AI/ML Techniques for 5G Coverage Drop Prediction

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

David Reiss de Fez

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS **ENGINEERING**

Advisor: Anna Umbert Juliana

Barcelona, June 2022

Abstract

This project introduces *Machine Learning* techniques to the wireless communication environment. In particular, the objective is to predict when the 5G network will not give coverage to a certain *User Equipment*. A sort of different techniques could be used to solve this, but the solution provided in this document uses a special type of *Artificial Neural Network*: The *Echo State Network.*

As will be seen through the document, the project will focus on predicting the 5G *Quality Parameter,* and by introducing the *Artificial Neural Networks* it is expected for the solution to be able to predict not just the immediate next value, but the next five to ten values. On the results section a comparison between how the final model predicts the 5G *Quality Parameter* on different time instances will be provided. Finally, on the conclusions, the objective accomplishment will be discussed.

Resum

Aquest projecte té com a objectiu introudiur tècniques de Machine Learning en l'entorn de les comunicacions inalàmbriques. En concret, es vol predir quan la xarxa de 5G no serà capaç de donar cobertura a un cert usuari. Moltes tècniques, tant dins com fora del Machine Learning, es podrien haver utilitzat, però la solució donada en aquest document s'ha fet a partir d'un tipus de Xarxa Neuronal Artificial: les Echo State Network.

Com es podrá veure a mesura que avancem pel document, l'objectiu serà predir el nivel de qualitat donat per la xarxa 5G. Introduint les xarxes neuronals artificials, el que es vol és que el model final sigui capaç de predir, no només l'instant posterior al valor rebut, sino els cinc o inclús els deu futurs valors. A la sección de Resultats veurem diverses comparatives entre la predicció donada pel model i els valors reals. Finalment, a les conclusions una breu discussió sobre l'acompliment dels objectius es durà a terme.

Resumen

Este proyecto tiene como objetivo introducir técnicas de Machine Learning al entorno de las comunicaciones inalámbricas. En particular, se quieren predecir esos momentos en los que la red 5G es incapaz de dar cobertura a un cierto usuario. Muchas técnicas, tanto dentro como fuera del Machine Learning, se podrían haber usado, pero la solución propuesta en este documento se basa en un tipo concreto de Red Neuronal Artificial: las Echo State Network.

Como se podrá observar a lo largo del documento, el objetivo será predecir el nivel de calidad futuro de la red 5G. Haciendo que el modelo esté basado en una Red Neuronal Artificial, se espera que pueda ser capaz de predecir no sólo el valor posterior al recibido, sino también de los cinco a los diez valores posteriores. En la sección de Resultados, diversas comparativas entre la predicción del modelo y el valor real se harán, y finalmente, en la sección de Conclusiones se hará una breve discusión sobre el complimiento de los diferentes objetivos del proyecto.

Acknowledgements

I want to recognise the work done by my supervisor Anna Umbert Juliana. She has guided me towards the good direction on some complex moments on which crucial decisions had to be made for the benefit of the project. Providing me different articles, papers and documentation that brought me some of the main ideas this project has been based on. Furthermore, I would like to thank the UPC for the provided software licenses and resources, without which the project would not have been viable.

Revision history and approval record

Revision	Date	Purpose
0	16/06/2022	Document creation
1	19/06/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
David Reiss de Fez	david.reiss@estudiantat.upc.edu
Anna Umbert Juliana	anna.umbert@upc.edu

Written by: David Reiss de Fez		Reviewed and approved by:	
Date	21/06/2022	Date	dd/mm/yyyy
Name	Degree Thesis	Name	Zzzzzzz Wwwwwww
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

1. Introduction

Mobile communications systems have been evolving in a revolutionary way since their inception, becoming an essential element in today's society.

5G is the new technology that is being deployed all over the world, and at the same time improving it is one of the first objectives to take us to systems Beyond 5G.

Due to the increasing potentiality the Artificial Intelligence and the Machine Learning are getting to, endowing the mobile communications networks with intelligence is on the first places of the future objectives list.

1.1. The Objectives

The aim of this project is to design and implement an algorithm that, based on the measurements received by a given *User Equipment* (signal power level, quality level, …), anticipates coverage problems. On a first instance, the main objective of the project, i.e., the final algorithm, should have been capable of taking a decision based on the predicted values —such as activating a relay, or deviating the signal through some other path so that no coverage drop is perceived— but, due to the short period on which the degree thesis is performed, that part has been removed and left for future lines.

Nevertheless, some subsequent objectives have appeared so that the main one can be accomplished. Some of those include understanding of the *Artificial Neural Networks* in general, and the *Echo State Networks* in particular, and the learning of the QUALIPOC and ROMES4 Replay software, recalling that both the *Artificial Neural Network* and the two software are completely new fields for me.

1.2. Requirements and procedures

Since the objective of the project is, somehow, to introduce *Machine Learning* techniques to the wireless environment, the requirements can be classified in two different blocks.

- Use of Machine Learning techniques suitable for mobile environments, since not all the *Artificial Neural Networks* are appropriate for wireless.
- Use of real data from commercial mobile networks to test the developed algorithm.

Therefore, the completion of the main requirements leads us to: previously study different ANN options to choose the most appropriate model —a research task—, and to have access to Mobile Network Testing (MNT) tools.

The MNT tools available for this project are the QUALIPOC, and the ROMES4 Replay software, both from Rohde&Schwarz company [10], [11].

Moving on to the procedures used, the first part of the project is pure research, as commented before, but, as the project advances the practical part of it arrives. The solution relies on the *Echo State Network* and, to do so the model proposed in [3] has been used. That comes in terms of a MATLAB toolbox which provides a model of a *Deep Echo State Network* with different functions, among those there can be found the ones to train the

network, to set the parameters, to compute the square error between the solution and the target solution, etc.

The procedure is to implement a MATLAB code —that relies on the mentioned model— setting the parameters so that they fit the problematic that concerns us. Some of the default parameters the model included have been modified, but that will later be explained precisely.

1.3. Work plan

Before starting the project, a general schedule needs to be set. It is not complex and can be divided on four Work Packages (WP) which, indeed, are really related with the procedures used.

First WP is to understand the problematic. That seems an easy task but sometimes it is not. On this first part, that should take a month, a study on how *Artificial Neural Networks* worked must carried out. Parallel to that, an introduction to how QUALIPOC and ROMES4 work is done. Not just that, but also the capabilities each software has too. Both things are completely new and require time to be understood. An important step is to start relating both fields to see how could and *Artificial Neural Network* help to solve the problematic.

On the second WP, corresponding to the second month of the project, the model has to be chosen. That means that a critical review needs to be done, and therefore, the chosen model is going to be the used one. On that part several options must be studied: the *Feed-Forward Neural Networks,* the *Recurrent Neural Networks,* the *Echo State Networks,* and some others too. The project development starts at this point of the too.

Third WP of the project (during the third month) consisted of the gathering process. In this part QUALIPOC and ROMES4 are used to analyse the environment and to gather different sequences on which the coverage given by the 5G network decreased. Not just that but starting to test the proposed model is also an objective of this part.

Finally, on the last month, the fourth WP is done. The results are obtained, and the document is written. An evaluation on which are the accomplished objectives and how the project has evolved over time is done on that final part.

2. State of the art of the technology used or applied in this thesis:

2.1. Problematic

The problematic that wants to be solved has its origin on the 5G non-coverage points. Basically, the model must predict when a given User Equipment (UE, from now on) will get to a 5G non-coverage point.

5G is the new mobile communications generation and it is not totally stablished yet. For the full network to be upgraded to the 5G technology the deployment has been divided in two phases. The first one —the one most 5G local networks are still on— focuses on the UE-to-base station connection. It is meant to stablish 5G technology on the signal path between the mobile equipment and the antenna. However, from there on, the treatment of the signal and the way the information is organized is still the one used on the LTE (4G). This first type of 5G networks are called non-standalone 5G. The purpose of the second phase, as you can imagine, is to set a general 5G network substituting the actual LTE network. That second type of network is called standalone 5G. For this project several values from a non-standalone 5G network have been gathered, which is, in fact, the present 5G network on the university campus.

In this project, the QUALIPOC and ROMES software have been used (later explained), those software give access to gather and post-process several of the parameters a given UE can receive. Among those values there can be found the SS-RSRP, the SS-RSRQ, the throughput, the perceived number of cells, the technology the UE is using at that moment, etc.

The question that must be made is which of those brings good information about the coverage given by a network. On a first attempt it seems like the most important parameter is the SS-RSRP (or the signal power level), but let's take a closer look. Let's make a reasoning based on two scenarios: On the first one a given UE is receiving a certain SS-RSRP level and the cell it is staying on is shared by five users. On the second one the received SS-RSRP is the same but now, the number of users has increased up to one thousand per cell and then the received value may not be enough. What gets clarified with that example is that a certain SS-RSRP level may be enough for some scenarios and not enough for some others. Indeed, the information brought by the SS-RSRP is very dependable on the user density. Nevertheless, the conclusion that must be made is not that the SS-RSRP is a value that brings no information; if the UE is receiving a -200dBm SS-RSRP level, it can be perfectly said that the UE is on a 5G non-coverage point.

Let's focus now on some of the other parameters. The throughput, for example, seems to bring good information about the coverage too, but, as before, the throughput itself depends very much on the user density. To acquire reliable measurements on the throughput averaging values on the same point over several days might be needed. The wasted time to measure a hundred points would have been too much. That is not of a high interest since the throughput is not even bringing that much of information of the coverage given by the network. Since the model must always work, it cannot rely on values that

depend so much on the user density (the training process of the neural network would become more complex). Moving on to some of the other parameters, the technology used, and the number of cells perceived by the user are found. Only 5G values are meant to be gathered so the technology used is not of much interest. The number of cells can be useful in those cases in which we are not static. If a vehicle is moving from the city to the mountain, it surely will perceive a decay on the number of cells perceived. That means it is getting to a zone on which there are no 5G base stations, but that is not the most interesting case (search on the 5G base stations distribution on Barcelona can be done easily). A different and more interesting case is that of a pedestrian or a low velocity vehicle moving through an area on which 5G is meant to be stablished. To predict a coverage drop on that second scenario is what interests us the most and where the focus of the project will be put.

The information brought by several parameters has been discussed and most of them have been discarded, however there is still one left, the SS-RSRQ or quality parameter. That is the one that brings good information about the coverage given by the network. It can be seen as a sum up of all the other parameters and it is not dependable on user density —as long as it is indirectly considering it—. If the throughput, and the SS-RSRP are low, the received signal will not be good, so the quality parameter will decrease. On the other hand, if those values raise the SS-RSRQ will increase as well. So, only one parameter seems to be enough to predict when will the UE see a 5G coverage drop.

This is the chosen route taken to solve our problem: Try to predict the SS-RSRQ parameter. Once the future value is obtained, the network can take a decision relying on it.

Further on, the SS_RSRQ parameter wants to be predicted, but the question is how. That is when the *Artificial Neural Networks* come into the game. But before moving on to how an *Artificial Neural Network* works let's explain how can be the different parameters that UE receives gathered and processed, so that they can be finally used to solve our problem.

There has not been a discussion over why the *ANN* will provide a good model to solve the problematic, but some works and projects that have been a source of inspiration have already used *NN* on the wireless field for prediction models and other problematics [4], [5], [7], [8]. Some were using *Recurrent Neural Networks* as its basis for the model [6], [9], and were used as a guide to know those type were useful for the model of this project.

2.2. The software: QUALIPOC, ROMES and MATLAB

Now that the problematic has been introduced let's focus on the data gathering and post-processing part.

On the one hand the QUALIPOC software will be used [10]. QUALIPOC allows measuring the different parameters —related to the wireless environment— that a given mobile is continuously receiving. It can be run simulating several mobile operating modes, i.e., downloading files (DL), uploading files (UL), IDLE state —just connected to the network neither receiving nor transmitting—, simulating a phone call, etc. Among all of them this project focuses just on the IDLE state, since for the main objective there is no need to use any of the other operating modes, even though it can be interesting for a future investigation. Not just the mentioned QUALIPOC's feature is used on this project, but also the capability of recollecting the received data over a certain period on what is called a 'campaign'. A campaign (also called 'job') allows to previously set the mobile operating mode and the time it is wanted to run in that mode. So, if for example, an examination on the downlink (DL) would like to be done for three minutes while moving around the university campus, a well-prepared campaign would be enough to do it. On the following sections it will be seen that the mobile was run in IDLE state and with a long-time value for the campaign. The idea behind is to gather different sequences on which the starting point is a high SS-RSRQ value and end with a non-acceptable one —or inverse sequences too, since they can be inverted—. Once different physical paths that lead to a non-acceptable SS-RSRQ value are found, several campaigns to gather all those sequences can be started, since they are valid to train our neural network (explained below).

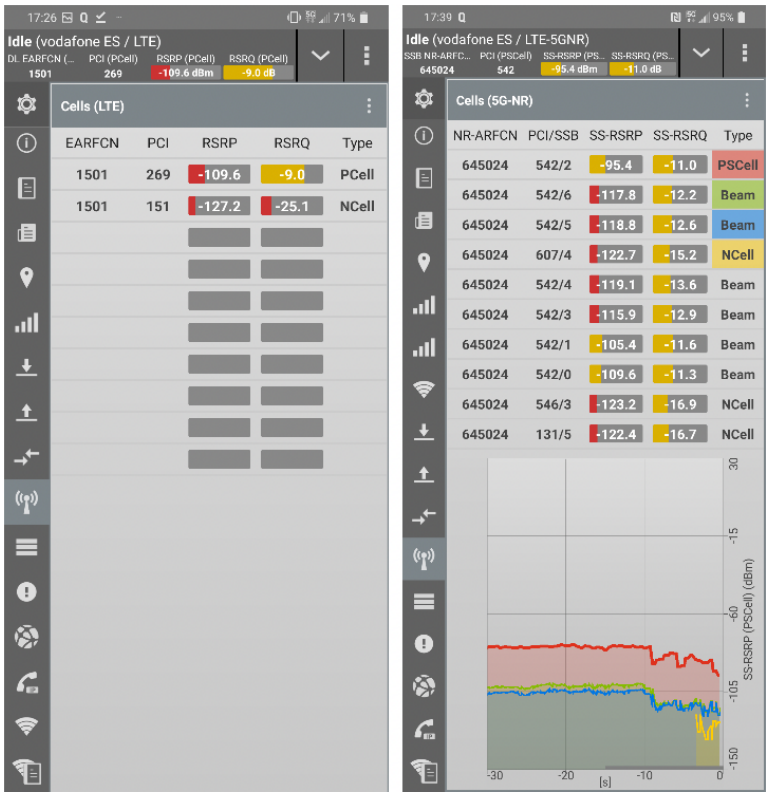

Figure 1. QUALIPOC environment; LTE values on the left-hand side and 5G values on the right-hand side.

On the other hand, the ROMES software is found [11]. ROMES software allows to post-process the gathered data. The objective is to extract a .csv file since it is the one used for the neural network training process. ROMES allows to do that and a lot more things. The followed path over a real map can be drawn, all the parameters the Qualipoc has received can be visualised (almost a thousand), the data can be plotted in several diagrams, and a lot more features that has not been exploited on this project. Therefore, the important feature is to extract .csv files.

Once the data can be gathered (Qualipoc part) and post-processed to obtain .csv files (ROMES part), it is also ready to be transferred onto MATLAB software where the *NN* is ready to be trained. This last step will be the topic of the next section, preceded by a general explanation on *Artificial Neural Networks (ANN)*, and a specific explanation on *Echo State Networks (ESN)*.

2.3. Artificial Neural Networks

2.3.1. Introduction

To explain how the *ANNs* work, it is necessary to explain a bit on *Artificial Intelligence (AI)* and *Machine Learning (ML)* fundamentals. *AI* objective is to give a machine the capability of taking decisions by itself, i.e., to endow them with intelligence. In contrast to the *ML*, on the *AI* framework you can also give the machine the rules (or algorithms) that must obey, e.g., the Pac-Man video game can be considered *AI* since the machine is chasing you by itself, but the algorithms that obeys have been given to it. So basically, there is an input, an output, and the rules between those can be given. When moving on to the *ML* framework, which is instead a part of the *AI*, there is also an input and an output but, the objective now is for the machine to find out which are the rules that relate those two. *ANN* belong to the *ML* framework. The process of learning is done on what is called the training part, on which the *ANN* will need to adjust its different parameters so that the given output matches the target output. In this project, the supervised learning is used, whose main characteristic is that the target output is known. This allows to generate an error function which will be the starting point of the training process, but this will be explained on the following sections. Some other types of learning exist, such as unsupervised learning (the target output is not known), reinforcement learning or semi-supervised learning.

Now that the basic concept has been introduced, the objective now is to get deepen on how an *ANN* works, the training process and the different types that exist, all this leading to an introduction to the *ESN.*

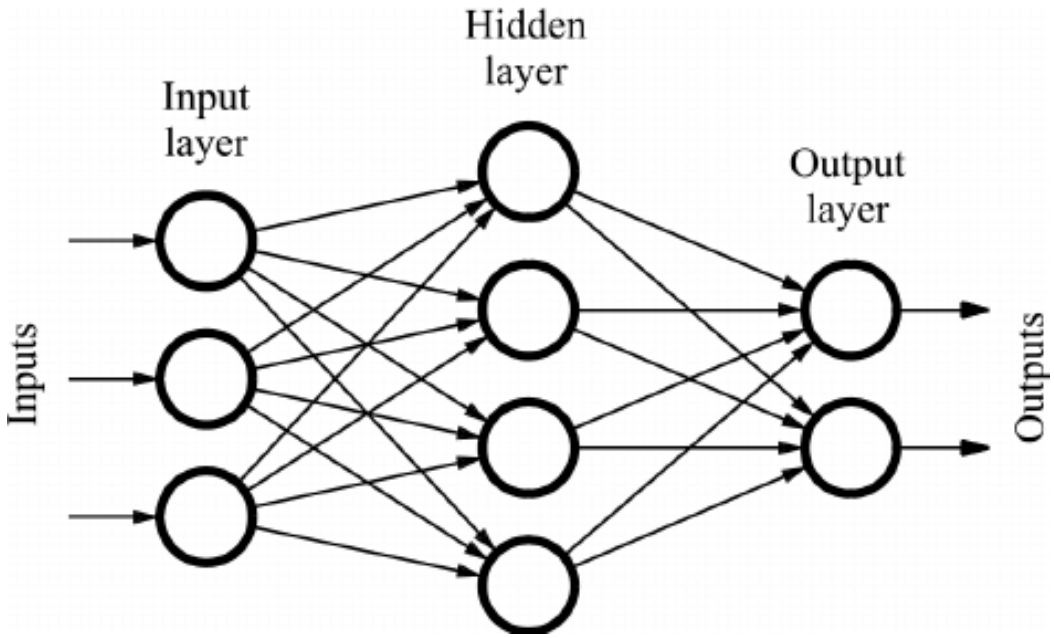

Figure 2. Feed-forward Neural Network.

The conceptual idea of an *ANN* is to mimic the brain, that is why the basic structure of an *ANN* follows the one shown in Fig 1. As it can be seen, the basic units, which are called *neurons*, are connected to each other. This simulates the brain neurons connections which are kind of similar but not the same, recall that the *ANN* are just inspired on the brain. The Fig 1. neural network is a *Feed-forward Neural Network (FNN),* which is, in fact, the most basic one since each neuron layer is connected just to the next one, with no loops or special connections. Now, let's define how the network mathematically works.

2.3.2. Mathematical Description

As observed in Fig 1., there is an input layer, a hidden layer, and an output layer. On the input layer the network is fed with the desired inputs, which are, the ones thought to bring good information for the neural network to perform correctly. Then the hidden layer is found, on which the learning is done. Finally, the output layer gives the output of the network. The connections between neurons on different layers are represented by a weight value (that is adjustable to optimize the network) that represents the strength of a given connection. When more than one neuron per layer exists —which is always—, the total number of connections increases too, indeed, there is a connection between one neuron and all the next layer neurons. That is why the so-called weight matrices are defined to compactly express the strength of all the neuron-to-neuron connections. The input of a hidden layer neuron is given then by the expression:

$$e_i = b_i + \sum_{j=1}^{N_{input}} w_{ij} \cdot u_j \qquad (1)$$

where e_i is the input of the $\mathrm{i}-\mathrm{th}$ hidden layer neuron, u_j is the $\mathrm{i}-\mathrm{th}$ input of the network and b_i represents the bias of the $\mathrm{i}-\mathrm{th}$ hidden layer neuron. The bias states around which point the input values will oscillate, i.e., if a network's most efficient range is from zero to one and the set of inputs range is from four to five, the bias should be set equal to -4. Anyway, if the set inputs do not have a defined or bounded range it can be set randomly at the initial time and then is going to optimized on the training part. w_{ij} represents the different weight values between the $\mathrm{i}-\mathrm{th}$ hidden layer neuron and the different input neurons. N_{input} can also be defined as the number of input neurons (network inputs). So, if the network has more than one hidden layer neuron (always), the connections among input and hidden layer neurons are defined by the input to hidden weight matrix. Equation (1) can be then expressed in matrix notation as:

$$\boldsymbol{e} = \boldsymbol{W^{in}} \cdot \boldsymbol{u} + \boldsymbol{b} \tag{2}$$

Equation (2) gives the vector that contains the input of all the hidden layer neurons, $\boldsymbol{e}$. $\boldsymbol{W^{in}}$ stands for the input to hidden weight matrix and $\boldsymbol{b}$ represents the bias vector which contains all the biases of all different neurons. To clarify the equation (2) let's express it this way:

$$\begin{pmatrix} e_1 \\ \vdots \\ e_{N_{hidden}} \end{pmatrix} = \begin{pmatrix} w_{11} & \cdots & w_{1N_{input}} \\ \vdots & \ddots & \vdots \\ w_{N_{hidden}1} & \cdots & w_{N_{hidden}N_{input}} \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_{N_{input}} \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_{N_{hidden}} \end{pmatrix} \tag{3}$$

It can be observed that the input to hidden weight matrix is on the $\boldsymbol{R}^{N_{hidden}\ x\ N_{input}}$ space, where N_{hidden} is the number of neurons of the hidden layer. There is also another more compact and frequently used equation to describe (2) on which, instead of having the sum of the bias vector, this last one is added as the first column of the $\boldsymbol{W^{in}}$ matrix and a one also to the input vector. The result is the same and the equation becomes more compact:

$$\boldsymbol{e} = \boldsymbol{W^{in}} \cdot [1; \boldsymbol{u}] \tag{4}$$

Next step is to explain the process that occurs inside the neuron. Once the input of a neuron is defined and, again, trying to mimic the brain neurons behaviour, a decision must be made: does that neuron output some value or not? This decision is made with the activation function. The activation function decides how much a neuron will activate or not for a given input. The easiest case is having a step function. What happens there is that, if the input of the neuron is higher than zero then the output will be one, if not, the output will be zero. This case is easy, but its applications are really restricted. Looking closely on the characteristics of using a step function, there are not that much of advantages since the only possible neuron outputs are '0' or '1' and that limits the network capacity. That is why other activation functions are preferred. Linear ones may seem useful, in fact they are to some applications but, they are not bounded, and this would mean that for a high input

value a high output value would be obtained and that is sometimes not a desired behaviour. The most used activation functions are typically the *tanh* and *sigmoid* functions, which are non-linear and bounded functions. So, the output of the hidden layer neurons is given by:

$$\boldsymbol{s} = f(\boldsymbol{e}) \tag{5}$$

Where f is the activation function. Having the vector of outputs $\boldsymbol{s}$, now the output of the network can be computed, which, as it can be imagined, that is the multiplication of the hidden layer output vector with the hidden to output weight matrix:

$$\boldsymbol{y} = \boldsymbol{W^{out}} \cdot \boldsymbol{s} \tag{6}$$

On the supervised learning framework, the learning process is done by computing a cost function which basically is the difference between the target output and the network given output (an error function optimization). The objective is to minimize the distance between both outputs. That is performed by adjusting all network weights and biases. Looking deeper and developing equation (6) —development performed on the Annex A—, expressing the output of the network as a function of the weights and biases is possible. This allows to change the performance of the neural network and, consequently, reducing the cost function. This is done by the gradient descent or backpropagation algorithms. For a complete explanation of these algorithms and a deeper introduction on *ANNs* [1].

Now that the basis of the mathematical process behind a neural network have been explained, the *Echo State Network (ESN)* will be introduced in the next section, for the differences it has in comparison with a normal *ANN*.

2.3.3. A Detailed Explanation on how ESN Work

In this section of the document, a clear explanation on how the *ESN* works is pretended to be given as it is fundamental for the whole project to be understood correctly.

On the one hand *ESNs* belong to the *Recurrent Neural Network (RNN)* domain. *RNN* are characterized by its intrinsic ability to analyse time series, such as mobility prediction, speech recognition, etc. To perform so well with the time dependent inputs, *RNN* are constructed a bit different to the usual *FNN*. While *FNN* have no loops, on the *RNN* some of those are introduced on the hidden layers. So, instead of having a structure like that in Fig 1., *RNN* follow a structure like that shown in Fig 2. As it can intuitively be thought, adding loops to a neural network will introduce some difficulties when the training part of the process arrives. Since each output is not just dependent on the immediate input but on other past inputs too, no normal algorithms like gradient descent or backpropagation can be used. Other more complex methods like backpropagation through time or real time training need to be used and that increases the complexity and the computational cost a lot.

On the other hand, *ESN* are on the *Reservoir Computing (RC)* framework. On the *RC* a 'reservoir' is always defined. A reservoir is a group of fixed parameters that define the structure of the proposed model; it can be composed by several scalars, vectors, matrices, etc. *ESN's* reservoir consists of one scalar, which is the leaking rate, and two matrices, which are the input-to-reservoir and recurrent weight matrices. On the *ESNs* domain the hidden structure will not be referred as hidden layer but as reservoir. This change is important; it makes no sense to represent the hidden part with a layered structure since no order is followed there. The reservoir neurons receive inputs from the input layer but also from other reservoir neurons and its outputs go to any of the other ones and to the output layer neurons. So, instead of representing the hidden part as those on the previous section, now they are represented as those in *Figure 2.*

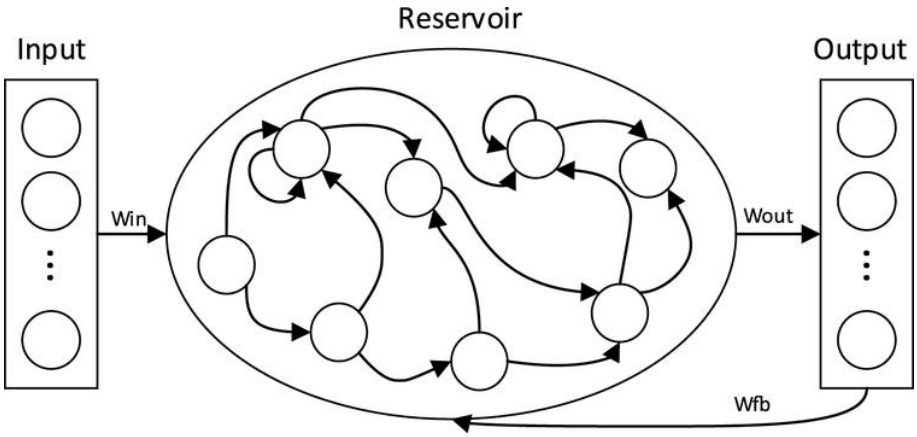

Figure 3. Schematic of an ESN

An important change has been introduced on the *ESN*; the input-to-reservoir and the recurrent weight matrices have become non-trainable matrices since they are fixed. The only trainable matrix will be the reservoir-to-output matrix and that allows to use normal training methods as in the *FNN* case. That was exactly one implicit objective our project had: to introduce recurrence in our neural network and to use normal training methods. Now it is known that imposing those conditions lead us to use an *ESN* as the proposed solution model.

A not commented feature of *ESN* is that seen on the *Figure 2*: the $\boldsymbol{W}_{fb}$**.** That matrix is the feedback matrix, which is sometimes used on the *ESN* model to feed the reservoir with the obtained outputs. It is meant to give more information and sometimes the teacher forcing technique is used. On that technique, the output of the network is modified on purpose to make the reservoir to learn faster, but in this project has not been neither studied nor used.

On the following sections, the general aspects of the *ESN* will be explained as the mathematical description too so that the advantages that the *ESN* model will give can be understood.

2.3.3.1. Global Parameters Definition

As a first contact with the ESN, this section will start by giving a brief list of which are the main ESN's parameters:

- N_u: the number of inputs
- N_y: the number of outputs
- N_R: the number of recurrent units (or neurons that compose the 'reservoir')
- $\boldsymbol{W}^{in}$, $\boldsymbol{W}^{out}$, $\boldsymbol{W}$: the input-to-reservoir, the reservoir-to-output, and the recurrent weight matrices, respectively
- $\boldsymbol{u}(n), \boldsymbol{y}(n), \boldsymbol{y}^{\boldsymbol{target}}(n)$: the input, the output, and the target output vectors, respectively
- $\boldsymbol{s}(n)$: the state vector

The main parameters are close to those on the *FNN*, but there is a new one: the reservoir state vector (state vector, from now on). The state vector contains the output of each of the reservoir neurons and as it will be later seen it plays a fundamental role on the output of the network. Another change appears with the recurrent weight matrix. This matrix substitutes the hidden weight matrix which defined the weights of the connections between the hidden layers. It is called recurrent because of the recurrence that the *ESNs* introduce.

Now an important distinction between the intern parameters and the global parameters must be made. Intern ones have been described above and are those that internally define the network. Global parameters are kind of external parameters and define how the network will train itself, how the inputs will be treated while passing through it, and how will be updated on each iteration. Among those the more important ones are the input scaling, the spectral radius, and the leaking rate. Not just the intern and global parameters define the whole neural network, but some constraints are imposed too. That is what it will be explained on the upcoming subsections as the mathematical description too.

2.3.3.2. Spectral Radius

ESN have so many advantages but, if the network is wanted to work properly the 'echo state property' needs to be fulfilled. Basically, what this property states is that for a long enough given input, the reservoir state should not depend on the existing conditions before that input. As explained in [1], this condition (or restriction) will be ensured if the spectral radius of our matrix is minor to one. The spectral radius of a matrix is the maximal eigenvalue. It is also known that the next equation is ensured:

$\|\boldsymbol{W} \cdot \boldsymbol{x}\| \leq |\lambda_{max}| \cdot \|\boldsymbol{x}\|$, where $\boldsymbol{W}$ is a matrix, λ_{max} is its maximal eigenvalue and $\boldsymbol{x}$ is a given vector.

There it is seen that by forcing the maximal eigenvalue to be lower than one the multiplication of the state vector with the recurrent weight matrix is forced to always decrease. That gives an idea of why that condition makes the echo state property to be ensured; a given input will enter the hidden weight matrix and its impact will be decreasing

on each of the network recurrent iterations, concluding on a zero impact of that input after a certain time. There is an optimal value for the spectral radius, which is in fact the one we have chosen for our model and is 0.9. This has been demonstrated and for further information you can check [].

That is one of the key design aspects, otherwise the whole network would not work properly, since the output values would be based on the initial conditions.

2.3.3.3. Sparse Reservoir and Non-Zero Elements Distribution

One of the constraints *ESN* impose is that the recurrent matrix (reservoir weight matrix) must be sparse. This can be directly related to the spectral radius condition that have been mentioned before. If we want that a given input is forgotten by the network when a period is finished, we need to restrict the connections among the reservoir neurons, otherwise it would be very difficult for the network to not to depend on all the historical inputs.

The usual process is to generate the input-to-reservoir and the recurrent matrices according to a zero mean Gaussian or Uniform distribution. Hence, the input scaling is usually the standard deviation if a Gaussian distribution is used, or the range value if a Uniform distribution is used. In this project case, the Uniform distribution is used with a range value equal to one. That is why the input scaling is set to one on a first instance.

2.3.3.4. Input Scaling

As commented on the previous section input scaling usually relates with the distribution used to generate the specified matrices. But let's look deeper on how it affects to the general working of the network.

The input scaling refers to the parameter the inputs are multiplied by. It is used over all the ML approaches. It allows, for example, to bound our inputs. A non-bounded set of inputs makes the network to have problems when feeding it with high values. It is not necessary for the inputs to be distributed on a [minus infinite, plus infinite] range to cause problems. Having inputs that go from -20 to 20, may not interest us. To fight against that, multiplying all the inputs by 0.2 would be an option, reducing that way the input range to [-4, 4]. That way the network may be working better and efficiently. in the model proposed on this project the initial value for the input scaling was one, but as you will see, it has been modified during the training process to set it to the value that made the model to give better results.

2.3.4. How Does an ESN Mathematically Work

Aforementioned, *ESN* have a new parameter whose impact has not been explained on the general *ANN* section. The reservoir state vector (state vector, for brevity) contains the reservoir information, i.e., the output of each of the reservoir neurons. Not only that, but the state vector and the input vector are the ones that jointly arrive to the output layer. That is

a difference too with the *FNN*: the input is passed to the reservoir but also to the output layer directly. That gives us an intuitive idea of why this neural network is called *Echo State Network*; the general scheme seems like a scenario on which the output layer is receiving a direct signal and an echo or distorted version of it (the one that goes through the reservoir). That is a key aspect when the training part of an *ESN* comes, but before that lets focus on the reservoir, which is the fundamental part of the network since it is the one on which the recurrence is introduced.

To start the reservoir explanation lets first define the recurrent equations:

$$\boldsymbol{s}(n) = (1 - \alpha) \cdot \boldsymbol{s}(n-1) + \alpha \cdot \tilde{\boldsymbol{s}}(n) \; ; \tag{7}$$

$$\tilde{\boldsymbol{s}}(n) = f(\boldsymbol{W^{in}} \cdot [1; \boldsymbol{u}(n)] + \boldsymbol{W} \cdot \boldsymbol{s}(n-1)) \; ; \tag{8}$$

$$\boldsymbol{y}(n) = \boldsymbol{W^{out}} \cdot [1; \boldsymbol{u}(n); \boldsymbol{s}(n)] \; ; \tag{9}$$

New important changes have appeared on those three equations. The first one, that has been already explained, is that, on contrary to an *FNN*, the output layer is fed with the state vector but also with the input vector, in concrete, is fed with the concatenation of both vectors and a '1' which represents the bias terms. Second change is the state vector. Since the recurrence is introduced on the reservoir of the network, it is known that the reservoir state must depend on its past values. The strength among those connections is defined by the recurrent matrix $\boldsymbol{W}$. Let's forget about the first equation for a moment and focus on the second one. What is defined there is an auxiliar state vector that, in fact, seems like the one defined to compute the output of a hidden layer on the *FNN*, but now another term is added to the expression: the past state vector. That is, as imagined, the point where the recurrence is added. A particular case —and the simplest— of an *ESN* is that on which the leaking rate (alpha parameter on the first equation) is set to one. In that case the auxiliar reservoir state vector becomes the reservoir state vector. Is the more intuitive case: feeding the output layer with the inputs and the inputs passed through the reservoir (state vector).

2.3.4.1. Leaking Rate

The equations [] [] have been mentioned, but not the first one. That is the first appearance of the leaking rate. The alpha parameter seen on equation [] is used to make the input to have its major impact on the state vector faster or slower. This can be seen with the extreme cases. First, the leaking rate is set to zero. That is a non-sense itself because we are making the state vector to depend just on its previous state and not on the input too. But instead of setting it to zero, imagine it is set to a value close to zero then, a given input would have an increasing impact on the state vector until the echo state property makes the network not to depend on it. This can be seen from another point of view: if the input $\boldsymbol{u}(n)$ is correlated with $\boldsymbol{u}(n-5)$ and the model is wanted to take that into account, the leaking rate must be a low value so that when the input $\boldsymbol{u}(n-5)$ enters the network coincides at the output layer with the state vector on which the input $\boldsymbol{u}(n)$ has had its major impact —remember that we feed the output layer with the concatenation of the input plus the state vector—. The second case is the opposite one; the input is correlated with the

next immediate input, so the first one has to fastly pass through the reservoir. That makes the leaking rate to be set closer to one. That is why the leaking rate is meant to be set so that the *ESN* matches the dynamics of the set of inputs.

Therefore, the used case is the second one. Since network must predict the following SS-RSRQ value the first value has to pass through the reservoir fast. That is why the leaking rate has been set to 0.9.

To sum up a bit, three of the most important global parameters have been defined. The spectral radius: makes the network to accomplish the 'echo state property'; the input scaling: helps to define the range on which the network is the most efficient, bounding in some cases the set of inputs; the leaking rate: makes the network to match the dynamics of the set of inputs. All those parameters are normally tunned until the optimal is found, as it can be appreciated on the following sections the initial value of some of those have been modified during the training process.

All said, let's move to the training part.

2.3.5. Training on an ESN, the Ridge Regression

As explained before we know that we can use normal algorithms to train an *ESN*. The most used one is to follow the ridge regression solution, which is also the one used on the Matlab Toolbox we will later use to train and test our network. We will now proceed to explain the ridge regression solution.

As we said before we are only training the hidden to output weight matrix, and we know that the equation that relates the hidden layer output and the final output is (9).

Thus, the problem we are trying to solve follows a linear regression model, i.e., we want to find the line (or the plane or hyperplane depending on the number of inputs) that is the closest to all the target points $\boldsymbol{y}^{\boldsymbol{target}}(n)$ to use it as our prediction basis. Basically, we want to find the $\boldsymbol{W}^{out}$ that minimize the square error between $\boldsymbol{y}(n)$ and $\boldsymbol{y}^{\boldsymbol{target}}(n)$. That is normally performed by solving and overdetermined system of linear equations:

$$\boldsymbol{Y} = \boldsymbol{W}^{\boldsymbol{out}} \cdot \boldsymbol{X} \tag{11}$$

For brevity on the nomenclature, we have added the temporal dimension to that equation. The $\boldsymbol{X}$ matrix, is on the $\boldsymbol{R}^{(1+N_{input}+N_{hidden})x\,T}$ space where T represents the total discrete time samples we are feeding the *ESN* with —the length of the input sequence—. Same happens with the $\boldsymbol{Y}$ which represents the output sequence our neural network will give us, this means is on the $\boldsymbol{R}^{N_y\,x\,T}$ space, where N_y is the total number of outputs.

The ridge regression solution, also known as regression with Tikhonov regularization, says that $\boldsymbol{W}^{\boldsymbol{out}}$ is given by the expression:

$$\boldsymbol{W}^{out} = \boldsymbol{Y}^{target}\boldsymbol{X}^{T}(\boldsymbol{X}\boldsymbol{X}^{T} + \beta\boldsymbol{I})^{-1} \quad (12)$$

Where β is a regularization coefficient whose main purpose is to counteract the effect of large weights values which exploit and amplify differences among different dimensions of the network.

In our case the regularization term has been set to zero, but as many others is a tuneable parameter. For deeper information on ridge regression and on *ESN* general description please check [2].

3. Methodology

The aim of this section is to clarify and sum up the development the whole project has gone through. This will be a guide of the project development section on which the most relevant parts of it will be explained in detail. On the preceding sections an idea of what the methodology is can be extracted, but now let's set a clearer version.

First and most important part of the project is the theoretical background. During the degree there has been no study on the *Artificial Intelligence* nor *Machine Learning* fields. Its applications cover a wide range of different fields on which the Telecommunications Engineering can be found. Although it is not the most exploited sector, the plan for the 6G is to completely rely on an intelligent network, and that is why the objective now is to take little step forward on combining both the Telecommunications and the *AI* fields. As you can see, on this project the prediction of future 5G values is carried out by using *Neural Networks* and not other methods —like temporal series prediction—.

Since the objective of the project is clear the first necessity that needed to be covered was the lack of *Artificial Neural Network* theoretical study. Studying which different types of *Neural Networks (NN)* exist and which provide a better model for a specific task was the first part of the project. An implicit task relies there; how does an *ANN* mathematically work, and how can it be re-designed so that it adapts better to a specific environment. Once the model is chosen a step forward can be done. That is the next part of the methodology: How the algorithm will be designed, and more important, is there an existing model of an *Echo State Network*?

Well, to answer those questions, a little research on which are the most used software for *ANN* designing had to be done. Among those *TensorFlow* and *MATLAB* were found. The decision was to use MATLAB because it has been used during the degree for several Telecommunications applications and no study on how it worked had to be done —as contrary to the *TensorFlow*—. Hence, the software was chosen, and research on which options on the *ANN* field the software gave us was carried out. That is when a toolbox of an *ESN* was found []. The presented model was a *Deep Echo State Network* —on which no research had been done— but, the different parameters could be modified, so that a one reservoir *ESN* was one among all the options the model gave.

With a theoretical background and a model that matched with the idea, another step was taken: Starting to gather and analyse the 5G signal environment. On this part of the project the main objective was to gather different decreasing 5G quality value sequences. If those were found, then the *ESN* training process could be started, but the first difficulty of the project appeared. The 5G network that is stablished on the university campus is a 5G non-standalone, which means that is not a complete 5G network, but a phase 1 network. Two problems appeared then: The first is due to the low 5G signal the *UE* was receiving. The second one was with the gathering software, the QUALIPOC. The QUALIPOC has an option on which you can force the mobile to use only 5G technology, but the problem is that whenever the 5G signal decreases, it automatically changes to the one that brings the better coverage, which was the LTE (4G). The mix of both problems made the final file, that contained all the different values gathered on a campaign, to be a mix of 5G and LTE signals. That meant that were no valid files. This problem will be further commented in detail on the Project Development section. The path followed subsequently was to train and test our model with LTE values, and left the 5G for the future, on which it is expected

for the 5G to be completely stablished giving an even better coverage than what the LTE network does.

From there on, the LTE values were used to train the *ESN*. A first experiment was carried out; generating decreasing sequences with an aleatory slope to train the model and see whether the network was predicting the results good or not. The results obtained were almost perfect. The network was predicting the testing sequence. That meant that the model works well, and the decision to keep this model until the end of the project was taken.

The next problem was that the quality parameter varied more with a form of a discontinuous function than a continuous function. This will further be explained, but basically that behaviour made the *ESN* prediction to be a bit temporally delayed compared to the target output. That little delay made the network prediction not to be as advanced to the present signal as it was supposed to be. Even though the results were not as good as expected and the prediction task was not totally accomplished some features of the model have been extracted for the future work that can be done.

The total power of the network was not totally exploited since it can be set as a *Deep Echo State Network* instead of a one reservoir *ESN*. Although the gathering process was complicated, the 5G network is not completely stablished and the LTE Quality Parameter is a non-continuous function a solution may exist: Getting higher sample frequency —decreasing the period between each sample of the LTE Quality Parameter—, and getting more precise values, i.e., with more decimals (three or four), would make the sequences to be more like a continuous than a non-continuous function.

Finally, as the final part of the methodology the elaboration of this document is done, trying to sum up all the different things learned and improved, also trying to give a good and clarifying version of the model for the future use of it.

All this and more technical details are commented in detail on the Project Development and Results sections but, hope a good methodology process can be extracted from this one.

4. Project Development

4.1. Echo State Network Model

To start this section the MATLAB model [3] needs to be defined so that the starting point of the practical part is clarified. As said before the used toolbox is a *Deep Echo State Network* model, which can be modified to be a one reservoir *ESN* —or what is called a 'shallow' *ESN*—. In the *ANN* field there is an implicit distinction; all the model that contain one hidden layer are called 'shallow' neural networks, while those with more than one hidden layer are part of the *Deep Learning* and are called 'deep' neural networks —remember that in the *ESN* the hidden part of the network is referred as reservoir, but the idea is still the same—. Hence, the model proposed is not a fixed model since all its parameters can be modified. Among those ones there is the N_l parameter which defines the number of reservoirs the network will have. So, as you can imagine, it will be set to one to have a 'shallow' *ESN* (the one that has been theoretically studied). That is a first step to define the starting point. Some of the other parameter values have been discussed over the project, most of them on the State-of-the-Art section, but to give a clearer version let's define them all:

Parameter	Initial Value
N_u	1
N_y	1
N_r	10
N_l	1
α	1
$input\ scaling$	1
$input\ scaling\ mode$	$byrange$
$bias$	1

Table 1. Initial Model Values

Now that the starting point is clear, let's move on to the first steps to validate the model. First, before going to the university campus to gather different values a first contact with the *ESN* was made. With an invented decreasing sequence, a .csv was defined using the Excel software. That was the input sequence to our *ESN.* Then the next step is to define the input and target output sequences. The model states that each column corresponds to an instance of time. Basically, the network input sequence is a row vector, on which each value (or column) represents the value of the parameter on a discrete time instance. The target output is not a vector but a matrix. Each column of the matrix represents the output

sequence the network must output for the corresponding input of the input sequence. Further on, each row must be an advanced version of the input sequence, this way the network will be trained to predict the future values. To clarify that procedure let's express both the input vector and the target output matrix:

$$u(n) = [u_1, u_2, u_3, \dots\dots, u_n] \tag{13}$$

$$Y^{target} = \begin{pmatrix} u(n+1) \\ u(n+2) \\ u(n+3) \\ \vdots \\ u(n+N_y) \end{pmatrix} = \begin{pmatrix} u_2 & \cdots & u_{n+1} \\ \vdots & \ddots & \vdots \\ u_{1+N_y} & \cdots & u_{n+N_y} \end{pmatrix} \tag{14}$$

As you can see there the network is told for each row to predict one instance ahead of the input signal. In the time domain, that corresponds to a total advance of one second —the sampling frequency of the QUALIPOC is equal to 10Hz—. A second may seem not that much of an advance but moving to the telecommunications domain knowing the '1s' future value of a signal gives a lot of margin to the network to take a decision —which was the original objective—. This was a first approach to what the final model should be capable of, but as it will be seen on the Results section, the obtained sequences will not be as good as expected.

Another important aspect is how does the network give its output, and how a good analysis and evaluation on the obtained results can be done. Well, the model does not just include the main methods to train, validate, design, and test the network, but also an important one which oversees computing the *Mean Square Error* between the target and the real output. The way the network gives its output is in the form of a matrix, which theoretically should be equal to the target output matrix. That allows the user to grab any of the obtained rows of the matrix and play with them in terms of plotting the values to compare them with different sequences that clarify whether the network is performing well or not. In this project, the most used comparison plots are between the output and the target output but, as you will further see, it is interesting to see the cases on which the model has worked perfectly and see how the input signal —the one that is meant to simulate the signal received by the user at the present instance— seems to be a delayed version of the prediction carried out by the network. That is the expected model behavior, but with the Quality Parameter sequences it will be seen that it does not happen practically never. That will be one among other topics of the next section: the unexpected problems.

The results on the first testing of the network are presented on the Results section, but as an advance, those were surprisingly good, and predicted almost perfectly the decreasing sequences the network was tested with.

4.2. Brief code explanation

In this sub-section the used MATLAB code is conceptually explained as it is an important part of the Project Development. As you know, our model relies on the ESN toolbox model. The toolbox has two main classes, the *ESN*, and the Task. First one is where all methods and attributes of the network are defined. Among the methods (or functions) there can be found the initialize, run states, train readout, train and test and some others, but the most important and used one is the train and test function. On that function a combination of all the others is performed to replicate the theory on which the ESN sustains. On the other hand, the Task file is found. That is used to define a specific task. There, a name, an input sequence, and a target output matrix are defined and, furthermore, this class is used to define how the data will be divided, i.e., over the whole data which of it is going to be used for the training, the validation, the design, and the testing parts. That is where the so-called test/training ratio is defined. The implemented algorithm is a combination of both classes.

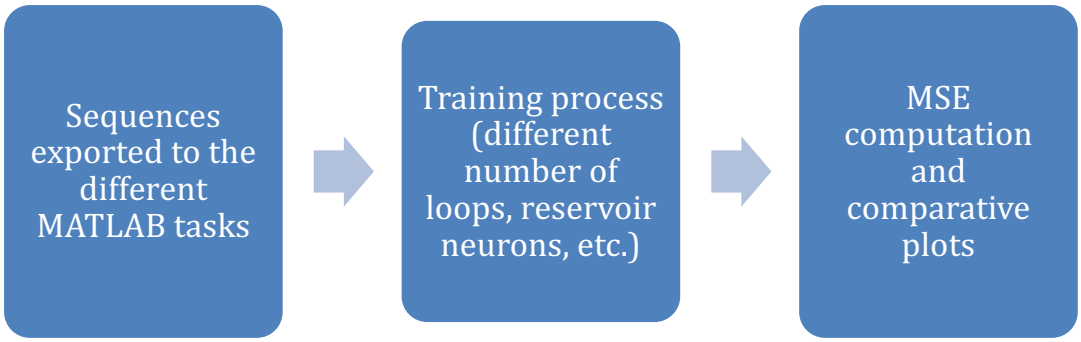

Figure 4. Schematic of the provided code.

First part of the code is thought to be a definition part, on which the different parameters of the network will be set, and several tasks with different sequences will be defined. Once this is defined, next part is the training and testing part. This part was implemented progressively. The first approach was to train just one sequence on which the test/training ratio was equal to 90%. So just the 10% of the samples were used to test the model –recall that the input sequences were 500 samples long -. Then the next move was to train the model with four different sequences. A difference was imposed between the first three ones, and the last one. On the first ones the training ratio was increased so that the 95% of the sequence was used to train the model, but on the fourth one the ratio was decreased to a 50%. That was made to see: first, a long test sequence, and second, how the model reacted. That was set that way for all the mentioned cases that will be presented on this section. For a better understanding on how the training part of the code is built, let's separate in two different modifiable features. The number of sequences is the first of them. That feature has been increased from beginning to end from one to ten. As this value increases the more information the network is fed with and, consequently, a better performance is achieved. The second feature is the number of loops. Instead of training the ten sequences one time a loop is introduced to achieve an even better result. As contrary as with the first feature, in this case increasing the loops to a very high value made the results to get worsen.

After several tries the final value of the total loops was set to ten, and sometimes to 20, but never larger values, because the model performance was affected negatively.

Before moving on to the final part of the code, it is important to remark the role of the washout value. The washout is in charge of making the network to forget about a given number of output values. On the first testing prove, the sequences were of decreasing type but, there was no separation between them as seen from the point of view of the network, so they did not coherently match each other. That made the network to learn to increase its value at the end of the prediction, and that was not a desired response. What it was done then, was to set a washout value equal to one hundred before the last sequence was input. That way it predicted the sequence perfectly. Otherwise, what was happening was that the network was learning that when a given sequence had finished, the next one was going to start at a value around -3. That is why it was increasing its value at the end of the prediction. To clarify this a comparison between a prediction with no washout and one with washout is plotted here.

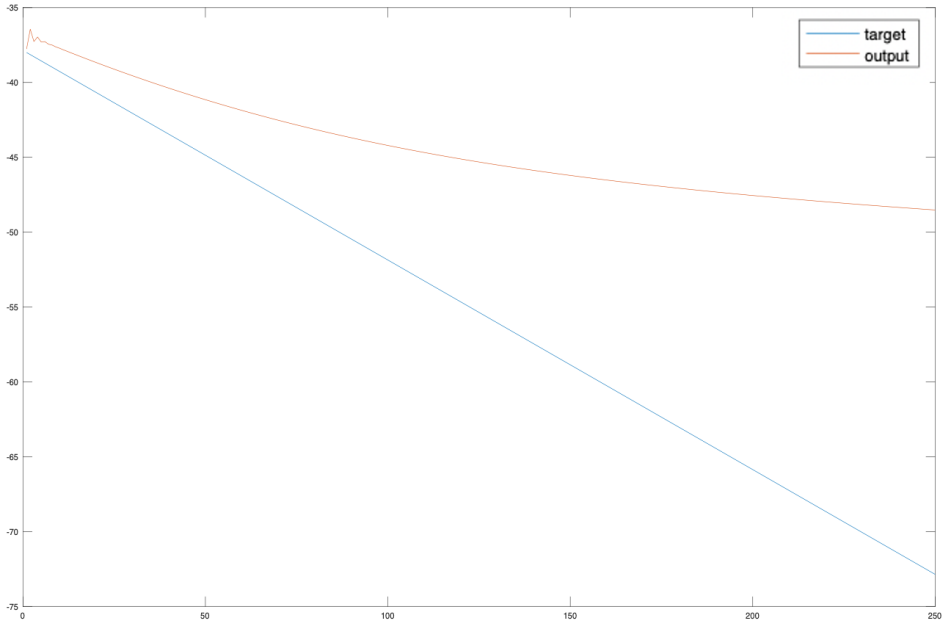

Figure 5. No washout case.

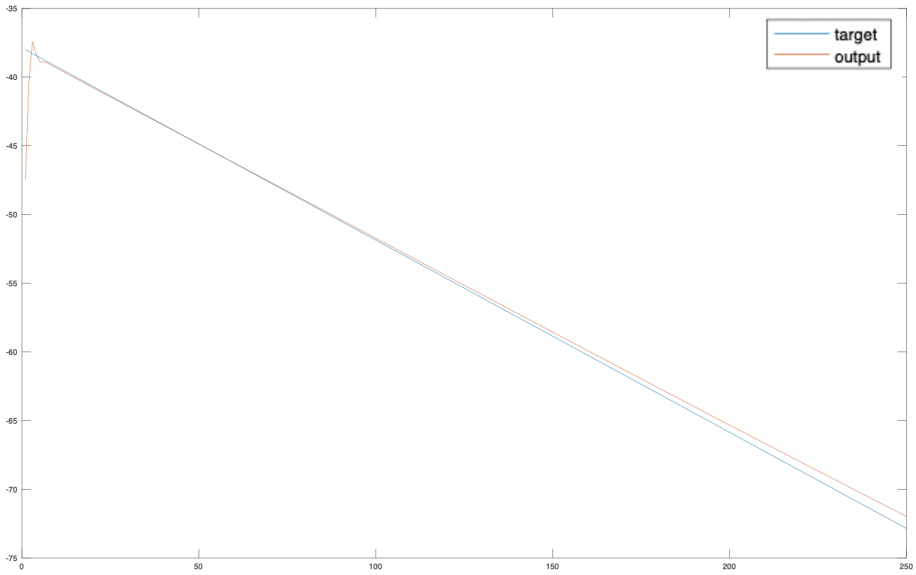

Figure 6. Washout case.

Taking profit of those graphics let's explain the last part of the code, which indeed is, the plotting part. The objective of that code section is to give us tools to compare and evaluate the model results. Plots and MSE values are computed there and presented in the form of the plots shown in this sub-section. The explanation on which sequences are plot and how the comparison is evaluated will be explained in the upcoming sections.

4.3. Data Collection Process

The topic of this section is about the gathering process that has been made to obtain reliable measurements on the 5G network. Not just the way in which different values were gathered but also the difficulties the environment was introducing will both be explained on this section.

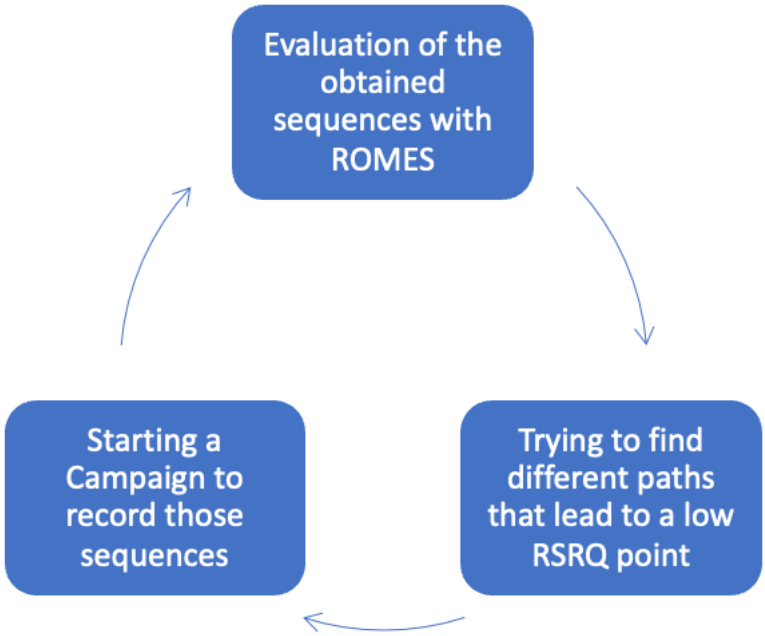

Figure 7. Data collection schematic

To introduce the main section theme let's talk about the QUALIPOC working environment. QUALIPOC gives several network analysing options. It allows to plot, visualize, and analyse all of the network parameters; and one of the most important features, in real time. As you move through the campus it is easy to see how different cells are perceived by the mobile, and the different values each one is giving to it too. Then, the first step would be to set the QUALIPOC to visualize the parameters that are of the most interest. In this project those are: the SS-RSRQ, the SS-RSRP, the number of cells perceived, the technology that is using, and the throughput. On a first approach those were the values that were thought to be useful or probably used on the future training process —at the end just the quality parameter has been used—, and that is why the QUALIPOC was set to visualize them.

Here is when some difficulties were found. The 5G signal is not that good on the university campus, and that made that some days moving to a point on which the previous day existed 5G connection, lead to a point on which there was no 5G. Even though that made the gathering process to be tedious, it is not the major of the problems. First taken steps were to analyse some SS-RSRQ sequences and that is when another problem arises. The files were not made just by SS-RSRQ values, but also form LTE RSRQ values too. The resulting files were a mix of 4G and 5G values. That meant that the UE was changing from 5G to 4G whenever the 5G network did not bring good coverage. This may seem a good thing because it means that when the file starts to be a '4G file' the UE has gotten to a 5G non-

coverage point. The problem is that this happened so many times over short time intervals (3min) that where too short sequences —even less than 50 values—. Not just that, but the pass from 5G to 4G was sometimes totally unexpected, making the transition on a value that other times was just a 5G normal value. Some other route had to be taken, so on the following campaigns, the QUALIPOC was set to force the 5G technology but, as contrary as you may think, forcing 5G was not going to correct anything. The results were files on which, instead of having a mix of 4G and 5G values, they were a mix of 5G values and void values. On those points on which there was no 5G connection the QUALIPOC stopped instead of keeping the recording of the 5G values even though there was no connection to the 5G network. So, basically, no advance had been done. The obtained 5G values were not useful to train the *ESN*. There, due to fact that there was not that much time left to finish the project, the decision was to train our model with LTE Quality Parameter values, which instead of being called SS-RSRQ are called RSRQ. From there on you will see that the project talks about those values and not about the 5G ones anymore.

Until now, only references to the QUALIPOC software have been made, but the QUALIPOC would be useless without the ROMES contribution. The explicit gathering process is to move around the campus doing a campaign (recording the data received) with the mobile that contains the QUALIPOC software. Once this is finished next step is to export those values to an intern folder of the software. Then the mobile is connected to a PC on which the ROMES software is installed, and with the help of it the exported results are opened. Once on the ROMES environment the data can start to be processed. The ROMES gives us several options. Among all of them the used one is the related with the exporting .csv files, as it has been explained on the State-of-the-Art section. Once the files can be exported from the ROMES the only missing step is to feed the model with them.

That is where the gathering process left the project. From here on, the results will be commented in detail giving reasons of why the model has or not performed correctly.

5. Results

This section is the most important on the whole document since it is the one on which the results obtained by the model will be presented. Following the same structure as on the previous sections the order to his one is: results of the first model testing prove, results on the RSRQ prediction case, and a little discussion on the reason of why that difference between both. All said, let's move on.

5.1. First testing results

There existed a necessity on pre-testing the model just in case the results were completely wrong and a change on the route to the solution could still be made but, that is not the case. The results of the first test were almost perfect and are presented below. Recall that on this first graphic the comparison is made between the first row of the target output matrix and the output matrix.

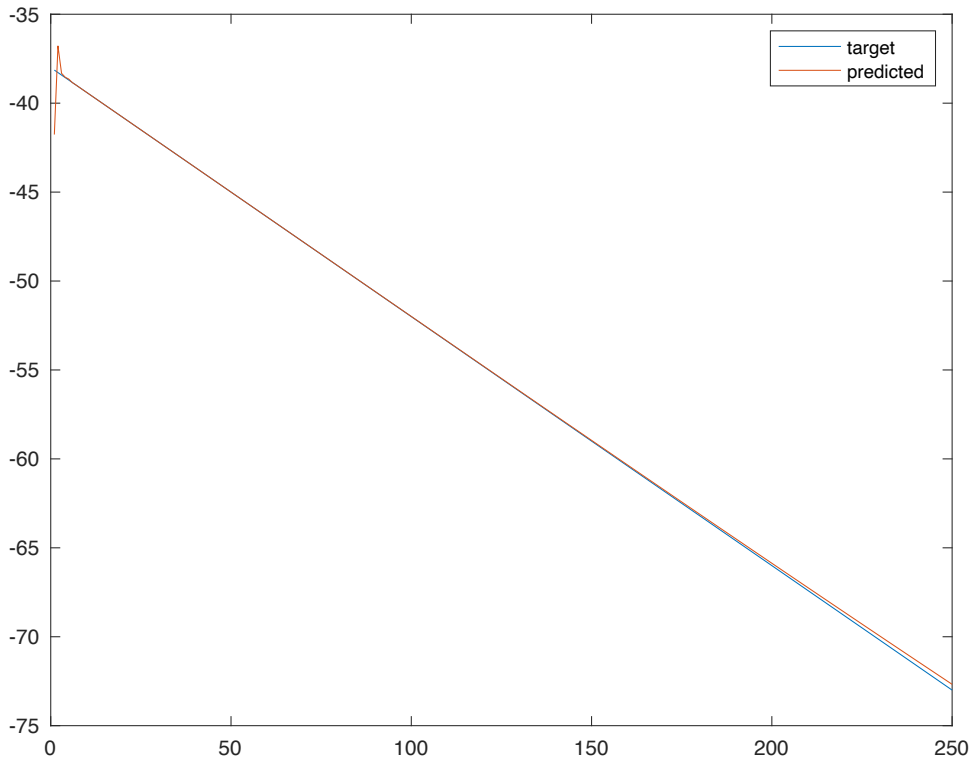

Figure 8. Comparison between the predicted and the target output of a generated decreasing sequence. MSE: 0.067

There, on *Figure 5*, the output of the network is the same as the target output (except for the initial and final values on which a little deviation appears), and the computed MSE value between both signals is equal to 0.067 —which is a much lower value than the ones obtained for the Quality Parameter—. To give another point of view to what the network is doing, let's see the next figure.

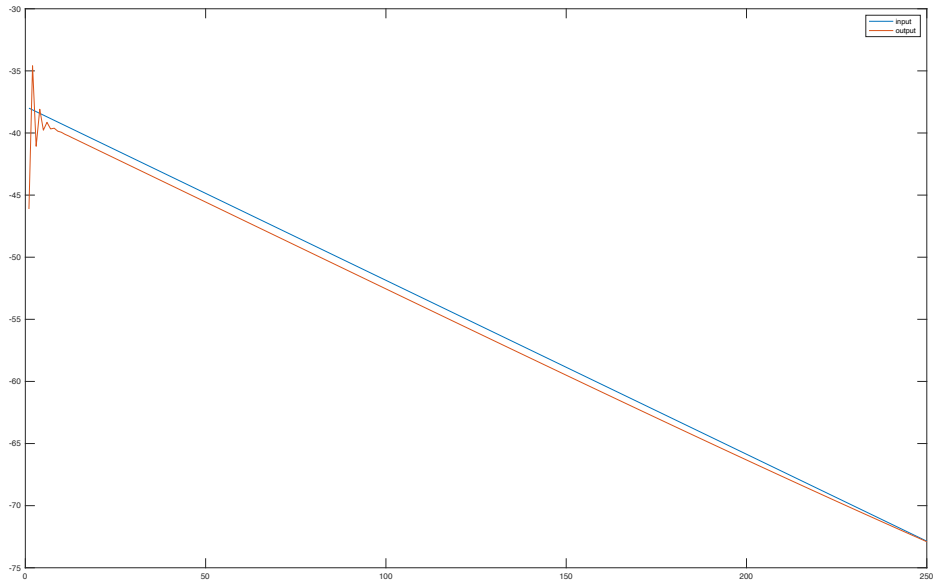

Figure 9. Comparison between the input sequence and the output corresponding to the fifth row of the output matrix

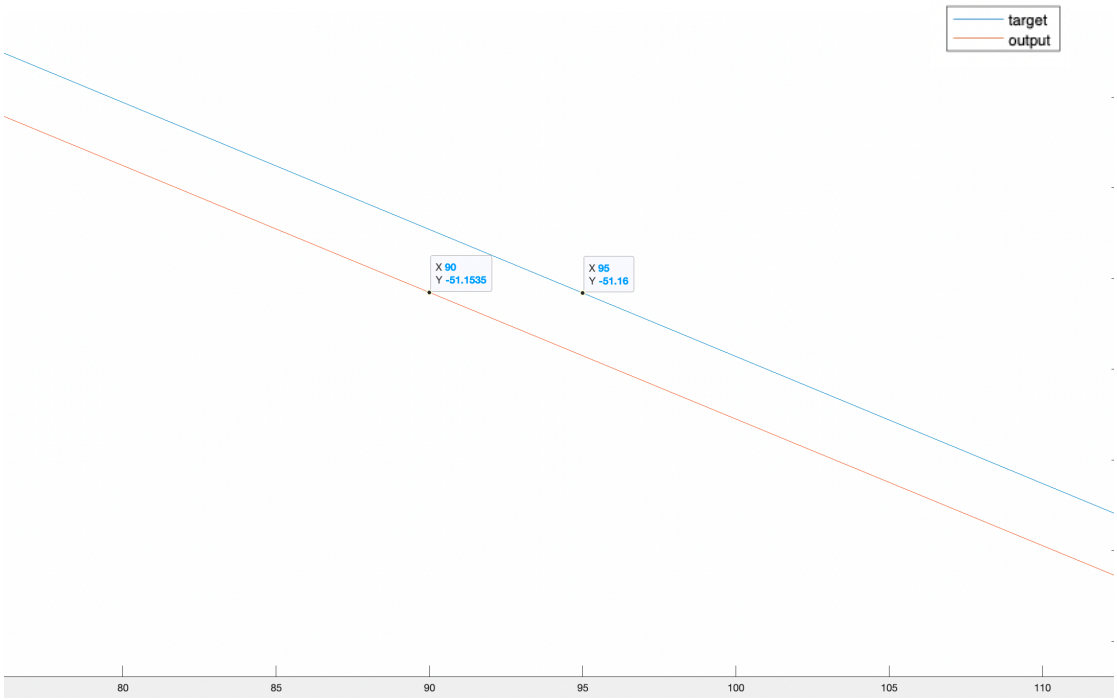

Figure 10. Zoom-in of the central part of the Figure 9

This comparison is a great example of how the model is predicting the decreasing sequence. On the first figure the whole sequence prediction is shown and by making use of a zoom and to markers the relationship between both signals can be seen. After an initial

transient the network predicts almost perfectly the fifth future value of the sequence —in the zoom-in figure, the 90th value of the prediction is equal to the 95th value of the input sequence.

The result of this experiment is of high importance. The network can predict decaying sequences. Of course, the Quality Parameter sequences were not expected to be as perfect as a line, but at the first moment they were thought to be almost a linearly decreasing function type. In the following parts of this section, it can be appreciated that the Quality Parameter is more like a non-continuous function that takes steps from one value to another. It does not follow a linear decaying function and that has been the highest difficulty introduced to the network. That combined with the above-mentioned difficulty to gather data and to find physical paths that lead to a non-coverage point, made the model not to predict the Quality Parameter sequences in a proper way. Concretely, the problem with the predicted values is that the network does not work as fast as it should to have enough advance on time to consider it a prediction. To evaluate this properly let's first show the obtained results with the same comparisons already done in the first case.

5.2. Results on the RSRQ sequences

5.2.1. Modifications on the initial values

Parameter	Modified Value
N_u	1
N_y	1
N_r	10
N_l	1
α	0.9
$input\ scaling$	0.1
$input\ scaling\ mode$	$byrange$
$bias$	1.5

Table 2. Modified Values Before RSRQ Testing

The observed modifications were part of the training process. The initial values made the model to perform not so good, and something had to be done. The modified values were obtained after a trial-and-error process on which several values were given to the different

parameters. At the end of the process the better performance was given when the model parameters were set to the ones shown above.

5.2.2. Results

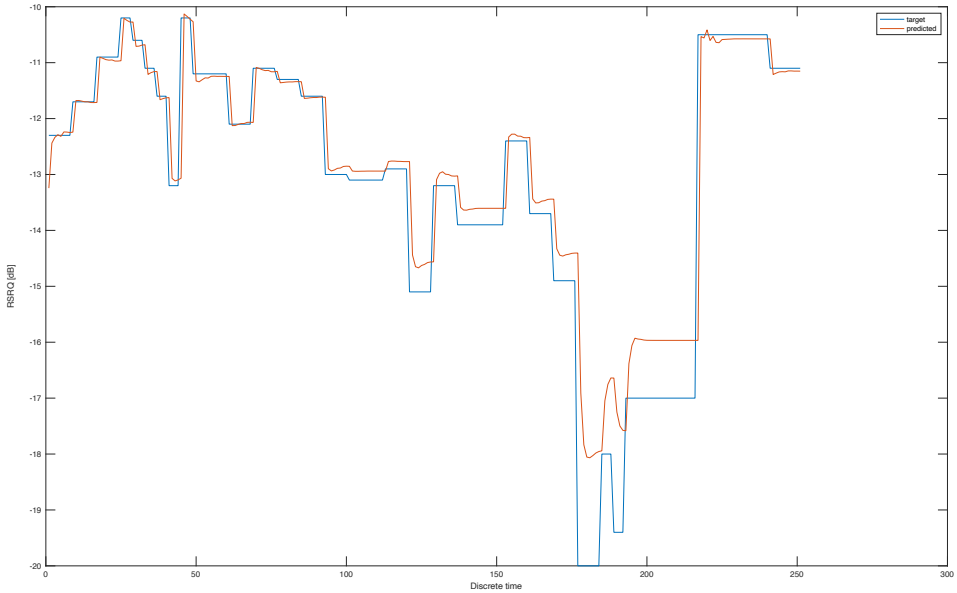

Figure 11. First row of the target output matrix compared with the first row of the output matrix. MSE equal to 0.6

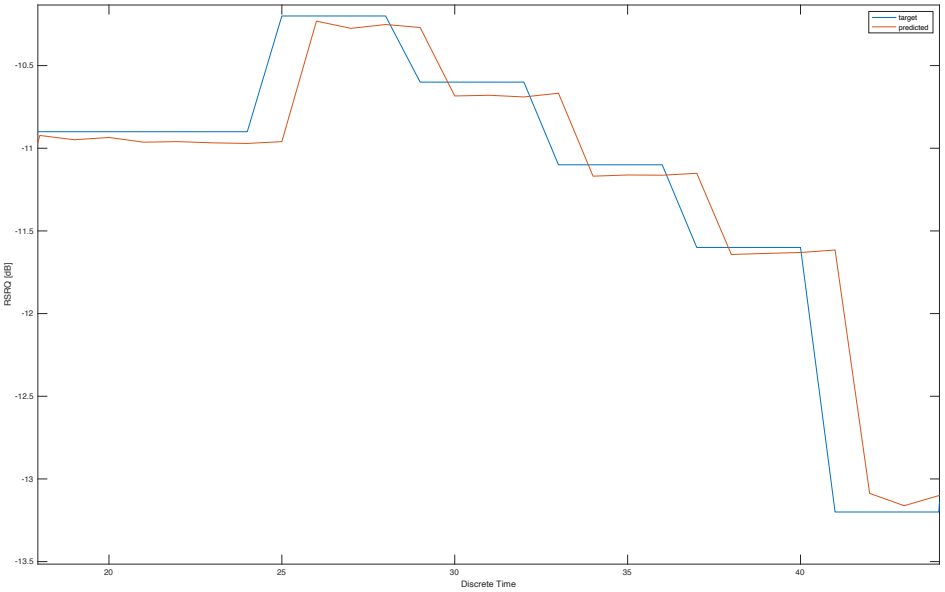

Figure 12. Zoom-in of the Figure 11 first decreasing sequence.

On those first plots, a mismatch in two aspects can be perceived. First of them is the time mismatch. The predicted sequence is a delayed version of its target sequence, and this

should not happen. That makes, as seen on *Figure 7*, that when the comparison is made with the input sequence no time advanced is done. The other mentioned mismatch is the one with the values, but in this case the mismatch is not that much, except for the huge low values perceived between the 170 and 220 discrete time samples on *Figure 5*. The worst of the mentioned mismatches is the temporal one because makes the prediction to be impossible. To see the importance of it lets plot the comparison between the input sequence and the first row of the output matrix.

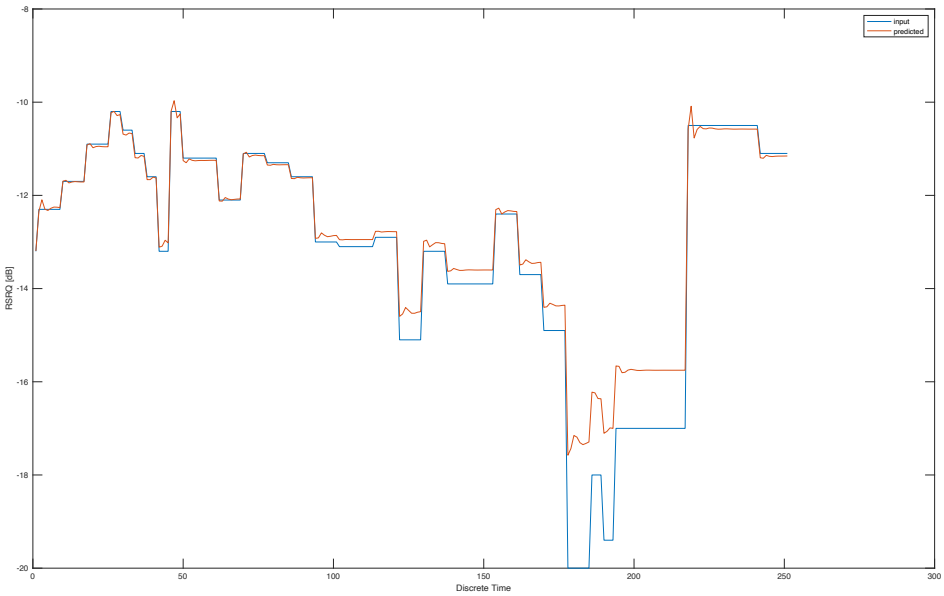

Figure 13. Comparison between the input sequence and the first row of the output matrix

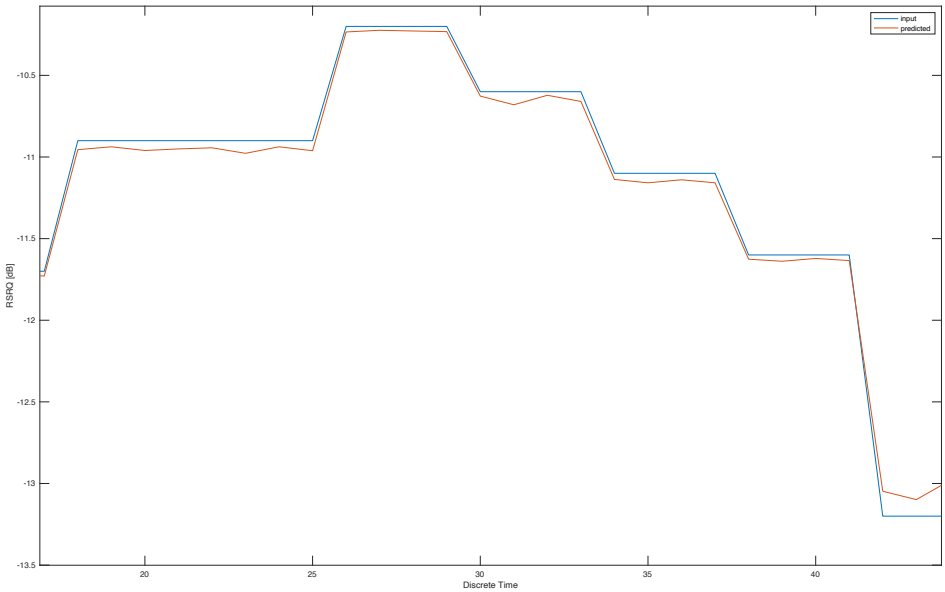

Figure 14. Zoom-in of the Same decreasing sequence of the Figure 11 but compared with the input sequence.

It can be appreciated on *Figure 7* and *Figure 8* is that the predicted values are accurate, but the accomplished time advance is not enough. In the zoom-in figure the predicted decreasing sequence is just a bit ahead of the input sequence, but in the increasing sequence —from samples 0 to 25—, the prediction is delayed in comparison with the input sequence, so that is a failure of the model. It cannot be said that a prediction is made. Recall that although it seems like the sequences are linearly decreasing form one level to another, that is just how MATLAB plots the discrete sequences but if a look is taken to the Annex B, where the sequences are shown in a table way, there is not an intermediate value between one level and another. Moving on when the results obtained for the fifth advanced value are plot things get worst.

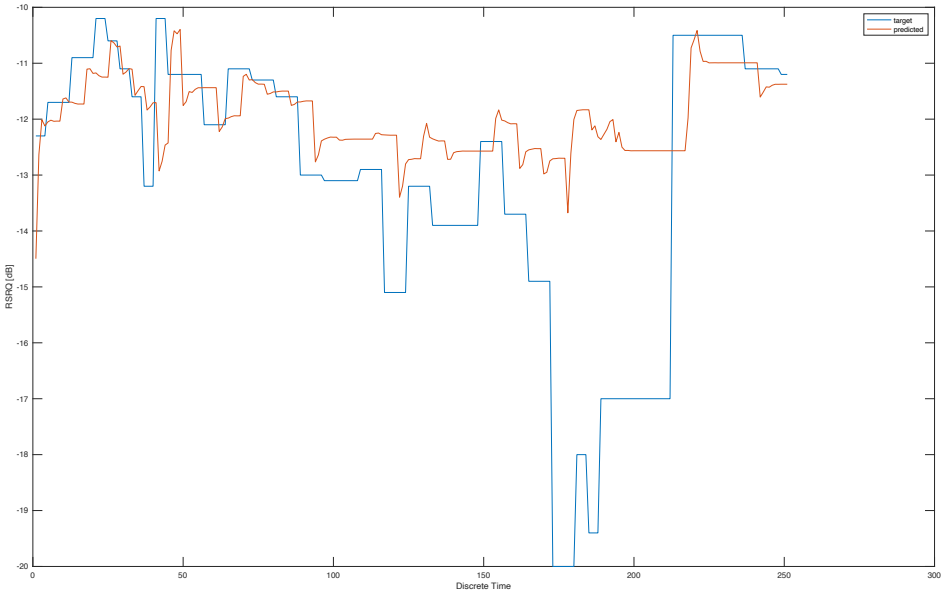

Figure 15. Comparison between the fifth row of the target output and the predicted output. MSE is equal to 6.09.

Here on *Figure 9*, the network is not able at all to match speeds with the sequences. Looking to the *MSE* in this case is not very useful since the temporal mismatch is so high that computing the *MSE* between the target and the predicted output makes no sense. Although the predicted values follow the shape of the target sequence, there is a clear tendency of an increasing temporal mismatch on the output matrix. To see the full comparison, check the Annex B, on which the plots and the *MSE* values between all the rows of the matrix is given for the brevity of this section.

As a final plot the comparison between the five-samples-ahead sequence and the input sequence is given on *Figure 14.*

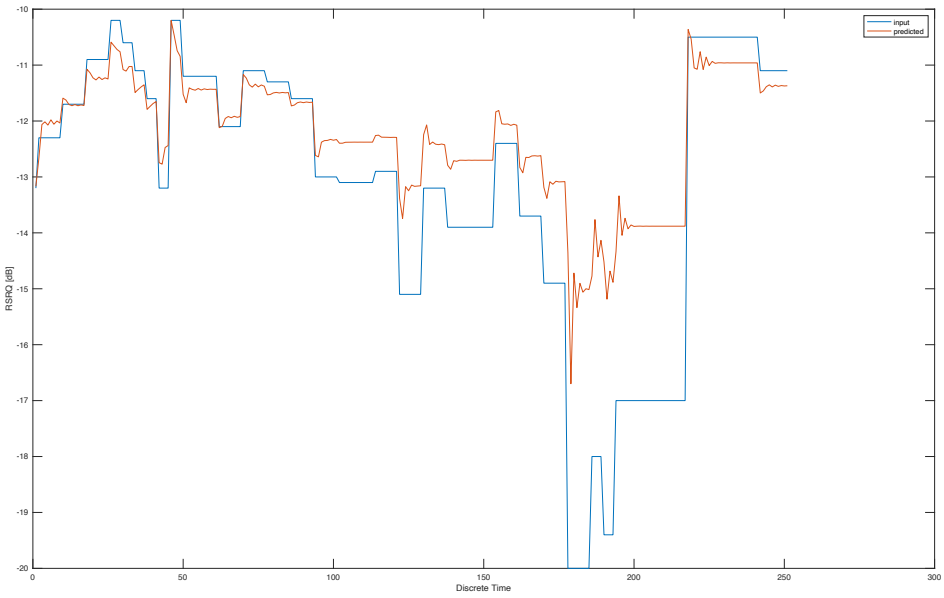

Figure 16. Comparison between the input sequence and the fifth row of the output matrix.

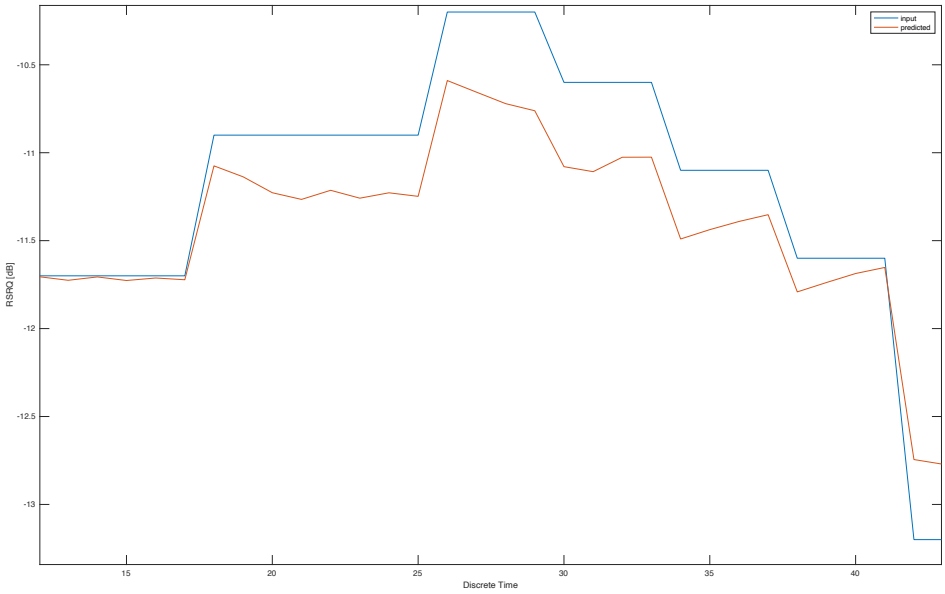

Figure 17. Zoom-in of the starting decreasing sequence of Figure 16.

On both figures the output is sometimes advanced and sometimes delayed, but as expected the values are not as accurate as those in the *Figure 7 & Figure 8*. With all said, the evaluation of the results, and the way the model should be improved in the future, is done on the Conclusion and Future Development section.

6. Budget

The budget of this project consists only on the software licenses spend. Each of the used software —MATLAB, ROMES4 Replay, and QUALIPOC— has its own license that must be paid. Fortunately, the UPC provides those licenses to its students, so this project has not any extra spend, in terms of physical products or software. Nonetheless, the hours that I personally spent working on this project as an engineer should also be added to the budget.

7. Future Development and Conclusions:

Future lines could be an infinite list of things to improve, but I think there are two of them which get on first place. Studying the *Deep ESN* is the first one. I think that those could provide a much powerful model capable of giving a better approach to the solution. Increasing the power of the network for it to become part of the *Deep Learning* seems like an easy task but must be done carefully. Studying which type of *DESN* could be the best and implementing an adequate model is one of the routes this project should take in the future. The second thing would be to wait until reliable measurements on the 5G network can be acquired and, if it is possible, in a way on which the training sequences are continuous function type. That has been discussed over the document, but with a higher sample frequency, and a more precise values the steps those functions take from one value to another may become linear increasing or decreasing sequences, on which the model performs well.

Many conclusions could be extracted from the work done on this project. On the one hand, there is then researching process I have carried out during the process. That has opened my eyes in terms of the amount of reliable information that can be extracted from professional organisations. I have to thank the UPC for letting me to access the IEEE resources, where most of the information I have used to study for this project is at. The learning process I have done during this project has been one of the most exciting of my life. Being alone against the AI and ML world, and, in particular, the ANNs, has made me realize that I want to learn more and deeply on this technology, because I think that on an immediate future this will be the topic of many different fields, and intelligent networks will surround the whole world. Indeed, the idea of combining AI with the Telecommunications world passionate me, and I want to continue working on it.

Concluding, more study on the ESNs must be done for future development of this project. Learning about how a Deep ESN works and which doors open is an interesting task that I left for the future due to be short on time. 5G technology still needs to be stablish and, personally, I think that when reliable measurements on a 5G network could be obtained, then the proposed model on this project still has a chance to perform well.

Bibliography:

[1] Chen, Mingzhe, et al. "Machine learning for wireless networks with artificial intelligence: A tutorial on neural networks." *arXiv preprint arXiv:1710.02913* 9 (2017).

[2] Lukoševičius, Mantas. "A practical guide to applying echo state networks." *Neural networks: Tricks of the trade*. Springer, Berlin, Heidelberg, 2012. 659-686.

[3] Gallicchio, Claudio, Alessio Micheli, and Luca Pedrelli. "Deep reservoir computing: A critical experimental analysis." *Neurocomputing* 268 (2017): 87-99.

[4] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," 2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2017, pp. 1-6, doi: 10.1109/SPAWC.2017.8227766.

[5] S. Aldossari and K. -C. Chen, "Predicting the Path Loss of Wireless Channel Models Using Machine Learning Techniques in MmWave Urban Communications," 2019 22nd International Symposium on Wireless Personal Multimedia Communications (WPMC), 2019, pp. 1-6, doi: 10.1109/WPMC48795.2019.9096057.

[6] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena, 404, 132306.

[7] Popoola, S. I., Adetiba, E., Atayero, A. A., Faruk, N., & Calafate, C. T. (2018). Optimal model for path loss predictions using feed-forward neural networks. *Cogent Engineering*, *5*(1), 1444345.

[8] Jo, H. S., Park, C., Lee, E., Choi, H. K., & Park, J. (2020). Path loss prediction based on machine learning techniques: principal component analysis, artificial neural network, and Gaussian process. Sensors, 20(7), 1927.

[9] Qiu, C., Zhang, Y., Feng, Z., Zhang, P., & Cui, S. (2018). Spatio-temporal wireless traffic prediction with recurrent neural network. IEEE Wireless Communications Letters, 7(4), 554-557.

[10] Co Kg, G. S. R. (s. f.). *QualiPoc Android*. Rohde & Schwarz. https://www.rohde-schwarz.com/nl/products/test-and-measurement/network-data-collection/qualipoc-android_63493-55430.html

[11] Co Kg, G. S. R. (s. f.-b). *R&S®ROMES4 Drive test software*. Rohde & Schwarz. https://www.rohde-schwarz.com/nl/products/test-and-measurement/network-data-collection/rs-romes4-drive-test-software_63493-8650.html

Annex A: Development of the ANN equations

Previously, on the *ANN* mathematical explanation section, it is said that the output of a *NN* is a function of the different weights that compose the whole network. That is the basis to understand the training process. Expressing the output as a function of the weights allows to carry on an optimization process. This section will be a step-by-step explanation until the desired expression is reached. To do so, the scalar notation will be used instead of the matrix notation. But first, let's define the nomenclature:

- The output of a given layer will be declared as: x^{n-l}, where *l* goes from '0' to '*n -1*', and *n* represents the total number of layers (the input layer does not count). So, if for example, our *NN* is composed of two hidden layers and the output layer, then *n = 3*. The output of the network corresponds to x^n, but we will refer to it as *y*. Our explanation goes from the output layer to the preceding ones, so, in the example, the output of the second layer will be $x^{3-1} = x^2$. In a generic way the output of the penultimate layer would be x^{n-1}; its previous one would be x^{n-2} , and so on.
- Same happens with the weights of the different inter-layer connections. From the output to the input layer, the groups of weights will be noted as $w^1, w^2, w^3, etc.$

The procedure will be to start from the output and move on to the input through the hidden layers. So, it is known that the i-th output fulfils this equation:

$$y_i = \sum_{j=1}^{N_{x^{n-1}}} w_{ij}^1 \cdot x_j^{n-1} + b_i$$

where $N_{x^{n-1}}$ is, as you can imagine, the number of outputs of the previous layer[1]. The same way y_i has been expressed as a function of the previous layer outputs, those can also be expressed as a function of its previous layer outputs too. That is:

$$x_j^{n-1} = \sum_{k=1}^{N_{x^{n-2}}} w_{jk}^2 \cdot x_k^{n-2} + b_j$$

Expressing the x_k^{n-2} term as a function of the previous layer outputs could be done here, but I think the idea is now understood so, let's move on.

Next step is to substitute the second equation on the first one:

$$y_i = \sum_{j=1}^{N_{x^{n-1}}} w_{ij}^1 \cdot \left(\sum_{k=1}^{N_{x^{n-2}}} w_{jk}^2 \cdot x_k^{n-2} + b_j \right) + b_i$$

There it is seen that the output has been expressed as a function of the inputs to the previous layer and as a function of the different weights too. The conclusion that must be made is that:

- First, the output can be expressed as a function of the *NN* input
- Second, the output can be expressed as a function of the weights

The second point is the one that is the most interesting for the training part. If it is fulfilled, we can differentiate the output expression over the different weights and find out which are the ones that minimize the cost function (the error between the output and the target).

[1] Note that, for the sake of simplicity, the activation function of the neurons has been omitted. In a real situation the summation plus the bias term would be the input of the activation function, as explained on the *ANN*'s section.

Annex B: Obtained Results for Different Time Advances

On this Annex more comparisons than those that appear on the Results section are given. Going from the prediction of the second future value to the eighth future value. As you will foresee, the MSE value decreases while moving to future values prediction. This is due to two commented mismatches. The one that increases the most is the temporal one.

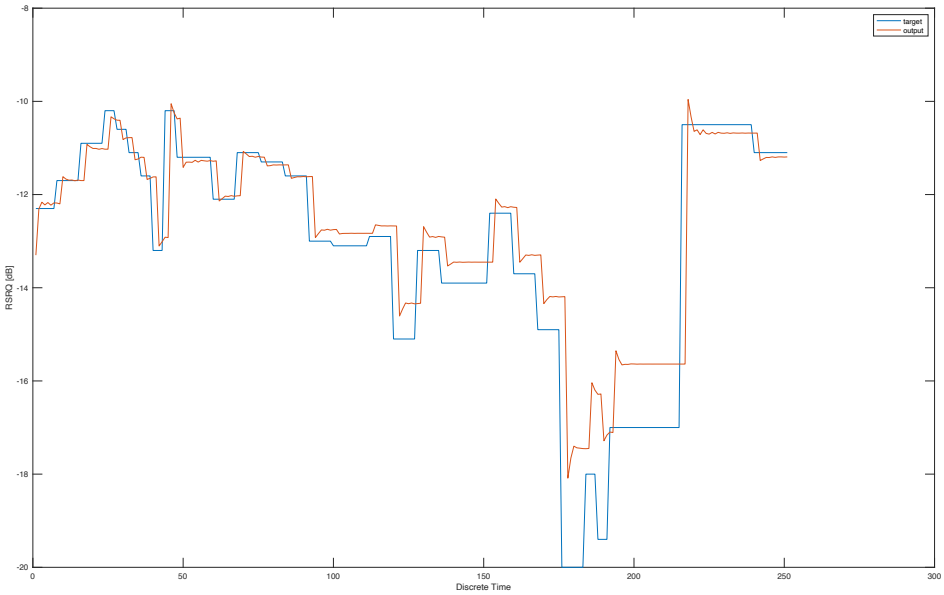

Figure 18. Comparison between the second row of both the target and the output matrices. MSE equal to 1.22

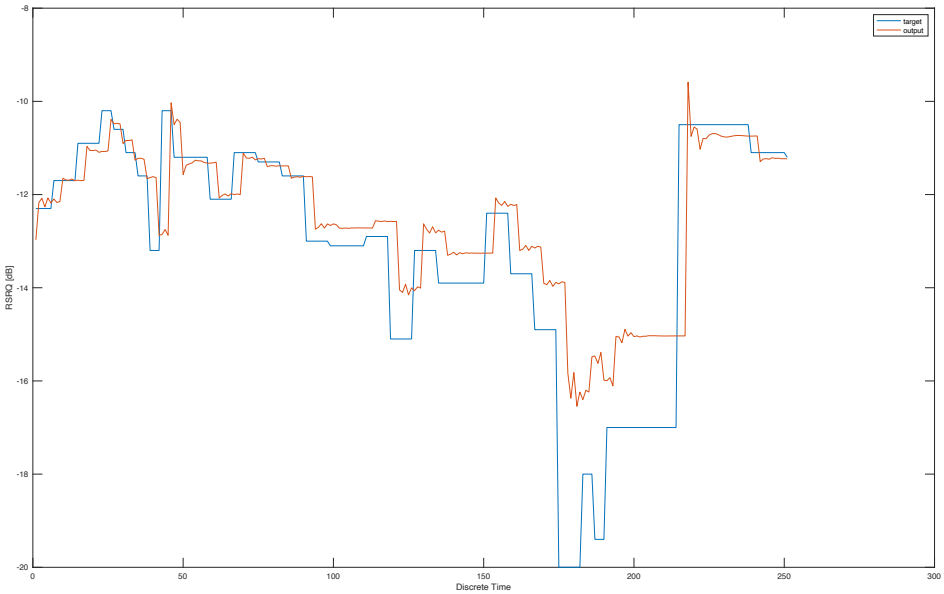

Figure 19. Comparison between the third row of both the target and output matrices. MSE equal to 2.06

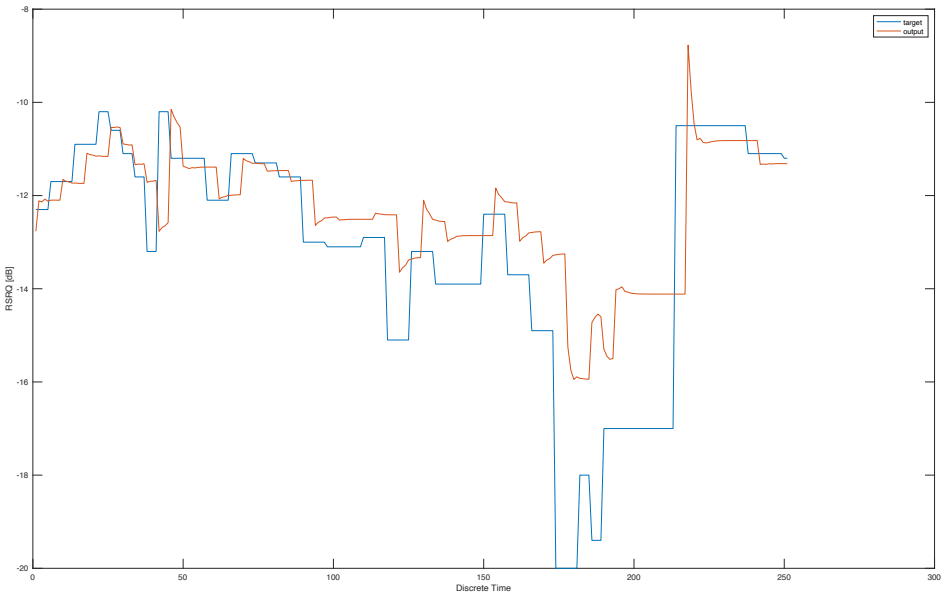

Figure 20. Comparison between the fourth row of both the target and the output matrices. MSE equal to 3.03.

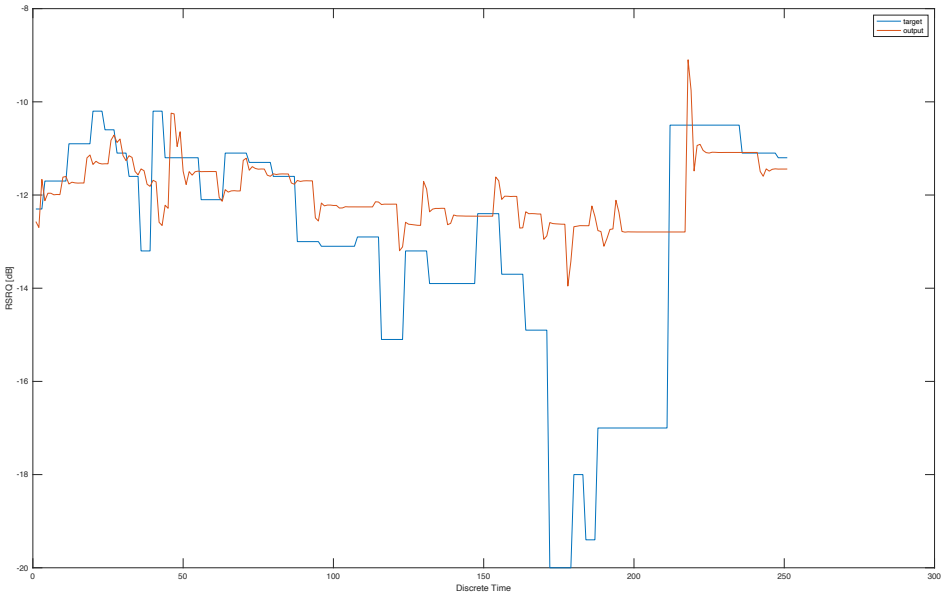

Figure 21. Comparison between the sixth row of both the target and output matrices. MSE equal to 5.67

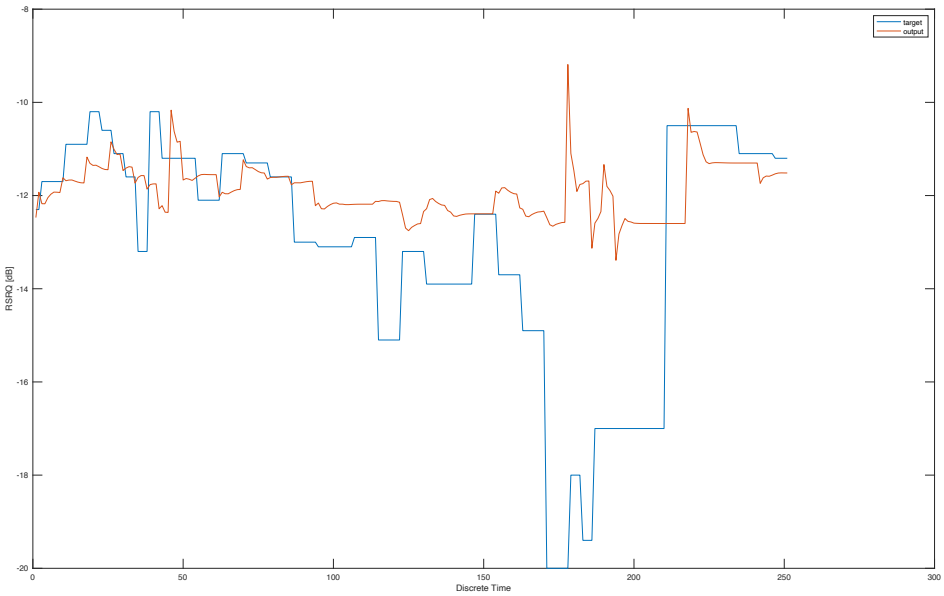

Figure 22. Comparison between the seventh row of both the target and output matrices. MSE equal to 6.7

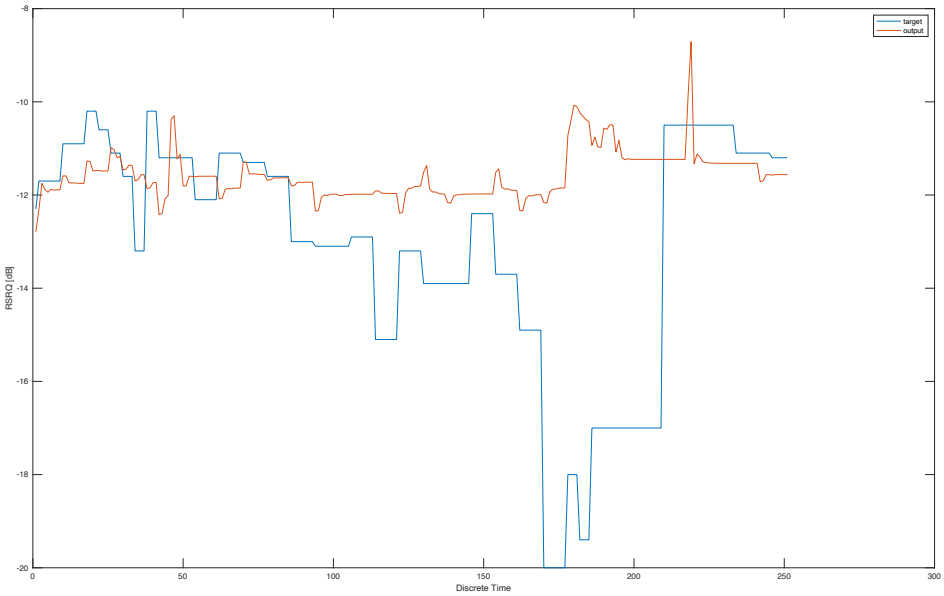

Figure 23. Comparison between the eighth row of both the target and output matrices. MSE equal to 9.11

There is a clear tendency for the model to fail as long as it is asked to predict a higher future sequence. Even though the MSE is not a clear indicator because of the temporal mismatch, on the graphics the value mismatch increases too. On the eighth prediction the model still follows the form of the target sequence —delayed version—, but the prediction has become totally unacceptable.

Glossary

NN: Neural Network

ANN: Artificial Neural Network

FNN: Feed-forward Neural Network

ESN: Echo State Network

DESN: Deep Echo State Network

MNT: Mobile Network Testing (tools)