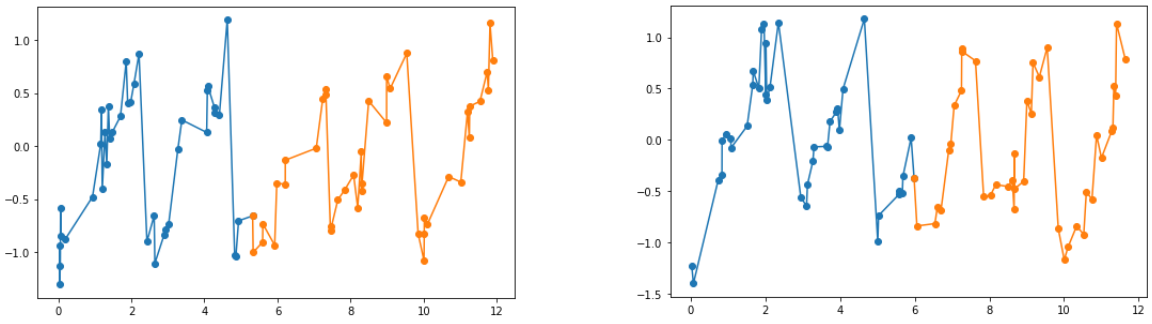


Figure 6.44 TA-CGAN - Examples of Sawtooth signals completed (second half)

Central Conditioning

The part given as conditioning was only the central half of the curve, asking the network to generate the first and the last quarter.

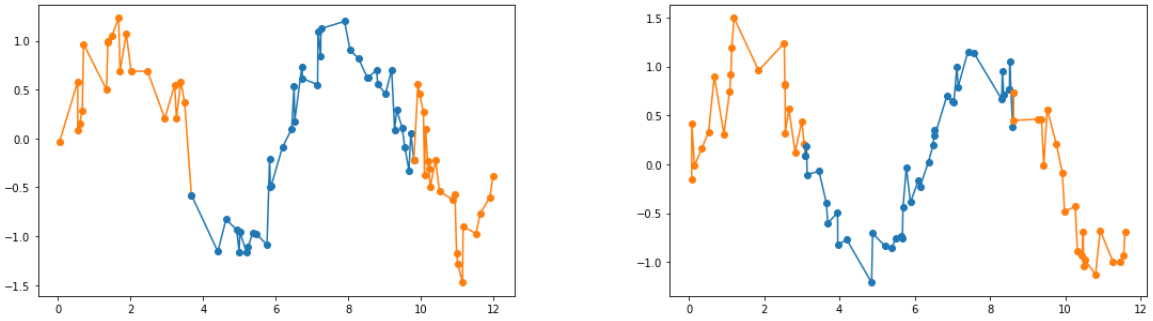


Figure 6.45 TA-CGAN - Examples of Sine wave signals completed (initial and final ends)

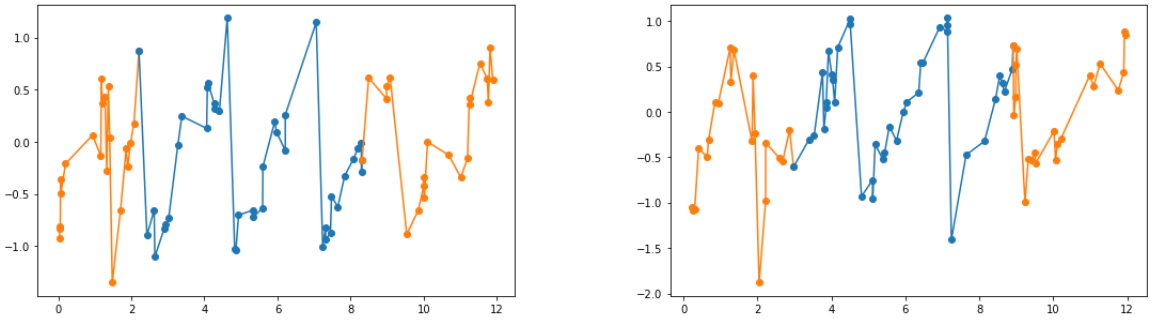


Figure 6.46 TA-CGAN - Examples of Sawtooth signals completed (initial and final ends)

Final Conditioning

The part given as conditioning was only the final half of the curve, asking the network to generate the first half.

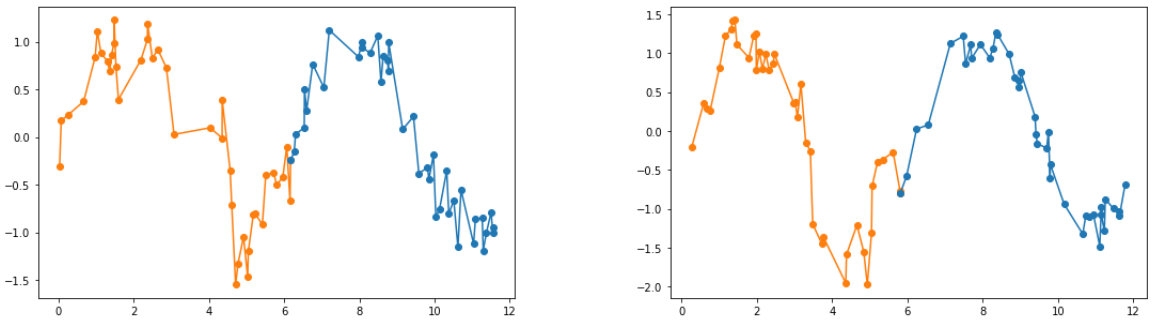


Figure 6.47 TA-CGAN - Examples of Sine wave signals completed (first half)

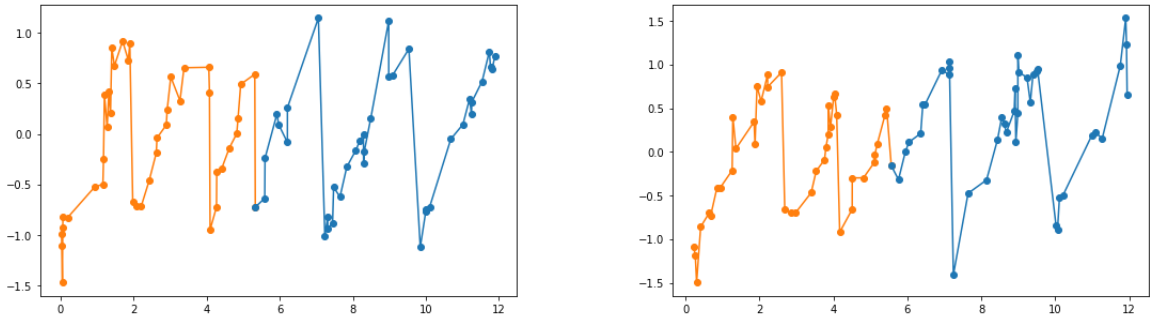


Figure 6.48 TA-CGAN - Examples of Sawtooth signals completed (first half)

Performance

The performance were obtained from a classifier which compared sine wave and sawtooth signals as opposite classes. As for the previous experiments, for each type of conditioning a table has been obtained. As can be seen from the tables, the AUROC values remain almost close to 1. With regard to periodic data, therefore, this second approach also performs well, both performance wise and in visual terms.

Analyzing in detail the values of the AUROC between the previous tables, we can verify that they are slightly better than T-CGAN ones.

	Time Series Length		
Training Set Size	40	80	120
40	0.99833	1.00000	1.00000
80	0.99917	1.00000	0.99333
120	0.99889	1.00000	1.00000

Table 6.13 AUROC - TA-CGAN - Periodic Synthetic Time Series - Initial Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	1.00000	1.00000	1.00000
80	1.00000	0.99917	0.99917
120	1.00000	1.00000	0.99944

Table 6.14 AUROC - TA-CGAN - Periodic Synthetic Time Series - Central Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0.96750	1.00000	1.00000
80	0.98500	1.00000	1.00000
120	0.98917	1.00000	1.00000

Table 6.15 AUROC - TA-CGAN - Periodic Synthetic Time Series - Final Conditioning

6.5.2 Experiments on Non Periodic Synthetic Time Series

In this section we will show the experiments conducted on non periodic synthetic data.

Initial Conditioning

The part given as conditioning was only the first half of the curve, asking the network to generate the second half. The half in blue is the conditioning, the half in orange is the generated one.

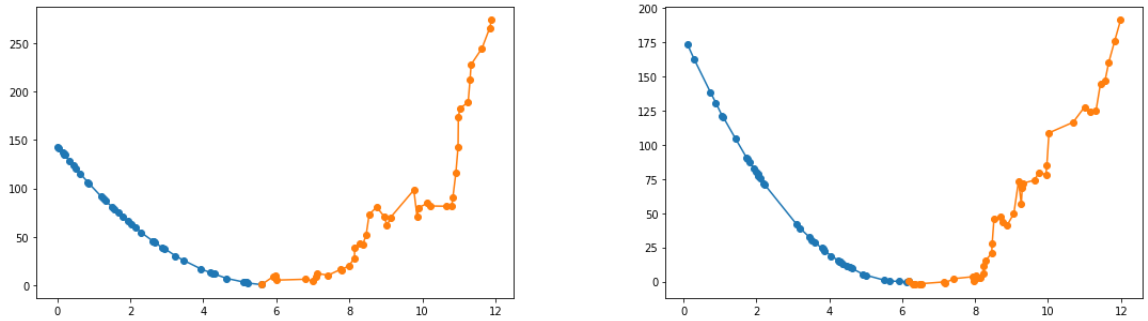


Figure 6.49 TA-CGAN - Examples of Parabolae completed (second half)

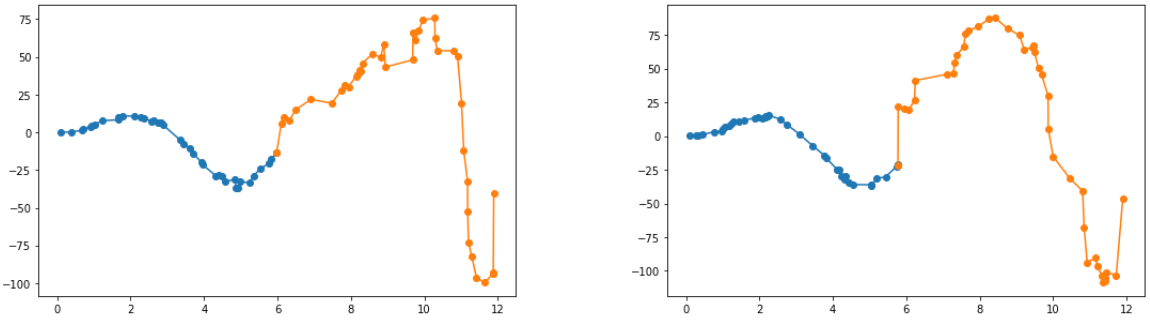


Figure 6.50 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (second half)

Central Conditioning

The part given as conditioning was only the central half of the curve, asking the network to generate the first and the last quarter.

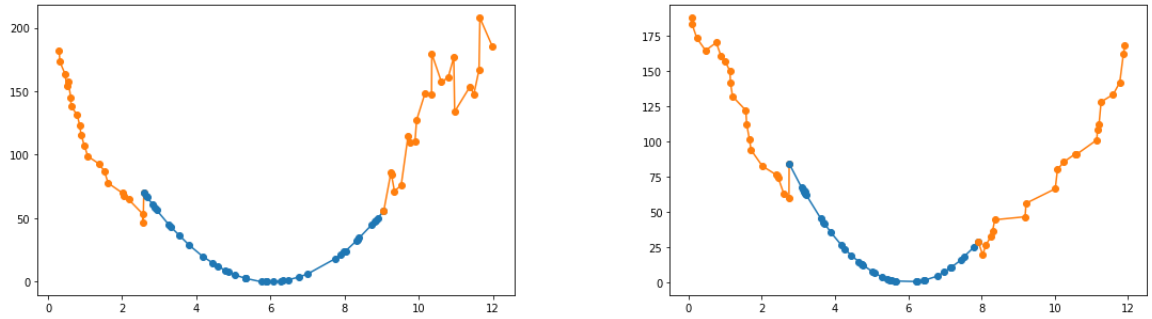


Figure 6.51 TA-CGAN - Examples of Parabolae completed (initial and final ends)

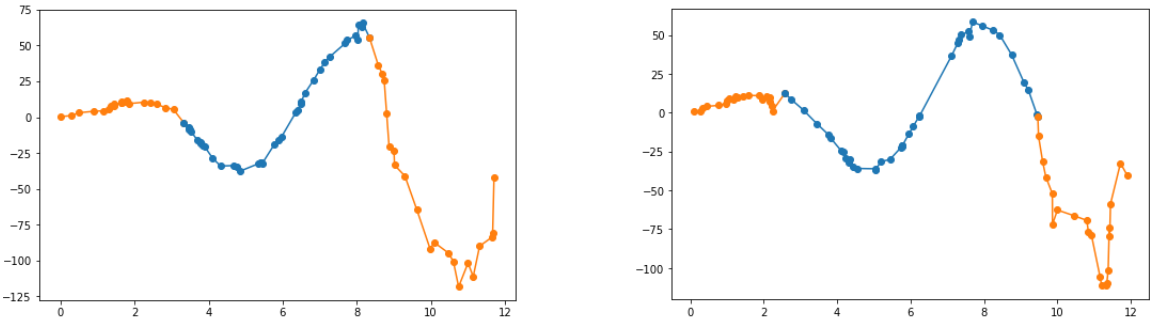


Figure 6.52 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (initial and final ends)

Final Conditioning

The part given as conditioning was only the final half of the curve, asking the network to generate the first half.

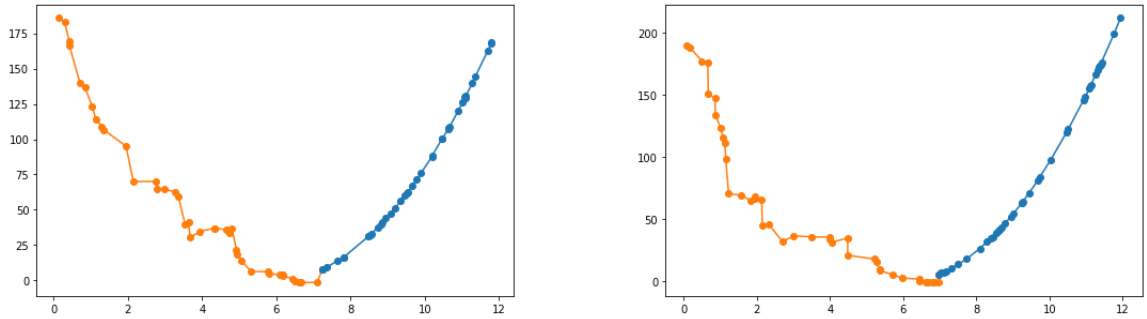


Figure 6.53 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (first half)

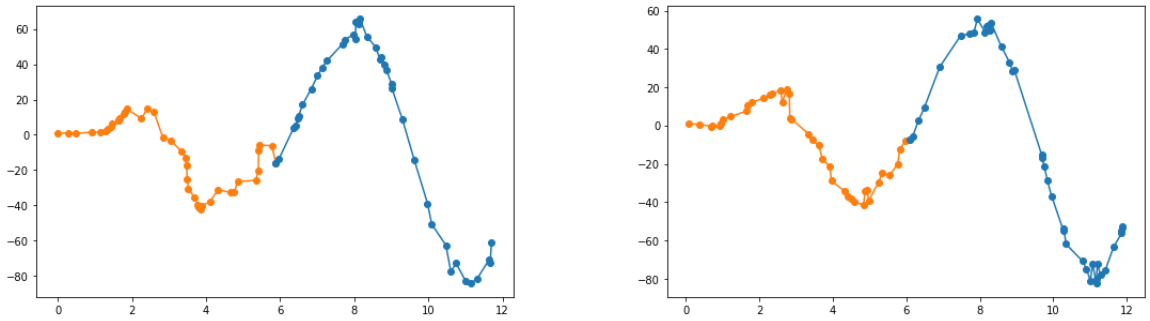


Figure 6.54 TA-CGAN - Examples of Sine wave signals with increasing amplitude completed (first half)

Performance

The performance were obtained from the classifier by comparing Parabolae and Sine wave signals with increasing amplitude as opposite classes. For each type of conditioning a table has been obtained as per usual. The experimental results, as well as the visual ones, confirm the goodness of this second methodology also for this series of experiments. The signals join together almost perfectly. It can be noticed only a few slight distances on the join point, mostly due to irregular sampling. It must be considered however that the shape of the curve is maintained as always. Analyzing the performance in detail it can be seen that even in this section, they are slightly better than the first method and clearly better than the baseline.

	Time Series Length		
Training Set Size	40	80	120
40	1.00000	1.00000	1.00000
80	0.99917	1.00000	1.00000
120	0.99944	1.00000	1.00000

Table 6.16 AUROC - TA-CGAN - Non Periodic Synthetic Time Series - Initial Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	1.00000	1.00000	1.00000
80	0.99750	0.99833	1.00000
120	0.99611	1.00000	1.00000

Table 6.17 AUROC - TA-CGAN - Non Periodic Synthetic Time Series - Central Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0.99833	1.00000	1.00000
80	1.00000	1.00000	1.00000
120	0.99833	1.00000	1.00000

Table 6.18 AUROC - TA-CGAN - Non Periodic Synthetic Time Series - Final Conditioning

6.5.3 Experiments on Real Time Series

Finally, also for the second method, we try to verify its capabilities on datasets coming from real scenarios. Experiments have been performed on SonyAIBORobotSurface and Wafer datasets described at the beginning of this chapter.

Initial Conditioning

The part given as conditioning was only the first half of the curve, asking the network to generate the second half.

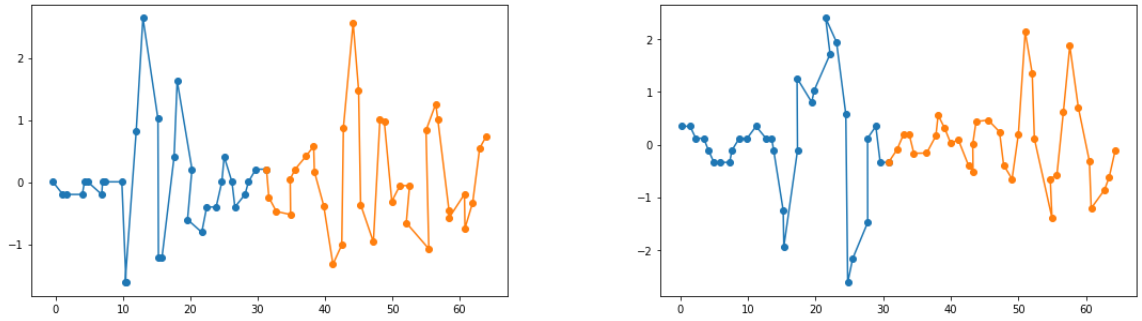


Figure 6.55 TA-CGAN - Examples of Sony 'Class 1' signals completed (second half)

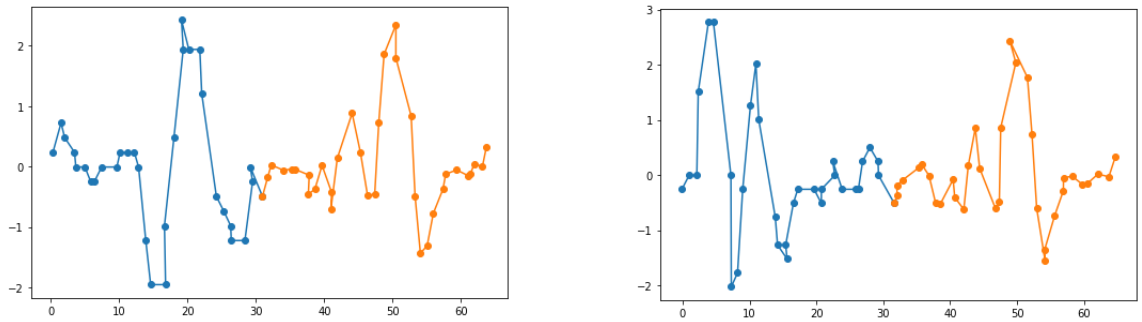


Figure 6.56 TA-CGAN - Examples of Sony 'Class 2' signals completed (second half)

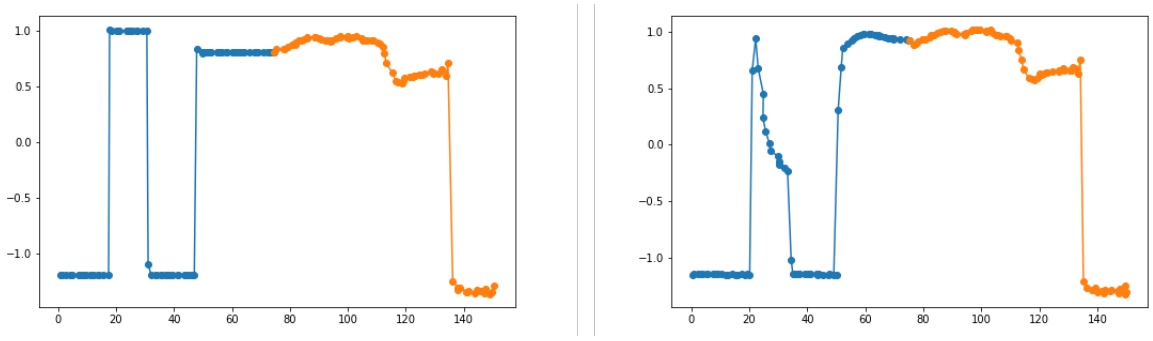


Figure 6.57 TA-CGAN - Examples of Wafer 'Positives' signals completed (second half)

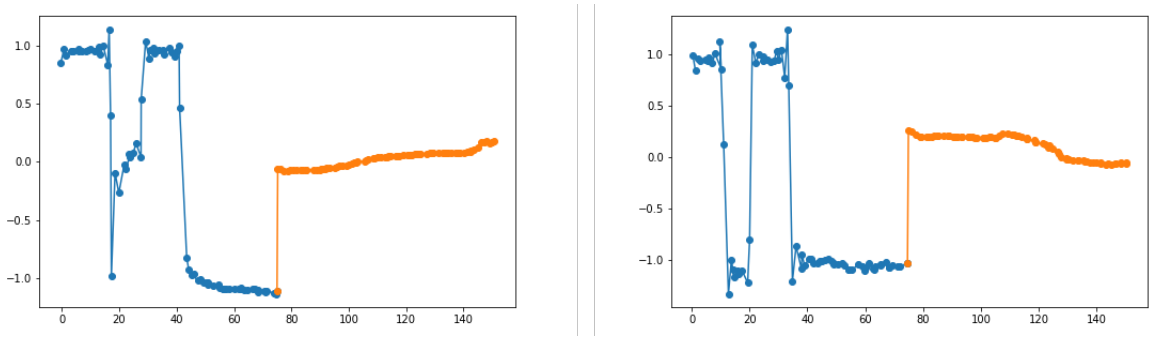


Figure 6.58 TA-CGAN - Examples of Wafer 'Negatives' signals completed (second half)

Central Conditioning

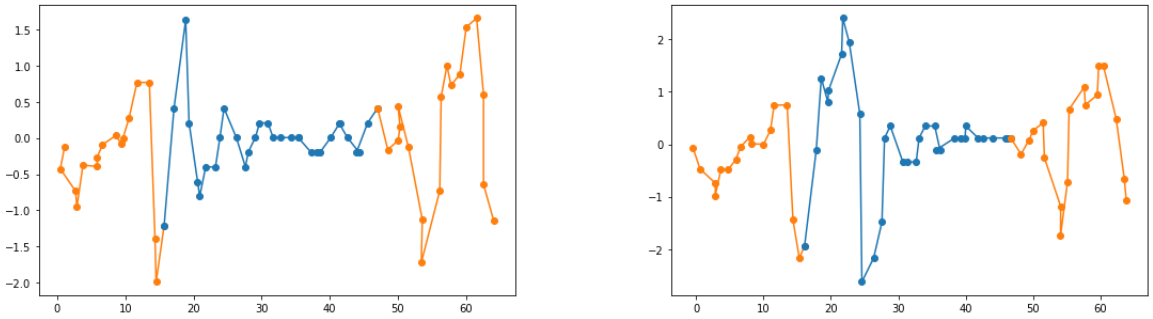


Figure 6.59 TA-CGAN - Examples of Sony 'Class 1' signals completed (initial and final ends)

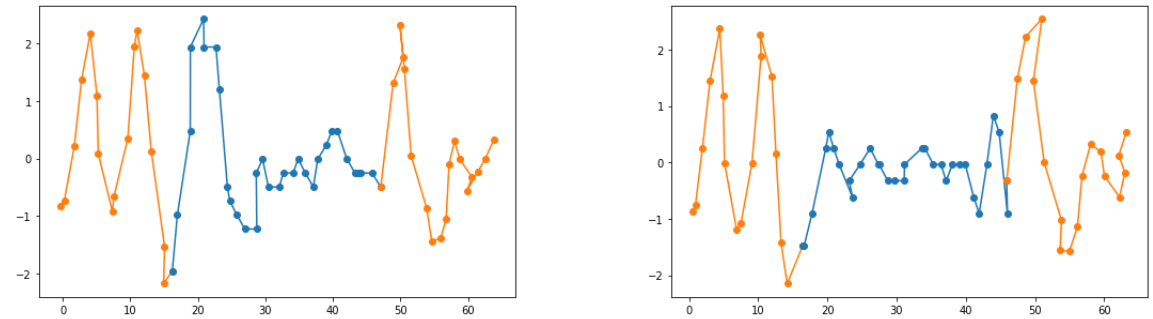


Figure 6.60 TA-CGAN - Examples of Sony 'Class 2' signals completed (initial and final ends)

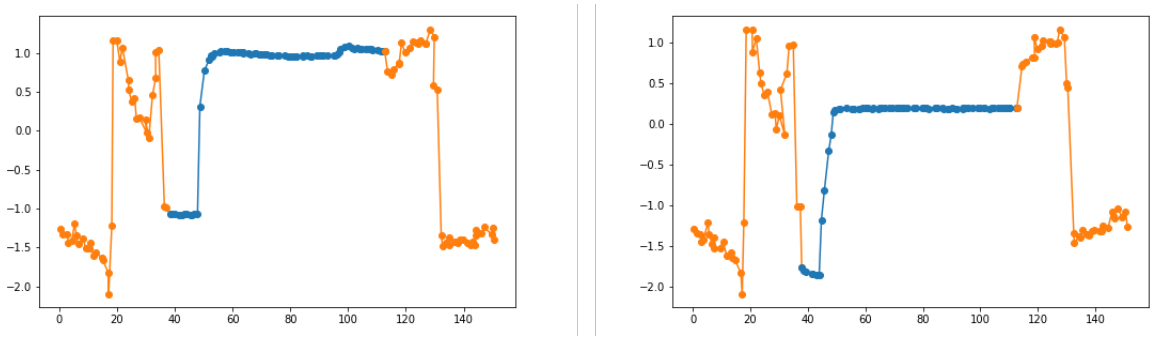


Figure 6.61 TA-CGAN - Examples of Wafer 'Positives' signals completed (initial and final ends)

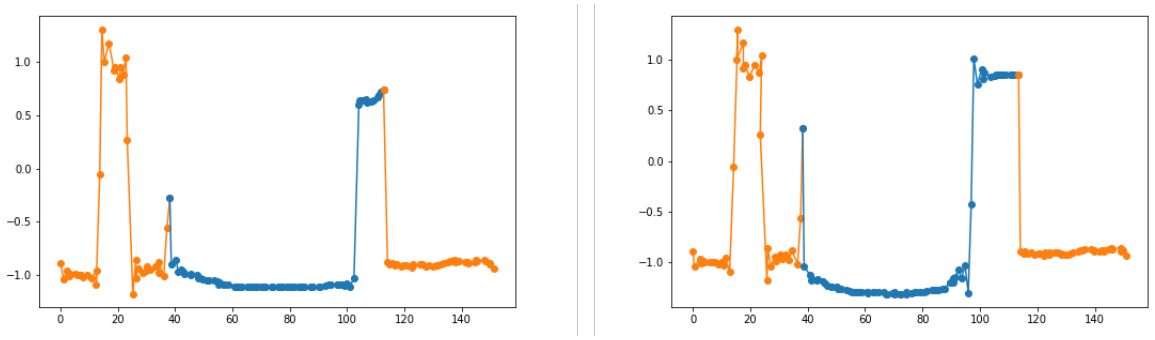


Figure 6.62 TA-CGAN - Examples of Wafer 'Negatives' signals completed (initial and final ends)

Final Conditioning

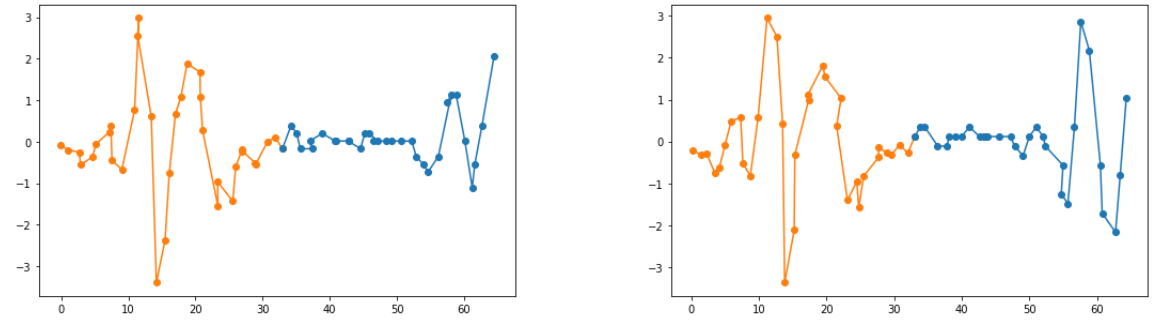


Figure 6.63 TA-CGAN - Examples of Sony 'Class 1' signals completed (first half)

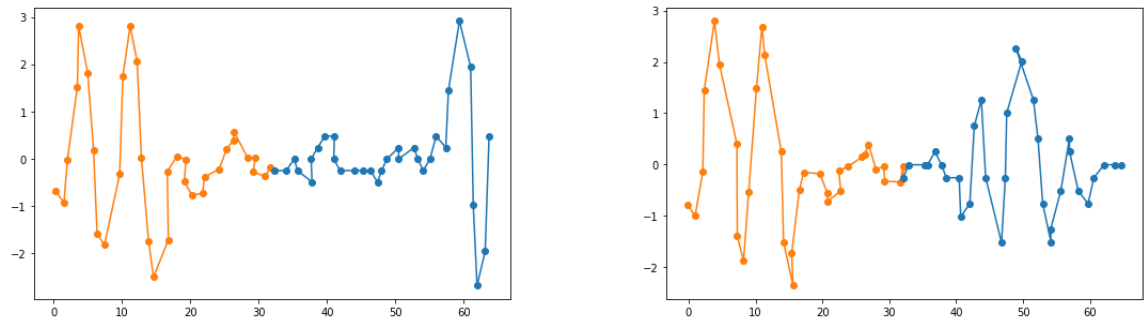


Figure 6.64 TA-CGAN - Examples of Sony 'Class 2' signals completed (first half)

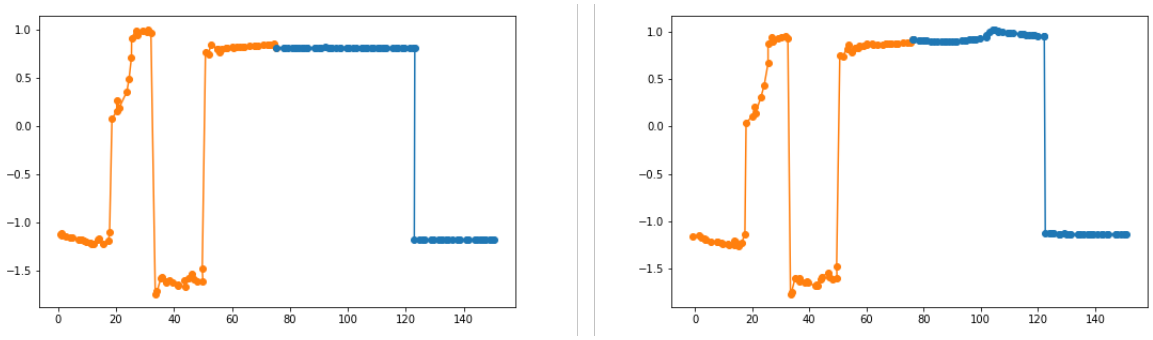


Figure 6.65 TA-CGAN - Examples of Wafer 'Positives' signals completed (first half)

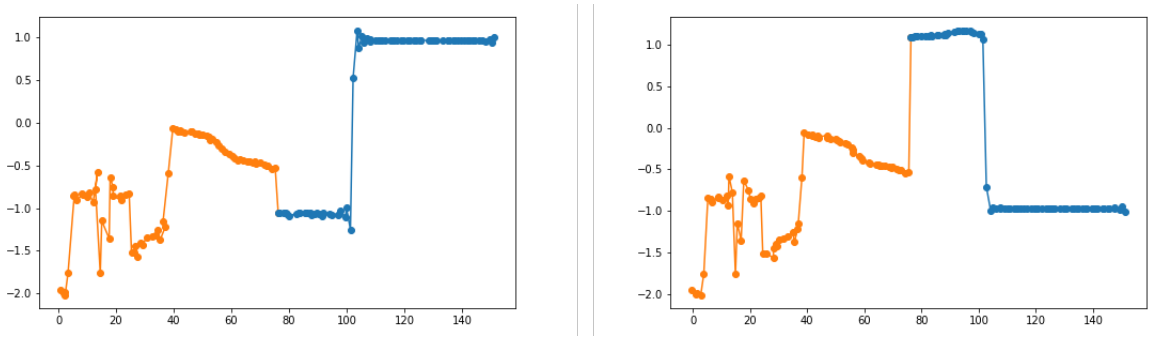


Figure 6.66 TA-CGAN - Examples of Wafer 'Negatives' signals completed (first half)

Performance

The performance were obtained from the classifier by comparing the two classes of SonyAIBORobotSurface dataset and the two classes of Wafer dataset. Analyzing the curves graphically, we note that they always faithfully maintain the shape of the real curves, joining together very well. We note however from the performance that we have a degradation regarding the version 1. In fact, the performance are about 0.15 lower on average. In the conclusions we will analyze in greater depth the possible reasons linked to this issue.

Initial Conditioning	Central Conditioning	Final Conditioning
0.8157576	0.76181817	0.79333334

Table 6.19 AUROC - TA-CGAN - Real Time Series - Sony

Initial Conditioning	Central Conditioning	Final Conditioning
0.8236943	0.729714272	0.697714288

Table 6.20 AUROC - TA-CGAN - Real Time Series - Wafer

6.6 LSTM GAN

This section deals with experiments using the LSTM GAN approach, explained in Section 4.5. As before, all the experimental images refer to the case studies (80, 80). At the bottom, the performance related to all the single experiments can be found.

Differently from the two previous approaches, due to reasons reported in Chapter 7, all time series, both synthetic and real, have been scaled to fit in the interval $[-1,+1]$. Moreover, Synthetic training sets are made of 8000 curves and the noise is sampled from a Gaussian with $\mu = 0$ and $\sigma = 0.05$.

6.6.1 Experiments on Periodic Synthetic Time Series

In this section we will show the experiments conducted on periodic synthetic data using the proposed LSTM GAN variant.

Complete Generation

This first batch of experiments is dedicated to the comparison between this approach and the one proposed in [25] to completely generate sine waves and sawtooth waves. As such, no conditioning information regarding the time series values is given to the generative model. Both networks however are still conditioned on time, as described in 4.5, with the differences between observation times.

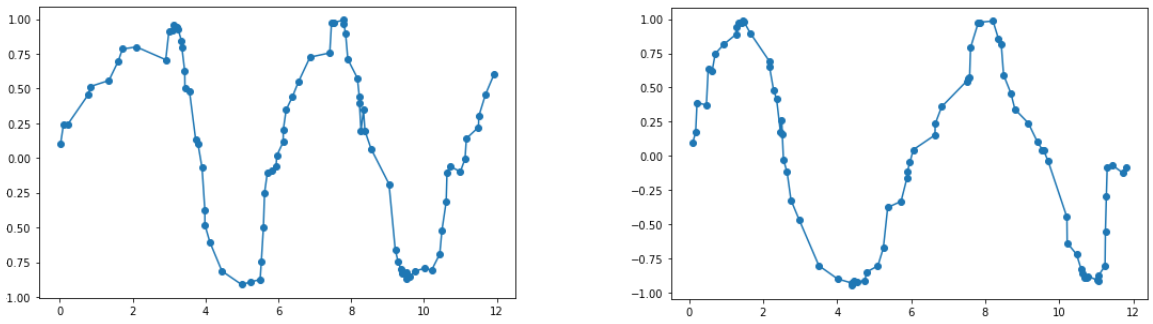


Figure 6.67 LSTM GAN - Examples of completely generated Sine wave signals

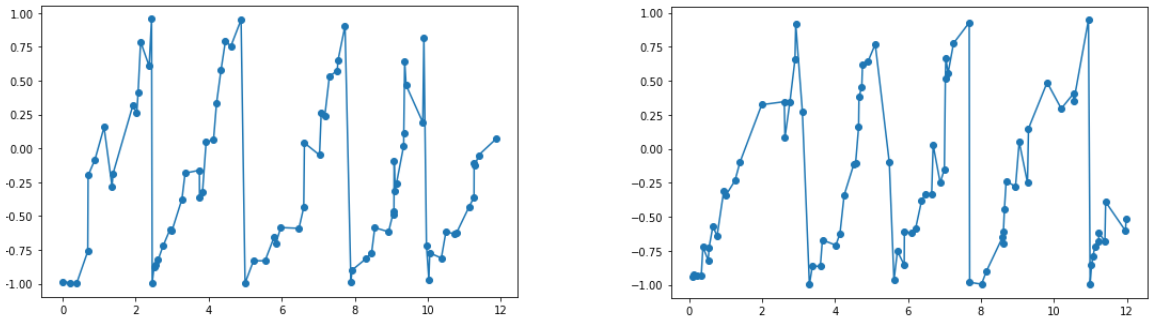


Figure 6.68 LSTM GAN - Examples of completely generated sawtooth signals

Training Set Size	T-CGAN				LSTM			
	Time Series Length				Time Series Length			
	40	60	80	100	40	60	80	100
40	0.970	0.995	0.995	1.0	0.94488	0.99388	0.98950	0.97144
60	0.988	0.997	1.0	0,994	0.98569	0.99744	0.99000	0.99888
80	0.981	0.971	0.990	0,981	0.98525	0.99625	0.99806	0.99569
100	0.937	0.955	1.0	0,998	0.99013	0.98869	0.99125	0.99519

Table 6.21 AUROC - LSTM GAN - Periodic Synthetic Time Series - Complete Generation

In this table the AUROC values for the experiment in which full-length time series were generated are reported. The four columns under *T-CGAN* have been taken from [25] and reported for comparison with our approach.

Table 6.21 shows that the LSTM approach has comparable performance results to T-CGAN, in some cases even surpassing it, when it comes to the complete generation of sine and sawtooth waves.

Initial Conditioning

The part given as conditioning was only the first half of the curve, asking the network to generate the second half. Half of the curve, in blue, indicates the half given in conditioning, half in orange instead that generated by the neural network.

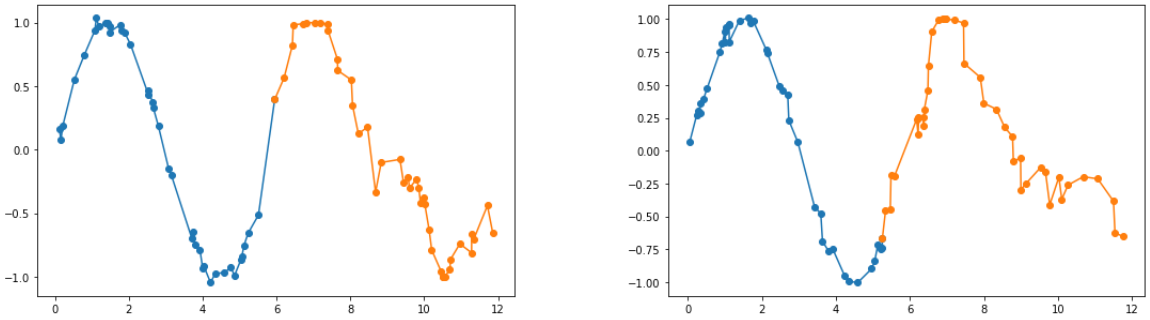


Figure 6.69 LSTM GAN - Examples of Sine wave signals completed (second half)

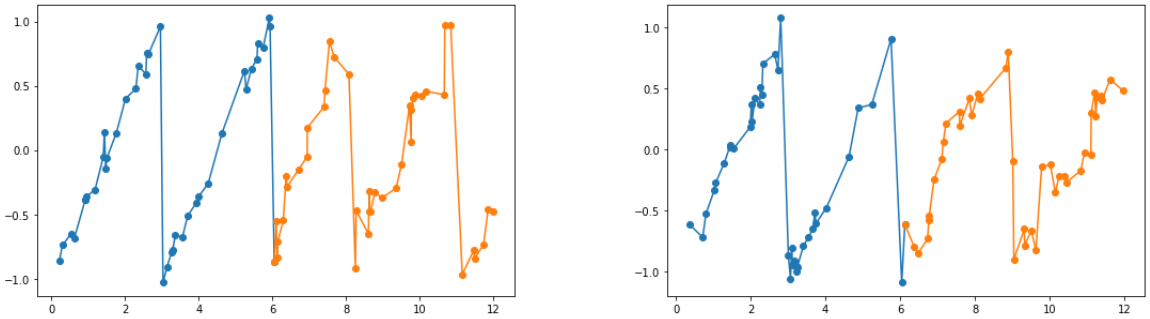


Figure 6.70 LSTM GAN - Examples of Sawtooth signals completed (second half)

Central Conditioning

The part given as conditioning was only the central half of the curve, asking the network to generate the first and the last quarter.

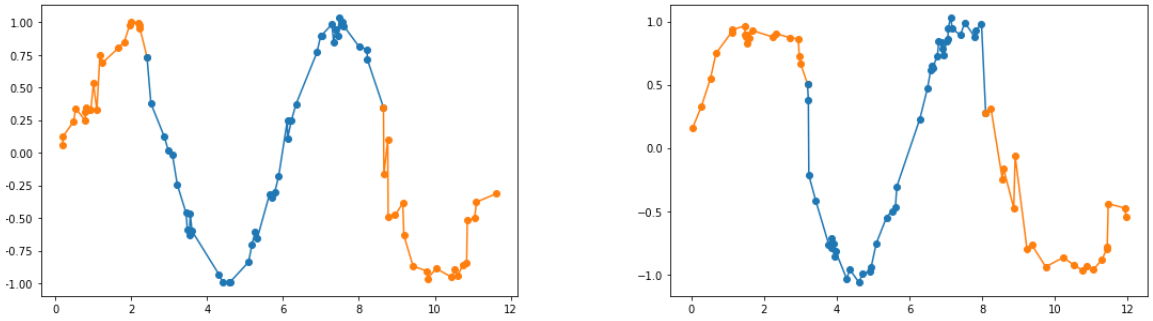


Figure 6.71 LSTM GAN - Examples of Sine wave signals completed (initial and final ends)

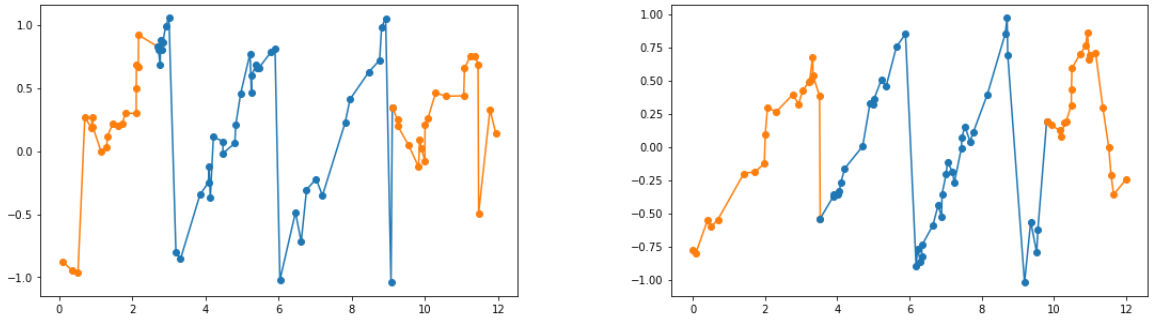


Figure 6.72 LSTM GAN - Examples of Sawtooth signals completed (initial and final ends)

Final Conditioning

The part given as conditioning was only the final half of the curve, asking the network to generate the initial part.

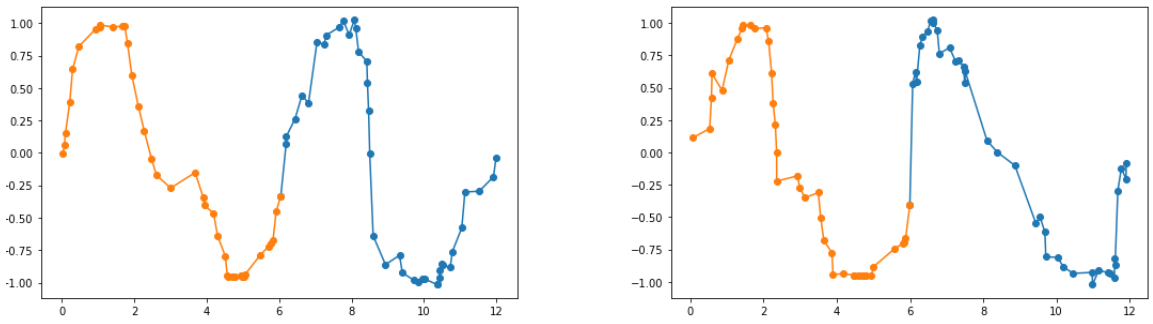


Figure 6.73 LSTM GAN - Examples of Sine wave signals completed (first half)

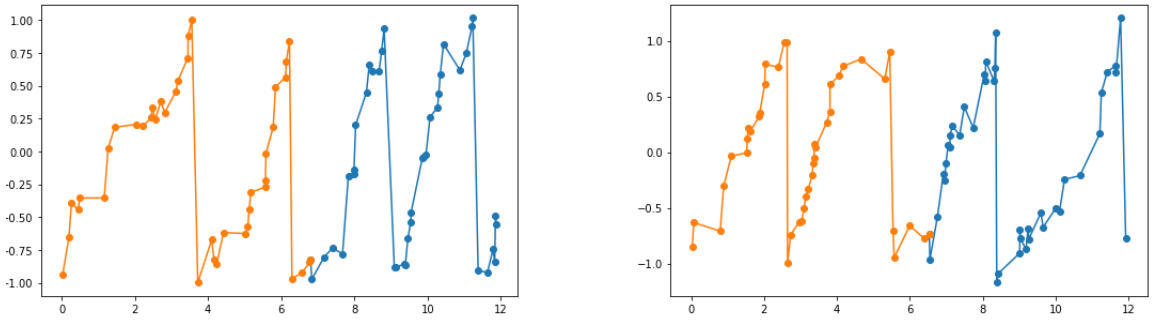


Figure 6.74 LSTM GAN - Examples of Sawtooth signals completed (first half)

Performance

The performance for this first part of the experiments on LSTM GAN are obtained, as for the previous methods, using sine wave signal and sawtooth signal in the classifier as the two separate classes. It can be noted that as far as periodic signals are concerned, graphically the curves join up perfectly, as in the two convolutional variants. Moreover, the shape of the curves is fully respected. However, analytically, we have to deal with a slight degradation in performance, as shown in the following tables.

	Time Series Length		
Training Set Size	40	80	120
40	0,96119	0,99975	0,99681
80	0,97619	0,99931	0,99888
120	0,96719	1,00000	0,99994

Table 6.22 AUROC - LSTM GAN - Periodic Synthetic Time Series - Initial Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0,97781	0,98550	0,99988
80	0,98956	0,99756	0,99953
120	0,99294	0,99938	1,00000

Table 6.23 AUROC - LSTM GAN - Periodic Synthetic Time Series - Central Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0,96638	0,99206	0,99919
80	0,97038	0,98900	0,99994
120	0,98119	0,99000	1,00000

Table 6.24 AUROC - LSTM GAN - Periodic Synthetic Time Series - Final Conditioning

6.6.2 Experiments on Non Periodic Synthetic Time Series

In this section we will show the experiments conducted on non periodic synthetic data.

Initial Conditioning

The part given as conditioning was only the first half of the curve, asking the network to generate the second half.

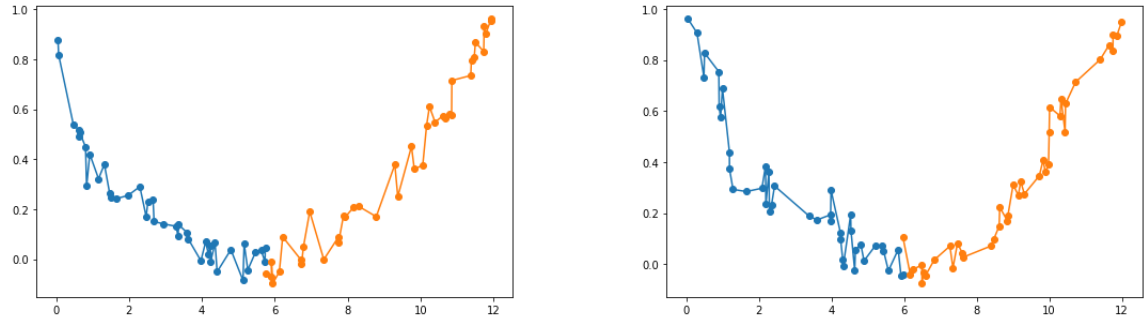


Figure 6.75 LSTM GAN - Examples of Parabolae completed (second half)

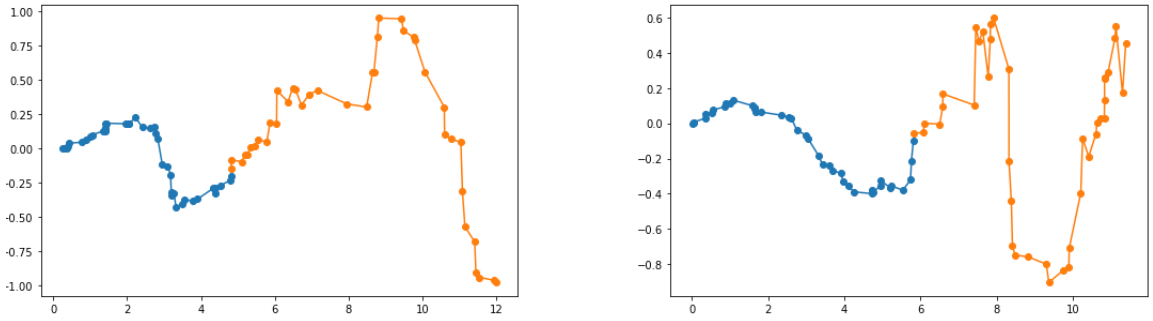


Figure 6.76 LSTM GAN - Examples of Sine wave signals with increasing amplitude completed (second half)

Central Conditioning

The part given as conditioning was only the central half of the curve, asking the network to generate the first and the last quarter.

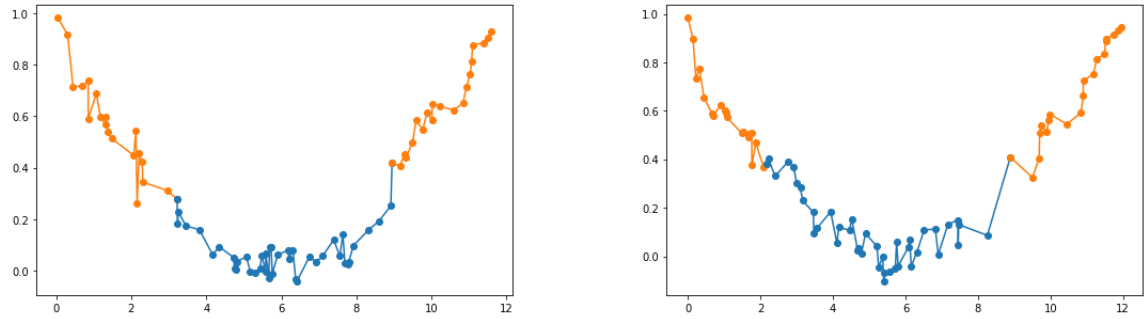


Figure 6.77 LSTM GAN - Examples of Parabolae completed (initial and final ends)

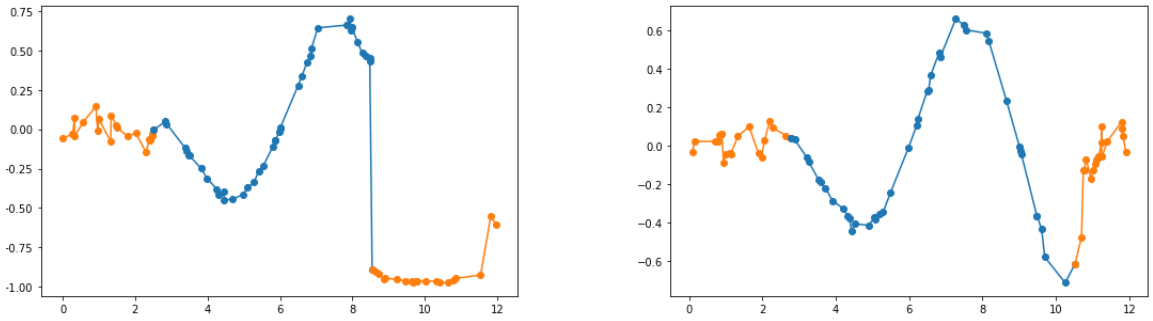


Figure 6.78 LSTM GAN - Examples of Sine wave signals with increasing amplitude completed (initial and final ends)

Final Conditioning

The part given as conditioning was only the final half of the curve, asking the network to generate the first half.

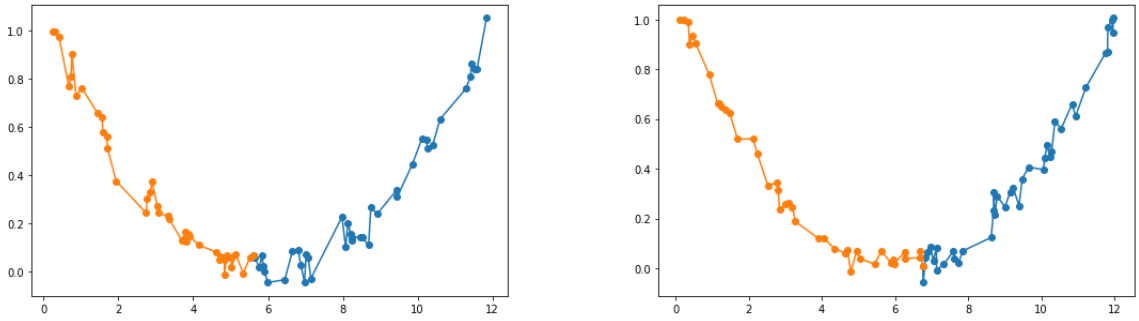


Figure 6.79 LSTM GAN - Examples of paraboale completed (first half)

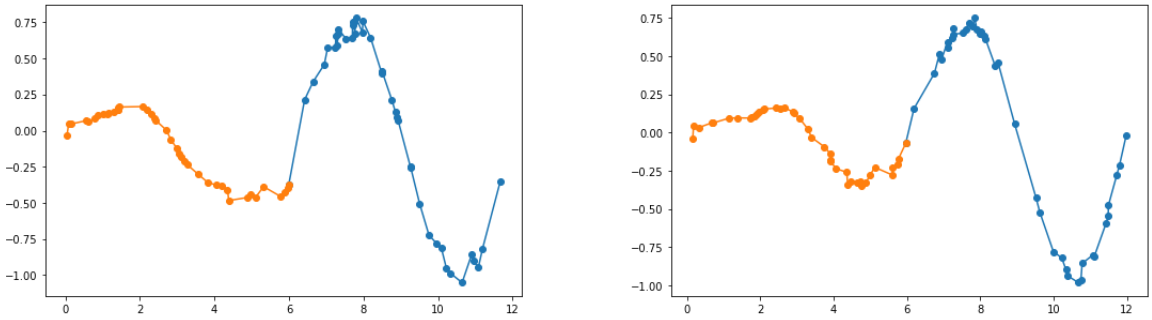


Figure 6.80 Examples of Sine wave signals with increasing amplitude completed (first half) by the LSTM GAN

Performance

The performance were obtained from the classifier by comparing parabolae and sine wave signals with increasing amplitude as opposite classes. The results, even in this case, are very good. Visually the signals are perfect, both in surroundings of the junction point and in the perfectly maintained form. The performance remain excellent, in line with those of the previous methods.

	Time Series Length		
Training Set Size	40	80	120
40	0,99981	1,00000	1,00000
80	0,99969	0,99994	1,00000
120	0,99975	1,00000	1,00000

Table 6.25 AUROC - LSTM GAN - Non Periodic Synthetic Time Series - Initial Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	1,00000	1,00000	1,00000
80	0,99925	1,00000	1,00000
120	0,99956	0,99994	1,00000

Table 6.26 AUROC - LSTM GAN - Non Periodic Synthetic Time Series - Central Conditioning

	Time Series Length		
Training Set Size	40	80	120
40	0,99906	0,99988	1,00000
80	0,99938	1,00000	1,00000
120	0,99938	1,00000	1,00000

Table 6.27 AUROC - LSTM GAN - Non Periodic Synthetic Time Series - Final Conditioning

6.6.3 Experiments on Real Time Series

Obviously also for the LSTM implementation, we tried to verify its capabilities on datasets coming from real scenarios. These experiments have been performed on SonyAIBORobotSurface and Wafer datasets described at the beginning of this chapter.

In the specific case of the Wafer dataset the time series have been sliced in half, with only the first one used for this batch of experiments.

Initial Conditioning

The part given as conditioning was only the first half of the curve, asking the network to generate the second half.

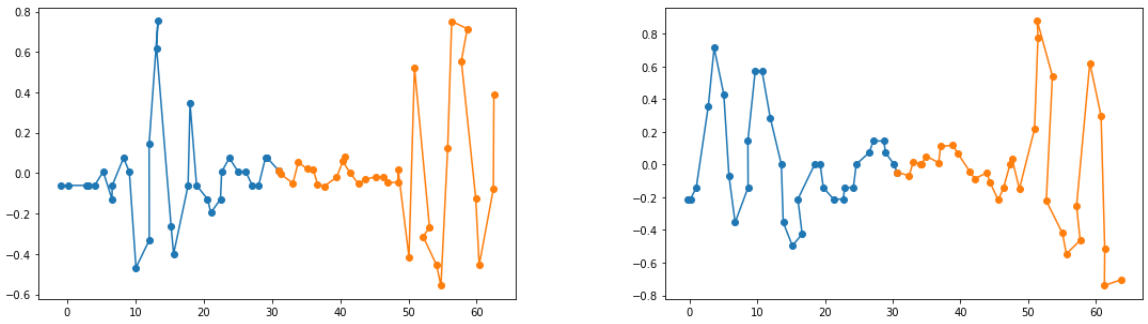


Figure 6.81 LSTM GAN - Examples of Sony 'Class 1' signals completed (second half)

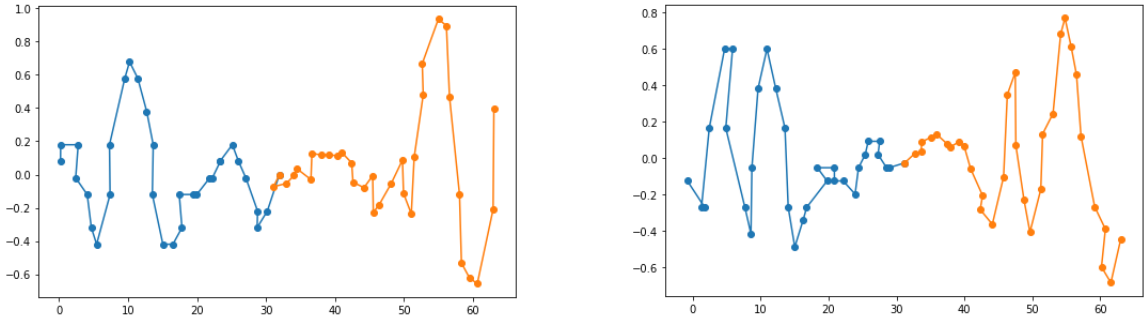


Figure 6.82 LSTM GAN - Examples of Sony 'Class 2' signals completed (second half)

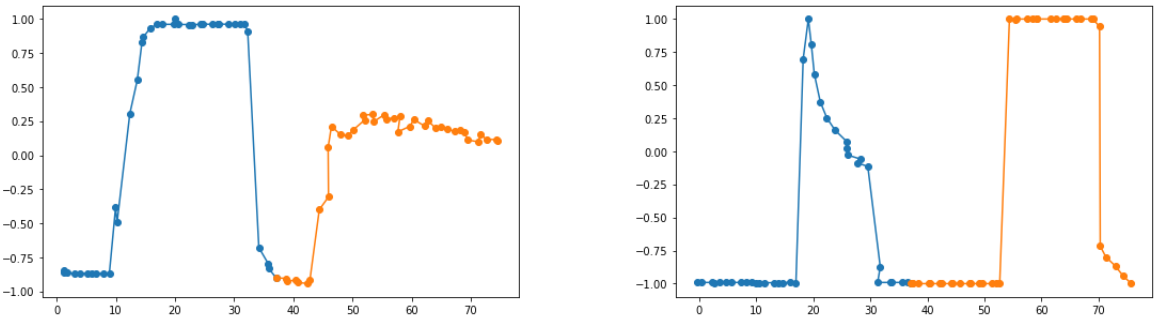


Figure 6.83 LSTM GAN - Examples of Wafer 'Positives' signals completed (second half)

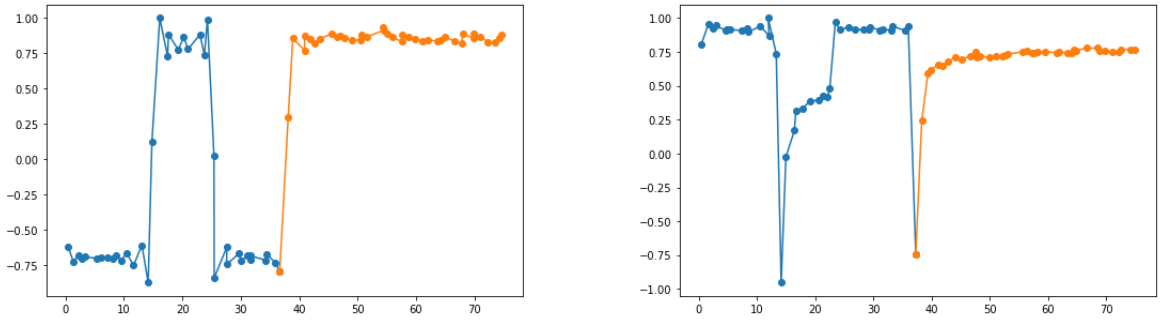


Figure 6.84 LSTM GAN - Examples of Wafer 'Negatives' signals completed (second half)

Central Conditioning

The part given as conditioning was only the central half of the curve, asking the network to generate the first and the last quarter.

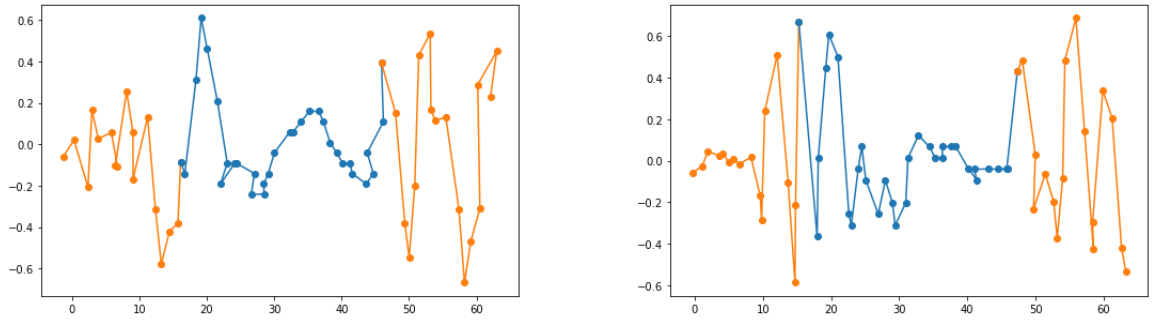


Figure 6.85 LSTM GAN - Examples of Sony 'Class 1' signals completed (initial and final ends)

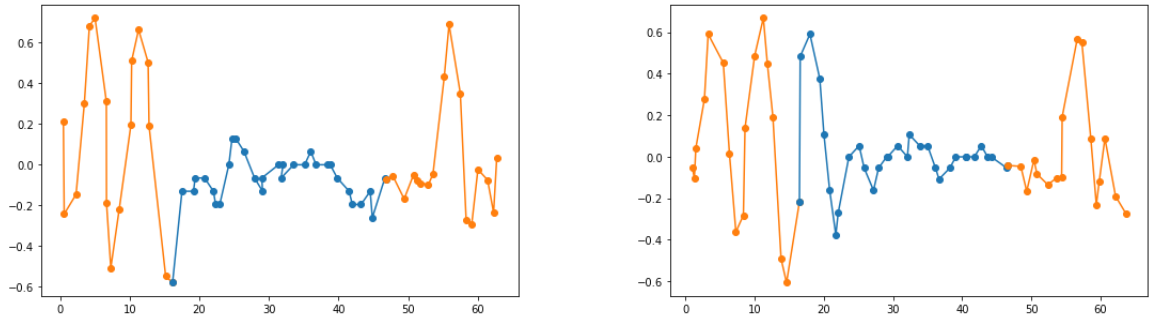


Figure 6.86 LSTM GAN - Examples of Sony 'Class 2' signals completed (initial and final ends)

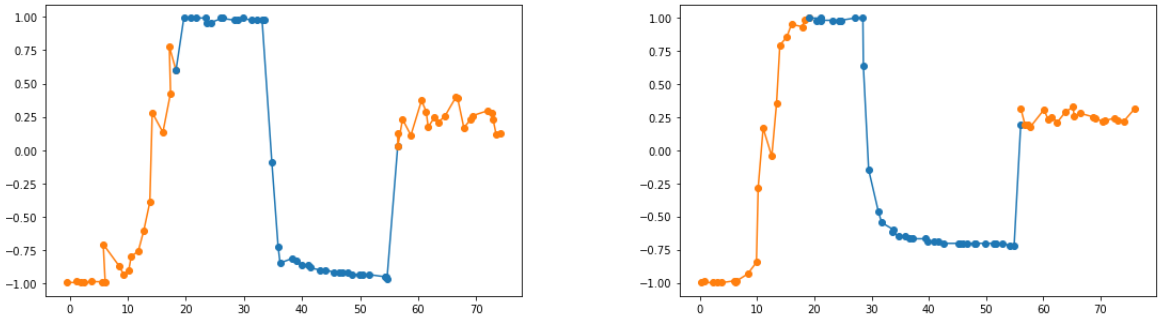


Figure 6.87 LSTM GAN - Examples of Wafer 'Positives' signals completed (initial and final ends)

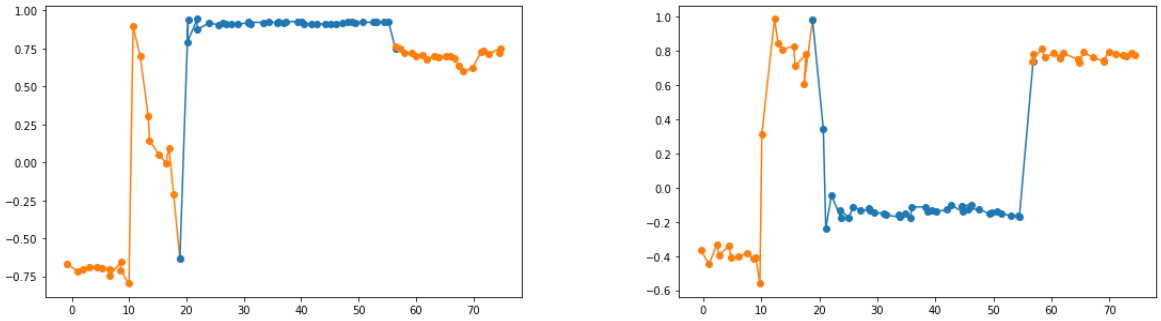


Figure 6.88 LSTM GAN - Examples of Wafer 'Negatives' signals completed (initial and final ends)

Final Conditioning

The part given as conditioning was only the final half of the curve, asking the network to generate the first half.

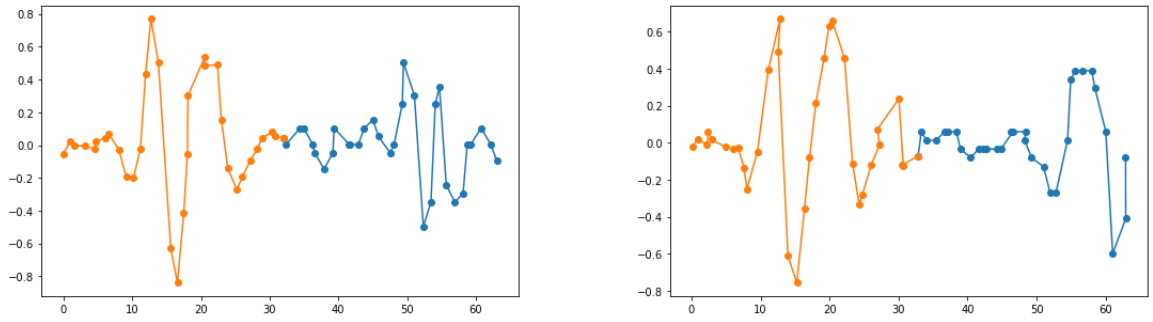


Figure 6.89 LSTM GAN - Examples of Sony 'Class 1' signals completed (first half)

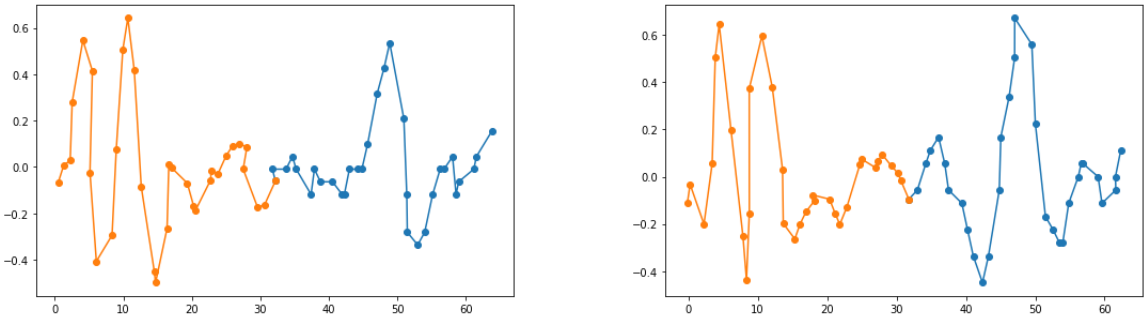


Figure 6.90 LSTM GAN - Examples of Sony 'Class 2' signals completed (first half)

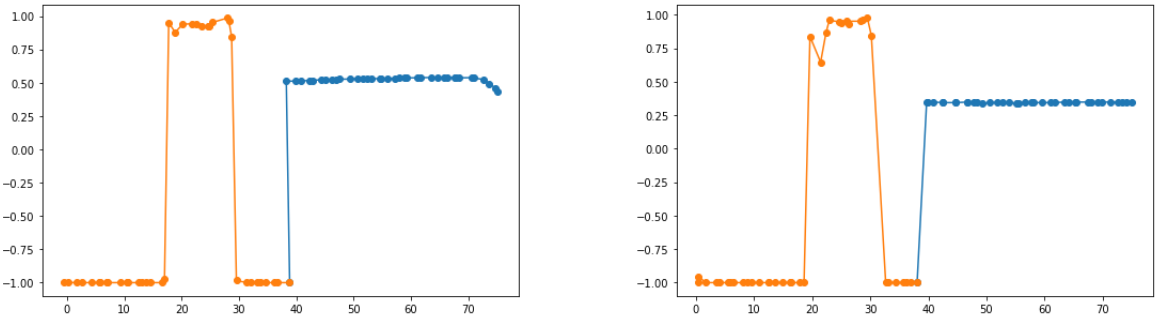


Figure 6.91 LSTM GAN - Examples of Wafer 'Positives' signals completed (first half)

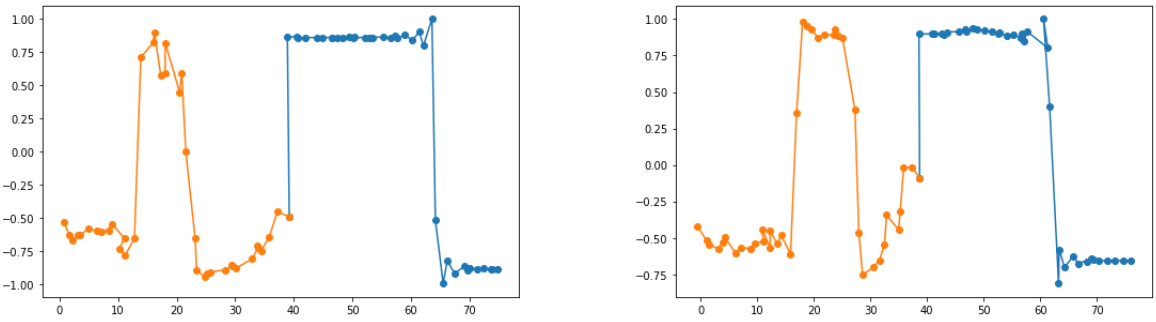


Figure 6.92 LSTM GAN - Examples of Wafer 'Negatives' signals completed (first half)

Performance

The performance were obtained from the classifier by comparing the two classes of Sony dataset and the two classes of Wafer dataset.

Initial Conditioning	Central Conditioning	Final Conditioning
0.631515	0.661818	0.781212

Table 6.28 AUROC - LSTM GAN - Real Time Series - Sony

Initial Conditioning	Central Conditioning	Final Conditioning
0.759428576	0.791428572	0.773142866

Table 6.29 AUROC - LSTM GAN - Real Time Series - Wafer (halved length)

6.7 Summary of results

To conclude, the comparisons between the proposed approaches are reported. Figure 6.93 shows the AUROC measurements in the case of synthetic periodic time series of T-CGAN [25], in blue, and the LSTM GAN approach proposed in this thesis, in orange. The upper plot shows the average AUROC when the size of the training set for the auxiliary classifier increases. On the other hand, the lower plot depicts the average AUROC when the length of the time series increases. The values are listed in detail in Table 6.21.

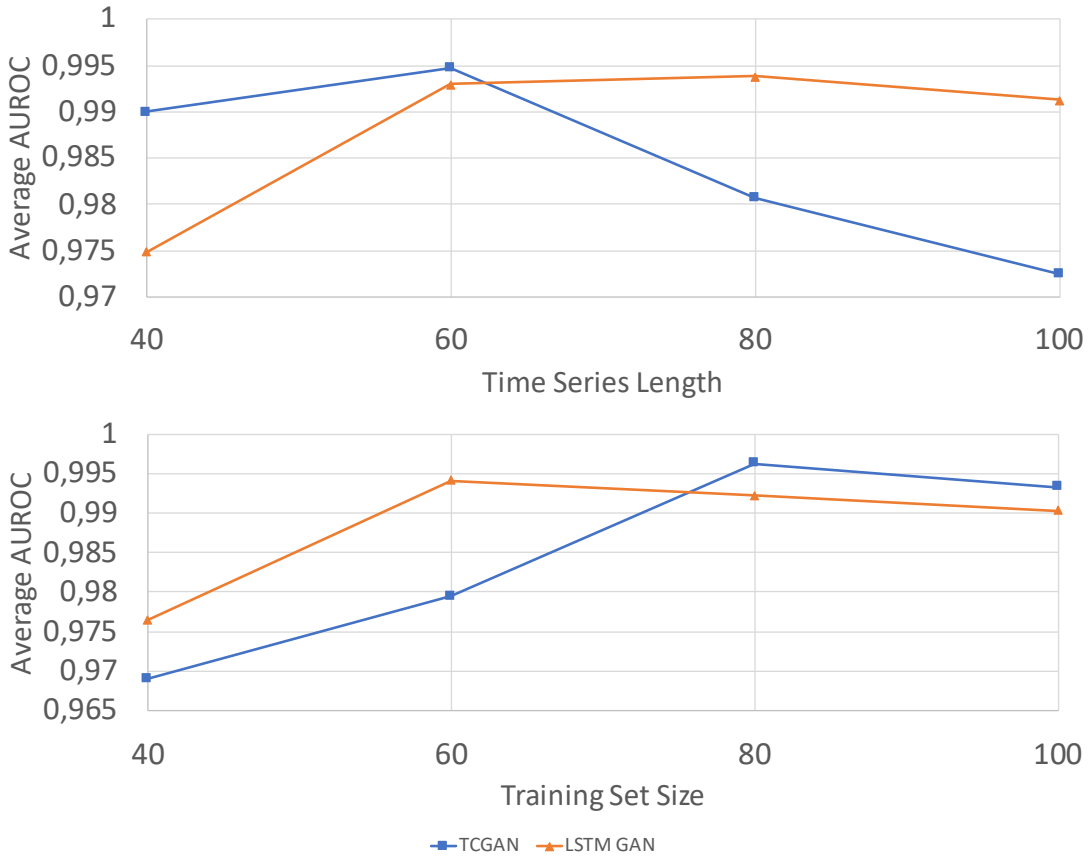


Figure 6.93 Comparison between T-CGAN and LSTM GAN for the complete generation of synthetic, periodic, time series

So, in the case of generating complete time series, T-CGAN performs worse on longer time series than LSTM GAN. The opposite happens if the training set size varies, showing slightly better results for T-CGAN.

In Figure 6.94, comparative charts of the performance of the three proposed methods can be observed. These plots were obtained by averaging the AUROC values for each individual method grouping all the initial, central and final cases. Although the classification

performance of the baseline is acceptable, the visual quality of the completed time series is not on the same level and the original and generated parts of the time series often do not connect correctly.

The results shown in Figure 6.94, as well as the ones in presented throughout this chapter, demonstrate how the proposed method are capable of exceeding the baseline performance. In general the trends of the different kinds of time series are captured successfully. When tested on synthetic time series TA-CGAN gives the most stable results as the time series length and the training set size increase, with the exception of the *Final* scenario. T-CGAN performs rather consistently with longer time series rather than larger training set. The opposite happens for LSTM GAN which has problems with longer time series but a large enough training set allows to achieve good classification performance regardless of the experimental context.

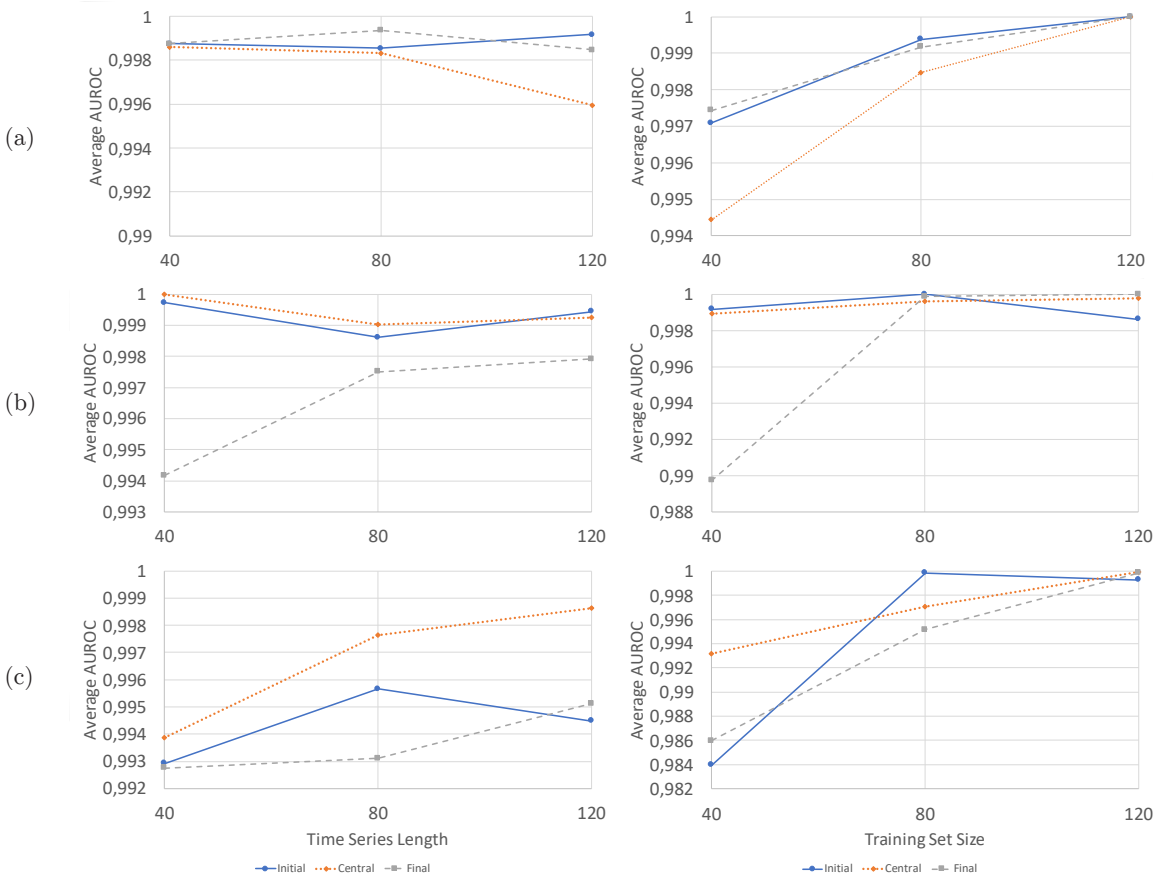


Figure 6.94 Average AUROC measurements for T-CGAN (a), TA-CGAN(b) and LSTM GAN (c) on sythetic data. The three experimental settings Initial (solid, blue), Central (dotted, orange) and Final (dashed, grey) are reported separately.

Chapter 7

Conclusion

This chapter summarizes the work presented in this thesis, outlines the problems and shortcomings of the proposed approaches and suggests some possible future continuations of the study.

7.1 Summary

The work presented in this thesis deals with Generative Adversarial Networks, a framework for estimating generative models through an adversarial process. The goal is to study their applicability in the context of data filling for irregularly sampled time series. Three different methods, as well as the ideas behind each one, have been proposed to address such problem. Two variants exploit Convolutional Neural Networks whereas the third technique is based on Long-Short Term Memory Networks. Leveraging the fundamental properties of the two families of networks the Generator G and Discriminator D have been conditioned with auxiliary information regarding time and the time series to complete. The quality of the results have been assessed via visual inspection and through the performance of an additional classifier trained on the newly completed time series. The methods presented allow to reconstruct the missing parts of different kind of time series, both synthetic and real world ones, regardless of where those gaps happen in the data.

7.2 Threats to validity

When building and experimenting with the proposed techniques few problems became evident:

- *Mode Collapse*: this is a well known problem when it comes to training GANs. This essentially means that, although the real data distribution is multimodal, the Generator ends up producing a limited variety of samples. Take as an example the MNIST dataset which has 10 modes, one for each digit. A collapsed Generator would produce only some, or even just one, of those 10 digits. This problem surfaced during the training TA-CGAN. The produced time series happened to have all the same trend regardless of the original shape.

- *Computational Limits*: despite the fact that we didn't record any precise measure, when training GANs with an increasing number of values, the time required to conclude this phase increased in non-linear way. This especially happened with the TA-CGAN approach where 120 values long time series took few hours of training became a full day of training when using the Wafer dataset, which has 152 values long time series. Moreover, LSTM GAN method requires much more iterations to train than its CNN-based counterparts.

- *Parametrization*: for none of the implemented GANs hyperparameters selection has been performed. This has proven to be particularly detrimental for the LSTM GAN approach which is unable to reliably complete, if at all, time series that are too long. This is the main reason for which the experiments on the Wafer datasets have been performed on sliced time series. Other obstacles with this third technique are the necessity to scale the time series in the $[-1, +1]$ range, the increase in training set size and the reduction of additive noise to have usable results. Experiments were conducted with different parameters but resulted in failures.

7.3 Future works

Future works for these three approaches include:

- Hyperparameter tuning to find better performing models and resolve the limitations of LSTM GAN method.

- Modify these techniques to be able to generate time series with an arbitrary number of samples, different from the input length of the generative model G.

- Study the possibility of extending these approaches to allow for a reliable generation process even outside of the time frame in which observation for the training time series are collected.

Bibliography

[1] Barratt, S. and Sharma, R. (2018). A note on the inception score.

[2] Baytas, I. M., Xiao, C., Zhang, X., Wang, F., Jain, A. K., and Zhou, J. (2017). Patient subtyping via time-aware lstm networks. In *KDD*.

[3] Chintala, S. (2016). How to train a gan?

[4] Cipra, T. and Hanzák, T. (2008). Exponential smoothing for irregular time series. *Kybernetika*, 44:385–399.

[5] Esteban, C., Hyland, S. L., and Rätsch, G. (2017). Real-valued (medical) time series generation with recurrent conditional gans.

[6] Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423.

[7] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

[8] Gretton, A., Borgwardt, K., Rasch, M. J., Scholkopf, B., and Smola, A. J. (2008). A kernel method for the two-sample problem.

[9] Hanzák, T. (2014). Methods for periodic and irregular time series.

[10] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium.

[11] Hoang, C. D. V., Cohn, T., and Haffari, G. (2016). Incorporating side information into recurrent neural network language models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1250–1255, San Diego, California. Association for Computational Linguistics.

[12] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[13] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[14] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

[15] LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–, London, UK, UK. Springer-Verlag.

[16] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2016). Photo-realistic single image super-resolution using a generative adversarial network.

[17] Luo, Y., Cai, X., ZHANG, Y., Xu, J., and xiaojie, Y. (2018). Multivariate time series imputation with generative adversarial networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 1596–1607. Curran Associates, Inc.

[18] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets.

[19] Mogren, O. (2016). C-rnn-gan: Continuous recurrent neural networks with adversarial training.

[20] Moritz, S. and Bartz-Beielstein, T. (2017). imputets: Time series missing value imputation in r.

[21] Moritz, S., Sardá, A., Bartz-Beielstein, T., Zaefferer, M., and Stork, J. (2015). Comparison of different methods for univariate time series imputation in r.

[22] Olszewski, R. T. (2001). *Generalized Feature Extraction for Structural Pattern Recognition in Time-series Data*. PhD thesis, Pittsburgh, PA, USA. AAI3040489.

[23] Pratama, I., Permanasari, A., Ardiyanto, I., and Indrayani, R. (2016). A review of missing values handling methods on time-series data. pages 1–6.

[24] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks.

[25] Ramponi, G., Protopapas, P., Brambilla, M., and Janssen, R. (2018). T-cgan: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling.

[26] Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative adversarial text to image synthesis.

[27] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

[28] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved techniques for training gans.

[29] Vail, D. L. and Veloso, M. M. (2004). Learning from accelerometer data on a legged robot.

[30] Vondrick, C., Pirsiavash, H., and Torralba, A. (2016). Generating videos with scene dynamics.

[31] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535.