FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Nonlinear Analog Equalizer for 5th Generation Communication Systems

João Ricardo Pais Correia

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Cândido Duarte

Second Supervisor: José Carlos Pedro

September 23, 2017

Resumo

A próxima geração de redes de comunicação (quinta geração) vai oferecer um maior débito de dados e operar a uma maior largura de banda do que a atual quarta geração. Para além disto está prevista uma mudança nas estações base, que agora cobrem uma área grande e têm um grande consumo de potência, para um maior número de estações de pequenas dimensões, que cobrem uma pequena área e consomem pouca potência.

Nas estações base atuais os amplificadores de potência operam quase no seu nível de saturação para assim garantirem uma maior eficiência energética. No entanto, isto faz com que o amplificador gere distorções no sinal de saída. Estas distorções são corrigidas por um bloco de pré-distorção digital que, devido ao seu método de operação, tem um grande consumo energético. Este consumo é pouco significante nas estações base atuais mas, se efetivamente existir a mudança para estações base com menor consumo, ter um pré-distorçor com grande consumo é impensável. Isto leva-nos a procurar métodos alternativos à pré-distorção digital. Um desses métodos é a pré-distorção analógica.

Esta trabalho serve como o primeiro passo na implementação de um pré-distorçor analógico baseado em redes neuronais. Neste trabalho desenhamos redes neuronais que usam uma função de ativação diferente das normalmente usadas, com o objetivo de esta ser mais facilmente implementada analogicamente. Fizemos também uma análise de sensibilidade a esta rede, de forma a determinar a precisão necessária que a rede deve ter de maneira a fazer uma boa pré-distorção.

Abstract

Fifth generation communication systems are expected to provide higher peak data rates and support larger bandwidths than the current fourth generation systems. As a consequence of this a change in the structure of the base stations is expected. This change will transform the current high power, high coverage central base stations into multiple lower power and lower coverage cells.

In the current base stations, power amplifiers operate close to their saturated power, the region where they are more efficient. This, however, leads to the amplifier generating nonlinearities and distorting the output signal. These distortions are corrected by a digital predistortion block that, by operating numerically on the sampled signal, is necessarily slow and consumes a lot of energy. This power consumption is affordable in the current high power base stations. However, if the migration to smaller low-power base station occurs, this power consumption will be too large. The way forward seems to be the change to an analog predistortion system.

This thesis serves as the first step towards an analog predistortion system based on neural networks. In this work we will propose a different activation function than the commonly used ones. This activation function has the advantage of being easy to implement in analog format. We will also test the necessary precision that this network will need to have in order to have a good predistortion performance.

Agradecimentos

Fica aqui o meu reconhecimento a todos que de alguma maneira me ajudaram no meu percurso académico.

Um obrigado especial aos meus pais e à minha irmã, que sempre me ajudaram e apoiaram ao longo destes anos.

Aos meus amigos, com quem vivi estes 5 anos. Ao lado deles vivi muitos momentos que dificilmente vou esquecer.

Aos meus orientadores, Cândido Duarte e José Carlos Pedro, pela disponibilidade e ajuda crucial no desenvolvimento desta tese.

E finalmente, à minha namorada que me motivou ao longo deste trabalho e me ajudou em bastantes momentos.

João Correia

"Ever tried. Ever failed.
No matter.
Try again. Fail again. Fail better."

Samuel Beckett

Contents

List of Figures

List of Tables

Abbreviations

Fifth Generation Communication Systems	5G
Base Station	BS
Power Amplifier	PA
Gallium Nitride	GaN
Artificial Neural Network	ANN
Gradient Descent	GD
Levemberg-Marquardt	LM
Normalized Mean Square Error	NMSE
Root Mean Square	RMS

Chapter 1

Introduction

In the current days 4th generation communications systems are already widespread and we are now looking into the next generation. In the future, personal devices will need to provide high quality video streaming, mobile TV and content production in a multi device, multi access network [1]. Furthermore, the next generation will need to support new radio interfaces, Internet of Things and other use cases. Fifth generation communication systems (5G) is expected to provide a peak data rate of 10 Gb/s for low mobility users and 1 Gb/s for high mobility users [2], and this number of users is ever increasing (because of the growth in human users but also, as was mentioned before, objects using Internet of Things). All this drives the need of larger bandwidths (estimates are in the 500 to 1000 MHz) [1]. To accommodate these needs one of the designed ideas for 5G is to change the base station (BS). The idea is to change the BSs from a central, high power and high coverage cell to several smaller, lower power and lower coverage cells. This makes it possible for various users to use the same spectrum [3].

This larger number of cells, allied with reports that the energy consumption of BSs contributes to over 60 percent of the electrical bill of cellular operators [4], means that making power efficient BSs is a necessity.

One of the main components of BSs is the Power Amplifier (PA). However, efficient PAs are mostly nonlinear [5]. This nonlinearity has severe consequences in the spectral density of the output signal since it spreads the bandwidth of the signal. This is a problem since spectral efficiency is very important.

To solve this problem and others arising from the nonlinearity of PAs, linearization methods have been developed. The one that is most commonly used is the predistortion method. This method is commonly done digitally and its inclusion means that there is one more block that consumes power in the transmission chain. In the 4G BS structure this power consumption is manageable since we have high power amplifiers, which means that the consumption of the predistortion block is small when compared to the global consumption of the BS. However, if we indeed change to a BS structure where we have more BS and these BS are low power, the power consumption of the digital predistortion block will be too large when compared to the global consumprion of the BS. This occurs because, although we can use a lower power PA, we cannot make

a proportionally lower power digital predistortion block. We can't afford to use a predistortion block that consumes more than the PA. One way to try to rectify this problem is to change the predistortion block from digital to analog.

The purpose of this work is to propose a predistortion block based on neural networks that serves as a first step towards the creation of an analog predistortion block. Neural networks have already been used to do digital predistortion [6] and there are even some works trying its analog implementation [7].

This work will take a new approach in the design of the neural network with its analog implementation in mind. The neural network will be created using an activation function that is easier to implement analogically than commonly used activation functions. With this we hope to facilitate a future implementation of the network in analog format.

The objectives of this work are to verify if this alternative activation function can have a similar performance as the commonly used activation functions for neural networks and also determine the necessary precision that the network needs to have in order to have a good performance.

In the following chapter of this document we will first look at the linearity and memory problems of PAs. In the next chapter we will give a detailed explanation of neural networks and the way they function. This will include both the functioning and the training of neural networks. We will look at two different training algorithms: the Gradient Descent algorithm and the Levenberg-Marquardt algorithm. These two chapters will come together in chapter 4, where we will discuss how neural networks can be used for predistortion purposes. This discussion will focus mainly in the way the inputs and outputs of the network can be defined with the purpose of maximizing the performance of the network in the predistortion case. Following this we will present the results of the tests that were done. These tests include the testing of networks of different structures with the goal of finding the best configuration of the network and to compare the new activation function with a commonly used activation function. Also included in this chapter is a sensibility test of the best performing network that uses the new activation function. This serves as a way to determine the required precision that the different zones of the network need to have in order to keep its performance. Finally we will have the chapter about the conclusions and future work.

Chapter 2

Power Amplifiers

2.1 Introduction

Power amplifiers are one of the most important components in radio frequency communications. They are responsible for amplifying the signal so that its power is high enough to be transmitted through an antenna and have a good area coverage.

The efficiency of the PA is important since they are one of the largest consuming elements of BSs. Also important in a mobile device is energy efficiency since battery life is a highly valued resource. However, as stated previously, efficient PAs are mostly nonlinear. This is inconvenient because it generates two problems. The first one is that the nonlinear behavior affects both the amplitude and phase of the output signal. Since data modulation schemes used nowadays involve both amplitude (AM) and phase modulation (PM), the AM/AM and AM/PM distortion can cause errors to occur in the demodulation, and therefore, incorrect symbol detections. The second problem is related to bandwidth usage. Nonlinearities of the PA spread the bandwidth of the output signal, which can cause interference in adjacent channels.

Due to this, it became important to devise linearization methods for PAs. It is important to note that modern PAs are not only non linear, but also have memory.

2.2 Linearity and Memory

In Mathematics the linearity of a function is defined as the satisfaction of two proprieties: additivity and homogeneity.

The additivity and homogeneity proprieties can be stated as follows:

$$f(x+y) = f(x) + f(y) \tag{2.1}$$

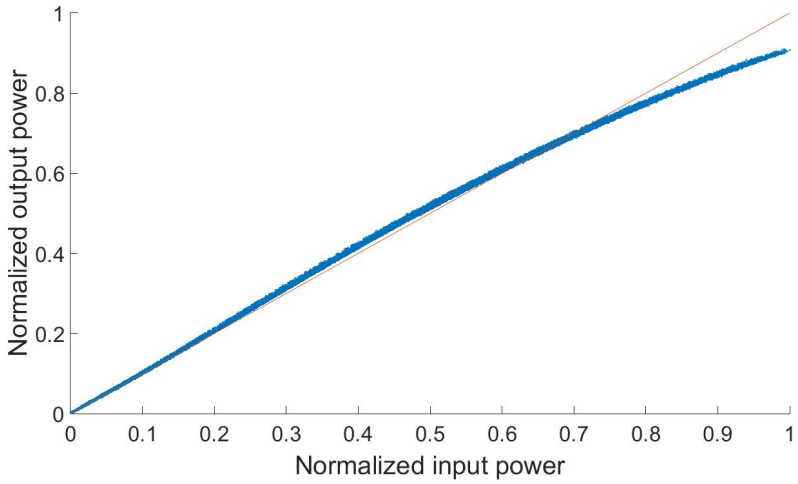
$$f(\alpha x) = \alpha f(x) \tag{2.2}$$

In the case under study we can simplify this by stating that a PA is not linear because the plot of its output power versus its input power is not a straight line. However, nonlinearity is not the only problem in PAs. If it was, one could model a PA with a simple polynomial equation with a large enough order.

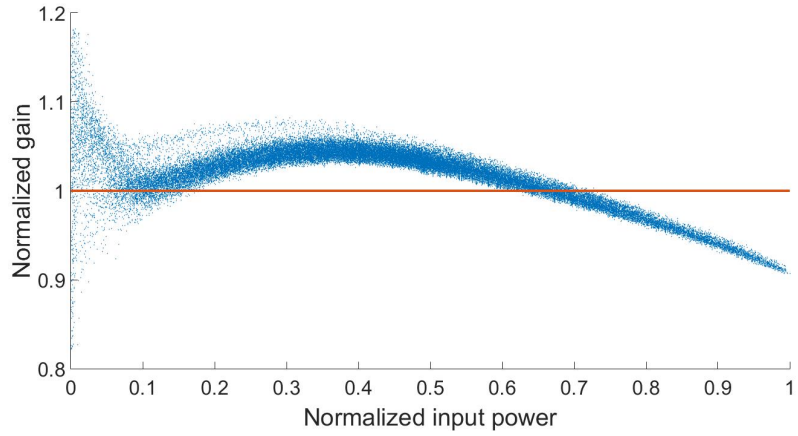
PAs also have memory. A system is said to have memory if its output value at any given time doesn't depend only on its input value at that time. Mathematically we can define $y(t)$ has having memory by:

$$y(t) = f\Big(x(t), x(t+k_1), x(t+k_2), x(t+k_3), \ldots\Big), k_1, k_2, k_3 \ldots \in \mathbb{R} \tag{2.3}$$

Since we are dealing with real systems and not theoretical ones, in the case of the PA we can say that its output value at any given time will depend on its current and past input values. So in 2.3 we will have $k_1, k_2, k_3 \ldots \in \mathbb{R}^-$.



(a)

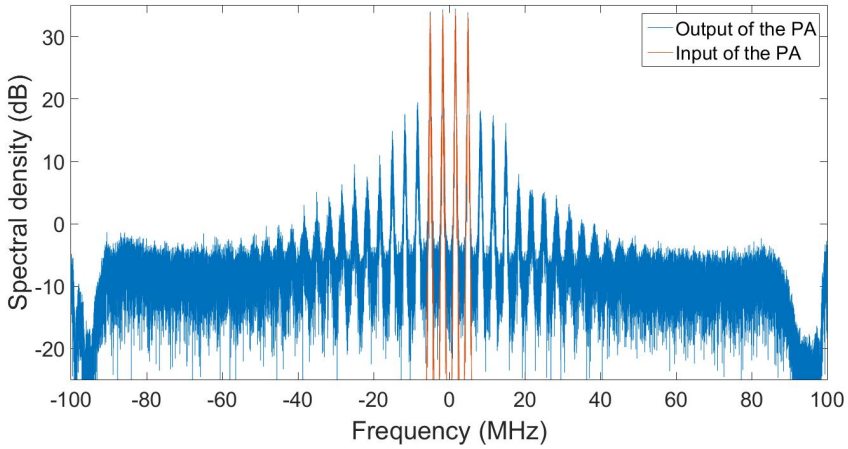


(b)

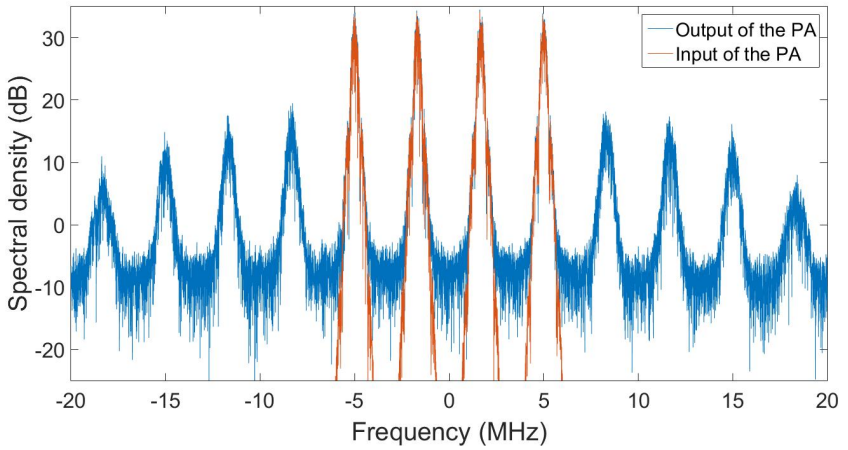
Figure 2.1: Plots of the normalized output power (a) and gain (b) of a PA versus the normalized input power.

Figures 2.1a and 2.1b show plots of a real PA. These plots were created using around 60000 samples from a Gallium Nitride (GaN) PA. In both plots an orange line was added for purposes of comparison, representing an ideal PA. It is possible to see in both plots the effects of nonlinearity and memory. Nonlinearity is noticeable because in both cases the samples don't form a straight line. Memory effects are seen since it is possible to observe different outputs/gain for the same input.

The nonlinearities result in a spread of the bandwidth, as can be seen in figure 2.2a and 2.2b. Of relevance when comparing these signals is the Adjacent Channel Power Ratio (ACPR), which is the ratio between the main channel power and the adjacent channel power. In this case the ACPR equals $3,1734$ dB, which is a very low value.



(a) Global view



(b) Center frequencies

Figure 2.2: Plot of the spectral density of the normalized output and input signal of the PA

2.3 Linearization techniques

To correct these issues in PAs, linearization techniques have been developed.

2.3.1 Feedback method

This linearization technique is based on a feedback loop. The use of a direct feedback loop between the output of the PA and its input has been well studied but it has not seen a lot of use in RF. A possible explanation for this has to do with concerns about amplifier stability and the difficulty in making networks with non-ideal components function over wide frequency bandwidths [8].

Indirect feedback methods have been developed and are more widely used. One family of these methods is modulation feedback. In this family we can find the Cartesian and the Polar feedback methods, each related to the modulation components used: Quadrature and In-phase, or Magnitude and Phase. In these modulation types, the components of the signal are the ones being compared.

A type of these systems revolves around low pass filtering the output signal so that it is possible to work with the baseband. This is then compared with the input, and the error is used to adjust the gain of the amplifier.

Feedback linearization techniques have the problem of limiting the gain of the linearized PA to the gain of the loop. Another limitation is imposed by the delay of the feedback loop, which needs to be small enough to guarantee stability. This makes it so that this technique can only be used in narrowband signal [9, Chapter 5].

2.3.2 Feedforward method

The feedforward linearization technique consists of using a correction signal in the output of the PA. A block diagram for this technique can be seen in figure 2.3 [9, Chapter 5]. In this method, we first calculate the difference between the scaled down output of the PA and the input signal (seen in the signal cancellation loop). After that, it is possible to calculate the scaled down value that the nonlinearity of the PA adds to the ideal output signal- the error. Writing it explicitly we are calculating $E = \frac{Y}{G} - X$, where X is the input of the signal, G is the ideal gain of the PA (without nonlinearities), Y is the real output of the PA and E is the error. It is possible to see that we can obtain the ideal output by doing $Y - E \times G$. It is exactly that that is done in the distortion correction loop.

This method provides a good linearization over a wide bandwidth. The drawback is that the delay lines must be very precise and the gain of the PA is non-trivial. It varies due to memory effects and operating conditions of the PA. Adaptation methods need to be implemented to achieve a good linearization. Power efficiency is also a major concern in this method since the error amplifier (Aux. PA in the figure) needs to be perfectly linear. This means that it is very power inefficient [10]. In third-generation PAs linearized by this method, the power efficiency was only of 10 to 15% [9, Chapter 5].

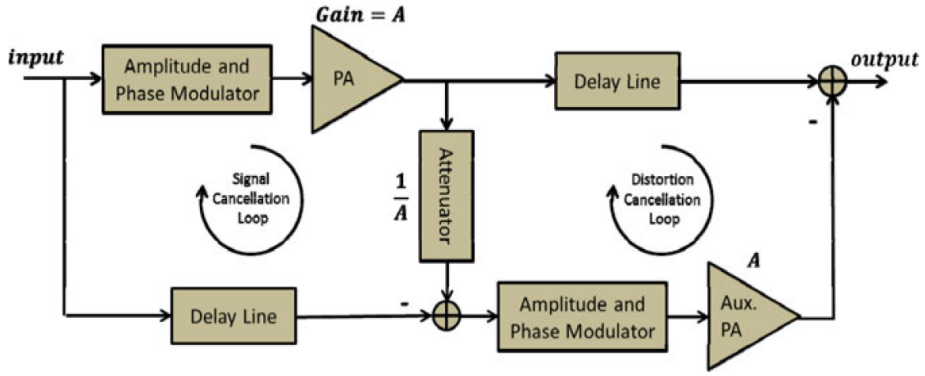


Figure 2.3: Block diagram of the feedforward linearization method

2.3.3 Predistortion method

This method is open-loop and is based on inserting a predistortion block before the PA as we can see in figure 2.4. This predistortion block can be inserted in the baseband or the RF part of the transmission chain. The idea is that the predistortion block distorts the signal in a way that corrects the distortion imposed by the PA so that the relation between the input of the predistortion block and the output of the PA is a linear one. This means that the predistortion block implements the inverse function of the PA. If we were to imagine a PA which implemented the following function $Y = X^2$, then the predistortion block needs to have $Y = \sqrt{X}$ as its implemented function. This would produce a linear cascade with unitary gain. If a different gain is desired, scaling factors need to be accounted for.

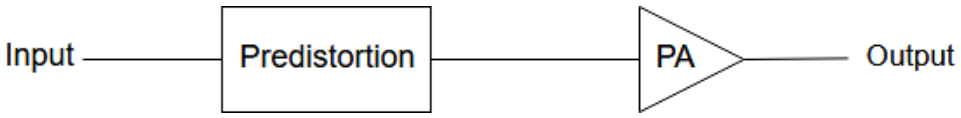


Figure 2.4: Block diagram of the predistortion linearization method.

Obviously the previous example was very simple and the real transfer function of the PA (and its inverse) is much more complex and difficult to find. Finding the model of the PA is therefore a very important part in this method of linearization. This is usually done through behavioral modeling. This has the advantage of not needing to know anything about the PA since in behavioral modeling this is considered a black box, and we only try to find the relation between the inputs and outputs.

There are several methods regarding the implementation of the predistorter such as: look up tables, complex polynomials, neural networks and fuzzy systems [11].

The main problem of this linearization method is that it relies on having a good model of the PA. It has the advantages of not having a bandwidth limitation like the feedback methods and of solving the problems of the feedforward method. The power efficiency of third generation PAs using digital predistorters was 30 to 45%.

Chapter 3

Neural Networks

Artificial neural networks (ANNs) have their origins in the attempt to find a mathmatical representation of information in biological systems [12]. These networks are constituted by the interconnection of various perceptrons, the core element of an ANN.

The purpose of an ANN is to model a complex system in which the transformations required to produce an output from an input are not clear or very difficult to know. These systems are quite varied and can go from stock market prediction [13], handwritten characters recognition [14], fingerprint recognition [15], the traveling salesman problem [16] among others. One of the main advantages of using neural networks is that there is no need for extensive knowledge of the system we are trying to model. We simply view it as a black box in which only the inputs and outputs are known. It is by using a large data set of inputs and outputs that the network is able to learn to behave like the system we are trying to model. Some knowledge of the system gives an advantage since it will provide information useful when creating the architecture of the network (for example in choosing a good activation function), and in the preprocessing of the inputs before they are fed to the network (which can severely increase its performance).

ANNs can be divided into different groups, depending on the way the perceptrons are connected. There are recurrent or non recurrent networks. We can have an organized layered structure or a more fluid structure. This chapter and the following work will focus on the multilayer perceptron.

A multilayer perceptron is a type of network where the perceptrons are organized in layers as follows: inputs, one or more hidden layers and the output layer. The way of counting layers varies according to authors and, in this work we will count only the layers from the hidden and output layers, since these are the only ones which fully implement the perceptron. So the first layer of the network will be the first hidden layer. All of the perceptrons of a given layer are connected to all the perceptrons in the following layer. So, the inputs of the network will be connected to all of the perceptrons in the fist layer, each perceptron of the first layer will be connected to all of the perceptrons in the second layer and so on until the output layer. The results obtained from the perceptrons in the output layer are the outputs of the network.

An example of a network of this kind can be seen in figure 3.1. In this example we have a

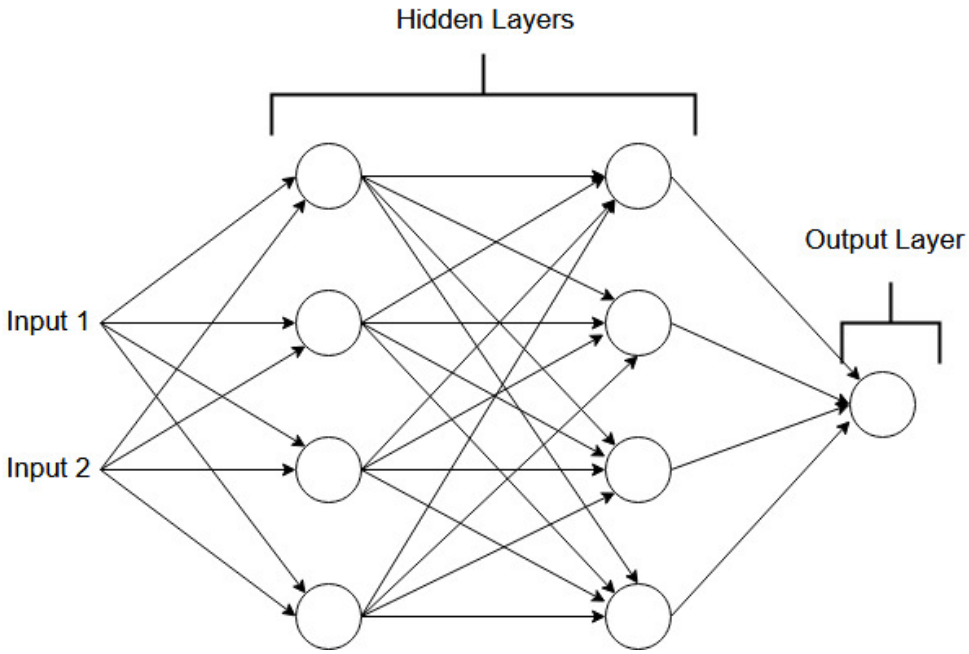


Figure 3.1: Overview of a multilayer perceptron

network with 2 inputs and 3 layers (2 hidden layers and 1 output layer). Since there is only one perceptron in the output layer, there is only one output. Also it is possible to see that there are 4 perceptrons in each hidden layer.

These networks have been well studied and it is believed that they can produce any continuous function with a network using one layer and a finite number of perceptrons [17] [18]. Furthermore, very positive results have been achieved using these types of network with predistortion purposes [6] [19] [20].

3.1 Perceptron

The basic unit that is used to create a neural network is the perceptron. As displayed in figure 3.1 a perceptron has multiple different inputs and will have a single output that will be repeated for each outward connection. A more detailed view of the perceptron can be seen in 3.2.

In the perceptron, the inputs first go through a weighted sum to which a bias is added. This generates the input of the activation function - z. z is written as follows:

$$z = b + \sum_k w_k \times I_k \tag{3.1}$$

In this equation the weights (w_k) and bias (b) are parameters of the perceptron. This sum is then used as the input for an activation function $\sigma(z)$ that produces the output of the perceptron -

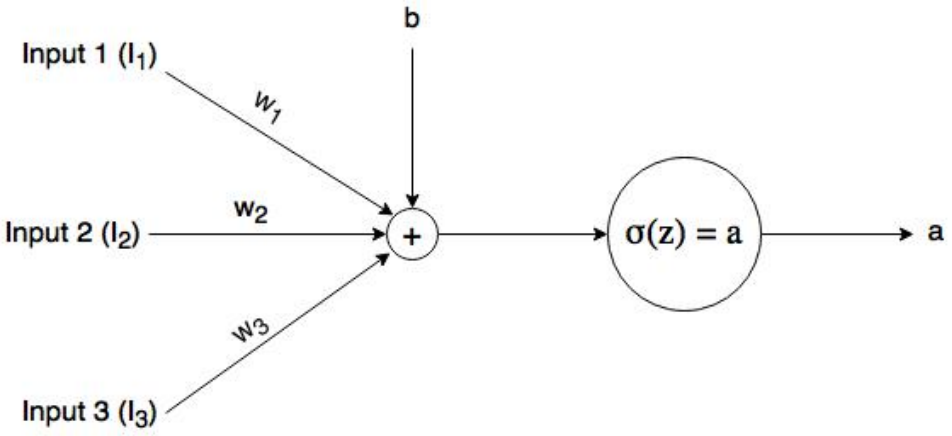


Figure 3.2: Diagram of the Perceptron

a. The output, often called the activation of the perceptron can therefore be defined as:

$$a = \sigma(b + \sum_k w_k \times I_k) \tag{3.2}$$

The manipulation of weights and biases, along with a good activation function, is what makes it possible for the multilayer perceptron to be able to replicate any continuous function. The most commonly used activation functions are sigmoid functions, such as the hyperbolic tangent. The activation function of the output layer is commonly a linear function or a sigmoid function. However, the activation functions chosen for each perceptron are totally dependent on the system one is trying to model.

It is possible to see that manipulating the biases and the weights of the perceptron is simply stretching and moving the activation function. For illustrative purposes lets imagine a neural network with one input, one hidden layer with one perceptron using the hyperbolic tangent as activation function and an output layer with one perceptron using a unitary linear activation function ($\sigma(z) = z$). As we have seen before, the output of the first perceptron is:

$$a_h = tanh(b_h + w_h \times I_1) \tag{3.3}$$

This will be used as the input of the output layer's perceptron. The output of the layer will be:

$$out = b_o + w_o \times a_h \tag{3.4}$$

Using 3.3 in 3.4 we obtain the full equation for the output of the network:

$$out = b_o + w_o \times tanh(b_h + w_h \times I_1) \tag{3.5}$$

In figure 3.3 we can see the output of the neural network as a function of the input I_1 for networks with different values of b_h, b_o, w_h, w_o. As we can see, by manipulating the parameters of the network we can obtain various different functions. If we imagine a network with more than

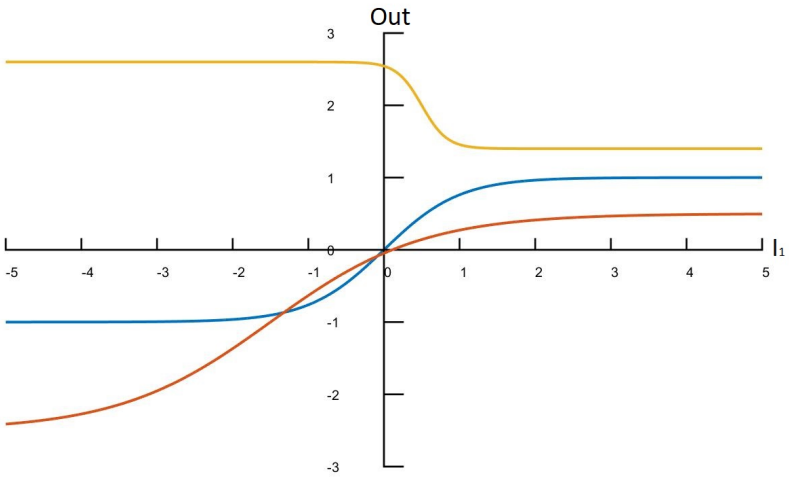


Figure 3.3: Output of the neural network versus the input I_1 for networks with different values of w_h, w_o, b_h and b_o

one perceptron in the hidden layer, we will have in the output a sum of various functions that will hopefully generate the desired output.

Now that we have seen how the perceptron is constituted and an example of a small and simple ANN, it is time to look at more complex networks and ways of simplifying the description of its equations.

3.2 Global structure

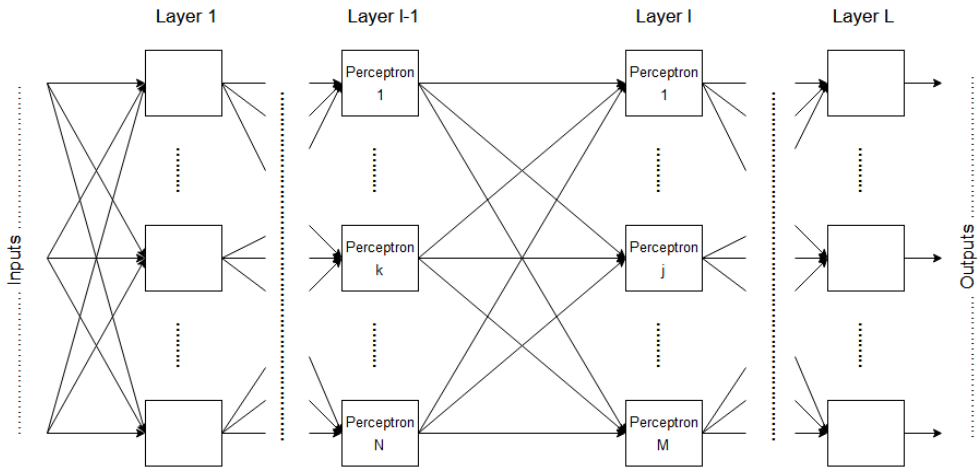


Figure 3.4: Block diagram of a generic ANN

In equation 3.5 we saw the output equation for a network with one input, one hidden layer with one perceptron and one perceptron in the output layer. This ANN is very simple and we will need

to create networks that are bigger and more complex. A way to simplify equations of the network is to use matrix notation.

In figure 3.4 we have a generic ANN. We can see that it has L layers and in layer $l-1$ and l we have N and M perceptrons respectively. If we want to write the input of the activation function of the perceptron j of layer l (which we will represent as z_j^l) we can follow equation 3.1 and get:

$$z_j^l = b_j^l + \sum_{k=1}^{N} w_{j,k}^l \times a_k^{l-1} \tag{3.6}$$

where b_j^l is the bias of the perceptron j in layer l, $w_{j,k}^l$ is the weight that connects the output of the perceptron k in layer $l-1$ to the perceptron j in layer l, and a_k^{l-1} is the output of perceptron k in layer $l-1$.

If we want to write the vector z^l with dimensions $M \times 1$, we can do it in the following way:

$$z^l = b^l + w^l \times a^{l-1} \tag{3.7}$$

where b^l is the vector with dimensions $M \times 1$ of the biases of each perceptron in layer l, w^l is the matrix with dimensions $M \times N$ where each element is $w_{j,k}^l$ as defined above, and a^{l-1} is the vector with dimensions $N \times 1$ of the outputs of each perceptron in layer $l-1$. On the first layer this equation has no meaning, since there is no a^0. We instead use the vector of the inputs in its place, giving us:

$$z^1 = b^1 + w^1 \times I \tag{3.8}$$

We can also define the vector a^l of the outputs of the perceptrons on layer l as:

$$a^l = \sigma^l(z^l) \tag{3.9}$$

where σ^l is the activation function used in layer l. Evidently the vector of outputs of the network, *out*, will be:

$$out = a^L \tag{3.10}$$

Now that we have a good mathematical description of the equations of the whole network, we can take a look into its training.

3.3 Network training

At this time we have a way to write the equations of a full network and easily find its output given a set of weights, biases, activation functions and inputs. But we still don't have a way to guarantee that the output is the one we desire. In fact, when we create a network, there is no way of knowing which value to attribute to each weight and bias in a way that transforms our inputs into the outputs

we want. That is unless the system we are trying to model is extremely simple. But, in that case we wouldn't be using a neural network.

When we create a network, most of the times, the weights and biases are set at random. The process in which the weights and biases are adjusted is called the training. For the training process, we need a large data set of inputs and its respective outputs in the system we are trying to model. With this large data set we can iteratively adjust the weights and biases in a way that minimizes a certain cost function. The cost function is defined in a way that can evaluate the effectiveness of the approximation of the neural network to the real system we are trying to model.

The training is an iterative process and each iteration can be divided into four steps:

1. forward propagation of a input through the network;
2. computation of the error and Cost function;
3. backwards propagation (this step has slight differences depending on the training algorithm used);
4. computation of the new weights and biases following a training algorithm.

The training can be done in batches or on the go. If we are training the network in batches, the first three steps will be done for all the samples in the data set and only after will the weights and biases be updated. On the other hand, if we are training the network on the go, we do all four steps for each sample. That is, we only do the forward propagation of the next sample after we update the weights and biases using the results from the current sample.

After we go through all the samples of the data set we have completed an epoch of training. Usually various epochs are required until we have a trained network.

3.3.1 Forward propagation and Cost function

As mentioned before, the first step of the training is the forward propagation. This is done by taking one input from the data set and running it through the network using equations 3.7, 3.8, 3.9 and 3.10.

In the end of the forward propagation we get the vector of the estimated outputs - *out*. Now that we have our estimated output, we need to evaluate the performance of the network. This is done by calculating the cost function. The cost function is the function that we want to minimize in the training process and is usually some variation of the sum of squared errors. A commonly used cost function is:

$$C = \frac{1}{2} \times \|out - y\|^2 \tag{3.11}$$

where *out* is the output vector of the network for a given input vector X, y is the output sample vector which corresponds to the same input vector X and $\|z\| = \sum_i z_i$. Since we want our *out* to be equal to y, we want to find the minimum of the cost function.

After calculating the cost function of the network we proceed with the backwards propagation. However, the way this step (and the following) is done depends on the training algorithm used.

There are multiple training algorithms that can be implemented such as the Gradient Descent (GD), Newton's Method, Gauss-Newton Method and the Levenberg-Marquardt (LM) algorithm. These algorithms have different degrees of computational complexity and performance. In this chapter we will explore the GD algorithm and the LM algorithm.

The GD algorithm is one of the most simple algorithms and the LM algorithm is one of the best performing algorithms, although at a higher complexity cost.

3.3.2 Gradient descent algorithm

The Gradient Descent algorithm [21] aims to update the weights and biases of the network using the following rules:

$$w^l_{j,k\,new} = w^l_{j,k} - \alpha \times \frac{\partial C}{\partial w^l_{j,k}} \tag{3.12}$$

$$b^l_{j\,new} = b^l_j - \alpha \times \frac{\partial C}{b^l_j} \tag{3.13}$$

where $w^l_{j,k\,new}$ and $b^l_{j\,new}$ will be the new weights and biases of the network, $\frac{\partial C}{\partial w^l_{j,k}}$ and $\frac{\partial C}{b^l_j}$ are the partial derivatives of the cost function with respect to each weight and bias, and α is the learning rate. The learning rate is a small number, usually less than one, that will control the convergence of the network. When using this algorithm it is necessary to find a good value for the learning rate by trial and error. This is one of the disadvantages of using this algorithm.

This algorithm converges to the local minimum which is nearer from the starting values of the weights and bias.

We will now look at how to compute $\frac{\partial C}{\partial w^l_{j,k}}$ and $\frac{\partial C}{b^l_j}$. The process of computing these values is called backpropagation. We will compute this step by step. Firstly we will compute the partial derivative of C with respect to the input of each perceptron in the output layer z^L_j. This will be referred to as $delta^L_j$:

$$delta^L_j = \frac{\partial C}{\partial z^L_j} = \frac{\partial a^L_j}{\partial z^L_j} \times \frac{\partial C}{\partial a^L_j} \tag{3.14}$$

Choosing the perceptron j in the output layer, the rightmost derivative in the equation will be:

$$\begin{aligned}\frac{\partial C}{\partial a_j^L} &= \frac{\partial(\frac{1}{2}\times\|out-y\|^2)}{\partial a_j^L} \\ &= \frac{1}{2}\times\frac{\partial(\sum_n(a_n^L-y_j)^2)}{\partial a_j^L} \\ &= \frac{1}{2}\times\frac{\partial(a_j^L-y_j)^2}{\partial a_j^L} = (a_j^L-y_j)\end{aligned} \tag{3.15}$$

The other derivative required to compute $delta_j$ is simply the derivative of the activation function with respect to its input:

$$\frac{\partial a_j^L}{\partial z_j^L} = \sigma'(z_j^L) \tag{3.16}$$

Using the results from 3.15 and 3.16 in 3.14, we get:

$$delta_j^L = \sigma'(z_j^L)\times(a_j^L-y_j) \tag{3.17}$$

We can rewrite this in matrix notation, so that $delta^L$ is the vector formed by the delta of each perceptron of the output layer.

$$delta^L = \sigma'(z^L)\odot(a^L-y) = \sigma'(z^L)\odot(out-y) \tag{3.18}$$

where $\odot$ is the Hadamard product, also known as the element wise multiplication of matrices.

With $delta^L$ we can easily compute the partial derivative of the cost function with respect to the weights and biases of the perceptrons in the output layer, but first lets see how to propagate $delta^l$ to a perceptron in the previous layer $delta_k^{l-1}$. Using figure 3.4 as example, we can see that there will be M paths connecting the perceptron k in layer $l-1$ to the layer l. So, $delta_k^{l-1}$ will be the sum of M propagations of $delta_j^l$. This is shown here:

$$\begin{aligned}delta_k^{l-1} &= \sum_{j=1}^{M}\left(\frac{\partial z_j^l}{\partial z_k^{l-1}}\times\frac{\partial C}{\partial z_j^l}\right) \\ &= \sum_{j=1}^{M}\left(\frac{\partial a_k^{l-1}}{\partial z_k^{l-1}}\times\frac{\partial z_j^l}{\partial a_k^{l-1}}\times delta_j^l\right) \\ &= \frac{\partial a_k^{l-1}}{\partial z_k^{l-1}}\times\sum_{j=1}^{M}\left(\frac{\partial z_j^l}{\partial a_k^{l-1}}\times delta_j^l\right)\end{aligned} \tag{3.19}$$

Computing $\frac{\partial a_k^{l-1}}{\partial z_k^{l-1}}$ is the same as in equation 3.16. As for the other derivative we have:

$$\begin{aligned}\frac{\partial z_j^l}{\partial a_k^{l-1}} &= \frac{\partial \left(b_j^l + \sum_{i=1}^N w_{j,i}^l \times a_i^{l-1}\right)}{\partial a_k^{l-1}} \\ &= \frac{\partial \left(w_{j,k}^l \times a_k^{l-1}\right)}{\partial a_k^{l-1}} \\ &= w_{j,k}^{l-1}\end{aligned} \tag{3.20}$$

We can now rewrite equation 3.19:

$$delta_k^{l-1} = \sigma^{'}(z_k^{l-1}) \times \sum_{j=1}^{M} \left(w_{j,k}^{l-1} \times delta^l\right) \tag{3.21}$$

This can be transformed into matrix notation so that we can write the equation to compute the vector of deltas of layer $l-1$:

$$delta^{l-1} = \sigma^{'}(z^{l-1}) \odot \left((w^{l-1})^T \times delta^l\right) \tag{3.22}$$

Using equations 3.22 and 3.18, we can compute the deltas of all the perceptrons of the network. To finally compute $\frac{\partial C}{\partial w_{j,k}^l}$ and $\frac{\partial C}{b_j^l}$ we simply need to do:

$$\begin{aligned}\frac{\partial C}{\partial w_{j,k}^l} &= \frac{\partial z_j^l}{\partial w_{j,k}^l} \times \frac{\partial C}{\partial z_j^l} \\ &= \frac{\partial \left(b_j^l + \sum_{i=1}^N w_{j,i}^l \times a_i^{l-1}\right)}{\partial w_{j,k}^l} \times delta_j^l \\ &= \frac{\partial \left(w_{j,k}^l \times a_k^{l-1}\right)}{\partial w_{j,k}^l} \times delta_j^l \\ &= a_k^{l-1} \times delta_j^l\end{aligned} \tag{3.23}$$

$$\begin{aligned}\frac{\partial C}{\partial b_j^l} &= \frac{\partial z_j^l}{\partial b_j^l} \times \frac{\partial C}{\partial z_j^l} \\ &= \frac{\partial \left(b_j^l + \sum_{i=1}^N w_{j,i}^l \times a_i^{l-1}\right)}{\partial b_j^l} \times delta_j^l \\ &= delta_j^l\end{aligned} \tag{3.24}$$

Or, in matrix notation:

$$\frac{\partial C}{\partial w^l} = a^{l-1} \times (delta^l)^T \tag{3.25}$$

$$\frac{\partial C}{\partial b^l} = delta^l \tag{3.26}$$

We have, at this point, found a way to efficiently compute all that is required in the backpropagation. We need to use equation 3.18 once, 3.22 once for each hidden layer, and equations 3.25 and 3.26 once for each layer.

After this we can finally update the weights and biases using equations 3.12 and 3.13.

This concludes the Gradient Descent algorithm. As stated before, when we use this algorithm, the update of the weights and biases can be done after each sample or after the whole batch of samples has been used.

3.3.3 Levenberg-Marquardt algorithm

The Levenberg-Marquardt algorithm [22] is much more complex than the gradient descent algorithm however it provides other advantages, such as a better overall performance, i.e we have a better minimization of the cost function. This algorithm is only used in batch mode, so we first need to compute the output of the network for all the input samples and only after will we compute new weights and biases.

This algorithm's update rule for the weights and biases is:

$$wb_{new} = wb + (J^T \times J + \alpha \times I)^{-1} \times J^T \times (y - out) \tag{3.27}$$

where wb is the vector of all the weights and biases, wb_{new} is the new vector of weights and biases, J is the jacobian matrix of the network, α is the learning rate (which will be automatically changed as we go through each epoch of training), I is the identity matrix of appropriate dimension, y is the vector of all the output samples and out is the vector of the outputs using all input samples. In some variations of the algorithm I is replaced by $J^T \times J$.

To further analyze we will look at the elements of the above equation in more detail. First we have wb, which is a vector of all the weights and biases. The order in which we insert the weights in this vector is not important if we use the same order in the other elements of the equation. For this work we will use the following order: first we will put the weights of layer 1, which will be arranged by first inserting the the fist line of the matrix, then the second and so on; then we will insert the weights of layer 2 using the same order as the ones in layer 1 and so on until we have all the weights in the vector; finally we will insert the biases of layer 1, then layer 2 and so forth. We

will get the following vector:

$$wb = \begin{bmatrix} w^1_{1,1} \\ w^1_{1,2} \\ w^1_{1,3} \\ \vdots \\ w^1_{1,N_1} \\ w^1_{2,1} \\ \vdots \\ w^1_{M_1,N_1} \\ w^2_{1,1} \\ \vdots \\ w^2_{M_2,N_2} \\ \vdots \\ w^L_{M_L,N_L} \\ b^1_1 \\ \vdots \\ b^1_{M_1} \\ b^2_1 \\ \vdots \\ b^L_{M_L} \end{bmatrix} \tag{3.28}$$

where M_i is the total number of perceptrons in layer i, N_i is the total number of perceptron in layer $i-1$ or, in the case of $i = 1$, the number of inputs. L is the total number of layers.

The vector *out* will be:

$$out = \begin{bmatrix} o_{1,1} \\ o_{1,2} \\ o_{1,3} \\ \vdots \\ o_{1,M} \\ o_{2,1} \\ \vdots \\ o_{I,M} \end{bmatrix} \tag{3.29}$$

where $o_{1,2}$ is the output of perceptron 2 of the output layer (out of M), when we use the first input (out of I) from the data set to get this output.

The vector y will have exactly the same structure as out. Finally, the Jacobian matrix J will be:

$$J = \begin{bmatrix} \frac{\partial o_{1,1}}{\partial wb_1} & \frac{\partial o_{1,1}}{\partial wb_2} & \cdots & \frac{\partial o_{1,1}}{\partial wb_N} \\ \frac{\partial o_{1,2}}{\partial wb_1} & \frac{\partial o_{1,2}}{\partial wb_2} & \cdots & \frac{\partial o_{1,2}}{\partial wb_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial o_{1,M}}{\partial wb_1} & \frac{\partial o_{1,M}}{\partial wb_2} & \cdots & \frac{\partial o_{1,M}}{\partial wb_N} \\ \frac{\partial o_{2,1}}{\partial wb_1} & \frac{\partial o_{2,1}}{\partial wb_2} & \cdots & \frac{\partial o_{2,1}}{\partial wb_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial o_{I,M}}{\partial wb_1} & \frac{\partial o_{I,M}}{\partial wb_2} & \cdots & \frac{\partial o_{I,M}}{\partial wb_N} \end{bmatrix} \tag{3.30}$$

where $o_{1,2}$ has the same meaning as above and wb_2 is the second element (out of N) of the vector wb. Here we can see that for this algorithm we need to compute the partial derivative of each of the outputs with regards to each weight and bias, whereas in the previous GD algorithm we had to compute the partial derivative of the Cost function with respect to the same parameters. This means that we will need to compute M times more partial derivatives than in the previous algorithm. In addition to this we have a matrix inversion and more multiplications. This is why this algorithm is more complex.

Of notice is the fact that computing $\frac{\partial o_{i,m}}{\partial wb_n}$ is very similar to computing $\frac{\partial C}{\partial w^l_{j,k}}$. In fact, the process is almost the same but, instead of computing a delta for each perceptron, we will need to compute M deltas for each perceptron, where M is the number of perceptrons in the output layer. Once more we start by computing the delta of the output layer. Since each output is only connected to its own perceptron, this will simplify the calculations necessary:

$$delta^L_{j,m,i} = \begin{cases} \sigma'(z^L_{j,i}), & \text{if } m = j \\ 0, & \text{if } m \neq j \end{cases} \tag{3.31}$$

where $delta^L_{j,m,i}$ is the delta for the perceptron j in the output layer with respect to the output m for the input sample i. With this we obtain $M \times I$ deltas for each perceptron (where most of them equal 0). We then need to propagate each of these deltas exactly the same way as we did in equation 3.22, but we must be careful to keep track of the output and input that each delta refers to.

$$delta^{l-1}_{m,i} = \sigma'(z^{l-1}_i) \odot \left((w^{l-1})^T \times delta^l_{m,i} \right) \tag{3.32}$$

where $delta^{l-1}_{m,i}$ and $delta^l_{m,i}$ are the vectors of the deltas with respect to output m and input i for each perceptron on layer $l-1$ and l respectively. To get all the deltas for layer $l-1$ we will need to use this equation once for each combination of output and input.

After getting all the deltas we can compute the elements of the Jacobian matrix by doing the same as in 3.25 and 3.26, again being careful to keep track of the output and input that each derivative corresponds to.

$$\frac{\partial o_{i,m}}{\partial w^l} = a_i^{l-1} \times (delta_{m,i}^l)^T \tag{3.33}$$

$$\frac{\partial o_{i,m}}{\partial b^l} = delta_{m,i}^l \tag{3.34}$$

where a_i^{l-1} is the vector of outputs of the perceptrons of layer $l-1$, applying the input i. Using this we can compute all that is needed for the Jacobian matrix. Afterwards we can compute the new weights and biases using the rule in equation 3.27.

Now comes another step that is different from the GD algorithm. We do a new forward propagation and compute the new cost function. If the cost function has a better result, we accept the new weights and biases, otherwise we remain with the previous ones. Either way, the α in equation 3.27 is adjusted depending on the result. The full flowchart of this algorithm can be seen in Figure 3.5. As it can be seen, after each iteration the learning rate α is updated. This changing learning rate is one of the biggest advantages of this algorithm. When α has a large enough value, this algorithm behaves in the same way as the GD algorithm. When α is very small, the algorithm has a behavior similar to the Gauss-Newton algorithm, which has a faster convergence when we are near the solution (minimum of the cost function). So, if we get closer to the minimum ($S_{new} < S$), α gets smaller so that the convergence can be faster. If we get further away from the minimum ($S_{new} > S$), α will get bigger. The factor that alters α can be changed but its value is usually within the $[2\ \ 10]$ interval. If, at any point, we get a S_{new} smaller than our maximum acceptable value, we can stop the training. Of course this mechanism can be disabled simply by doing $S_{max} = 0$. One final thing worth noting is that if we get more than five consecutive scenarios where $S_{new} > S$, we will accept the new weights anyway. This mechanism is present to allow us to get away from the current point in the hyper-surface and try to escape a local minimum in order to try to find the global minimum, or a better local minimum.

This finishes our analyses of the training algorithms. It is important to note that in spite of the fact that generally the LM obtains better results, in some occasions the GD will perform better. So it is important to keep this in mind and alternate between training algorithms when we are developing a neural network.

3.3.4 Stopping the training and validation

The training algorithms for neural networks are iterative. Knowing when to stop is important so that we don't waste time when it is impossible to get better results.

This can be achieved by setting a stopping condition such as the one we saw in the LM algorithm. This stopping condition is useful when we know the limits our network will ever achieve or when we know that we don't need any more precision than a set known value. For example, if

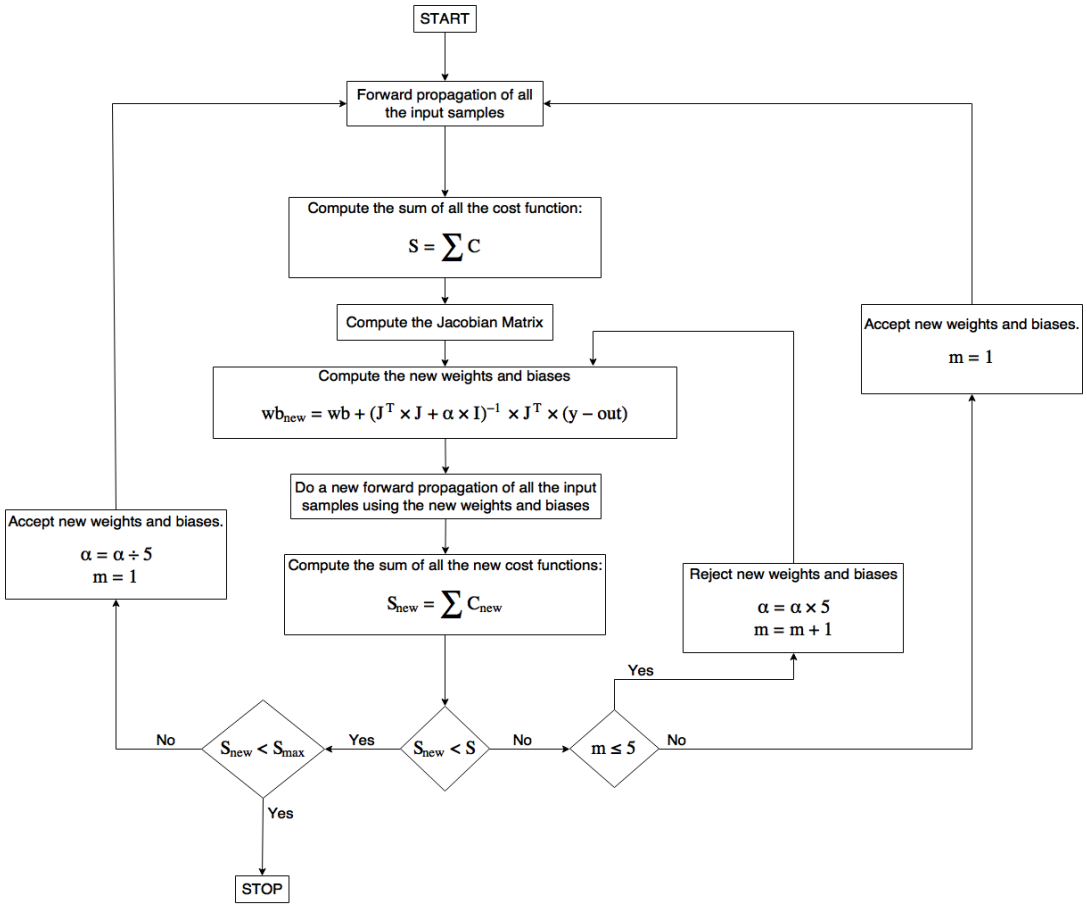


Figure 3.5: Flowchart of the LM algorithm

we are using the output of our network as the input of a system that has an error in the 10^{-1} power order, it is wasteful training the network to have a precision in the 10^{-4} power order (assuming that the error propagation is linear).

However, sometimes we don't have any of these conditions. Therefore, the best way to know when to stop the training is to observe the evolution of the mean of the cost function over all the input samples. The curve of the mean of the cost function versus epoch will eventually stagnate, with each new epoch giving insignificant improvements to the cost function. As soon as we reach that point, we can stop the training and proceed to the next step, the validation.

The validation of the network is a crucial step after its training. It exists with the purpose of ensuring that the network can perform its task outside of the training data set. For the validation we use a different data set and simply run the inputs through the network and compute the mean cost function. This result will give us a feel for how the network will behave with new samples and if it will be able to provide a good model. Naturally, the new mean cost function will be slightly worse than the last one obtained in the training, but the difference should be small. If that is not the case it is possible that the network has suffered from overfitting. Overfitting is an occurrence in machine learning when the model under training (in our case the ANN) fails to understand the underlying relation between inputs and outputs and, instead, tries to fit all the data points.

An exaggerated example of such a case can be seen in figure 3.6. In it we can observe that the desired separation curve between the red and blue dots is in black. However due to some misplaced dots (that can be the result of noise or another random error) the network overfits and draws the green line to separate the dots. Using this data set, the green line produces better results than the black line, but if we give random coordinates to the network and ask it if the dot will be red or blue, the black line will produce better results. And the purpose of creating a network is using it in the future to predict results, so we don't want our network to overfit.

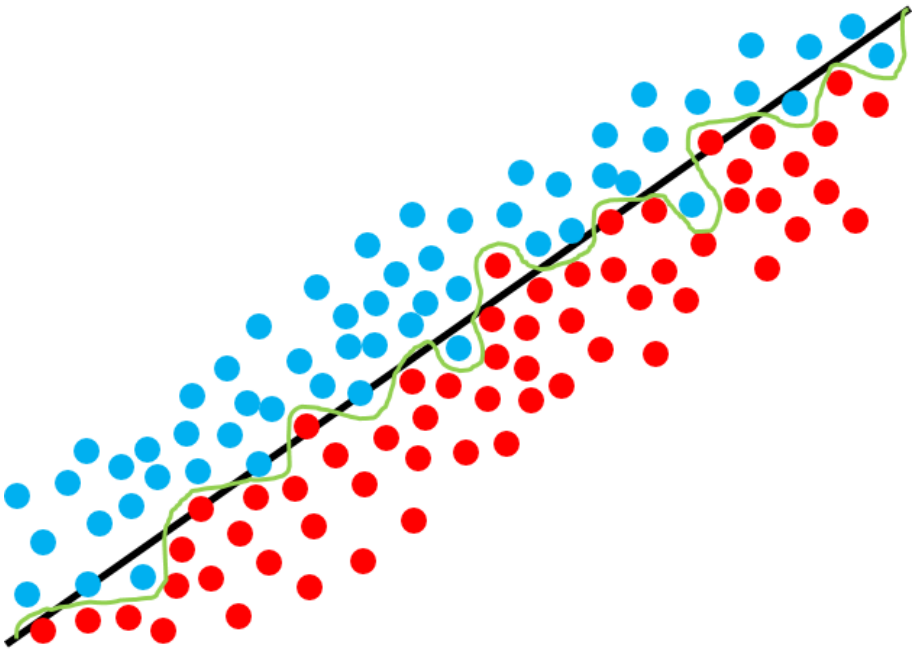


Figure 3.6: Example of overfitting

Overfitting can be the result of a network with too many perceptrons or hidden layers. However, if we don't use enough perceptrons we may be unable to find a good relation between inputs and outputs - underfitting. This tradeoff is obviously very important and must be taken into account, along with many others, when designing the network. This just serves to prove that although there are some guidelines we can follow when designing a network, it is certainly not and exact and direct process. Trial and error is necessary to get good results.

Chapter 4

Neural Networks for Predistortion

Now that we've looked at the functioning of ANNs we need to see how they will be integrated in the global architecture of the system so that they can do predistortion.

Figure 2.4 of section 2.3.3 displays the basic block diagram for the predistortion method. We will want this predistortion block to be an ANN that does the inverse of the PA. We saw that we can train an ANN to be any continuous function. However the ANN will be static, that is, it won't have any memory effect. As we saw previously, PAs have memory effects and, as a consequence, so will their inverse function.

Therefore, we need to come up with a method to integrate memory effects in our predistortion block. That can be achieved by manipulating what the inputs and outputs of the ANN will be.

4.1 Inputs and Outputs of the ANN

We know that we can create an ANN with any number of inputs and outputs. Deciding what will be inputs and outputs of the network is an extremely important factor in the success of the ANN.

To begin with, we know that the inputs of the predistortion block will be the current input signal and the output of the predistortion block is going to be the predistorted signal that will go on to the PA (after some modifications). This signal will be complex so we can view it in one of two ways: either as a single complex signal or as two signals, separating the real part from the imaginary one (cartesian form) or by separating its phase and magnitude (polar form). For now we will disregard these possibilities and treat it as a single signal.

First lets focus on the problem of not having memory effects in our ANN. This is corrected by using a tap-delay sequence in the inputs of the ANN, as it can be seen figure 4.1. As shown, the inputs of the ANN will be the current signal and the M previous inputs. Thus, the current output of the ANN will be impacted by these M previous input samples and therefore, the whole system will have memory.

Now lets look at the type of input and output signals that we want. Using a single complex signal would be impractical because that would mean that we needed activation functions that could take complex inputs. Therefore we are left with two choices: we can separate the signal in

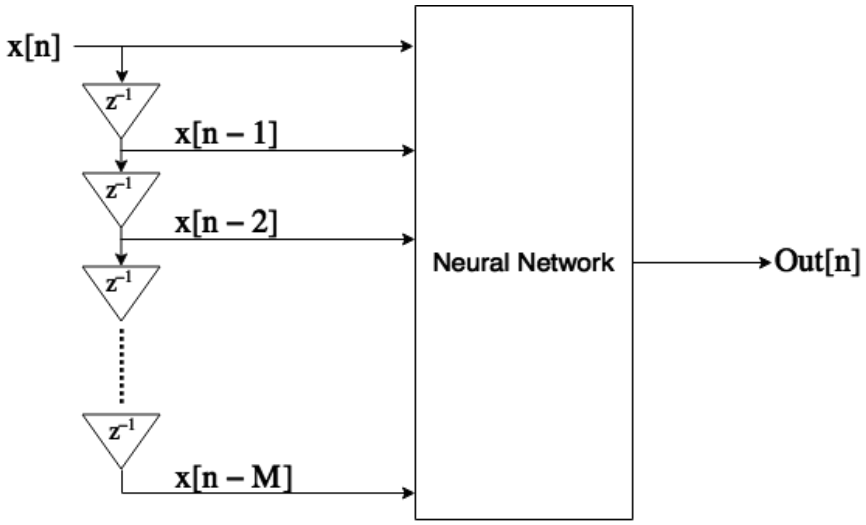


Figure 4.1: Neural network with a tap-delay sequence in the input

the cartesian form or in the polar form. The former is the most used and consists in separating the input signals in their Quadrature and In-phase parts. Evidently each of these parts will have their own tap-delay sequence. The resulting structure can be seen in figure 4.2. In it we have a complex input and output signal x and y such that:

$$x[n] = x_Q[n] + jx_I[n] \tag{4.1}$$

$$Out[n] = Out_Q[n] + jOut_I[n] \tag{4.2}$$

Predistortion blocks using this structure have been used with different degrees of success.

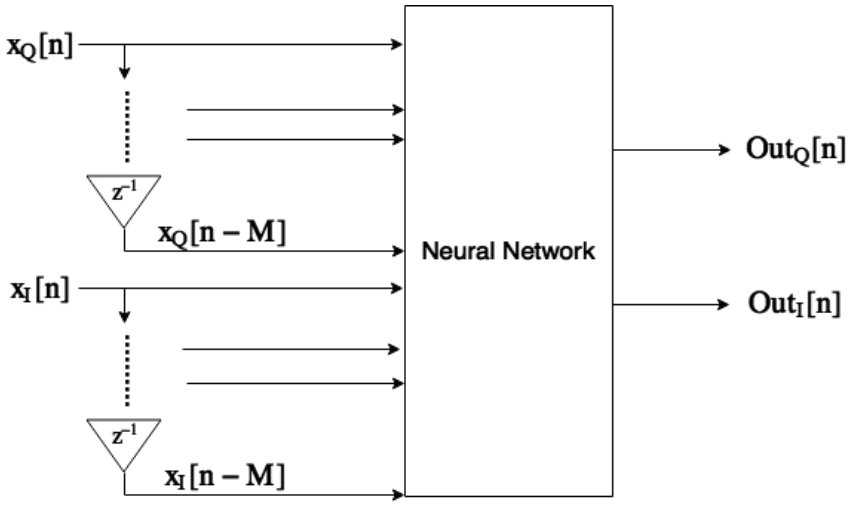


Figure 4.2: ANN with inputs and outputs in the cartesian form

In this structure we can see that we have two outputs and $2 \times (M+1)$ inputs. However a

recent paper [23] states that this selection of inputs and outputs is not optimal when the input is a wideband signal. This was concluded because it was shown that an ANN of this type produces non-physical terms, i.e terms that a PA, or its inverse function, are unable to produce. Hence, these terms are unwanted and introduce an error. The same study proposed an alternative ANN that uses the polar form. The proposed ANN is shown in figure 4.3. In it we have a complex input and output signal x and y such that:

$$x[n] = a[n] \times e^{j\theta[n]} \tag{4.3}$$

$$Out[n] = b[n] \times e^{j\alpha[n]} = b[n] \times e^{j(\theta[n]+\varphi[n])} \tag{4.4}$$

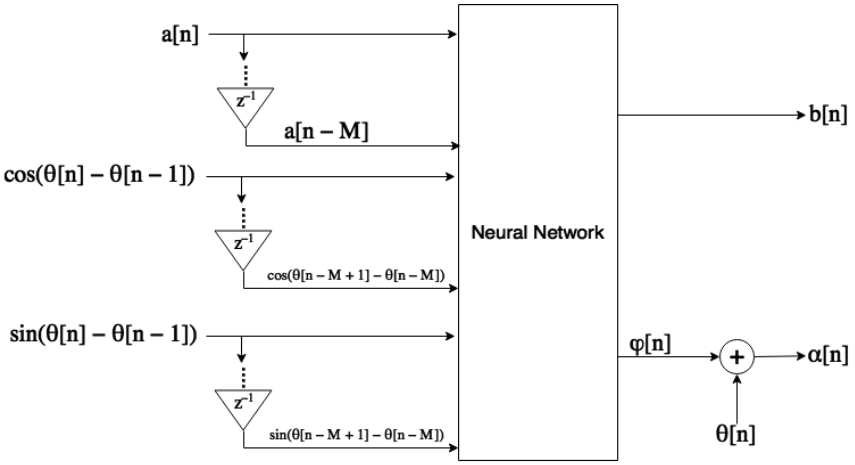


Figure 4.3: ANN with inputs and outputs in the polar form

In this structure we have two outputs and $M+1+M+M$ inputs. It is noticeable that the parts of the complex signal are not used in a straightforward way as it was done in the previous structure. We have as the input of the network the magnitude of the signal and the cosine and sine of the difference between the current and previous phase of the signal, as is possible to see in the figure. Each one of these has its own tap-delay sequence. This leads to a higher number of inputs than in the cartesian structure for almost all values of M. To be precise, the polar structure has $M-1$ more inputs. This would mean that for any M greater than 1 the polar structure has more inputs. In the output of the ANN we will get the magnitude of the signal and the phase correction which will be added to the phase of the current input signal to get the current output signal.

Still in the same paper, it was proven that this structure produces only physically meaningful terms and should therefore be better than the cartesian structure.

4.2 Global structure of the system

Regardless of what structure for the ANN is chosen, either the one from figure 4.2 or the one from figure 4.3, they both need to learn how to implement the inverse of the PA. For that purpose it is

necessary to add a section to figure 2.4.

The full system will look like the one seen in figure 4.4. This figure does not portrait the tap-delay sequences and the different types of inputs and outputs for the neural networks for simplicity. We can assume that the parts of the complex signal are combined in the Upconverter block and are separated in the Downconverter block. It is also important to note that this system is designed to work in baseband, since it will require a lower sampling rate to obtain the same results due to the frequency being much lower. If needed we could implement it in order to work in intermediate frequencies or in RF, using the same idea displayed in the figure.

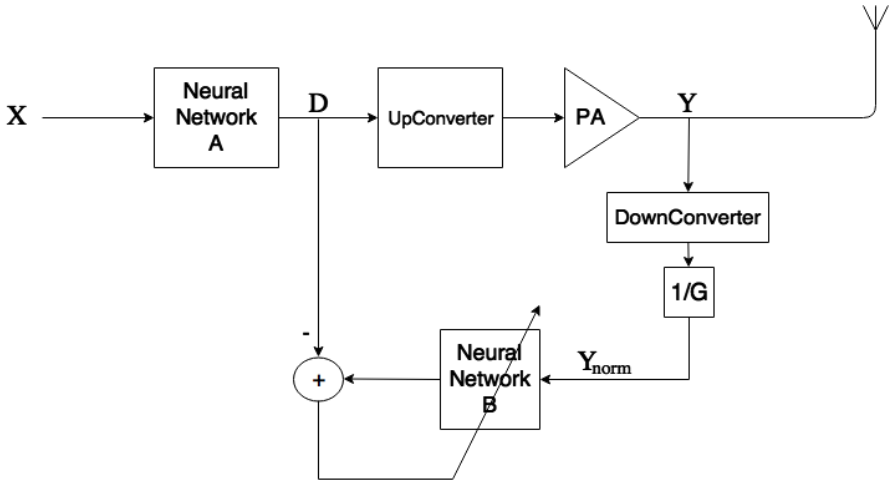


Figure 4.4: Full block diagram for predistortion using Neural Networks

The way the system works is that first the baseband input signal X will go through the predistortion block (Neural Network A) which generates signal D. In turn, this signal will go through the Upconverter block, which converts the signal from baseband to RF. Then the signal goes through the PA and we obtain the signal Y. From here, this signal will be transmitted through an antenna. At the same time we will take the Y signal, pass it through the Downconverter block, which will convert the RF signal to baseband and then we will attenuate the signal by G. This will result in a normalized Y signal that is in baseband, Y_{norm}.

Y_{norm} will be used as the input for Neural Network B. The output of this network will be compared with the corresponding D signal so that we can train Neural Network B. It is possible to see that Neural Network B is being trained to do the inverse function of the PA, in baseband. This neural network will then be copied to Neural Network A. That way Neural Network A is suffering an indirect training so that it can do the inverse function of the PA, which is our objective.

Chapter 5

Tests

To verify the capabilities of predistortion of Neural Networks, a series of tests were done with various degrees of complexity.

All tests in chapter 5.1 were done in a MatLab environment. All the code used was of original creation (except the functions of the RFWebLab) and no toolbox was used, with the exception of the signal processing toolbox, which was only used for its align signals function. In section 5.2 and 5.3 in addition to MatLab, a PA excited with real signals was used, along with the necessary instruments to generate and measure the signal. At all points we worked with digital predistortion.

5.1 Inverse of the PA

The first step taken towards the construction of the final system was the training and validation of a neural network for it to be able to produce the inverse function of the PA.

The objectives of this step were:

- the construction of a working script for training and validation of a neural network;
- to compare the performance of the GD and the LM training algorithms;
- to compare the performance of polar and cartesian structure of inputs and outputs of the neural network;
- to get a feel for the necessary number of delays (M) and the number of perceptrons necessary in the hidden layer;
- to compare the performance of the hyperbolic function against a log exponential function as activation functions of the hidden layer.

The neural network will have one hidden layer and one output layer as its internal structure. The output layer will have two perceptrons given that both output structures (polar and cartesian) have two outputs. The activation function of the output layer will be simply $\sigma(z) = z$. The number of perceptrons in the hidden layer and the number of delays will vary from test to test with the

goal of finding the minimum necessary to achieve a certain goal. The activation function of the hidden layer will be either the hyperbolic tangent function or a log exponential defined as:

$$\sigma(z) = \frac{1}{2} \times \left(z + \log(e^{z} + e^{-z})\right) \tag{5.1}$$

A plot of this function can be seen in figure 5.1. It is possible to see that this function has an exponential behavior around and before zero and presents a linear behavior after. This function is used because it is possible to easily implement it in analog format either by using a diode or transistors in the sub-threshold region. Since the end goal is to have an analog predistortion block, having a easily implementable activation function is an advantage. But first we need to know that it can have a performance at least similar with the hyperbolic tangent function, which is an activation function that has already been used with some degree of success in predistortion.

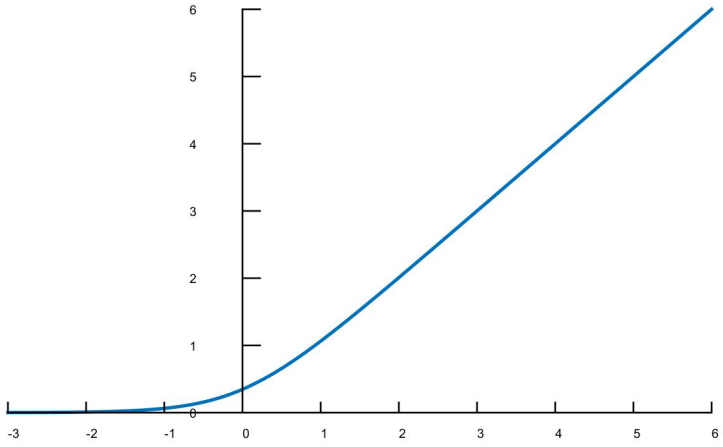


Figure 5.1: Plot of the log exponential function

Two MatLab scripts were created to reach these goals, one for each of the training algorithms. These scripts implemented functions that took as parameters:

- the number of perceptrons in the hidden layer - $hl1$;
- the number of delays to use - M;
- the structure for the input and outputs desired - *polar*, which if set to 1 would use the polar structure. Otherwise it would use the cartesian structure;
- the activation function to be used in the hidden layer - *act*, which if set to 1 would use the log exponential function. Otherwise it would use the hyperbolic tangent function;
- The number of iterations of the training algorithm - $itTrain$;
- In the case of the GD script, the learning rate to be used in the training - *eta*

The cost function chosen for the training was in both cases:

$$C[n] = \frac{1}{2} \times (Out_{real}[n] - Y_{real}[n])^2 + (Out_{imag}[n] - Y_{imag}[n])^2 \tag{5.2}$$

where $Y_{real}[n]$ and $Y_{imag}[n]$ are the real and imaginary parts of the targets for the sample n from the dataset. $Out_{real}[n]$ and $Out_{imag}[n]$ are the real and imaginary parts of the complex signal obtained from the outputs of the network for input sample n. The complex output signal of the network is in equation 4.2 for the cartesian case and in equation 4.4 for the polar case.

The function in both scripts evaluated the normalized mean square error (NMSE) of the output in relation to the targets at the end of each iteration of training. The NMSE can be computed in the following manner:

$$NMSE = \frac{\sum_n \|Out[n] - Y[n]\|^2}{\sum_n \|Y[n]\|^2} \tag{5.3}$$

No early stopping was set for the training since we just want to see the performance of the algorithms and we have no NMSE value as a goal.

After the training, the function performed the validation of the network by calculating the NMSE of the network using the dataset defined for validation (which is different than the one for the training).

Finally the function returned all the NMSE values obtained in the training and the NMSE obtained in the validation.

It is important to note that in the training process for the polar structure special attention is needed because the cost function doesn't use the values of the output of the network directly. So we will need to take some time to analyze this problem before continuing with the results of the tests.

In the GD training this has an impact when we compute $\frac{\partial C}{\partial a_j^L}$, which in this case will correspond to $\frac{\partial C[n]}{\partial b[n]}$ and $\frac{\partial C[n]}{\partial \varphi[n]}$. For the former we will have (we will omit the sample indication n for simplicity):

$$\begin{aligned}
\frac{\partial C}{\partial b} &= \frac{1}{2} \times \frac{\partial \left((Out_{real} - Y_{real})^2 + (Out_{imag} - Y_{imag})^2 \right)}{\partial b} \\
&= \frac{1}{2} \times \frac{\partial \left(\left(b \times \cos(\theta + \varphi) - Y_{real} \right)^2 + \left(b \times \sin(\theta + \varphi) - Y_{imag} \right)^2 \right)}{\partial b} \\
&= \frac{1}{2} \times \left(2 \times \frac{\partial (b \times \cos(\theta + \varphi) - Y_{real})}{\partial b} \times (b \times \cos(\theta + \varphi) - Y_{real}) + \right. \\
&\left. + 2 \times \frac{\partial (b \times \sin(\theta + \varphi) - Y_{imag})}{\partial b} \times (b \times \sin(\theta + \varphi) - Y_{imag}) \right) \\
&= \cos(\theta + \varphi) \times (b \times \cos(\theta + \varphi) - Y_{real}) + \sin(\theta + \varphi) \times (b \times \sin(\theta + \varphi) - Y_{imag}) \\
&= b - Y_{real} \times \cos(\theta + \varphi) - Y_{imag} \times \sin(\theta + \varphi)
\end{aligned} \tag{5.4}$$

And for the other output we have:

$$\begin{aligned}\frac{\partial C}{\partial \varphi} &= \frac{1}{2} \times \frac{\partial \Big((Out_{real} - Y_{real})^2 + (Out_{imag} - Y_{imag})^2 \Big)}{\partial \varphi} \\ &= \frac{1}{2} \times \Bigg(2 \times \frac{\partial (b \times \cos(\theta + \varphi) - Y_{real})}{\partial \varphi} \times (b \times \cos(\theta + \varphi) - Y_{real}) + \\ &+ 2 \times \frac{\partial (b \times \sin(\theta + \varphi) - Y_{imag})}{\partial \varphi} \times (b \times \sin(\theta + \varphi) - Y_{imag}) \Bigg) \\ &= b \times \Big(-\sin(\theta + \varphi) \Big) \times (b \times \cos(\theta + \varphi) - Y_{real}) + b \times \cos(\theta + \varphi) \times (b \times \sin(\theta + \varphi) - Y_{imag}) \\ &= b \times Y_{real} \times \sin(\theta + \varphi) - b \times Y_{imag} \times \cos(\theta + \varphi)\end{aligned} \tag{5.5}$$

The rest of the algorithm follows as usual.

In the LM training we also need to make changes. Since the cost function uses Out_{real} and Out_{imag}, those are the outputs of the network as far as the learning algorithm is concerned. So eq. 3.31 will need to be changed. In the case of the perceptron 1 of the output layer (the one responsible for b) we will get (again we will not include the sample indication n and its equivalent in eq. 3.31, i)

$$\begin{aligned} delta^2_{1,m} &= \begin{cases} \sigma'(z^2_1) \times \dfrac{\partial Out_{real}}{\partial b}, & \text{if } m = 1 \\ \sigma'(z^2_1) \times \dfrac{\partial Out_{imag}}{\partial b}, & \text{if } m = 2 \end{cases} \\ &= \begin{cases} \sigma'(z^2_1) \times \dfrac{\partial \Big(b \times \cos(\theta + \varphi) \Big)}{\partial b}, & \text{if } m = 1 \\ \sigma'(z^2_1) \times \dfrac{\partial \Big(b \times \sin(\theta + \varphi) \Big)}{\partial b}, & \text{if } m = 2 \end{cases} \\ &= \begin{cases} \sigma'(z^2_1) \times \cos(\theta + \varphi), & \text{if } m = 1 \\ \sigma'(z^2_1) \times \sin(\theta + \varphi), & \text{if } m = 2 \end{cases} \end{aligned} \tag{5.6}$$

where L was substituted by 2 since we are in the output layer and the network only has 2 layers. So $delta^2_{1,m}$ is the delta of the perceptron 1 of the layer number 2 for the output m, z^2_1 is the z of perceptron 1 of the layer number 2. In this case we considered Out_{real} to be output 1, so $m = 1$ refers to it. Out_{imag} was considered output 2. We could have done it the other way around, given that we were consistent with our choice.

The delta of the perceptron 2 of the output layer will be:

$$
\begin{aligned}
delta_{2,m}^{2} &= \begin{cases} \sigma^{'}(z_2^2) \times \dfrac{\partial Out_{real}}{\partial \varphi}, & \text{if } m = 1 \\ \sigma^{'}(z_2^2) \times \dfrac{\partial Out_{imag}}{\partial \varphi}, & \text{if } m = 2 \end{cases} \\
&= \begin{cases} \sigma^{'}(z_2^2) \times \dfrac{\partial \Big(b \times \cos(\theta + \varphi)\Big)}{\partial \varphi}, & \text{if } m = 1 \\ \sigma^{'}(z_2^2) \times \dfrac{\partial \Big(b \times \sin(\theta + \varphi)\Big)}{\partial \varphi}, & \text{if } m = 2 \end{cases} \\
&= \begin{cases} -\sigma^{'}(z_2^2) \times b \times \sin(\theta + \varphi), & \text{if } m = 1 \\ \sigma^{'}(z_2^2) \times b \times \cos(\theta + \varphi), & \text{if } m = 2 \end{cases}
\end{aligned}
\tag{5.7}
$$

where we made substitutions similar to the previous equation.

Now that we have seen the modifications needed to be done, we can proceed to the results obtained.

5.1.1 Results

Multiple tests were done. There were eight groups of tests, one for each combination of training algorithm, input/output structure and activation function. In each group various tests were run by changing the memory considerations (M) from 1 to 9, and for each of these the number of perceptrons in the hidden layer was changed from 5 to 30 with a step of 5. So a total of $8 \times 9 \times 6 = 432$ tests were run.

The dataset used for these tests was a 4 carrier GSM signal that passed through a GaN PA. The signals were already aligned and the full dataset contained 65000 samples, of which about 40000 sequential samples were selected for the training and about 15000 sequential samples were selected for the validation.

In the case of the tests using the GD algorithm, the learning rate used was of 0.001. This value was the one that produced the best results from the ones tried.

To guarantee the maximum comparability of the data, it was used the same seed for the random number generator in all tests. The number of iterations of the training were set to 300 for the LM algorithm and 400 for the GD algorithm.

The results can be seen in table A.1 in Appendix A. In the table, "Train. Alg" is the training algorithm used, "In/Out" is the structure of inputs and outputs used, "Act. Func." is the activation function used in the hidden layer, "M" is the number of delays used for each type of input, "Nº Perceptrons" is the number of perceptrons used in the hidden layer, "NMSE T. (dB)" is the dB value of the NMSE at the end of the training and "NMSE V. (dB)" is the dB value of the NMSE

obtained in the validation. The value used to determine the performance of each network is the value of the NMSE in the validation. Obviously we want this value to be as low as possible.

In the following sections we will use these results to compare the parameters used in the training.

5.1.1.1 Training algorithm

We'll start with the training algorithm. It is very clear that the LM algorithm performs much better as expected. Out of the 432 tests, the 110 best ones use the LM training algorithm. The best GD result for the NMSE in the validation was of -37.45 dB, whereas the LM algorithm was able to achieve -43.81 dB.

Added to this is the fact that we don't need to "search" for the best learning algorithm value when we use the LM algorithm.

Because of these clear advantages, in the next sections we will only use the results from the LM algorithm for the comparisons.

5.1.1.2 Input and Output structure

For this analysis two tables were created from the total results table. The first table (5.1) uses only the results that have the LM training algorithm and the hyperbolic tangent activation function. From those, the best result from every In/Out and M combination was selected. That is, the best result with In/Out=Polar and M=1 was selected, then the best result with In/Out=Polar and M=2 was selected and so on. The results were then organized according to the NMSE V., from lowest to highest. Table 5.2 was created similarly but for the log exponential activation function.

In both tables it is possible to see that the best results were achieved using the Polar structure (the best six in the hyperbolic tangent case and the best five the the log exponential case). The best Cartesian result lags around 1 dB behind the best Polar result in both tables. Looking at the other end of the spectrum it is also true that the worst results were obtained using the Polar structure. The Cartesian structure seems to perform in a more stable manner with the difference between the best and worst result being of around 1 dB in table 5.1 and around 2 dBs in table 5.2.

5.1.1.3 Activation Function of the hidden layer

As it was possible to see from tables 5.1 and 5.2, both activation functions produce results within the same range. This is a positive initial sign for the log exponential since, as was mentioned before, this activation function can be easily implemented analogically and we just want to make sure that it performs at the same level as that hyperbolic tangent function.

5.1.1.4 M and Nº of Perceptrons in the hidden layer

It is possible to see in the tables 5.1 and 5.2 that for the Polar structure, a low M achieves the best results. In the Cartesian case, mid values seem to be best. However, disregarding the low Polar

Table 5.1: Selection of results taken from table A.1, with the purpose of comparing of the In/Out structure for the hyperbolic tangent case.

Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE V. (dB)
LM	Polar	tanh	1	10	-43,79
LM	Polar	tanh	4	5	-43,71
LM	Polar	tanh	5	20	-43,70
LM	Polar	tanh	3	20	-43,68
LM	Polar	tanh	7	15	-43,63
LM	Polar	tanh	2	10	-43,34
LM	Cartesian	tanh	7	30	-42,83
LM	Cartesian	tanh	4	25	-42,69
LM	Cartesian	tanh	5	25	-42,60
LM	Cartesian	tanh	3	30	-42,55
LM	Cartesian	tanh	2	25	-42,45
LM	Cartesian	tanh	6	25	-42,44
LM	Cartesian	tanh	1	25	-42,38
LM	Cartesian	tanh	8	25	-41,64
LM	Cartesian	tanh	9	30	-41,61
LM	Polar	tanh	6	5	-37,74
LM	Polar	tanh	9	5	-34,19
LM	Polar	tanh	8	25	-29,77

Table 5.2: Selection of results taken from table A.1, with the purpose of comparing of the In/Out structure for the log exponential case.

Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE V. (dB)
LM	Polar	log_exp	1	20	-43,81
LM	Polar	log_exp	2	10	-43,45
LM	Polar	log_exp	3	5	-43,42
LM	Polar	log_exp	6	5	-43,41
LM	Polar	log_exp	7	5	-43,36
LM	Cartesian	log_exp	2	30	-42,76
LM	Cartesian	log_exp	7	30	-42,68
LM	Cartesian	log_exp	9	25	-42,68
LM	Cartesian	log_exp	8	25	-42,66
LM	Cartesian	log_exp	1	30	-42,62
LM	Cartesian	log_exp	6	20	-42,62
LM	Cartesian	log_exp	4	30	-42,58
LM	Cartesian	log_exp	5	20	-42,22
LM	Cartesian	log_exp	3	30	-40,63
LM	Polar	log_exp	8	10	-36,35
LM	Polar	log_exp	4	5	-33,63
LM	Polar	log_exp	5	10	-30,53
LM	Polar	log_exp	9	5	-29,15

results, varying M doesn't seem to make a significant impact. This gives the advantage to low values of M since they result in less complex neural networks (since there are less inputs) and, therefore, less computations needed in the training.

As for the N^o of Perceptrons, again it seems that the Polar structure requires only a low number of them to produce the best results. The Cartesian structure seems to work better with more perceptrons. This would give an advantage to the Polar structure since less perceptrons translates in a less complex neural network and, again, this would result in less computations needed in the training.

All in all, the most significant and clear conclusion from this set of tests is that the LM algorithm performs much better than the GD algorithm. We can also say that the polar structure is better than the cartesian one since it produced the best results with less memory and perceptrons. But as mentioned this advantage is not significant enough so that we can disregard the cartesian structure completely.

5.2 Complete Predistortion System

Now that we have a working script for the training of a network we can advance to the test of the entire system.

The complete system was already shown in chapter 4 in figure 4.4. The figure will be repeated here for convenience.

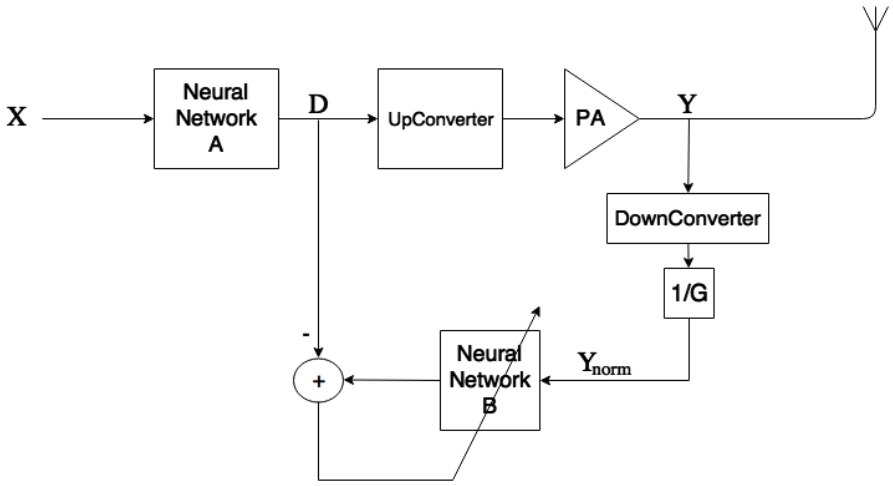


Figure 5.2: Full block diagram for predistortion using Neural Networks (repeated)

As was stated before, the way the system works is as follows: the input signal X goes through the predistortion block (Neural Network A), generating signal D. D goes through an upconvertion and the PA, generating signal Y. Signal Y is downconverted and normalized and the resulting signal (Y_{norm}) serves as the input data of a neural network to be trained. The target used in the training will be signal D. When Neural Network B finishes its training, its parameter are copied to neural network A. That is, Neural Network A is always a copy of Neural Network B.

Regarding the test, the way it is done is in batches. We first select a large sequential number of samples from the input signal (we used 40000). These are our input samples X. Then we generate the 40000 samples of D, by running all the samples of X through the neural network. Then all the samples D are transformed into samples Y_{norm}, using a tool that will be described later. Finally we train Neural Network B using as input samples the 40000 samples of Y_{norm}, and the 40000 samples of D as targets. After this we copy the parameters (the weights and biases) from Neural Network B to A and we begin the process again. Obviously, after doing all the necessary iterations of this process, we select a different set of input samples X and run it through the system to validate it.

After every iteration we can compare X to Y_{norm} so that we can see the effectiveness of the predistortion. This is done by computing the NMSE and the ACPR. We want the former to be as low as possible and the later to be as high as possible. The performance of the predistortion will be evaluated based on its ACPR, but the NMSE is also an important indicator.

As was mentioned before, we must have a way to generate Y_{norm} from D. For this the RF WebLab from Chalmers University was used [24].

5.2.1 RF WebLab

The RF WebLab is a platform created with the purpose of facilitating the evaluation of the performance of digital predistortion algorithms.

By using it we can remotely access a measurement system that includes a signal generator, a nonlinear GaN power amplifier and a signal analyzer. The full description of the measurement setup can be found in the webpage of the WebLab and is quoted here:

> "The measurement setup is based on a PXI Chassi (PXIe-1082) with embedded host PC from National Instruments. The chassis is equipped with a Vector Signal Transceiver (PXIe-5646R VST) with 200 MHz instantaneous bandwidth. The signal generated from the VST transmitter is fed to a linear driver amplifier before the GaN PA DUT (Cree CGH40006-TB, testboard for the transistor CGH40006P). A 30 dB RF attenuator is placed at the DUT output from which the output signal is connected to the VST receiver. A PC embedded in the PXI chassi is used to control the instruments and to down- and upload datafiles at request from the users. The DUT is suppled by a power source module (PXI-4130) which also measures the current consumption of the power amplifier."[25]

One of the ways to use this WebLab is through a MatLab script that is provided. This script implements a function that takes as inputs a vector that is the desired input signal of the PA and a value representing the desired root mean square (RMS) power level setting, in dBm of the signal generator. The function returns the vector of the measured IQ-data sampled at 200 MHz with a bandwidth of 160 MHz, the RMS output power in dBm and the measured current and voltage.

The inputs are restricted in order to protect the system from being damaged. The restrictions are:

> " The maximum allowed peak power from the signal generator is -8 dBm. The maximum allowed rms power level (Pin,RMS) depends on the peak-to-average power ratio (PAPR) of the input signal and will ensure that the peaks of the input signals stay mostly below -8 dBm. Some more input RMS power is allowed for high PAPR signals. There is also a maximum peak-to-average power ratio of the IQ-data signal of 20 dB. If any of these limits are exceeded, the system will not perform the measurement. The output in these cases is a data-file that contains a single “-inf”-value."[26]

One important thing to note is that the output signal provided by the RF WebLab is not aligned with the input signal. An alignment is therefore needed after receiving the output. This makes it so that some samples are lost in the process. But very few samples are lost (values measured are around 5 or 6 samples), which is insignificant because we are using data sets with around 40000 samples.

The RF WebLab will be responsible for the Upconvertion, PA and Downconvertion parts of the full system, leaving us only with the parts responsible for the training and predistortion to worry about. Let us then look at the results obtained.

5.2.2 Results

In the full system shown in figure 5.2 it is important to note that we can distinguish two different trainings. The first one is the training of the Neural Network B, which we will call network training. The second is the system training where each iteration corresponds to a propagation of the X data set to generate the D (by running X through neural network A) and Y_{norm} (by using D as input for the function of the RF WebLab and normalizing its output) and the corresponding network training using the current D and Y_{norm} data sets. So, in each iteration of the system training, we have a full network training.

The system training needs to be an iterative process (as opposed to doing it just once) because the PA works as an operator, so if we change its input function (in this case, the input signal), the output function will also change (in this case the output signal). So when we do the network training we are changing the predistortion block and, therefore, D (and consequently the input of the PA) will also be changed. This will make it so that the nonlinearities imposed to the output will also change. However this change is small enough so that the system training is able to converge. That is, once we do some iterations of the system training, the change of the nonlinearities of the PA due to the change of the predistortion block is so small that the network training will produce only a small change to the parameters of the neural network. This will result in only a small change in the D values and that will, therefore, produce only a small change in the nonlinearities of the PA.

In the first iteration of the system training, there is no Neural Network A defined, so we do $D = A$. After the first iteration, the process goes as described before. It is also important to note that the network training of the first iteration of the system training will need to be more time consuming (in number of iterations) than the later ones. That is because we will start with

a network with random values of weights and biases. In the following iterations of the system training the network training won't need to have a large number of iterations because we know that the network that we have is already close to the desired result. We can see the first iteration of the system training as getting an estimate for the desired network and the other iterations as fine tuning that network.

To test the system, a MatLab script was created. The script can be seen in Appendix C. The functions related to the RF WebLab are not included since they are freely provided in the site. The functions developed for the previous set of tests were adapted and used for the network training. The script takes as input the same parameters as in the previous example, except the *eta*, because it was decided to not use the GD algorithm in this set of tests. This choice was made because of the results of the previous set of tests, where the LM algorithm clearly outperformed the GD algorithm. Both input/output structures were tested because, although the polar structure obtained the best results, the difference between the results obtained by the cartesian structure was not large and, therefore, it was not possible to conclude with certainty that the polar structure would perform better than the cartesian structure. In the script the first network training was set to have 200 iterations and all the other network trainings were set to have 50 iterations. The system training was set to have 25 iterations. When using the function of the RF WebLab, the input rms is set to be as high as possible within the restrictions, so that we use the PA as efficiently as possible. This corresponded to a value between -15 dBm and -18 dBm most of the times.

The ACPR and NMSE values were computed at the end of every iteration of the system training. Also a validation check was done every iteration of the system training, and its ACPR and NMSE were computed. This validation check naturally used a different vector X. These systematic validation checks were done because, as it will be possible to see later, the ACPR and NMSE begin to fluctuate when we reach a certain stability in the training, The purpose of these checks is to see at what iteration was the best NMSE and ACPR achieved. The network that achieved these values would be the selected network to do the predistortion after the system training ended. Naturally, the values obtained in these validation checks were never used for the network training as that would defeat the purpose of the validation.

The script was run for different combinations of M (ranging again from 1 to 9), N^{o} of perceptrons in the hidden layer (ranging from 5 to 40, with steps of 5), hidden layer activation function (hyperbolic tangent or log exponential function) and input/output structure (polar or cartesian). The number of perceptrons in the hidden layer tested was changed to go to 40 instead of the previous 30 because some of the results in tables 5.1 and 5.2 were achieved using 30 perceptrons. So it is possible that a larger number of perceptrons would produce better results. The new maximum value set was of 40 because, even if better results were produced with a larger number of perceptrons, these better results wouldn't be worth the higher complexity of the network. The total number of tests was of $9 \times 8 \times 2 \times 2 = 288$.

The full results table can be seen in table B.1 of Appendix B. The table has entries almost equal to the ones of table A.1 with the added ACPR T. and ACPR V. which are the ACPR of the best iteration of the system training and the best ACPR obtained in the validation checks. The

NMSE V. is the value of the NMSE corresponding to the same iteration as the one that produced the best validation ACPR. No column was allocated for the training algorithm since we always use the LM training algorithm.

As was done in the previous set of tests we will now discuss the results obtained by analyzing each parameter. The method of evaluating the performance of each test is its ACPR V. result obtained.

5.2.2.1 Input and Output structure

Contrary to the previous set of tests, in this set it was very clear to see that the polar structure was definitely better than the cartesian structure. Out of the 288 tests, the best 100 were obtained using the polar structure. That is to say that more than two thirds of the polar networks were better than the best cartesian network.

The best cartesian network achieved an ACPR in the validation of 45,6 while the best polar network achieved a value of 50,03.

Because of these results, in the following sections we will only analyze the results obtained by the tests that used the polar structure.

Table 5.3: Selection of results taken from table B.1

In/Out	Act. F.	M	Nº P.	ACPR V. (dB)
Polar	tanh	2	20	51,74
Polar	tanh	6	15	51,44
Polar	log_exp	4	20	51,31
Polar	tanh	3	25	51,30
Polar	tanh	4	20	51,29
Polar	log_exp	5	10	51,26
Polar	tanh	9	25	51,25
Polar	log_exp	1	30	51,24
Polar	tanh	1	20	51,07
Polar	log_exp	6	10	51,01
Polar	log_exp	9	10	50,77
Polar	log_exp	3	15	50,56
Polar	log_exp	2	20	50,37
Polar	tanh	7	10	50,15
Polar	tanh	5	10	49,59
Polar	log_exp	7	25	49,29
Polar	log_exp	8	40	49,10
Polar	tanh	8	40	46,10

5.2.2.2 Activation Function of the hidden layer

For this analysis and the analysis of the M and Nº of Perceptrons in the hidden layer, a table was created from the total results table. The table (5.3) was created in a similar way as tables 5.1 and

5.2. The elements of the table were ordered according to the ACPR V. obtained.

It is possible to see that the log exponential activation function produces results in the same range as the hyperbolic tangent function. The best results were obtained using the tanh function, but the difference between these results and the best polar result is of only 0.43 dB.

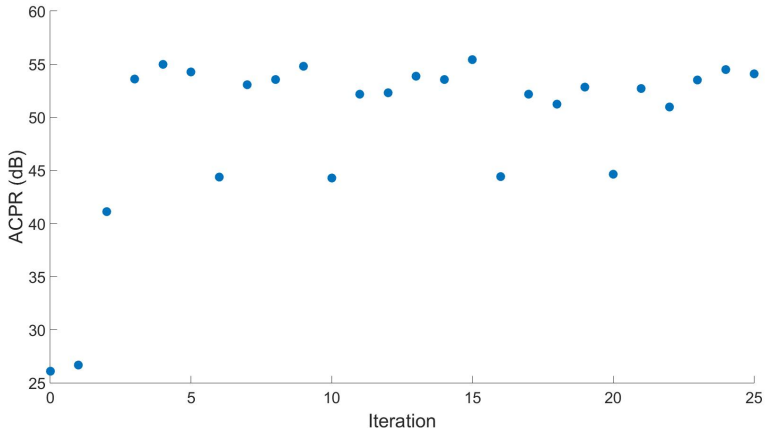
5.2.2.3 M and N of Perceptrons in the hidden layer

It is possible to see that many of the best results for each M were obtained using a number of perceptrons around 20.

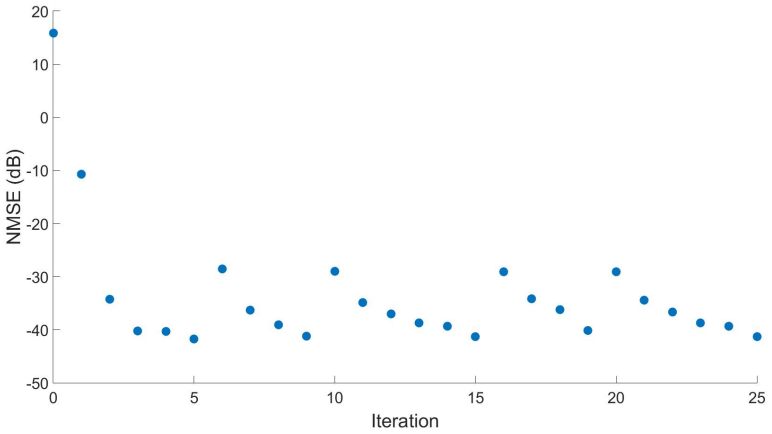
It was also possible to see that setting 40 as a maximum number of perceptrons was enough since the only Ms that got their best results using 40 were the worst performing ones. In all other cases the best result always used 30 or less perceptrons.

5.2.3 Performance of the predistortion

We will now look at the performance of the predistortion we were able to achieve.



(a) Evolution of the ACPR in the training phase



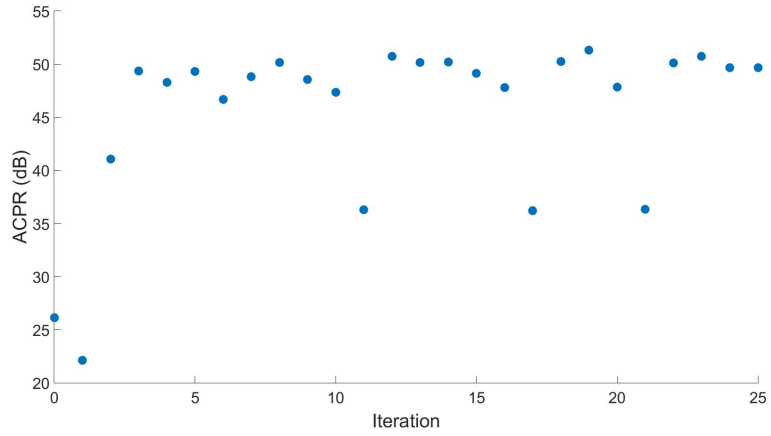
(b) Evolution of the NMSE in the training phase

Figure 5.3: Plots of the results of the training phase

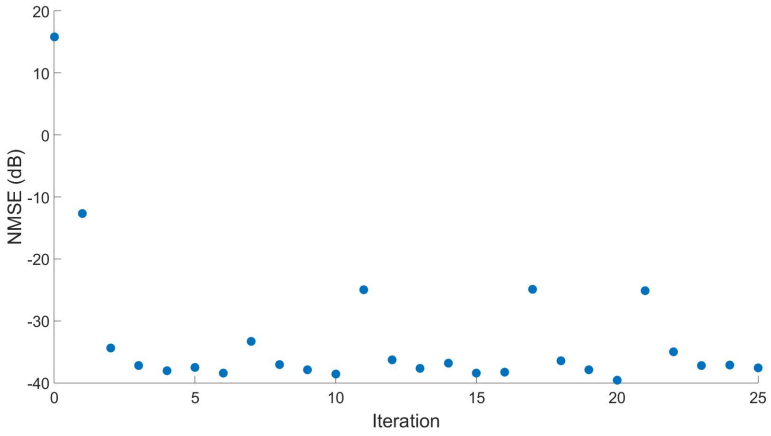
To do this we will look at what was achieved using the network that got the best results for the log exponential function. That is, we will use a network with polar input/output structure, the log exponential function, M set to 4, and 20 hidden layer perceptrons.

Figure 5.3 shows the evolution of the system training. In the figures, iteration 0 corresponds to the results when no predistortion block is used. In both figures it is possible to see that the values fluctuate when we are near the best achievable results. The best ACPR value was obtained in the 15^{th} iteration and is equal to $55,42$ dB. Its corresponding NMSE is $-41,33$ dB, a value very close to the minimum achieved of $-41,72$ dB in the 5^{th} iteration.

While it is interesting to see the evolution of the training, the plots that are of most interest are the ones created using the results from the validation checks. These plots are displayed in figure 5.4.



(a) Evolution of the ACPR in the validation checks



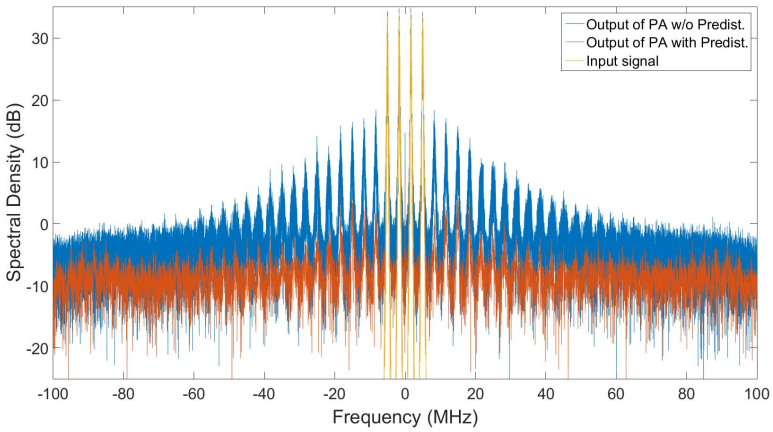
(b) Evolution of the NMSE in the validation checks

Figure 5.4: Plots of the results of the validation checks

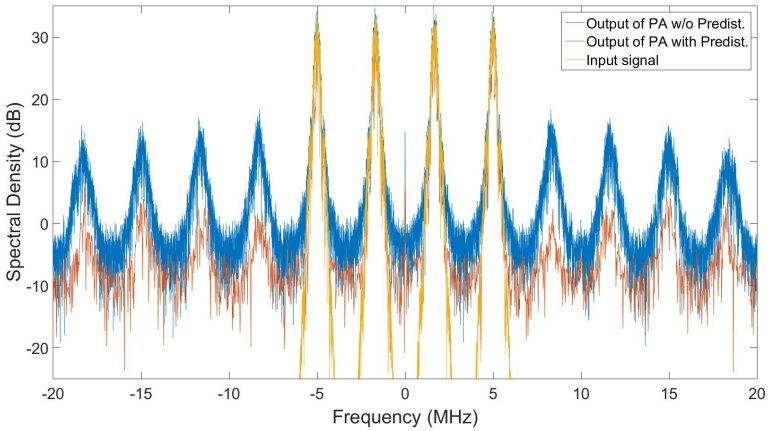
As before, it is possible to see a fluctuation of the NMSE and the ACPR when we are near the best achievable result. The best ACPR value was obtained in the 19^{th} iteration and is equal to $51,31$ dB. Its corresponding NMSE is $-37,92$ dB. The best NMSE value was obtained on the 20^{20}

iteration and was equal to $-39,59$ dB. This best ACPR value results in a improvement of $25,20$ dB over the initial ACPR of $26,11$ dB. The corresponding NMSE improves the original (without predistortion) value by $-53,79$ dB over the initial value of $15,87$ dB.

The improvement of the ACPR becomes very clear if we plot the spectral density of the output of the PA with and without a predistortion block being present. That is what we can see in figure 5.5. In both figures we have plotted the input signal (X in the figure 5.2) in yellow, the output of the PA with predistortion in red and the output of the PA without predistortion in blue. All signals were normalized before being plotted. The two figures show the global view and a zoom in of the important area. In both figures the effect of the predistortion block is clearly visible.



(a) Global view of the spectral density



(b) Zoom in of the important section

Figure 5.5: Plot of the spectral density of the input and output signal with and without predistortion

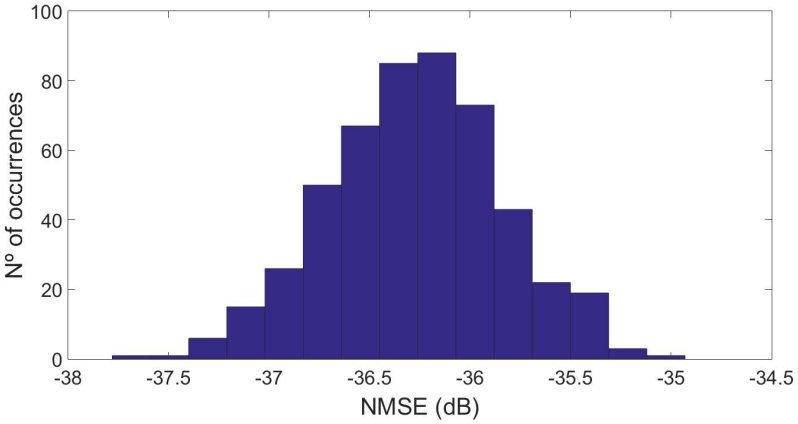
5.3 Sensitivity test

Given that the we are making this model with intentions of it being implemented analogically it is important to do a sensitivity test. Analog circuits don't have perfect precision so it is important to

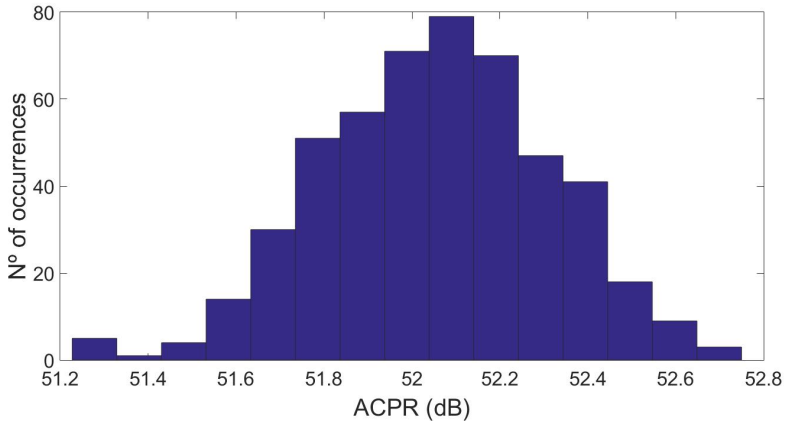
know what degree of precision is needed in order to still have a good end result.

To do this we will take the best network that used the log exponential activation function, the same that was used in section 5.2.3:polar input/output structure, the log exponential function, M set to 4, and 20 hidden layer perceptrons. We are using the network that, using this configuration, obtained the best ACPR in the validation.

Before doing any sensitivity test of the network itself, it is important to note that in our system we already have a section that is not constant. Everything from the output of the signal generator in the RF WebLab until the signal analyzer is subject to various noise sources that are outside of our control. To test the influence and variability of them, a batch of 10000 samples of predistorted signal was sent to the WebLab 500 times. The batch sent was the same for all situations, and in each situation the resulting ACPR was NMSE was computed, giving us 500 different values of each. The resulting histograms can be seen in figures 5.6a and 5.6b.



(a)



(b)

Figure 5.6: NMSE (a) and ACPR (b) histograms of measurements of the PA output signal when the same signal sent to the RF Weblab

The lowest NMSE measured was of $-37,78$ dB and the highest was of $-34,93$ dB. The resulting mean was of $-36,27$ dB and the standard deviation was of $0,43$ dB.

As for the ACPR, the highest value measured was of $52,75$ dB and the lowest was of $51,23$ dB. The mean of the measurements was of $52,05$ dB, with a standard deviation of $0,26$ dB.

It is curious to note that the value of the best ACPR and corresponding NMSE measured in the validation of the test run on section 5.2, for the same network structure was of $51,31$ dB and $-37,92$ dB. This would correspond to a really bad ACPR (since it is near the lowest value measured) and a really good NMSE (since it is lower than the lowest value measured) when these histograms are taken into account. However one must be careful when doing a direct comparison of these values since another factor that can influence the ACPR and NMSE are the operating conditions of the PA (such as aging, temperature and usage). While it is true that these conditions of the PA can be assumed to be the same in all 500 measurements since they were done in quick succession, the same cannot be said when comparing these 500 tests to the results obtained in 5.2 because the measurements were done several days apart. We could say that the aging and usage of the PA is practically the same in all measurements but there is no guarantee regarding the temperature since no information is provided by the RF WebLab regarding this.

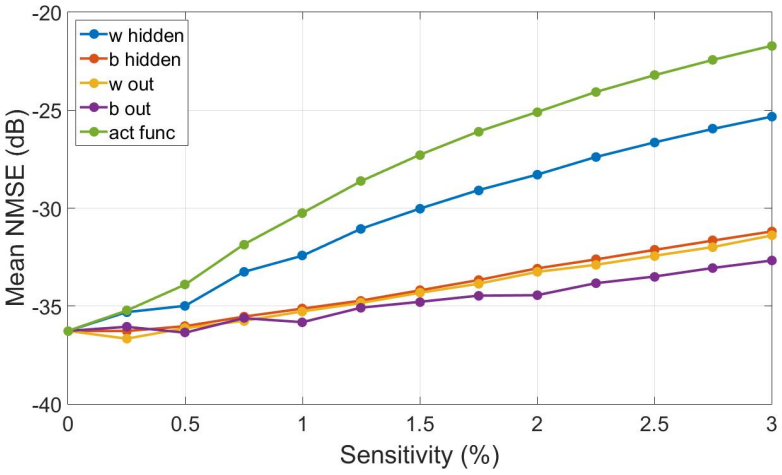
What can be concluded from the histogram is that the order of performance seen in table 5.3 might not be correct because the difference between the ACPR of some entries is small enough that a new measurement might give a different order. However no new measurements are going to be made in order to compare different structures since the specific order of the table is of no crucial importance. What is important is to know that a certain network structure is capable of performing predistortion with an ACPR/NMSE around a certain number. From what was seen in the histograms, that number won't vary greatly (more so in the ACPR case). Networks that achieve an ACPR within $0,5$ dB of each other could be said to have an equal performance.

After knowing what influence the noise effects in the RF Weblab have in the NMSE and ACPR we are ready to start testing the sensitivity of our network. We will test the sensitivity of five different zones of the network:

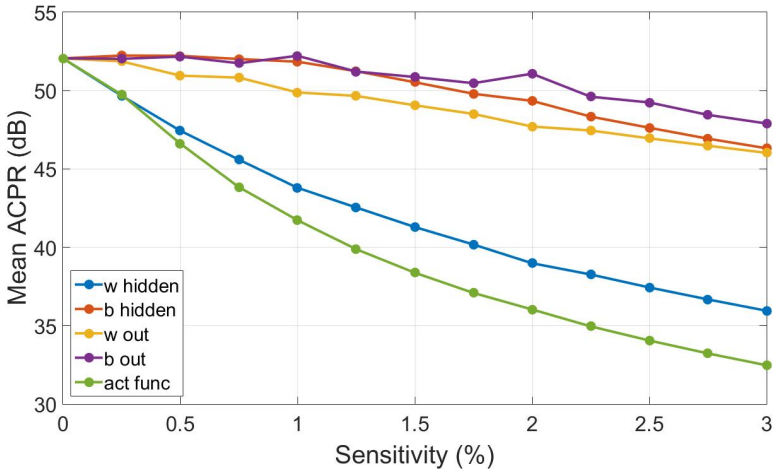
- the weights of the hidden layer;
- the biases of the hidden layer;
- the output of the hidden layer activation function;
- the weights of the output layer;
- the biases of the output layer.

The sensitivity test will be done for each zone separately. To do this test we will create a normal distribution for each element of the zone to be tested. After that we will select a random value from each normal distribution and the selected values will be the ones used in the test. We will run a batch of 10000 input samples (the same used in the previous test) through the network and then through the RF WebLab. Finally, the resulting NMSE and ACPR is measured. After this we will begin a new test where new values are selected from the normal distribution of each element. We will run a total of 500 tests. This procedure will be called test sequence.

The normal distribution used in the test sequence will be created according to the six sigma rule and using a percentage of the value of the element to define the standard deviation. For example, if we are testing the biases of the output layer we know that this zone has 2 elements: the bias of the first perceptron of the output layer b_1 and the bias of the second perceptron of the output layer b_2. Lets assume that $b_1 = 10$ and $b_2 = 20$. If we are testing this zone for a sensitivity of 1% then we will create two normal distributions, one for each element. The means of these distributions will be the value of its corresponding element. So distribution 1 will have $\mu_1 = 10$ and distribution 2 will have $\mu_2 = 20$. Given that we are testing for a sensitivity of 1% that corresponds to the interval $[9,9\ \ 10,1]$ for case 1 and $[19,8\ \ 20,2]$ for case 2. According to the six sigma rule this will generate the following standard deviations for each case: $\sigma_1 = \frac{10,1-10}{6}$ and $\sigma_2 = \frac{20,2-20}{6}$. The generated distribution will guarantee that around only 1 in 500 million values will be outside the referred intervals.



(a)



(b)

Figure 5.7: Plots of the mean NMSE (a) and ACPR (b) for sensitivity tests of different zones of the network

Using this we are able to do test sequences for the various zones for different percentages of sensitivity. For each zone 12 test sequences were done with a sensitivity varying from $0,25\%$ to 3% with steps of $0,25\%$. At the end of each test sequence the mean and standard deviation of the ACPR and NMSE were computed. The resulting plots for the mean can be seen in figures 5.7a and 5.7b.

In the plots, the 0% points correspond to the mean NMSE and ACPR measured when we did the histograms of figure 5.6. Analyzing the plots it becomes clear that the two most sensitive zones are the weights of the hidden layer and the output of the activation function of the PA. The other three zones have a very similar and low sensitivity comparatively. In fact we can see that with a sensitivity of 3% in the bias of the hidden layer, the ACPR will be reduced from $52,05$ dB to $46,32$ dB. This corresponds to a $5,73$ dB decrease. A sensitivity of 3% in the weights of the hidden layer and the output of the activation function of the hidden layer will result in an ACPR of $35,95$ dB and $32,47$ dB respectively. This will equal a decrease of $16,10$ dB and $19,58$ dB, which is almost 3 times and $3,5$ times worse than the decrease resulting from the bias of the hidden layer.

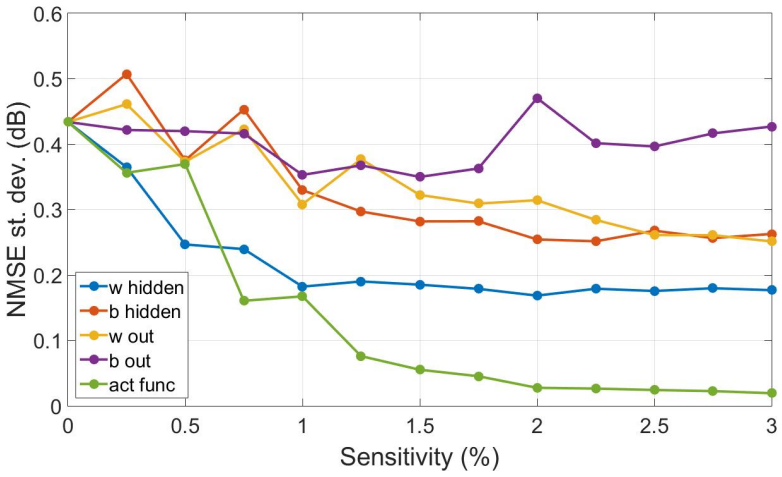
This higher sensitivity of the weights of the hidden layer and the output of the activation function of the hidden layer is somewhat expected given that the neural network used is a multilayer perceptron. So alterations induced in the first layer will have an effect in the following layers as well. The bias of the hidden layer, although also being a part of the first layer, doesn't have this high sensitivity. This can be explained by the fact that the bias plays most of the times a minor role given that it is added with all the weighted inputs and thus has only a minor influence in the total sum. So an error in the bias will result in only a small shift to the corresponding input of the activation function (z) when compared to the values of the weighted inputs. If we had a network where the bias was a large contributor to z then the bias of the hidden layer would also be highly sensitive.

Along with the mean, after each test sequence the standard deviation was also computed. The resulting plots are displayed in figures 5.8a and 5.8b.

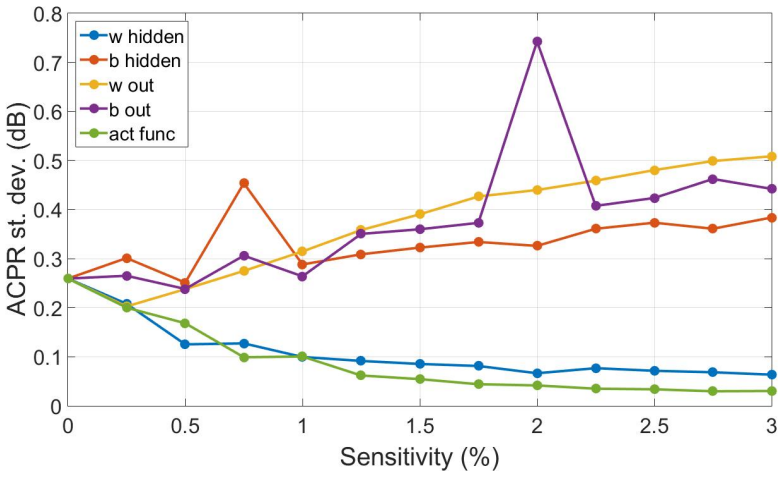
Analyzing the plots it is possible to see that in both cases, the zones that are more sensitive (weights of the hidden layer and output of the activation function of the hidden layer) also see their standard deviations decrease as we increase the percentage of the test. Regarding the other zones, we can see that the standard deviation of the ACPR increases slightly as we increase the percentage of the test. The standard deviation of the NMSE of the output bias doesn't vary much. The standard deviation of the NMSE for the remaining two zones decreases slightly as we increase the percentage of the test.

For the sake of completeness, a final set of tests were run where we tested the sensitivity of all the zones simultaneously. Since we were testing all zones simultaneously, the range of percentages tested was lowered. The sensitivity tested ranged from $0,2\%$ to $1,6\%$ with steps of $0,2\%$. The results are displayed in figures 5.9a and 5.9b.

As expected, the performance of the network decreases rapidly. The performance of the network when sensitivity was set to $1,6\%$ is around the same as the performance of the network when we test a sensitivity of 3% in only the output of the activation function of the hidden layer.



(a)

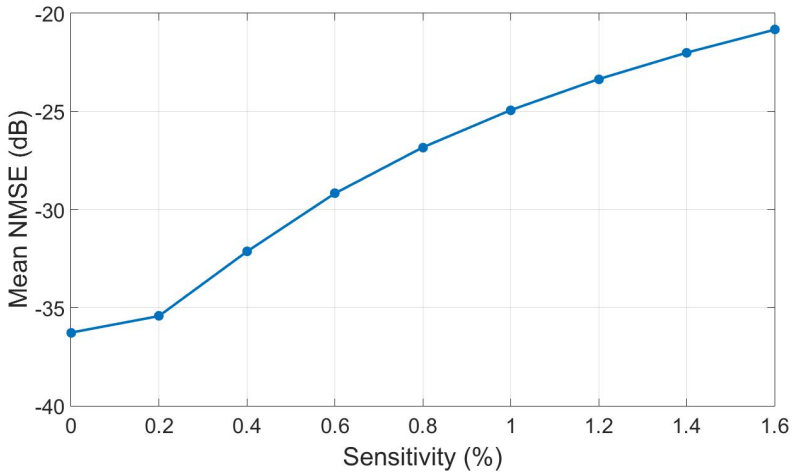


(b)

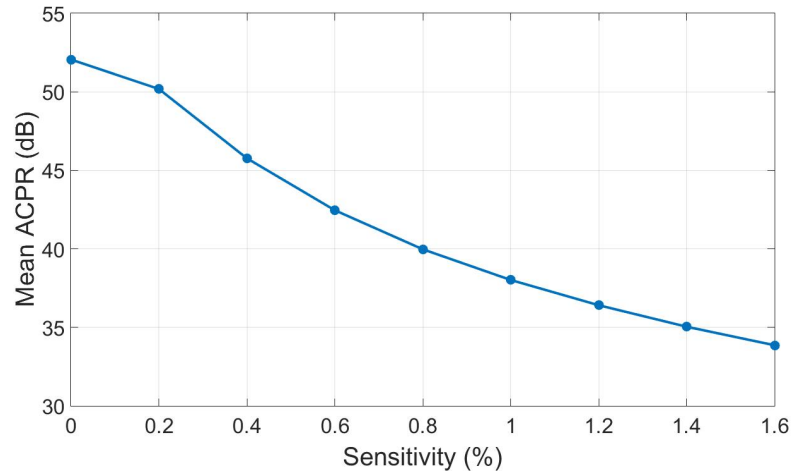
Figure 5.8: Plots of the standard deviation of the NMSE (a) and ACPR (b) for sensitivity tests of different zones of the network

Finally, in order to get the scope of what these percentages mean we will present the range of values that each zone has:

- weights of the hidden layer - $[-7,2664 \;\; 8,6213]$;
- bias of the hidden layer - $[-3,4185 \;\; 2,6797]$;
- weights of the output layer - $[-2,4416 \;\; 2,5260]$;
- bias of the output layer - $[-0,3477 \;\; 3,8580]$;
- output of the activation function of the hidden layer - $[6,6906 \times 10^{-12} \;\; 12,0569]$ (note that due to the function used, the minimum possible value would be 0).



(a)



(b)

Figure 5.9: Plots of the mean NMSE (a) and ACPR (b) for sensitivity tests of all zones simultaneously

Given that that this work has the purpose of serving as a support for an analog implementation the ranges of values in the other zones of the network will also be presented here:

- input corresponding to a in figure 4.3 - $[0,0045 \;\; 0,9949]$ (the other inputs won't be mentioned since they are the result of a cosine or a sine);
- input of the activation function of the hidden layer - $[-12,5186 \;\; 12,0569]$;
- output corresponding to b in figure 4.3 - $[0,0076 \;\; 0,9761]$;
- output corresponding to φ in figure 4.3 - $[-0,4314 \;\; 0,4891]$;

Overall this sensitivity analysis serves as a reference for the precision that the corresponding circuits for each zone of the network will need to have in order to obtain a good analog predistortion block. The discrepancy in the sensitivity of the different zones and the fact that the more

sensitive zones are in the first layer of the network open the possibility of implementing these zones in a digital format. For example, we could do the weight zone digitally, ensuring a better precision, and only after begin the analog part of the circuit. However, we would need to analyze what the energy consumption of this digital block would be, since the purpose of implementing a predistortion block analogically is to be more energy efficient. Nevertheless, it is certainly an option to consider.

Chapter 6

Conclusions

The goal of this work was to create a model that could serve as a first step towards the creation of an analog predistortion block based on neural networks. To achieve this we created neural networks using the log exponential function as its activation function, since this function would have an easier analog implementation than the commonly used functions. Furthermore, we had the objective of finding the necessary precision the network needs to have in order to maintain a good predistortion performance.

In the work developed we concluded that the LM algorithm has a better performance than the GD algorithm. Furthermore it is more convenient to use since we don't need to find the appropriate α. In the tests done it was also confirmed that the polar structure proposed in [23] has a significantly better performance, when doing predistortion, than the more commonly used cartesian structure.

The log exponential function was tested and it was concluded that this function has a performance similar to the hyperbolic tangent function. This is a good sign since the hyperbolic tangent function is one of the activation functions commonly used when using neural networks for predistortion. As mentioned before, the log exponential function could have an easy (when compared to the hyperbolic tangent function) implementation in analog by using diodes or transistors operating in the sub-threshold region.

It also became clear that an analog implementation of the neural network would need to be highly precise. However when analyzing the results of the sensitivity test, the possibility of doing part of the network in digital format appeared. This option would need to be analyzed to see if the power consumption of the digital block would be low enough for its use to be viable.

6.1 Future work

For the future work on this theme several options are available.

First of all, and the most obvious one, is continuing this work by designing an analog circuit to implement the neural network proposed. This includes the design of a circuit that can implement the neural network as well as a study to determine the power consumption of the circuit. As

mentioned before it is also possible to look into the possibility of doing the part of the weights of the hidden layer in digital form. This would require the analyzes of the power consumption of this section and its comparison to the corresponding analog section with the precision that both can achieve in mind. Also in the analog implementation one thing to keep in mind is the possibility of slightly modifying the training method in a way that makes it possible to limit the values accepted in each zone of the network. This could be done by using the modifications to the LM algorithm proposed in [7]. With this modification it is possible, for example, to limit the values of the weights to the $[-5\ \ 5]$ range. This would obviously have drawbacks in terms of the performance of the predistortion achieved, but it could be an advantage if it would enable the design of a more precise circuit.

Another option is to repeat the tests for different types of neural networks other than the multilayer perceptron which was analyzed in this work. Out of all the possibilities, looking at a recurrent neural network seems like an interesting choice as these neural networks already have memory effects built into them. Other options include designing a neural network so that some inputs have naturally a greater impact in the output. This would mean designing each individual connection between perceptrons. The point of this is to make it so that the inputs of the current sample have more impact in the output (since we know that the current output is very close to the current input).

In terms of the tests ran, new tests could be run in a more controlled environment, where we have access or control of the operating conditions of the PA. This would allow us to guarantee the comparability of tests run at different times.

Appendix A

Results of tests for the inverse of the PA

Table A.1: Results from the tests of a neural network trained to be the inverse of a PA

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
1	GD	Cartesian	log_exp	1	5	-26,94	-24,31
2	GD	Cartesian	log_exp	1	10	-28,13	-25,39
3	GD	Cartesian	log_exp	1	15	-34,35	-30,31
4	GD	Cartesian	log_exp	1	20	-32,91	-28,66
5	GD	Cartesian	log_exp	1	25	-35,00	-32,17
6	GD	Cartesian	log_exp	1	30	-33,92	-30,49
7	GD	Cartesian	log_exp	2	5	-28,64	-25,31
8	GD	Cartesian	log_exp	2	10	-29,50	-26,19
9	GD	Cartesian	log_exp	2	15	-33,09	-29,79
10	GD	Cartesian	log_exp	2	20	-31,24	-27,33
11	GD	Cartesian	log_exp	2	25	-32,25	-28,93
12	GD	Cartesian	log_exp	2	30	-32,57	-29,02
13	GD	Cartesian	log_exp	3	5	-27,72	-25,25
14	GD	Cartesian	log_exp	3	10	-32,02	-29,26
15	GD	Cartesian	log_exp	3	15	-31,66	-28,33
16	GD	Cartesian	log_exp	3	20	-31,70	-28,11
17	GD	Cartesian	log_exp	3	25	-31,42	-28,10
18	GD	Cartesian	log_exp	3	30	-32,04	-27,59
19	GD	Cartesian	log_exp	4	5	-26,95	-23,44
20	GD	Cartesian	log_exp	4	10	-29,78	-26,62
21	GD	Cartesian	log_exp	4	15	-29,85	-27,01
22	GD	Cartesian	log_exp	4	20	-30,72	-28,02
23	GD	Cartesian	log_exp	4	25	-31,81	-29,02
24	GD	Cartesian	log_exp	4	30	-31,21	-26,60
25	GD	Cartesian	log_exp	5	5	-27,65	-24,64

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
26	GD	Cartesian	log_exp	5	10	-28,75	-23,92
27	GD	Cartesian	log_exp	5	15	-30,94	-28,35
28	GD	Cartesian	log_exp	5	20	-30,48	-26,68
29	GD	Cartesian	log_exp	5	25	-30,32	-26,42
30	GD	Cartesian	log_exp	5	30	-30,96	-26,34
31	GD	Cartesian	log_exp	6	5	-26,65	-23,57
32	GD	Cartesian	log_exp	6	10	-29,78	-26,55
33	GD	Cartesian	log_exp	6	15	-29,88	-26,91
34	GD	Cartesian	log_exp	6	20	-30,17	-26,71
35	GD	Cartesian	log_exp	6	25	-30,17	-26,95
36	GD	Cartesian	log_exp	6	30	-29,88	-25,24
37	GD	Cartesian	log_exp	7	5	-26,99	-24,19
38	GD	Cartesian	log_exp	7	10	-28,44	-24,07
39	GD	Cartesian	log_exp	7	15	-28,49	-24,79
40	GD	Cartesian	log_exp	7	20	-30,02	-26,78
41	GD	Cartesian	log_exp	7	25	-29,82	-26,55
42	GD	Cartesian	log_exp	7	30	-28,67	-24,56
43	GD	Cartesian	log_exp	8	5	-27,61	-23,71
44	GD	Cartesian	log_exp	8	10	-29,37	-25,68
45	GD	Cartesian	log_exp	8	15	-29,23	-26,02
46	GD	Cartesian	log_exp	8	20	-30,38	-27,17
47	GD	Cartesian	log_exp	8	25	-28,89	-23,75
48	GD	Cartesian	log_exp	8	30	-30,19	-26,26
49	GD	Cartesian	log_exp	9	5	-26,71	-23,09
50	GD	Cartesian	log_exp	9	10	-28,33	-24,22
51	GD	Cartesian	log_exp	9	15	-28,30	-24,56
52	GD	Cartesian	log_exp	9	20	-28,87	-24,43
53	GD	Cartesian	log_exp	9	25	-29,17	-25,14
54	GD	Cartesian	log_exp	9	30	-29,74	-25,87
55	GD	Cartesian	tanh	1	5	-27,78	-24,20
56	GD	Cartesian	tanh	1	10	-29,44	-25,08
57	GD	Cartesian	tanh	1	15	-33,26	-29,51
58	GD	Cartesian	tanh	1	20	-35,21	-30,88
59	GD	Cartesian	tanh	1	25	-34,82	-30,37
60	GD	Cartesian	tanh	1	30	-38,11	-34,26
61	GD	Cartesian	tanh	2	5	-29,87	-26,19
62	GD	Cartesian	tanh	2	10	-29,62	-25,61
63	GD	Cartesian	tanh	2	15	-33,98	-29,75
64	GD	Cartesian	tanh	2	20	-33,52	-28,49

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
65	GD	Cartesian	tanh	2	25	-34,43	-31,39
66	GD	Cartesian	tanh	2	30	-35,00	-31,05
67	GD	Cartesian	tanh	3	5	-27,98	-22,22
68	GD	Cartesian	tanh	3	10	-30,54	-25,88
69	GD	Cartesian	tanh	3	15	-31,17	-26,82
70	GD	Cartesian	tanh	3	20	-34,00	-28,88
71	GD	Cartesian	tanh	3	25	-32,61	-27,96
72	GD	Cartesian	tanh	3	30	-33,80	-27,10
73	GD	Cartesian	tanh	4	5	-28,14	-22,22
74	GD	Cartesian	tanh	4	10	-30,83	-26,44
75	GD	Cartesian	tanh	4	15	-30,02	-26,48
76	GD	Cartesian	tanh	4	20	-31,11	-27,09
77	GD	Cartesian	tanh	4	25	-32,06	-26,90
78	GD	Cartesian	tanh	4	30	-31,26	-26,54
79	GD	Cartesian	tanh	5	5	-27,64	-23,20
80	GD	Cartesian	tanh	5	10	-30,17	-25,50
81	GD	Cartesian	tanh	5	15	-31,49	-28,18
82	GD	Cartesian	tanh	5	20	-31,07	-26,62
83	GD	Cartesian	tanh	5	25	-30,55	-24,90
84	GD	Cartesian	tanh	5	30	-30,66	-27,02
85	GD	Cartesian	tanh	6	5	-27,17	-22,15
86	GD	Cartesian	tanh	6	10	-29,52	-25,76
87	GD	Cartesian	tanh	6	15	-30,13	-26,46
88	GD	Cartesian	tanh	6	20	-31,42	-25,67
89	GD	Cartesian	tanh	6	25	-31,42	-26,13
90	GD	Cartesian	tanh	6	30	-29,78	-26,27
91	GD	Cartesian	tanh	7	5	-27,30	-22,17
92	GD	Cartesian	tanh	7	10	-28,08	-22,89
93	GD	Cartesian	tanh	7	15	-30,54	-27,17
94	GD	Cartesian	tanh	7	20	-30,99	-26,73
95	GD	Cartesian	tanh	7	25	-29,19	-26,07
96	GD	Cartesian	tanh	7	30	-29,29	-24,06
97	GD	Cartesian	tanh	8	5	-27,83	-22,47
98	GD	Cartesian	tanh	8	10	-29,18	-23,68
99	GD	Cartesian	tanh	8	15	-28,95	-25,59
100	GD	Cartesian	tanh	8	20	-29,72	-25,04
101	GD	Cartesian	tanh	8	25	-29,44	-23,88
102	GD	Cartesian	tanh	8	30	-30,48	-26,92
103	GD	Cartesian	tanh	9	5	-26,76	-22,25

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
104	GD	Cartesian	tanh	9	10	-28,46	-24,26
105	GD	Cartesian	tanh	9	15	-28,39	-24,55
106	GD	Cartesian	tanh	9	20	-29,10	-24,76
107	GD	Cartesian	tanh	9	25	-29,44	-24,83
108	GD	Cartesian	tanh	9	30	-28,27	-23,04
109	GD	Polar	log_exp	1	5	-37,17	-34,25
110	GD	Polar	log_exp	1	10	-37,65	-34,85
111	GD	Polar	log_exp	1	15	-36,83	-34,03
112	GD	Polar	log_exp	1	20	-38,74	-35,99
113	GD	Polar	log_exp	1	25	-36,20	-33,51
114	GD	Polar	log_exp	1	30	-38,96	-36,24
115	GD	Polar	log_exp	2	5	-33,33	-31,04
116	GD	Polar	log_exp	2	10	-34,72	-31,85
117	GD	Polar	log_exp	2	15	-30,98	-28,68
118	GD	Polar	log_exp	2	20	-32,04	-29,70
119	GD	Polar	log_exp	2	25	-34,64	-31,35
120	GD	Polar	log_exp	2	30	-35,08	-31,85
121	GD	Polar	log_exp	3	5	-33,52	-31,39
122	GD	Polar	log_exp	3	10	-33,37	-31,28
123	GD	Polar	log_exp	3	15	-34,28	-32,19
124	GD	Polar	log_exp	3	20	-35,12	-32,44
125	GD	Polar	log_exp	3	25	-31,50	-28,13
126	GD	Polar	log_exp	3	30	-29,76	-27,53
127	GD	Polar	log_exp	4	5	-32,65	-29,96
128	GD	Polar	log_exp	4	10	-25,62	-23,41
129	GD	Polar	log_exp	4	15	-26,28	-23,51
130	GD	Polar	log_exp	4	20	-28,11	-25,17
131	GD	Polar	log_exp	4	25	-27,38	-25,09
132	GD	Polar	log_exp	4	30	-25,40	-22,29
133	GD	Polar	log_exp	5	5	-33,27	-30,37
134	GD	Polar	log_exp	5	10	-29,32	-26,26
135	GD	Polar	log_exp	5	15	-23,10	-20,23
136	GD	Polar	log_exp	5	20	-26,74	-24,42
137	GD	Polar	log_exp	5	25	-27,93	-25,33
138	GD	Polar	log_exp	5	30	-23,35	-20,72
139	GD	Polar	log_exp	6	5	-32,53	-30,06
140	GD	Polar	log_exp	6	10	-24,49	-21,84
141	GD	Polar	log_exp	6	15	-22,59	-19,91
142	GD	Polar	log_exp	6	20	-27,40	-24,38

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
143	GD	Polar	log_exp	6	25	-25,10	-22,07
144	GD	Polar	log_exp	6	30	-22,09	-19,91
145	GD	Polar	log_exp	7	5	-33,83	-31,25
146	GD	Polar	log_exp	7	10	-34,38	-31,21
147	GD	Polar	log_exp	7	15	-24,06	-21,23
148	GD	Polar	log_exp	7	20	-28,79	-25,94
149	GD	Polar	log_exp	7	25	-22,01	-19,12
150	GD	Polar	log_exp	7	30	-21,71	-19,35
151	GD	Polar	log_exp	8	5	-25,11	-22,56
152	GD	Polar	log_exp	8	10	-32,77	-29,78
153	GD	Polar	log_exp	8	15	-29,95	-26,86
154	GD	Polar	log_exp	8	20	-25,94	-22,59
155	GD	Polar	log_exp	8	25	-20,11	-17,38
156	GD	Polar	log_exp	8	30	-21,74	-19,46
157	GD	Polar	log_exp	9	5	-28,76	-26,25
158	GD	Polar	log_exp	9	10	-28,28	-25,74
159	GD	Polar	log_exp	9	15	-30,36	-27,98
160	GD	Polar	log_exp	9	20	-27,01	-23,92
161	GD	Polar	log_exp	9	25	-23,68	-20,71
162	GD	Polar	log_exp	9	30	-25,29	-21,89
163	GD	Polar	tanh	1	5	-35,56	-32,81
164	GD	Polar	tanh	1	10	-34,47	-31,14
165	GD	Polar	tanh	1	15	-37,10	-34,49
166	GD	Polar	tanh	1	20	-40,46	-37,45
167	GD	Polar	tanh	1	25	-37,07	-34,56
168	GD	Polar	tanh	1	30	-39,20	-36,65
169	GD	Polar	tanh	2	5	-35,54	-33,06
170	GD	Polar	tanh	2	10	-35,55	-32,97
171	GD	Polar	tanh	2	15	-34,94	-32,48
172	GD	Polar	tanh	2	20	-32,66	-30,62
173	GD	Polar	tanh	2	25	-33,75	-30,94
174	GD	Polar	tanh	2	30	-32,71	-30,86
175	GD	Polar	tanh	3	5	-33,07	-30,61
176	GD	Polar	tanh	3	10	-34,83	-32,33
177	GD	Polar	tanh	3	15	-34,13	-31,53
178	GD	Polar	tanh	3	20	-34,26	-31,43
179	GD	Polar	tanh	3	25	-30,19	-27,02
180	GD	Polar	tanh	3	30	-32,42	-29,76
181	GD	Polar	tanh	4	5	-30,99	-28,40

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
182	GD	Polar	tanh	4	10	-31,22	-28,54
183	GD	Polar	tanh	4	15	-28,61	-25,73
184	GD	Polar	tanh	4	20	-29,23	-26,81
185	GD	Polar	tanh	4	25	-28,61	-25,87
186	GD	Polar	tanh	4	30	-28,56	-25,79
187	GD	Polar	tanh	5	5	-32,26	-29,73
188	GD	Polar	tanh	5	10	-30,76	-27,59
189	GD	Polar	tanh	5	15	-27,61	-25,02
190	GD	Polar	tanh	5	20	-32,74	-29,76
191	GD	Polar	tanh	5	25	-26,20	-23,60
192	GD	Polar	tanh	5	30	-27,43	-23,91
193	GD	Polar	tanh	6	5	-33,01	-30,55
194	GD	Polar	tanh	6	10	-31,76	-29,56
195	GD	Polar	tanh	6	15	-25,40	-22,25
196	GD	Polar	tanh	6	20	-29,37	-26,78
197	GD	Polar	tanh	6	25	-23,46	-20,32
198	GD	Polar	tanh	6	30	-24,12	-21,35
199	GD	Polar	tanh	7	5	-29,04	-26,51
200	GD	Polar	tanh	7	10	-33,28	-30,81
201	GD	Polar	tanh	7	15	-29,71	-27,00
202	GD	Polar	tanh	7	20	-27,23	-24,45
203	GD	Polar	tanh	7	25	-23,75	-21,09
204	GD	Polar	tanh	7	30	-22,95	-19,67
205	GD	Polar	tanh	8	5	-30,46	-27,93
206	GD	Polar	tanh	8	10	-32,22	-29,56
207	GD	Polar	tanh	8	15	-29,28	-26,91
208	GD	Polar	tanh	8	20	-26,35	-23,47
209	GD	Polar	tanh	8	25	-23,89	-20,95
210	GD	Polar	tanh	8	30	-28,62	-26,01
211	GD	Polar	tanh	9	5	-29,66	-27,21
212	GD	Polar	tanh	9	10	-27,68	-25,10
213	GD	Polar	tanh	9	15	-28,67	-25,89
214	GD	Polar	tanh	9	20	-27,28	-24,06
215	GD	Polar	tanh	9	25	-27,74	-25,04
216	GD	Polar	tanh	9	30	-24,85	-21,87
217	LM	Cartesian	log_exp	1	5	-30,54	-27,32
218	LM	Cartesian	log_exp	1	10	-41,62	-40,63
219	LM	Cartesian	log_exp	1	15	-42,34	-41,74
220	LM	Cartesian	log_exp	1	20	-43,36	-42,56

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
221	LM	Cartesian	log_exp	1	25	-43,35	-42,56
222	LM	Cartesian	log_exp	1	30	-43,41	-42,62
223	LM	Cartesian	log_exp	2	5	-31,34	-28,39
224	LM	Cartesian	log_exp	2	10	-41,30	-40,18
225	LM	Cartesian	log_exp	2	15	-43,37	-42,19
226	LM	Cartesian	log_exp	2	20	-43,57	-42,43
227	LM	Cartesian	log_exp	2	25	-43,70	-42,42
228	LM	Cartesian	log_exp	2	30	-43,97	-42,76
229	LM	Cartesian	log_exp	3	5	-31,71	-29,47
230	LM	Cartesian	log_exp	3	10	-40,67	-39,35
231	LM	Cartesian	log_exp	3	15	-43,33	-42,53
232	LM	Cartesian	log_exp	3	20	-42,77	-41,68
233	LM	Cartesian	log_exp	3	25	-44,00	-42,66
234	LM	Cartesian	log_exp	3	30	-44,09	-42,87
235	LM	Cartesian	log_exp	4	5	-31,37	-28,18
236	LM	Cartesian	log_exp	4	10	-41,94	-41,29
237	LM	Cartesian	log_exp	4	15	-43,15	-41,94
238	LM	Cartesian	log_exp	4	20	-43,61	-42,49
239	LM	Cartesian	log_exp	4	25	-43,75	-42,52
240	LM	Cartesian	log_exp	4	30	-44,21	-42,58
241	LM	Cartesian	log_exp	5	5	-31,46	-28,85
242	LM	Cartesian	log_exp	5	10	-37,95	-35,93
243	LM	Cartesian	log_exp	5	15	-42,01	-40,69
244	LM	Cartesian	log_exp	5	20	-43,59	-42,22
245	LM	Cartesian	log_exp	5	25	-43,30	-42,08
246	LM	Cartesian	log_exp	5	30	-43,14	-42,18
247	LM	Cartesian	log_exp	6	5	-31,32	-28,32
248	LM	Cartesian	log_exp	6	10	-38,57	-36,71
249	LM	Cartesian	log_exp	6	15	-42,13	-41,02
250	LM	Cartesian	log_exp	6	20	-43,70	-42,62
251	LM	Cartesian	log_exp	6	25	-43,79	-42,41
252	LM	Cartesian	log_exp	6	30	-44,01	-42,43
253	LM	Cartesian	log_exp	7	5	-29,37	-26,18
254	LM	Cartesian	log_exp	7	10	-40,37	-38,64
255	LM	Cartesian	log_exp	7	15	-43,69	-42,49
256	LM	Cartesian	log_exp	7	20	-43,77	-42,54
257	LM	Cartesian	log_exp	7	25	-43,90	-42,57
258	LM	Cartesian	log_exp	7	30	-44,17	-42,68
259	LM	Cartesian	log_exp	8	5	-32,05	-29,25

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
260	LM	Cartesian	log_exp	8	10	-40,88	-39,26
261	LM	Cartesian	log_exp	8	15	-42,16	-40,93
262	LM	Cartesian	log_exp	8	20	-43,69	-42,19
263	LM	Cartesian	log_exp	8	25	-44,32	-42,66
264	LM	Cartesian	log_exp	8	30	-43,98	-42,38
265	LM	Cartesian	log_exp	9	5	-30,18	-27,11
266	LM	Cartesian	log_exp	9	10	-37,67	-35,50
267	LM	Cartesian	log_exp	9	15	-43,16	-41,87
268	LM	Cartesian	log_exp	9	20	-40,99	-38,98
269	LM	Cartesian	log_exp	9	25	-44,45	-42,68
270	LM	Cartesian	log_exp	9	30	-42,36	-40,55
271	LM	Cartesian	tanh	1	5	-36,56	-34,11
272	LM	Cartesian	tanh	1	10	-38,11	-35,53
273	LM	Cartesian	tanh	1	15	-43,26	-42,28
274	LM	Cartesian	tanh	1	20	-43,62	-42,29
275	LM	Cartesian	tanh	1	25	-43,72	-42,38
276	LM	Cartesian	tanh	1	30	-43,68	-42,22
277	LM	Cartesian	tanh	2	5	-37,13	-34,88
278	LM	Cartesian	tanh	2	10	-42,17	-40,78
279	LM	Cartesian	tanh	2	15	-43,72	-42,34
280	LM	Cartesian	tanh	2	20	-43,91	-42,39
281	LM	Cartesian	tanh	2	25	-44,15	-42,45
282	LM	Cartesian	tanh	2	30	-44,12	-42,30
283	LM	Cartesian	tanh	3	5	-29,19	-26,05
284	LM	Cartesian	tanh	3	10	-38,75	-36,02
285	LM	Cartesian	tanh	3	15	-43,32	-41,67
286	LM	Cartesian	tanh	3	20	-44,11	-42,43
287	LM	Cartesian	tanh	3	25	-44,24	-42,32
288	LM	Cartesian	tanh	3	30	-44,58	-42,55
289	LM	Cartesian	tanh	4	5	-38,14	-36,03
290	LM	Cartesian	tanh	4	10	-33,72	-31,25
291	LM	Cartesian	tanh	4	15	-43,44	-41,98
292	LM	Cartesian	tanh	4	20	-44,14	-42,46
293	LM	Cartesian	tanh	4	25	-44,75	-42,69
294	LM	Cartesian	tanh	4	30	-44,34	-42,52
295	LM	Cartesian	tanh	5	5	-29,28	-26,19
296	LM	Cartesian	tanh	5	10	-38,43	-36,03
297	LM	Cartesian	tanh	5	15	-43,46	-41,74
298	LM	Cartesian	tanh	5	20	-43,85	-41,91

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
299	LM	Cartesian	tanh	5	25	-44,79	-42,60
300	LM	Cartesian	tanh	5	30	-44,68	-42,56
301	LM	Cartesian	tanh	6	5	-36,00	-33,32
302	LM	Cartesian	tanh	6	10	-31,99	-29,22
303	LM	Cartesian	tanh	6	15	-43,81	-42,07
304	LM	Cartesian	tanh	6	20	-43,71	-41,78
305	LM	Cartesian	tanh	6	25	-44,54	-42,44
306	LM	Cartesian	tanh	6	30	-44,70	-42,29
307	LM	Cartesian	tanh	7	5	-32,60	-30,23
308	LM	Cartesian	tanh	7	10	-37,96	-35,53
309	LM	Cartesian	tanh	7	15	-39,69	-37,51
310	LM	Cartesian	tanh	7	20	-42,88	-40,64
311	LM	Cartesian	tanh	7	25	-43,93	-41,68
312	LM	Cartesian	tanh	7	30	-44,86	-42,83
313	LM	Cartesian	tanh	8	5	-38,35	-36,08
314	LM	Cartesian	tanh	8	10	-37,76	-35,30
315	LM	Cartesian	tanh	8	15	-37,84	-35,39
316	LM	Cartesian	tanh	8	20	-38,24	-35,31
317	LM	Cartesian	tanh	8	25	-43,95	-41,64
318	LM	Cartesian	tanh	8	30	-43,24	-40,85
319	LM	Cartesian	tanh	9	5	-29,95	-26,92
320	LM	Cartesian	tanh	9	10	-35,62	-33,25
321	LM	Cartesian	tanh	9	15	-39,57	-37,23
322	LM	Cartesian	tanh	9	20	-41,97	-39,87
323	LM	Cartesian	tanh	9	25	-43,90	-41,56
324	LM	Cartesian	tanh	9	30	-44,03	-41,61
325	LM	Polar	log_exp	1	5	-40,08	-38,09
326	LM	Polar	log_exp	1	10	-43,87	-43,38
327	LM	Polar	log_exp	1	15	-44,18	-43,76
328	LM	Polar	log_exp	1	20	-44,26	-43,81
329	LM	Polar	log_exp	1	25	-42,40	-40,55
330	LM	Polar	log_exp	1	30	-42,71	-40,71
331	LM	Polar	log_exp	2	5	-43,70	-43,31
332	LM	Polar	log_exp	2	10	-44,01	-43,45
333	LM	Polar	log_exp	2	15	-43,06	-41,51
334	LM	Polar	log_exp	2	20	-27,18	-26,29
335	LM	Polar	log_exp	2	25	-42,23	-39,13
336	LM	Polar	log_exp	2	30	-43,55	-39,73
337	LM	Polar	log_exp	3	5	-43,79	-43,42

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
338	LM	Polar	log_exp	3	10	-37,36	-34,56
339	LM	Polar	log_exp	3	15	-38,57	-35,24
340	LM	Polar	log_exp	3	20	-43,39	-41,69
341	LM	Polar	log_exp	3	25	-35,38	-31,39
342	LM	Polar	log_exp	3	30	-37,27	-33,76
343	LM	Polar	log_exp	4	5	-37,09	-33,63
344	LM	Polar	log_exp	4	10	-34,11	-30,80
345	LM	Polar	log_exp	4	15	-30,39	-29,67
346	LM	Polar	log_exp	4	20	-37,65	-31,23
347	LM	Polar	log_exp	4	25	-31,85	-26,17
348	LM	Polar	log_exp	4	30	-33,60	-28,13
349	LM	Polar	log_exp	5	5	-30,19	-26,72
350	LM	Polar	log_exp	5	10	-43,48	-30,53
351	LM	Polar	log_exp	5	15	-35,47	-29,74
352	LM	Polar	log_exp	5	20	-32,05	-27,21
353	LM	Polar	log_exp	5	25	-15,17	-14,16
354	LM	Polar	log_exp	5	30	-30,30	-26,12
355	LM	Polar	log_exp	6	5	-44,39	-43,41
356	LM	Polar	log_exp	6	10	-29,11	-26,04
357	LM	Polar	log_exp	6	15	-29,35	-26,13
358	LM	Polar	log_exp	6	20	-31,58	-28,45
359	LM	Polar	log_exp	6	25	-12,74	-10,54
360	LM	Polar	log_exp	6	30	-24,61	-21,28
361	LM	Polar	log_exp	7	5	-44,37	-43,36
362	LM	Polar	log_exp	7	10	-32,10	-28,65
363	LM	Polar	log_exp	7	15	-33,64	-30,36
364	LM	Polar	log_exp	7	20	-33,09	-24,75
365	LM	Polar	log_exp	7	25	-26,38	-23,29
366	LM	Polar	log_exp	7	30	-28,63	-24,26
367	LM	Polar	log_exp	8	5	-41,52	-35,96
368	LM	Polar	log_exp	8	10	-44,55	-36,35
369	LM	Polar	log_exp	8	15	-25,92	-20,53
370	LM	Polar	log_exp	8	20	-21,21	-17,71
371	LM	Polar	log_exp	8	25	-20,02	-17,37
372	LM	Polar	log_exp	8	30	-22,64	-17,42
373	LM	Polar	log_exp	9	5	-31,76	-29,15
374	LM	Polar	log_exp	9	10	-29,25	-22,53
375	LM	Polar	log_exp	9	15	-22,45	-18,36
376	LM	Polar	log_exp	9	20	-25,52	-20,70

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
377	LM	Polar	log_exp	9	25	-27,68	-20,93
378	LM	Polar	log_exp	9	30	-24,09	-18,53
379	LM	Polar	tanh	1	5	-43,82	-43,39
380	LM	Polar	tanh	1	10	-44,39	-43,79
381	LM	Polar	tanh	1	15	-27,74	-25,18
382	LM	Polar	tanh	1	20	-43,60	-42,00
383	LM	Polar	tanh	1	25	-32,23	-29,96
384	LM	Polar	tanh	1	30	-43,03	-41,23
385	LM	Polar	tanh	2	5	-42,17	-40,36
386	LM	Polar	tanh	2	10	-44,26	-43,34
387	LM	Polar	tanh	2	15	-44,38	-42,89
388	LM	Polar	tanh	2	20	-41,12	-35,62
389	LM	Polar	tanh	2	25	-44,57	-42,74
390	LM	Polar	tanh	2	30	-44,10	-40,43
391	LM	Polar	tanh	3	5	-43,71	-43,34
392	LM	Polar	tanh	3	10	-44,17	-42,50
393	LM	Polar	tanh	3	15	-26,97	-25,86
394	LM	Polar	tanh	3	20	-45,30	-43,68
395	LM	Polar	tanh	3	25	-40,34	-35,51
396	LM	Polar	tanh	3	30	-25,18	-23,84
397	LM	Polar	tanh	4	5	-43,91	-43,71
398	LM	Polar	tanh	4	10	-23,62	-22,84
399	LM	Polar	tanh	4	15	-41,32	-32,12
400	LM	Polar	tanh	4	20	-29,77	-25,86
401	LM	Polar	tanh	4	25	-39,57	-26,36
402	LM	Polar	tanh	4	30	-21,63	-19,38
403	LM	Polar	tanh	5	5	-19,78	-17,45
404	LM	Polar	tanh	5	10	-44,53	-31,64
405	LM	Polar	tanh	5	15	-25,32	-20,72
406	LM	Polar	tanh	5	20	-45,68	-43,70
407	LM	Polar	tanh	5	25	-45,31	-35,93
408	LM	Polar	tanh	5	30	-24,68	-20,63
409	LM	Polar	tanh	6	5	-42,05	-37,74
410	LM	Polar	tanh	6	10	-29,22	-23,23
411	LM	Polar	tanh	6	15	-30,67	-25,63
412	LM	Polar	tanh	6	20	-41,82	-31,66
413	LM	Polar	tanh	6	25	-23,51	-18,98
414	LM	Polar	tanh	6	30	-34,50	-24,28
415	LM	Polar	tanh	7	5	-31,18	-25,35

#	Train. Alg.	In/Out	Act. Func.	M	Nº Perceptrons	NMSE T. (dB)	NMSE V. (dB)
416	LM	Polar	tanh	7	10	-38,21	-29,68
417	LM	Polar	tanh	7	15	-45,62	-43,63
418	LM	Polar	tanh	7	20	-43,76	-34,97
419	LM	Polar	tanh	7	25	-16,75	-12,65
420	LM	Polar	tanh	7	30	-34,59	-24,57
421	LM	Polar	tanh	8	5	-22,81	-19,42
422	LM	Polar	tanh	8	10	-20,93	-18,30
423	LM	Polar	tanh	8	15	-24,53	-19,52
424	LM	Polar	tanh	8	20	-17,87	-14,17
425	LM	Polar	tanh	8	25	-40,34	-29,77
426	LM	Polar	tanh	8	30	-18,45	-13,60
427	LM	Polar	tanh	9	5	-44,11	-34,19
428	LM	Polar	tanh	9	10	-35,36	-28,79
429	LM	Polar	tanh	9	15	-16,31	-11,05
430	LM	Polar	tanh	9	20	-37,15	-24,19
431	LM	Polar	tanh	9	25	-31,26	-24,92
432	LM	Polar	tanh	9	30	-29,61	-23,33

Appendix B

Results of tests of the Predistortion system

Table B.1: Results from the tests of the Predistortion system

#	In/Out	Act. F.	M	N^{o} P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
1	Cartesian	tanh	1	5	-15,33	-15,99	35,13	32,47
2	Cartesian	tanh	1	10	-23,82	-23,22	36,48	33,85
3	Cartesian	tanh	1	15	-18,56	-23,00	35,43	33,15
4	Cartesian	tanh	1	20	-23,55	-23,21	37,25	33,47
5	Cartesian	tanh	1	25	-23,53	-23,11	38,44	33,25
6	Cartesian	tanh	1	30	-23,83	-23,22	37,23	33,16
7	Cartesian	tanh	1	35	-23,41	-23,15	38,56	32,76
8	Cartesian	tanh	1	40	-22,93	-23,21	38,22	33,13
9	Cartesian	tanh	2	5	-15,89	15,70	30,27	26,31
10	Cartesian	tanh	2	10	-18,42	-18,91	34,81	29,39
11	Cartesian	tanh	2	15	-23,75	-22,92	36,92	33,49
12	Cartesian	tanh	2	20	-23,27	-23,27	37,06	33,19
13	Cartesian	tanh	2	25	-23,55	-23,22	37,02	33,30
14	Cartesian	tanh	2	30	-23,53	-23,17	38,76	33,37
15	Cartesian	tanh	2	35	-23,03	-23,15	38,05	32,34
16	Cartesian	tanh	2	40	-22,72	-22,86	37,42	32,16
17	Cartesian	tanh	3	5	-16,47	-15,84	34,71	31,15
18	Cartesian	tanh	3	10	-23,03	-23,20	36,81	33,77
19	Cartesian	tanh	3	15	-23,23	-19,68	36,99	29,15
20	Cartesian	tanh	3	20	-23,38	-23,13	37,28	32,74
21	Cartesian	tanh	3	25	-23,07	-23,27	37,74	32,64
22	Cartesian	tanh	3	30	-23,58	-23,37	39,29	33,63
23	Cartesian	tanh	3	35	-23,22	-23,32	39,45	33,54

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
24	Cartesian	tanh	3	40	-23,43	-22,98	39,32	33,34
25	Cartesian	tanh	4	5	-15,52	-16,15	34,84	31,19
26	Cartesian	tanh	4	10	-18,06	-18,39	35,04	30,97
27	Cartesian	tanh	4	15	-23,29	-19,07	37,15	29,92
28	Cartesian	tanh	4	20	-23,34	-23,14	37,32	33,85
29	Cartesian	tanh	4	25	-23,46	-19,63	37,59	29,51
30	Cartesian	tanh	4	30	-23,54	-23,31	38,95	34,51
31	Cartesian	tanh	4	35	-23,20	-23,05	38,94	33,66
32	Cartesian	tanh	4	40	-23,59	-23,24	39,66	33,62
33	Cartesian	tanh	5	5	-17,75	-17,70	33,28	26,39
34	Cartesian	tanh	5	10	-18,50	-18,64	35,16	29,94
35	Cartesian	tanh	5	15	-19,47	-23,47	33,72	33,70
36	Cartesian	tanh	5	20	-23,23	-19,54	37,17	29,48
37	Cartesian	tanh	5	25	-23,56	-23,18	37,25	33,60
38	Cartesian	tanh	5	30	-23,42	-23,15	37,32	33,88
39	Cartesian	tanh	5	35	-22,71	-23,08	38,06	33,76
40	Cartesian	tanh	5	40	-23,00	-23,36	38,84	33,24
41	Cartesian	tanh	6	5	-17,87	-17,75	33,53	29,10
42	Cartesian	tanh	6	10	-18,30	-17,67	35,24	31,63
43	Cartesian	tanh	6	15	-18,29	-16,16	35,87	30,44
44	Cartesian	tanh	6	20	-19,58	-19,79	34,69	29,85
45	Cartesian	tanh	6	25	-24,19	-23,34	37,09	34,90
46	Cartesian	tanh	6	30	-23,67	-23,23	37,53	35,05
47	Cartesian	tanh	6	35	-23,40	-23,25	37,55	34,78
48	Cartesian	tanh	6	40	-23,52	-23,11	38,49	33,00
49	Cartesian	tanh	7	5	-17,52	-17,42	32,96	26,70
50	Cartesian	tanh	7	10	-18,19	-17,97	35,38	30,09
51	Cartesian	tanh	7	15	-18,61	-18,76	34,96	29,33
52	Cartesian	tanh	7	20	-18,84	-19,56	35,28	29,22
53	Cartesian	tanh	7	25	-23,71	-23,71	37,90	34,71
54	Cartesian	tanh	7	30	-19,40	-19,65	34,43	30,20
55	Cartesian	tanh	7	35	-18,67	-19,47	33,65	29,20
56	Cartesian	tanh	7	40	-23,28	-19,47	37,42	29,14
57	Cartesian	tanh	8	5	-14,38	15,66	28,08	26,38
58	Cartesian	tanh	8	10	-18,24	-17,94	35,65	30,84
59	Cartesian	tanh	8	15	-18,54	-18,53	35,17	29,55
60	Cartesian	tanh	8	20	-19,17	-19,69	34,32	29,97
61	Cartesian	tanh	8	25	-23,67	-23,51	37,55	33,68
62	Cartesian	tanh	8	30	-23,64	-23,60	38,10	34,62

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
63	Cartesian	tanh	8	35	-23,51	-23,26	37,54	34,50
64	Cartesian	tanh	8	40	-23,57	-19,52	38,64	29,67
65	Cartesian	tanh	9	5	-12,79	-15,75	30,80	27,26
66	Cartesian	tanh	9	10	-18,20	-17,90	35,67	30,61
67	Cartesian	tanh	9	15	-19,73	-23,52	34,60	34,60
68	Cartesian	tanh	9	20	-18,82	-19,86	34,71	30,11
69	Cartesian	tanh	9	25	-19,64	-19,77	35,10	30,91
70	Cartesian	tanh	9	30	-24,16	-19,72	37,78	29,83
71	Cartesian	tanh	9	35	-18,54	-18,86	34,11	29,22
72	Cartesian	tanh	9	40	-19,22	-19,59	34,52	30,35
73	Cartesian	log_exp	1	5	-28,93	-24,74	37,89	33,25
74	Cartesian	log_exp	1	10	-31,77	-23,41	39,66	33,76
75	Cartesian	log_exp	1	15	-33,83	-29,98	45,97	39,75
76	Cartesian	log_exp	1	20	-34,79	-31,23	45,83	39,72
77	Cartesian	log_exp	1	25	-38,47	-32,51	49,64	40,67
78	Cartesian	log_exp	1	30	-36,44	-32,70	49,56	40,54
79	Cartesian	log_exp	1	35	-34,00	-30,54	48,96	40,00
80	Cartesian	log_exp	1	40	-38,63	-34,77	50,98	45,02
81	Cartesian	log_exp	2	5	-28,05	-26,46	37,67	33,79
82	Cartesian	log_exp	2	10	-23,79	-23,42	37,47	33,99
83	Cartesian	log_exp	2	15	-35,85	-24,29	44,75	34,65
84	Cartesian	log_exp	2	20	-36,23	-32,88	48,01	40,56
85	Cartesian	log_exp	2	25	-36,93	-34,73	49,41	43,53
86	Cartesian	log_exp	2	30	-36,30	-33,38	49,97	42,29
87	Cartesian	log_exp	2	35	-38,15	-33,50	49,96	41,95
88	Cartesian	log_exp	2	40	-35,54	-33,53	49,92	42,70
89	Cartesian	log_exp	3	5	-27,56	-25,10	37,76	33,69
90	Cartesian	log_exp	3	10	-32,97	-23,89	41,37	34,63
91	Cartesian	log_exp	3	15	-34,50	-31,40	46,32	40,26
92	Cartesian	log_exp	3	20	-35,13	-30,88	46,83	41,90
93	Cartesian	log_exp	3	25	-37,58	-33,37	48,86	43,37
94	Cartesian	log_exp	3	30	-39,82	-36,10	51,42	44,93
95	Cartesian	log_exp	3	35	-36,61	-34,33	50,18	43,40
96	Cartesian	log_exp	3	40	-33,10	-31,64	48,81	40,80
97	Cartesian	log_exp	4	5	-28,11	-24,54	37,63	32,78
98	Cartesian	log_exp	4	10	-31,72	-29,16	40,78	38,12
99	Cartesian	log_exp	4	15	-33,34	-31,36	46,08	39,72
100	Cartesian	log_exp	4	20	-35,02	-31,26	46,00	41,05
101	Cartesian	log_exp	4	25	-38,57	-33,93	50,29	42,55

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
102	Cartesian	log_exp	4	30	-39,60	-34,17	50,83	43,75
103	Cartesian	log_exp	4	35	-37,96	-34,42	51,39	44,43
104	Cartesian	log_exp	4	40	-38,59	-34,33	51,87	45,08
105	Cartesian	log_exp	5	5	-27,82	-24,87	37,23	32,09
106	Cartesian	log_exp	5	10	-30,55	-25,75	39,24	35,44
107	Cartesian	log_exp	5	15	-34,44	-29,53	45,80	38,82
108	Cartesian	log_exp	5	20	-36,81	-33,46	48,11	43,30
109	Cartesian	log_exp	5	25	-35,58	-33,40	48,87	42,00
110	Cartesian	log_exp	5	30	-36,41	-34,71	50,86	44,65
111	Cartesian	log_exp	5	35	-38,18	-33,47	51,55	43,15
112	Cartesian	log_exp	5	40	-38,05	-34,76	52,07	44,62
113	Cartesian	log_exp	6	5	-29,05	-25,33	38,05	33,79
114	Cartesian	log_exp	6	10	-31,50	-26,36	39,99	35,51
115	Cartesian	log_exp	6	15	-35,16	-24,07	45,83	34,44
116	Cartesian	log_exp	6	20	-37,55	-32,93	49,19	42,09
117	Cartesian	log_exp	6	25	-36,37	-33,48	48,62	40,83
118	Cartesian	log_exp	6	30	-38,70	-33,68	49,48	41,60
119	Cartesian	log_exp	6	35	-38,48	-33,50	51,31	44,80
120	Cartesian	log_exp	6	40	-38,67	-35,31	50,52	44,28
121	Cartesian	log_exp	7	5	-28,43	-25,20	37,90	33,37
122	Cartesian	log_exp	7	10	-32,69	-26,69	42,08	34,34
123	Cartesian	log_exp	7	15	-35,20	-30,86	46,98	39,46
124	Cartesian	log_exp	7	20	-37,03	-31,96	48,20	40,35
125	Cartesian	log_exp	7	25	-36,61	-34,25	49,90	43,75
126	Cartesian	log_exp	7	30	-37,68	-34,25	50,70	44,07
127	Cartesian	log_exp	7	35	-37,38	-33,20	49,14	43,04
128	Cartesian	log_exp	7	40	-36,03	-32,98	49,06	40,82
129	Cartesian	log_exp	8	5	-28,11	-25,15	37,55	32,67
130	Cartesian	log_exp	8	10	-24,92	-24,77	37,91	34,32
131	Cartesian	log_exp	8	15	-36,61	-24,72	46,68	33,77
132	Cartesian	log_exp	8	20	-37,52	-33,37	47,79	40,44
133	Cartesian	log_exp	8	25	-37,48	-31,97	48,78	40,82
134	Cartesian	log_exp	8	30	-38,88	-34,06	48,77	43,09
135	Cartesian	log_exp	8	35	-36,96	-34,37	50,64	43,38
136	Cartesian	log_exp	8	40	-36,99	-33,90	50,31	42,32
137	Cartesian	log_exp	9	5	-28,34	-24,66	37,28	32,60
138	Cartesian	log_exp	9	10	-32,19	-27,09	42,34	35,01
139	Cartesian	log_exp	9	15	-35,66	-31,29	46,54	40,33
140	Cartesian	log_exp	9	20	-35,80	-31,41	47,75	40,93

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
141	Cartesian	log_exp	9	25	-37,95	-32,34	49,00	41,37
142	Cartesian	log_exp	9	30	-38,88	-34,76	50,35	42,55
143	Cartesian	log_exp	9	35	-36,01	-32,06	48,44	40,91
144	Cartesian	log_exp	9	40	-36,03	-33,73	50,53	42,69
145	Polar	tanh	1	5	-42,92	-38,62	54,57	49,61
146	Polar	tanh	1	10	-41,68	-36,50	55,94	50,49
147	Polar	tanh	1	15	-44,05	-37,79	57,36	50,36
148	Polar	tanh	1	20	-41,30	-36,76	56,49	51,07
149	Polar	tanh	1	25	-43,07	-38,50	57,41	50,83
150	Polar	tanh	1	30	-44,73	-37,31	56,49	49,63
151	Polar	tanh	1	35	-40,63	-37,18	55,04	50,76
152	Polar	tanh	1	40	-38,71	-34,86	53,52	49,07
153	Polar	tanh	2	5	-43,49	-37,55	54,43	48,34
154	Polar	tanh	2	10	-43,57	-38,24	56,03	49,57
155	Polar	tanh	2	15	-40,16	-36,42	54,51	48,80
156	Polar	tanh	2	20	-43,34	-36,93	56,34	51,74
157	Polar	tanh	2	25	-43,50	-37,38	56,26	51,33
158	Polar	tanh	2	30	-42,83	-34,40	54,89	47,57
159	Polar	tanh	2	35	-40,51	-36,65	52,25	47,68
160	Polar	tanh	2	40	-41,25	-36,21	52,81	47,57
161	Polar	tanh	3	5	-38,97	-35,39	50,36	47,43
162	Polar	tanh	3	10	-42,65	-37,66	57,23	50,03
163	Polar	tanh	3	15	-44,92	-37,55	57,17	50,95
164	Polar	tanh	3	20	-36,29	-31,48	48,21	41,87
165	Polar	tanh	3	25	-44,72	-37,24	56,80	51,30
166	Polar	tanh	3	30	-37,53	-32,50	52,32	43,39
167	Polar	tanh	3	35	-41,23	-36,70	54,18	48,96
168	Polar	tanh	3	40	-38,88	-33,11	51,71	46,78
169	Polar	tanh	4	5	-38,70	-36,73	53,04	48,43
170	Polar	tanh	4	10	-44,75	-38,31	57,83	49,57
171	Polar	tanh	4	15	-38,55	-32,85	53,11	46,34
172	Polar	tanh	4	20	-43,72	-36,50	57,44	51,29
173	Polar	tanh	4	25	-35,47	-33,32	51,30	46,38
174	Polar	tanh	4	30	-37,20	-36,12	52,04	45,11
175	Polar	tanh	4	35	-37,86	-34,38	51,77	47,09
176	Polar	tanh	4	40	-41,41	-36,38	55,68	48,06
177	Polar	tanh	5	5	-40,71	-36,72	50,96	48,37
178	Polar	tanh	5	10	-42,46	-35,74	56,77	49,59
179	Polar	tanh	5	15	-42,22	-36,31	55,28	48,38

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
180	Polar	tanh	5	20	-38,83	-35,47	53,38	47,07
181	Polar	tanh	5	25	-36,90	-35,35	50,03	45,63
182	Polar	tanh	5	30	-32,21	-31,37	45,16	42,11
183	Polar	tanh	5	35	-36,42	-35,18	52,79	48,48
184	Polar	tanh	5	40	-30,90	-28,46	43,66	38,23
185	Polar	tanh	6	5	-39,59	-37,28	53,35	48,11
186	Polar	tanh	6	10	-34,72	-29,94	47,17	40,62
187	Polar	tanh	6	15	-42,71	-37,09	57,44	51,44
188	Polar	tanh	6	20	-38,62	-32,81	52,24	47,78
189	Polar	tanh	6	25	-38,18	-34,35	55,20	44,73
190	Polar	tanh	6	30	-20,13	-20,95	33,04	31,90
191	Polar	tanh	6	35	-42,30	-37,41	56,05	49,68
192	Polar	tanh	6	40	-34,42	-34,65	49,56	45,65
193	Polar	tanh	7	5	-40,75	-36,91	53,46	48,21
194	Polar	tanh	7	10	-42,56	-37,34	56,24	50,15
195	Polar	tanh	7	15	-23,86	-21,41	34,88	30,55
196	Polar	tanh	7	20	-30,04	-26,41	41,51	38,38
197	Polar	tanh	7	25	-22,81	-22,80	35,90	33,75
198	Polar	tanh	7	30	-18,83	-17,21	31,66	28,15
199	Polar	tanh	7	35	-34,00	-34,34	52,85	47,40
200	Polar	tanh	7	40	-18,09	-17,37	29,32	27,22
201	Polar	tanh	8	5	-33,99	-34,06	39,88	42,06
202	Polar	tanh	8	10	-18,69	-21,25	34,91	31,29
203	Polar	tanh	8	15	-21,09	-19,37	33,07	28,50
204	Polar	tanh	8	20	-29,33	-29,30	40,73	41,13
205	Polar	tanh	8	25	-28,49	-27,55	40,80	38,39
206	Polar	tanh	8	30	-32,53	-29,86	45,78	41,87
207	Polar	tanh	8	35	-27,94	-26,00	41,29	38,76
208	Polar	tanh	8	40	-33,99	-31,14	50,54	46,10
209	Polar	tanh	9	5	-41,70	-35,97	53,63	47,83
210	Polar	tanh	9	10	-42,36	-36,87	54,86	49,41
211	Polar	tanh	9	15	-16,75	15,69	27,82	26,37
212	Polar	tanh	9	20	-27,18	-24,05	37,33	32,61
213	Polar	tanh	9	25	-42,22	-37,93	57,14	51,25
214	Polar	tanh	9	30	-22,45	-24,26	37,26	36,48
215	Polar	tanh	9	35	-26,60	-28,22	40,06	39,89
216	Polar	tanh	9	40	-33,67	-31,42	46,41	40,49
217	Polar	log_exp	1	5	-42,18	-37,57	54,34	48,18
218	Polar	log_exp	1	10	-40,63	-37,43	56,18	49,30

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
219	Polar	log_exp	1	15	-42,38	-37,61	55,49	48,79
220	Polar	log_exp	1	20	-40,94	-36,26	55,78	50,04
221	Polar	log_exp	1	25	-42,12	-36,67	56,28	49,45
222	Polar	log_exp	1	30	-42,44	-36,70	56,74	51,24
223	Polar	log_exp	1	35	-43,41	-35,73	55,30	50,28
224	Polar	log_exp	1	40	-40,30	-35,79	54,96	49,60
225	Polar	log_exp	2	5	-44,30	-37,45	55,37	48,85
226	Polar	log_exp	2	10	-42,23	-39,76	54,73	48,51
227	Polar	log_exp	2	15	-39,67	-35,42	51,89	45,65
228	Polar	log_exp	2	20	-42,23	-38,11	55,75	50,37
229	Polar	log_exp	2	25	-41,98	-37,04	55,47	48,76
230	Polar	log_exp	2	30	-38,67	-35,50	52,54	49,27
231	Polar	log_exp	2	35	-39,11	-34,95	51,44	45,57
232	Polar	log_exp	2	40	-36,14	-31,68	47,92	42,65
233	Polar	log_exp	3	5	-41,44	-37,55	53,69	47,99
234	Polar	log_exp	3	10	-44,66	-38,36	57,30	49,01
235	Polar	log_exp	3	15	-44,00	-36,76	57,30	50,56
236	Polar	log_exp	3	20	-44,69	-38,34	57,60	48,41
237	Polar	log_exp	3	25	-39,96	-36,75	54,16	48,96
238	Polar	log_exp	3	30	-19,91	-21,03	29,63	32,35
239	Polar	log_exp	3	35	-43,59	-35,38	56,56	49,35
240	Polar	log_exp	3	40	-44,07	-35,48	56,66	49,89
241	Polar	log_exp	4	5	-34,70	-34,32	42,44	43,32
242	Polar	log_exp	4	10	-44,34	-37,75	57,17	50,74
243	Polar	log_exp	4	15	-44,51	-37,37	56,68	50,04
244	Polar	log_exp	4	20	-41,33	-37,92	55,42	51,31
245	Polar	log_exp	4	25	-42,86	-36,92	54,31	48,37
246	Polar	log_exp	4	30	-42,78	-36,85	56,60	49,88
247	Polar	log_exp	4	35	-26,10	-25,06	37,94	37,35
248	Polar	log_exp	4	40	-24,04	-21,75	37,57	32,93
249	Polar	log_exp	5	5	-25,87	-26,37	39,50	37,74
250	Polar	log_exp	5	10	-42,63	-36,33	57,35	51,26
251	Polar	log_exp	5	15	-42,70	-38,22	57,51	50,56
252	Polar	log_exp	5	20	-41,30	-36,38	53,11	48,65
253	Polar	log_exp	5	25	-34,13	-28,91	47,98	41,38
254	Polar	log_exp	5	30	-43,66	-36,72	57,04	50,30
255	Polar	log_exp	5	35	-44,55	-36,77	56,68	50,03
256	Polar	log_exp	5	40	-43,66	-36,65	55,66	48,50
257	Polar	log_exp	6	5	-43,60	-36,99	54,14	48,50

#	In/Out	Act. F.	M	Nº P.	NMSE T. (dB)	NMSE V. (dB)	ACPR T. (dB)	ACPR V. (dB)
258	Polar	log_exp	6	10	-44,59	-36,41	57,24	51,01
259	Polar	log_exp	6	15	-44,83	-38,39	57,73	50,27
260	Polar	log_exp	6	20	-29,00	-27,46	41,23	38,05
261	Polar	log_exp	6	25	15,68	15,68	26,39	26,39
262	Polar	log_exp	6	30	-40,42	-37,84	55,61	49,78
263	Polar	log_exp	6	35	-43,24	-36,16	54,44	48,22
264	Polar	log_exp	6	40	-43,31	-36,70	55,43	49,17
265	Polar	log_exp	7	5	-42,65	-36,66	56,13	49,20
266	Polar	log_exp	7	10	-25,36	-26,18	39,78	38,34
267	Polar	log_exp	7	15	-27,45	-25,73	42,94	37,05
268	Polar	log_exp	7	20	-43,03	-35,90	56,01	48,98
269	Polar	log_exp	7	25	-41,25	-37,32	54,96	49,29
270	Polar	log_exp	7	30	-43,67	-38,76	54,20	46,99
271	Polar	log_exp	7	35	-26,57	-20,26	38,75	31,53
272	Polar	log_exp	7	40	-26,84	-24,63	38,52	35,95
273	Polar	log_exp	8	5	-40,22	-37,49	53,90	47,43
274	Polar	log_exp	8	10	-42,70	-37,36	56,25	47,45
275	Polar	log_exp	8	15	-27,95	-22,81	40,95	34,82
276	Polar	log_exp	8	20	-24,35	-21,42	37,01	33,44
277	Polar	log_exp	8	25	-41,52	-37,19	54,79	48,34
278	Polar	log_exp	8	30	15,68	15,68	26,39	26,39
279	Polar	log_exp	8	35	15,71	15,71	26,29	26,29
280	Polar	log_exp	8	40	-43,50	-37,26	56,18	49,10
281	Polar	log_exp	9	5	-43,59	-38,58	55,09	47,72
282	Polar	log_exp	9	10	-43,55	-38,45	56,14	50,77
283	Polar	log_exp	9	15	15,67	15,67	26,38	26,38
284	Polar	log_exp	9	20	-43,08	-37,23	56,61	49,27
285	Polar	log_exp	9	25	15,70	15,70	26,35	26,35
286	Polar	log_exp	9	30	-32,04	-27,14	44,35	38,42
287	Polar	log_exp	9	35	-35,11	-35,96	47,70	48,33
288	Polar	log_exp	9	40	-41,17	-38,84	55,69	50,11

Appendix C

MatLab code

```
function [NMSEtraindB, NMSEValdB, ACPR, ACPRtrain, ACPRbest,
    WHminmax, BHminmax, WOminmax, BOminmax, ZHminmax, N1minmax,
    O1minmax, O2minmax, wh1, wout, bin1, bout]= predist_grid(hl1 ,M,
    actfunc, itTrain, polar, debug)
%main function
NMSEtraindB=zeros(1, itTrain+1);
NMSEValdB=zeros(1, itTrain+1);
ACPR=zeros(1, itTrain+1);
ACPRtrain=zeros(1, itTrain+1);
nI=M+1;

data=load('GSM_data.mat');

%%params
hiddenLayerSize1=hl1;
if polar==1

    InputLayerSize=3*nI-2;
else
    InputLayerSize=2*nI;
end
rng(90);

%test output without predistortion (D=X)
[yNovoPrimeiro, RMSout, Idc, Vdc]=RFWebLab_PA_meas_v1_1(data.xa
    ,-15.5);
yNovoPrimeiro=yNovoPrimeiro-mean(yNovoPrimeiro);

```

```
25 [outAlignPrimeiro, inAlignPrimeiro, D]=alignsignals(
       yNovoPrimeiro,data.xa);
26 yalignPrimeiro=outAlignPrimeiro(abs(D)+1:end);
27 xalignPrimeiro=inAlignPrimeiro(abs(D)+1:end);
28 if D<0
29     xalignPrimeiro=xalignPrimeiro(1:end-abs(D));
30
31 else
32     yalignPrimeiro=yalignPrimeiro(1:end-abs(D));
33
34 end
35 yalignComp=yalignPrimeiro/max(abs(yalignPrimeiro));
36 xalignComp=xalignPrimeiro/max(abs(xalignPrimeiro));
37 Xnorm=sum(abs(xalignPrimeiro).^2);
38 NMSEtraindB(1)=-10*log10(1/(sum(abs(xalignComp-yalignComp).^2)/
       Xnorm));
39 NMSEValdB(1)=-10*log10(1/(sum(abs(xalignComp-yalignComp).^2)/
       Xnorm));
40 Mham = length(yalignPrimeiro);
41 w = hanning(Mham);
42 yw = w.*yalignPrimeiro;
43 fftY=fftshift(fft(yw/max(abs(yw))));
44 n=[-100:1/(length(fftY)-1)*200:100];
45 pointsInf=find((-5-10/3-10/6)<n & n<(-5-10/3+10/6));
46 pointsSup=find((5+10/3-10/6)<n & n<(5+10/3+10/6));
47 pointsMain=find((-5-10/6)<n & n<(-5+10/6));
48 powerMain=sum(fftY(pointsMain).*conj(fftY(pointsMain))*1/(length
       (fftY)-1)*200*10^6);
49 powerSup=sum(fftY(pointsSup).*conj(fftY(pointsSup))*1/(length(
       fftY)-1)*200*10^6);
50 powerInf=sum(fftY(pointsInf).*conj(fftY(pointsInf))*1/(length(
       fftY)-1)*200*10^6);
51 ACPRtrain(1)=min([10*log10(powerMain/powerSup) 10*log10(
       powerMain/powerInf)]);
52 ACPR(1)=min([10*log10(powerMain/powerSup) 10*log10(powerMain/
       powerInf)]);
53
54
55 %bias init
56 bin1=zeros(hiddenLayerSize1,1); %biases of the hidden layer
```

```
57 bout= randn(2,1); %biases of the output layer
58 %weights init
59 wh1=randn(hiddenLayerSize1,InputLayerSize); % weights from input
       to hidden layer
60 wout=randn(2,hiddenLayerSize1); %weights from hidden layer to
      output layer
61
62 [wh1, wout, bin1,bout]=train_network_LM(M,hl1,actfunc,3,data.xa,
      data.ya,wh1,wout,bin1,bout,polar,debug);
63 nVal=2;
64
65 for ntrain=2:itTrain
66
67
68
69
70           [Xcompare,NMSEVal, xalignVal, yalignVal ]=
                 generate_D_Y_val(data.xa,data.ya,hl1,M,actfunc,wh1,
                 wout,bin1,bout,polar,debug);
71           Mham = length(yalignVal);%compute ACPR
72           w = hanning(Mham);
73           yw = w.*yalignVal;
74           fftY=fftshift(fft(yw/max(abs(yw))));
75           n=[-100:1/(length(fftY)-1)*200:100];
76           pointsInf=find((-5-10/3-10/6)<n & n<(-5-10/3+10/6));
77           pointsSup=find((5+10/3-10/6)<n & n<(5+10/3+10/6));
78           pointsMain=find((-5-10/6)<n & n<(-5+10/6));
79           powerMain=sum(fftY(pointsMain).*conj(fftY(pointsMain))
                 *1/(length(fftY)-1)*200*10^6);
80           powerSup=sum(fftY(pointsSup).*conj(fftY(pointsSup))*1/(
                 length(fftY)-1)*200*10^6);
81           powerInf=sum(fftY(pointsInf).*conj(fftY(pointsInf))*1/(
                 length(fftY)-1)*200*10^6);
82           ACPR(nVal)=min([10*log10(powerMain/powerSup) 10*log10(
                 powerMain/powerInf)]);
83
84
85
86           if ACPR(nVal)==max(ACPR)
87               save('best_network.mat','wh1','wout','bin1','bout')
```

```
88          end
89          NMSEValdB(nVal)=-NMSEVal;
90          if debug
91
92              figure(4)
93              subplot(2,1,1)
94              nscatter=(0:nVal-1).*2;
95              scatter(nscatter,NMSEValdB(1:nVal))
96              title('NMSE Val');
97              drawnow
98              subplot(2,1,2)
99              scatter(nscatter,ACPR(1:nVal))
100             title('ACPR Val');
101             drawnow
102         end
103         nVal=nVal+1;
104
105
106     [Xcompare,NMSEnovo, xalign, yalign,WHmin, WHmax,BHmin,BHmax,
            WOmin,WOmax,BOmin,BOmax,ZHmin,ZHmax,N1min,N1max,O1min,
            O1max,O2min,O2max ]=generate_D_Y(data.xa,data.ya,hl1,M,
            actfunc,wh1,wout,bin1,bout,polar,debug);
107
108
109     if ntrain==2
110         WHminmax=[WHmin WHmax];
111         BHminmax=[BHmin BHmax];
112         WOminmax=[WOmin WOmax];
113         BOminmax=[BOmin BOmax];
114         ZHminmax=[ZHmin ZHmax];
115         N1minmax=[N1min N1max];
116         O1minmax=[O1min O1max];
117         O2minmax=[O2min O2max];
118     else
119         WHminmax=[min(WHminmax(1),WHmin) max(WHminmax(2),WHmax)
                ];
120         BHminmax=[min(BHminmax(1),BHmin) max(BHminmax(2),BHmax)
                ];
121         WOminmax=[min(WOminmax(1),WOmin) max(WOminmax(2),WOmax)
                ];
```

```
122             BOminmax=[min(BOminmax(1),BOmin) max(BOminmax(2),BOmax)
                    ];
123             ZHminmax=[min(ZHminmax(1),ZHmin) max(ZHminmax(2),ZHmax)
                    ];
124             N1minmax=[min(N1minmax(1),N1min) max(N1minmax(2),N1max)
                    ];
125             O1minmax=[min(O1minmax(1),O1min) max(O1minmax(2),O1max)
                    ];
126             O2minmax=[min(O2minmax(1),O2min) max(O2minmax(2),O2max)
                    ];
127         end
128         %%compute ACPR
129         Mham = length(yalign);
130         w = hanning(Mham);
131         yw = w.*yalign;
132         fftY=fftshift(fft(yw/max(abs(yw))));
133         n=[-100:1/(length(fftY)-1)*200:100];
134         pointsInf=find((-5-10/3-10/6)<n & n<(-5-10/3+10/6));
135         pointsSup=find((5+10/3-10/6)<n & n<(5+10/3+10/6));
136         pointsMain=find((-5-10/6)<n & n<(-5+10/6));
137         powerMain=sum(fftY(pointsMain).*conj(fftY(pointsMain))*1/(
                length(fftY)-1)*200*10^6);
138         powerSup=sum(fftY(pointsSup).*conj(fftY(pointsSup))*1/(
                length(fftY)-1)*200*10^6);
139         powerInf=sum(fftY(pointsInf).*conj(fftY(pointsInf))*1/(
                length(fftY)-1)*200*10^6);
140         ACPRtrain(ntrain)=min([10*log10(powerMain/powerSup) 10*log10
                (powerMain/powerInf)]);
141         NMSEtraindB(ntrain)=-NMSEnovo;
142
143         if debug
144
145             figure(3)
146             subplot(2,1,1)
147             nscatter=(0:ntrain-1);
148             scatter(nscatter,NMSEtraindB(1:ntrain))
149             title('NMSE train');
150             drawnow
151             subplot(2,1,2)
152             scatter(nscatter,ACPRtrain(1:ntrain))
```

```
153             title('ACPR train');
154             drawnow
155         end
156
157         [wh1, wout, bin1,bout]=train_network_LM(M,hl1,actfunc,50,
                xalign',yalign,wh1,wout,bin1,bout,polar,debug);
158     end
159     ACPRbest=max(ACPR);
160     load('best_network.mat');
161     end
162
163
164
165     function r=act(n)
166     %%log exponential function
167
168     n(find(n>700))=700;
169     n(find(n<-700))=-700;
170     r=1/2*n+log(exp(n)+exp(-n))/2;
171     end
172
173     function r=d_act(n)
174     %%derivative of the log exponential function
175
176     n(find(n>700))=700;
177     n(find(n<-700))=-700;
178     r=1/2+1/2*((exp(n)-exp(-n))./(exp(n)+exp(-n)));
179     end
180
181     function [Xcompare,NMSEnovo, xalign, yalign, WHmin, WHmax,BHmin,
        BHmax,WOmin,WOmax,BOmin,BOmax,ZHmin,ZHmax,N1min,N1max,O1min,
        O1max,O2min,O2max ]=generate_D_Y(X_in,Y_in,hl1,M,actfunc,wh1,
        wout,bin1,bout,polar,debug)
182     % generates signal D and Y
183
184     nI=M+1;
185
186     if polar==1
187
188         InputLayerSize=3*nI-2;
```

```
189 else
190     InputLayerSize=2*nI;
191 end
192 batchSize=40050;
193 numInputs=nI;
194 rng(90);
195
196
197 Xtotal=X_in;
198
199
200 X=zeros(InputLayerSize, batchSize);
201 k=10000;
202
203
204 if polar==1
205
206     [Xtotal_Theta,Xtotal_Mag]=cart2pol(real(Xtotal),imag(Xtotal)
        );
207     Xtotal_Mag=Xtotal_Mag/max(Xtotal_Mag);
208
209
210     X_Theta=zeros(1,batchSize);
211
212     for ncr= 1:batchSize
213         X_Mag=Xtotal_Mag(k+numInputs:-1:1+k);
214
215         X_cos=cos(Xtotal_Theta(k+numInputs:-1:2+k)-Xtotal_Theta(
            k+numInputs-1:-1:1+k));
216         X_sin=sin(Xtotal_Theta(k+numInputs:-1:2+k)-Xtotal_Theta(
            k+numInputs-1:-1:1+k));
217         X_Theta(ncr)=Xtotal_Theta(k+numInputs);
218         if nI==1
219             X(:,ncr)=X_Mag;
220         else
221             X(:,ncr)=[X_Mag;X_cos;X_sin];
222         end
223         k=k+1;
224     end
225 else
```

```
226         Xtotal=Xtotal/max(abs(Xtotal));
227         for ncr= 1:batchSize
228
229             X(:,ncr)=[real(Xtotal(k+numInputs:-1:k+1));imag(Xtotal(k
                    +numInputs:-1:k+1))];
230             k=k+1;
231         end
232     end
233
234
235     k=10000;
236
237     %%%%forward propagation%%%%
238     z1=wh1*X+bin1*ones(1,batchSize);
239     if actfunc== 1
240         N1=act(z1);
241     else
242         N1=tansig(z1);
243     end
244
245     %out
246     out=wout*N1+bout*ones(1,batchSize);
247
248     if polar==1
249
250         outplot=[out(1,:);X_Theta+out(2,:)];
251
252         outplot2PAMag=outplot(1,:);
253         outplot2PA=outplot2PAMag.*exp(1i*(outplot(2,:)));
254     else
255         outplot=[out(1,:);out(2,:)];
256
257         outplot2PA=outplot(1,:)+1i*outplot(2,:);
258     end
259
260     if mod(length(outplot2PA),2)==1
261         outplot2PA=outplot2PA(1:end-1);
262
263     end
264
```

```
265 RMSi=-15;
266 while 1 %%get best RMS
267     if check_inputs_WebLab(outplot2PA, RMSi);
268         break
269     else
270         RMSi=RMSi-0.5;
271     end
272 end
273 if debug
274     RMSi
275 end
276
277 [yNovo, RMSout, Idc, Vdc]=RFWebLab_PA_meas_v1_1(outplot2PA,RMSi)
    ;
278 yNovo=yNovo-mean(yNovo); %%correct output signal
279
280 [outAlign, inAlign, D]=alignsignals(yNovo,outplot2PA); %%align
    and crop signals
281 yalign=outAlign(abs(D)+1:end);
282 xalign=inAlign(abs(D)+1:end)';
283 if D<0
284     xalign=xalign(1:end-abs(D));
285     Xcompare=X_in(k+numInputs:k+numInputs+length(yalign)-1);
286
287 else
288     yalign=yalign(1:end-abs(D));
289     Xcompare=X_in(k+numInputs+abs(D):k+numInputs+length(yalign)+
        abs(D)-1);
290
291 end
292 Xcompare=Xcompare/max(abs(Xcompare));
293 yalignComp=yalign/max(abs(yalign));
294 Xnorm=sum(abs(Xcompare).^2);
295 NMSEnovo=10*log10(1/(sum(abs(Xcompare-yalignComp).^2)/Xnorm));
296 WHmin=min(min(wh1));
297 WHmax=max(max(wh1));
298 BHmin=min(bin1);
299 BHmax=max(bin1);
300 WOmin=min(min(wout));
301 WOmax=max(max(wout));
```

```
BOmin=min(bout);
BOmax=max(bout);
ZHmin=min(min(z1));
ZHmax=max(max(z1));
N1min=min(min(N1));
N1max=max(max(N1));
O1min=min(min(out(1,:)));
O1max=max(max(out(1,:)));
O2min=min(min(out(2,:)));
O2max=max(max(out(2,:)));
end


function [Xcompare, NMSEnovo, xalign, yalign]=generate_D_Y_val(
    X_in,Y_in,hl1,M,actfunc,wh1,wout,bin1,bout, polar,debug)
%%generates signal D and Y for validation

nI=M+1;
if polar==1

    InputLayerSize=3*nI-2;
else
    InputLayerSize=2*nI;
end

numInputs=nI;
rng(90);
batchSize=10050;

Xtotal=X_in;



X=zeros(InputLayerSize, batchSize);
k=51000;


if polar==1

```

```
340         [Xtotal_Theta ,Xtotal_Mag]=cart2pol(real(Xtotal),imag(Xtotal)
                ) ;
341         Xtotal_Mag=Xtotal_Mag/max(Xtotal_Mag) ;
342
343
344         X_Theta=zeros(1,batchSize) ;
345
346         for ncr= 1:batchSize
347             X_Mag=Xtotal_Mag(k+numInputs:-1:1+k) ;
348
349             X_cos=cos(Xtotal_Theta(k+numInputs:-1:2+k)-Xtotal_Theta(
                    k+numInputs-1:-1:1+k)) ;
350             X_sin=sin(Xtotal_Theta(k+numInputs:-1:2+k)-Xtotal_Theta(
                    k+numInputs-1:-1:1+k)) ;
351             X_Theta(ncr)=Xtotal_Theta(k+numInputs) ;
352             if nI==1
353                 X(:,ncr)=X_Mag;
354             else
355                 X(:,ncr)=[X_Mag;X_cos;X_sin];
356             end
357             k=k+1;
358         end
359     else
360         Xtotal=Xtotal/max(abs(Xtotal)) ;
361         for ncr= 1:batchSize
362
363             X(:,ncr)=[real(Xtotal(k+numInputs:-1:k+1));imag(Xtotal(k
                    +numInputs:-1:k+1))];
364             k=k+1;
365         end
366     end
367
368
369     k=51000;
370
371     %%%%forward propagation%%%%
372     z1=wh1*X+bin1*ones(1,batchSize) ;
373     if actfunc== 1
374         N1=act(z1) ;
375     else
```

```
376     N1=tansig(z1);
377 end
378
379 out=wout*N1+bout*ones(1,batchSize);
380
381
382 if polar==1
383
384     outplot=[out(1,:);X_Theta+out(2,:)];
385
386     outplot2PAMag=outplot(1,:);
387     outplot2PA=outplot2PAMag.*exp(1i*(outplot(2,:)));
388 else
389     outplot=[out(1,:);out(2,:)];
390
391     outplot2PA=outplot(1,:)+1i*outplot(2,:);
392 end
393
394 if mod(length(outplot2PA),2)==1
395     outplot2PA=outplot2PA(1:end-1);
396 end
397 RMSi=-15;
398 while 1 %%get best RMS
399     if check_inputs_WebLab(outplot2PA, RMSi);
400         break
401     else
402         RMSi=RMSi-0.5;
403     end
404 end
405 if debug
406     RMSi
407 end
408
409 [yNovo, RMSout, Idc, Vdc]=RFWebLab_PA_meas_v1_1(outplot2PA,RMSi)
    ;
410 yNovo=yNovo-mean(yNovo); %%correct output signal
411
412 [outAlign, inAlign, D]=alignsignals(yNovo,outplot2PA); %%align
    and crop signals
413 yalign=outAlign(abs(D)+1:end);
```

```
414 xalign=inAlign(abs(D)+1:end)';
415 if D<0
416     xalign=xalign(1:end-abs(D));
417     Xcompare=X_in(k+numInputs:k+numInputs+length(yalign)-1);
418
419 else
420     yalign=yalign(1:end-abs(D));
421     Xcompare=X_in(k+numInputs+abs(D):k+numInputs+length(yalign)+
            abs(D)-1);
422
423 end
424 Xcompare=Xcompare/max(abs(Xcompare));
425 yalignComp=yalign/max(abs(yalign));
426 Xnorm=sum(abs(Xcompare).^2);
427 NMSEnovo=10*log10(1/(sum(abs(Xcompare-yalignComp).^2)/Xnorm));
428 end
```

References

[1] NGMN Alliance. 5g white paper. *Next generation mobile networks, white paper*, 2015.

[2] Cheng-Xiang Wang, Fourat Haider, Xiqi Gao, Xiao-Hu You, Yang Yang, Dongfeng Yuan, Hadi Aggoune, Harald Haas, Simon Fletcher, and Erol Hepsaydir. Cellular architecture and key technologies for 5g wireless communication networks. *IEEE Communications Magazine*, 52(2):122–130, 2014.

[3] Nicola Marchetti. Towards the 5th generation of wireless communication systems. *arXiv preprint arXiv:1702.00370*, 2017.

[4] Congzheng Han, Tim Harrold, Simon Armour, Ioannis Krikidis, Stefan Videv, Peter M Grant, Harald Haas, John S Thompson, Ivan Ku, Cheng-Xiang Wang, et al. Green radio: radio techniques to enable energy-efficient wireless networks. *IEEE communications magazine*, 49(6), 2011.

[5] Ilari Teikari et al. *Digital predistortion linearization methods for RF power amplifiers*. Teknillinen korkeakoulu, 2008.

[6] Yeqing Qian and Fuqiang Liu. Neural network predistortion technique for nonlinear power amplifiers with memory. In *Communications and Networking in China, 2006. ChinaCom'06. First International Conference on*, pages 1–5. IEEE, 2006.

[7] Melin Ngwar. *An Analog Neural Network for Wideband Predistortion of Pico-cell Power Amplifiers*. PhD thesis, thesis, Carleton University, 2015. Available: https://curve. carleton. ca/2d167908-4475-4ab3-b192-eeb4c656e2de, 2015.

[8] Allen Katz. Linearization: Reducing distortion in power amplifiers. *IEEE microwave magazine*, 2(4):37–49, 2001.

[9] Abbas Mohammadi and Fadhel M Ghannouchi. *RF transceiver design for MIMO wireless communications*, volume 145. Springer Science & Business Media, 2012.

[10] Sonam Goyal and Jyoti Gupta. Review of power amplifier linearization techniques in communication systems. *International Journal of Engineering Research & Technology*, 2(6), 2013.

[11] Mohammad Javad Rezaei, Abdollah Amirkhani Shahraki, and Shahriar B Shokouhi. A review of intelligent predistortion methods for the linearization of rf power amplifiers. In *Computer Applications Technology (ICCAT), 2013 International Conference on*, pages 1–6. IEEE, 2013.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[13] Abhishek Kar. Stock prediction using artificial neural networks. *Dept. of Computer Science and Engineering, IIT Kanpur*, 1990.

[14] Rókus Arnold and Póth Miklós. Character recognition using neural networks. In *Computational Intelligence and Informatics (CINTI), 2010 11th International Symposium on*, pages 311–314. IEEE, 2010.

[15] WF Leung, SH Leung, WH Lau, and Andrew Luk. Fingerprint recognition using neural network. In *Neural Networks for Signal Processing [1991]., Proceedings of the 1991 IEEE Workshop*, pages 226–235. IEEE, 1991.

[16] SM Abdel-Moetty. Traveling salesman problem using neural network techniques. In *Informatics and Systems (INFOS), 2010 The 7th International Conference on*, pages 1–6. IEEE, 2010.

[17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[19] M Ibukahla, J Sombria, Francis Castanie, and Neil J Bershad. Neural networks for modeling nonlinear memoryless communication channels. *IEEE transactions on communications*, 45(7):768–771, 1997.

[20] Nikos Naskas and Y Papananos. Adaptive baseband predistorter for radio frequency power amplifiers based on a multilayer perceptron. In *Electronics, Circuits and Systems, 2002. 9th International Conference on*, volume 3, pages 1107–1110. IEEE, 2002.

[21] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[22] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[23] Eduardo G Lima, Telmo R Cunha, and José C Pedro. A physically meaningful neural network behavioral model for wireless transmitters exhibiting pm–am/pm–pm distortions. *IEEE Transactions on Microwave Theory and Techniques*, 59(12):3512–3521, 2011.

[24] Rf weblab. http://dpdcompetition.com/rfweblab/. Accessed: 20-07-2017.

[25] Rf weblab - measurement setup. http://dpdcompetition.com/rfweblab/measurement-setup/. Accessed: 20-07-2017.

[26] Rf weblab - access details. http://dpdcompetition.com/rfweblab/access-details/. Accessed: 20-07-2017.