# RF direct and inverse modeling for radioastronomy using neural networks

B. Censier

October 17, 2019

## Contents

**1 Introduction** **1**
1.1 Neural networks . . . 2
1.2 Motivation for neural networks . . . 2
1.3 Limitations of neural networks . . . 2
1.4 State of the art . . . 3

**2 Application to S-AAIR electronics** **3**
2.1 Data . . . 3
2.2 Note on data normalization . . . 4
2.3 Neural network implementation . . . 6
2.4 Direct modeling . . . 6
2.4.1 Network inputs/outputs . . . 6
2.4.2 Network topology optimization . . . 6
2.4.3 Training . . . 7
2.4.4 Results . . . 7
2.5 Inverse modeling . . . 9
2.5.1 The non uniqueness problem . . . 9
2.5.2 Hybrid method . . . 11
2.5.3 I/O and Network topology . . . 11
2.5.4 Results . . . 12

**3 Conclusions** **12**

**4 Perspectives** **12**

**Abstract**

Direct and inverse modeling with neural networks of a complex front end electronics for the S-AAIR project are evaluated. While direct model outputs S-parameters based on 4 electronic inputs parameters, inverse modeling has to output 4 optimal electronic parameters with respect to 5 S-parameters and noise constraints inputs. The advantages to be expected from neural netwoks are execution speed without loss of accuracy, and the possibility of modeling a system based on measurements only. This is a crucial step for designing a fully adaptive front-end electronics, with a single electronic board capable of adapting to any kind of observation constraints like signal over noise, bandpass, dynamic range, power consumption, and several other possible features. In this first study, simulations data are used together with a simple python library for neural networks to evaluate the performances of such methods. Direct model is shown to have between 0.1 and 0.5 dB RMS errors, which sufficiently accurate but could be easily improved given the simplicity of the network used so far. Inverse modeling is realized without any errors on a multi layer network, using a dedicated hybrid method. Those first tests confirm neural networks are valid tools for complex RF electronic modeling, with several perpectives for improving and/or extending the results.

## 1 Introduction

Following the CSD recommandation on artifical intelligence applied to radioastronomy intrumentation, and within the framework of the S-AAIR project that aims at building an adaptive electronics front end for radioastronomy, a robust, accurate and fast modeling method for complex electronic systems is needed. The complexity of current RF front ends is indeed such that simulation are growing more and more heavy on resources for both memory and computation time. On the other hand measurements, when they are possible, may fully characterize a system but needs to be put under a

functional and fast form, a modeling that could allow to dramatically improve the time needed for optimizing a set of parameters with respect to some user defined requirements.

Roughly speaking, such an adaptive electronics would take the form of a complex RF electronic board associating several non linear RF components. That system should be able to adapt its specifications (gain, return loss, noise level, energy consumption...) dynamically, i.e. find the best way to fit the performances that a user needs for a given observation. It will have Nin parameters as inputs, representing Nin different submodules settings like impedance, voltage or current values. Those input parameters will allow to tune the chosen specifications. We may thus represent the system as a Nin-input/Nout-output model, where outputs are selected performance criteria.

In the following, we'll deal with 2 types of modeling:

- *direct modeling*, where a model reproduces the output vs. input relationship, i.e. predict the performances of the system for any given set of input electronics parameters
- *inverse modeling*, where a model reproduces the input vs. output relationship, i.e. predict the electronics parameters that will optimize the performances of the system with respect to user given constraints.

## 1.1 Neural networks

In short, artificial neural networks are a simplified modeling of how biological neurons work. An artifical neuron simply takes N inputs, sum them while punderating them with a set of weights, and pass the result in a given thresholding function (the "activation function"). An artificial neural network is a set of interconnected neurons, with one input layer, one output layer, and one or more hidden layers of neurons.

The neural network is set to follow a measured and/or analytical and/or simulated and/or effective model based on a training step. During this training step, several examples drawn from measured or simulated data are presented to the network. For each pair of input/output in the training set, the network compares its output with the desired output, and modify the weights in order to minimize a given loss function related to the error between computed and desired output. This training step thus involves optimization methods, here we'll use so called feedforward network architecture, with weights being adjusted iteratively during training by a basic gradient-based algorithm.

## 1.2 Motivation for neural networks

Complex RF electronics systems often exhibit highly nonlinear behaviour, without any analytical representation. Their characterization thus often rely on simulations and/or measurements. One may roughly expect on the order of 10% accuracy with simulations, which are rather heavy on computer resources and time consuming. Measurements on the other hand, may reach on the order of 1% accuracy, with effective values without any prior hypothesis on the system, but needs dedicated instrumentation and are also time consuming.

When a modeling of a given electronics system is needed, one can thus use a detailed physical modeling that may be accurate but slow and heavy on resources, or use approximate modeling which will be faster but with a loss of accuracy.

Neural networks offer a way of approximating an unknown function with both high accuracy, fast response, and no priors on the model.

Following the universal approximation theorem (see [1]), about any function may be theoretically approximated with the desired accuracy by a single hidden layer neural network.

The training based modeling involves no prior and allows to agreggate information from various data sources, in time domain, frequency domain, or any parametric form. Here we'll present a preliminary work based on simulations which hold its own priors, but the ultimate goal is to use measured data so that an effective modeling based on measurement only is achieved.

The basic goal is to build a fast direct model, allowing to sum up all the informations about the system behaviour in a function. This function may in turn then be used as a fast block model in RF simulations, massively speeding up the process. If trained on measured data, this offers the possibility to do a kind of snapshot of the electronic behaviour based on measurements, and then use that effective model as a submodule in higher level simulations.

Concerning inverse modeling, there are 2 main possibilities:

- use the direct model in a traditional optimization algorithm, a fast loss function being computed based on the network response, allowing to efficiently minimize it and find the electronics parameter best fitting the desired specifications.
- directly train the neural network so that it takes specifications as inputs, and ouputs optimal electronics parameters with respect to those specifications.

## 1.3 Limitations of neural networks

As already stated, neural network may approximate about any function. However this is in practice limited by so called *learnability*: the theoretical possibility does not assure that a learning algorithm exist and/or that the needed number of neurons is not too large to be computed.

Another possible drawback is "overfitting". While a network may reproduce a sample of data used during training accurately, it may also lead to large errors on another set of data from the same system and the same input parameter space that were not used during training. One may use the aalogy of using a high degree polynomial for fitting a set

of points following a linear trend: while the resulting function may exactly pass through every points of the set, it may oscilates wildly between them, failing to reproduce the linear trend. This effect is monitored by systematically splitting the data in 2 sets: the training set used for training, and the validation or test set used to evaluate the error between neural outputs and data. The validation set error should follow closely the training set error during training, and if the validation set error starts to increase, it means the learning has to be stopped because of some overfitting.

Last but not least, neural networks are very powerful and adaptive tools, but they have a complex behaviour and are out of the traditional engineering method: building more or less from scratch, assembling parts and testing each step so that some specifications are met. Here neural networks are mostly black box: there is no manual tweaking of the weights, no design of a given part based on fundamental principles, and the specifications lies in the data themselves, so they don"t have to be explicit. Working with neural network is thus involving lots of tweaking and is mostly based on trial/error:

> "Unlike statistics, machine learning tends to deal with large, complex datasets (such as a dataset of millions of images, each consisting of tens of thousands of pixels) for which classical statistical analysis such as Bayesian analysis would be impractical. As a result, machine learning, and especially deep learning, exhibits comparatively little mathematical theory—maybe too little—and is engineering oriented. It's a hands-on discipline in which ideas are proven empirically more often than theoretically.[2]"

As a rule of thumb, a network with few neurons may lack accuracy reproducing the training data, while increasing the number of neurons allows to increase this accuracy. However increasing the number of neurons is equivalent to increasing the number of internal parameters to be optimized during learning, hence the increase of accuracy, but at the risk of overfitting the data.

### 1.4 State of the art

While neural networks have been known for several decades, their intensive use really began around the beginning of the $21^{st}$ century. Introduction of several methods including backpropagation [3], multilayers network [4] (so called *deep learning*) or convolutional networks [5], associated to a rise in hardware performances and accessible data for training.

It has then been applied succesfuly to several real-world problems that classical methods were unable to solve, and/or with better performances. Concerning RF electronics and electromagnetic modeling, several applications has been studied including direct modeling of RF filters, discrete components or complex systems, EM-based modeling, hybrid approach gathering simulation, analytical and effective modeling with measurements, or inverse modeling based on several methods. This domain shows a continuous effort since the the 90's, following the progresses and new methods introduced in the field of neural network since then. Some reviews and more detailed studies may be found e.g. in [6], [7], [8] [9], [11], and several applications deal with the associated problem of antenna modeling [10].

In [6], the main benefits that may be expected from neural networks are presented:

> "[...] At the circuit level, the neural network speeds up optimization by replacing repeated circuit simulations. This method is faster than direct optimization of original device and circuit models. Compared to existing polynomial or table look-up models used in analysis and optimization, the proposed approach has the capability to handle high-dimensional and highly nonlinear problems."

## 2 Application to S-AAIR electronics

### 2.1 Data

The system under study is a one stage amplifier described by 4 input parameters (x1, x2, x3, x4 in the following). Those 4 input parameters are impedances and active sources settings, they are each set by a value coded on 4 bits, allowing 16 different values (0-15). The output specifications corresponding to each set of the 4 inputs have been simulated on the electronic simulation software ADS. Those outputs includes the 4 scattering parameters S11, S12, S21 and S22 vs. frequency in 12 frequency channels between 400 and 1500 MHz, and the noise parameters.

The first step consists in evaluating the nature of the data, restrict them to a meaningful domain for training and analyze its main characteristics.

We first note the a big part of the data shows output specifications that are not compatible with a practical use. This is essentially less than unity gain data (S21 $<$ 0 dB) and high input return loss (S11 $>$ -10 dB) That part of the data will be of no practical use, and it introduces large variations of some specifications that would complicates the modeling for no benefits.

We thus restrict the data to S21 $>$ 0 dB, S11 $<$ -10 dB, and S22 $<$ -10 dB. Input parameters x1, x2, x3 and x4 each vary from 0 to 15 on the whole simulated range, which gives $16^4 = 65536$ simulated points. The restriction used corresponds to $6 <$ x1 $< 16$, $1 <$ x2 $< 15$, $5 <$ x3 $< 16$, and $3 <$ x4 $< 16$, which is 14040 data points. The rough ditribution of S21 and S11 values before and after selection can be seen on figure 1. The selected domain in the 4-D space is the smallest 4D cube containing all the data satisfying the restriction rules presented above. Some selected values are thus outside of those restrictions (for exemple some S11 values are more than -10 dB), which just means we will train the network on an input parameter space slightly larger than needed.

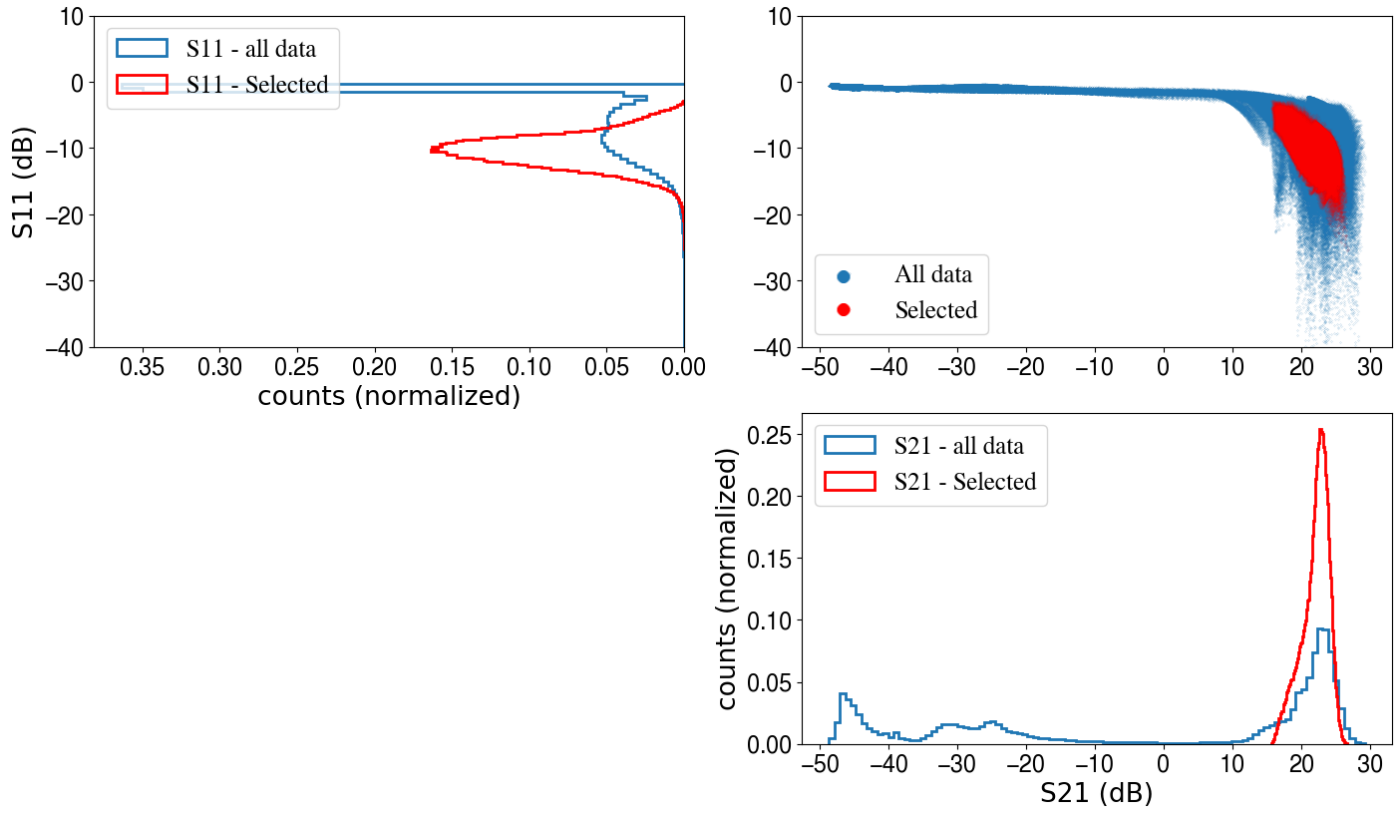


Figure 1: S11 vs. S21 scatter plot (top right panel) for all data points including all frequency channels, before and after selection. Upper left and lower right panels shows histogram of resp. S11 and S21 values.

A 4 dimensional space parameters is not easy to visualize, however some statistical parameters may reveal some interesting features. Computing the gradient of outputs versus inputs, it's clear that S11 has the largest variations, followed by S21, with S22 and S12 showing a rather smooth behaviour with respect to the 4 input parameters.

S11 data looks quite different than the 3 other S-parameters, with respect to large variations and a sometime noise-like behaviour with respect to the $2^{nd}$ input parameters (see figures 2 and 3). That noisy behaviour may limit the reachable error between simulated data and neural output to about 0.5 dB on S11, depending on the number of neurons. A relatively small network will lead to something similar to a non-linear regression on noisy data, smoothing the relationship between S11 and x2, hence the 0.5 dB limitation. On the other hand, if better accuracy is needed, a more complex network could be used in order for the neural output to reproduce the exact simulated behaviour.

## 2.2 Note on data normalization

Neural network input parameters normalization is generally needed, depending on the data and on the activation function used. Most activation functions takes a limited values range as input, most often [-1,1] or [0,1]. Here we use the sigmoid activation function, that calls for a normalization between 0 and 1.

The output data may also be normalized. While not mandatory, this can significantly ease the training process and make it faster. The goal here is to normalize the output data so that each output neurons will have to model variations that are not too different from each other. To illustrate further with an extreme example, if a network has 2 output neurons, and the first neuron has to model data varying between 0 and 0.1, while the other is modeling data varying between $10^{33}$ and $10^{34}$, the optimization process setting the weights through the network will have much difficulty converging. On the other hand normalizing the data so that both outputs are between 0 and 1 will substantially improve the performances of the training.

Both input and output data used for training and validation are thus normalized following the scheme:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{1}$$

Where $x_{norm}$ is the normalized version of $x$, and $x_{min}$ and $x_{max}$ are respectively the minimum and maximum values for $x$.

For scattering parameters output data, each output neurons will represent a given frequency channel for one of the 4 s-parameters (see section 2.4.1). Each of those outputs are normalized using minimum and maximum values over all possible combination of the 4 input parameters, and with different min/max values for each frequency channel.

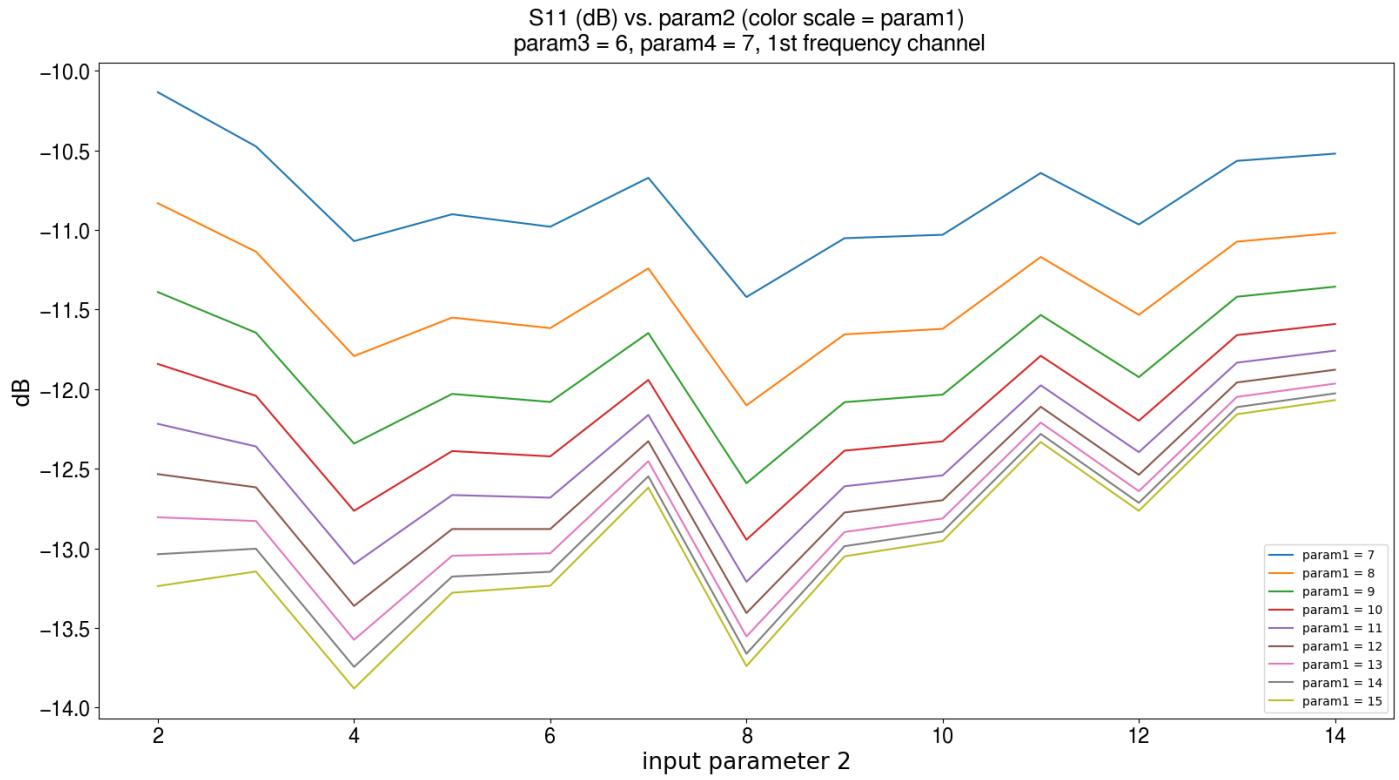


Figure 2: S11 vs. x2, for several values of x1 (color coded, see legend). The noise like behaviour may be due to unresolved oscillations. It is characterized by a $\approx 0.5$ dB dispersion and may thus limit the accuracy of modeling.

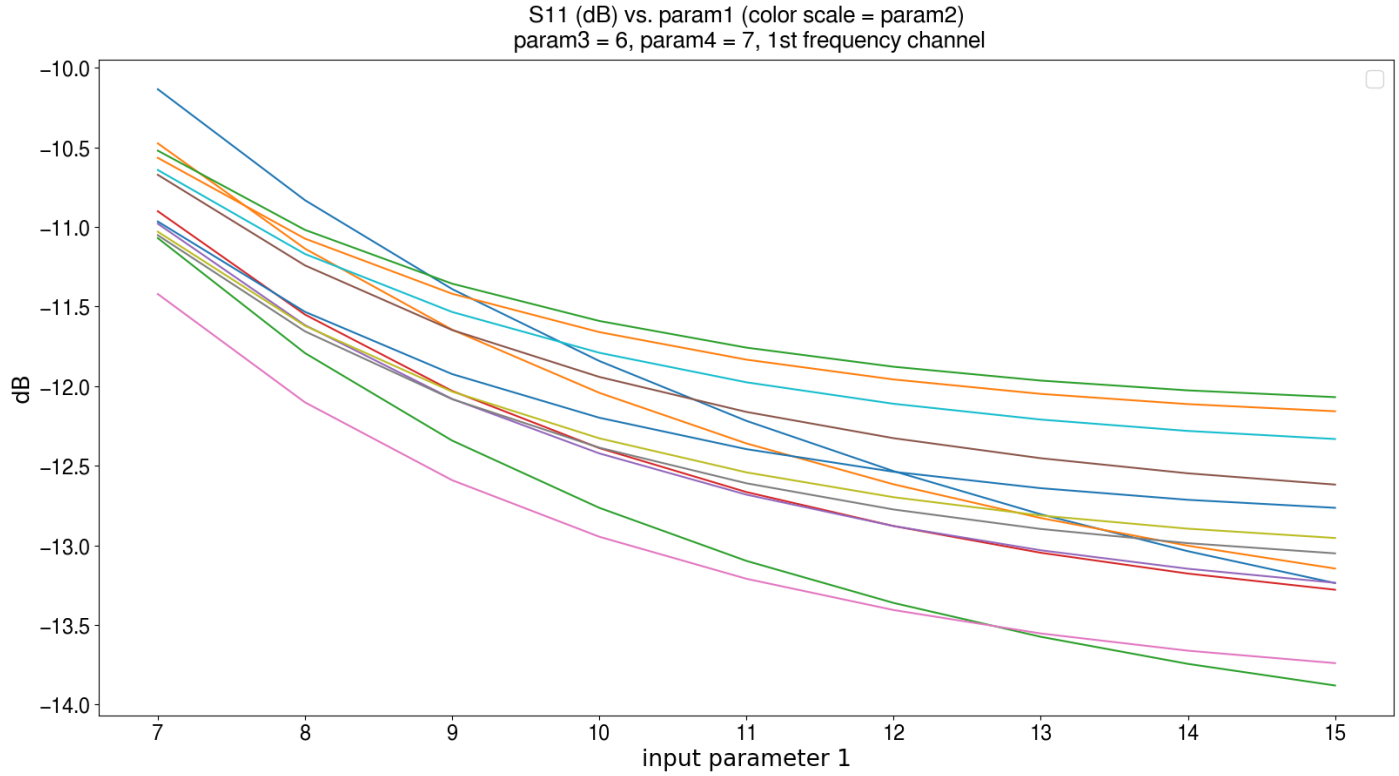


Figure 3: S11 vs. x1, for several values of x2 (color coded, see legend). To be compared to figure 2, here the curves stay smooth without the noise-like behaviour seen w.r.t. x2.

## 2.3 Neural network implementation

We'll use feedforward networks with sigmoid as activation function, and a basic backpropagation algorithm for training. Every neurons layer is fully connected to the previous and next layer, meaning that each neurons from a layer has a connection and a correponding weight to any neurons of a contiguous layer.

The implementation is done with the *Pybrain* python library. While quite confidential, limited in the possible settings, and not maintained anymore, that library is fairly easy to use and appears as an apdapted tool for a first study. In later works, it is planned to use one the most used framework for deeplearning (i.e. roughly speaking neural network with several hidden layers), which is *Keras*, using the well known *Tensorflow* backend. That framework will allow to go into more details and testing about activation function, big sized networks, and training methods.

## 2.4 Direct modeling

### 2.4.1 Network inputs/outputs

For a sketch of the neural network structure used for direct modeling, see figure 4.

The network used for direct modeling has 4 input neurons corresponding to the 4 input parameters x1,x2,x3,x4 of the model.

The outputs are the 4 scattering parameters as a function of frequency channel. That's 4 times 12 frequency channels, i.e. 48 output neurons. If SXX represents the 12-vector of one of the 4 S-parameters vs. frequency, the output may be seen as a simple flattening of the concatenated vector [S11, S12, S21, S22]

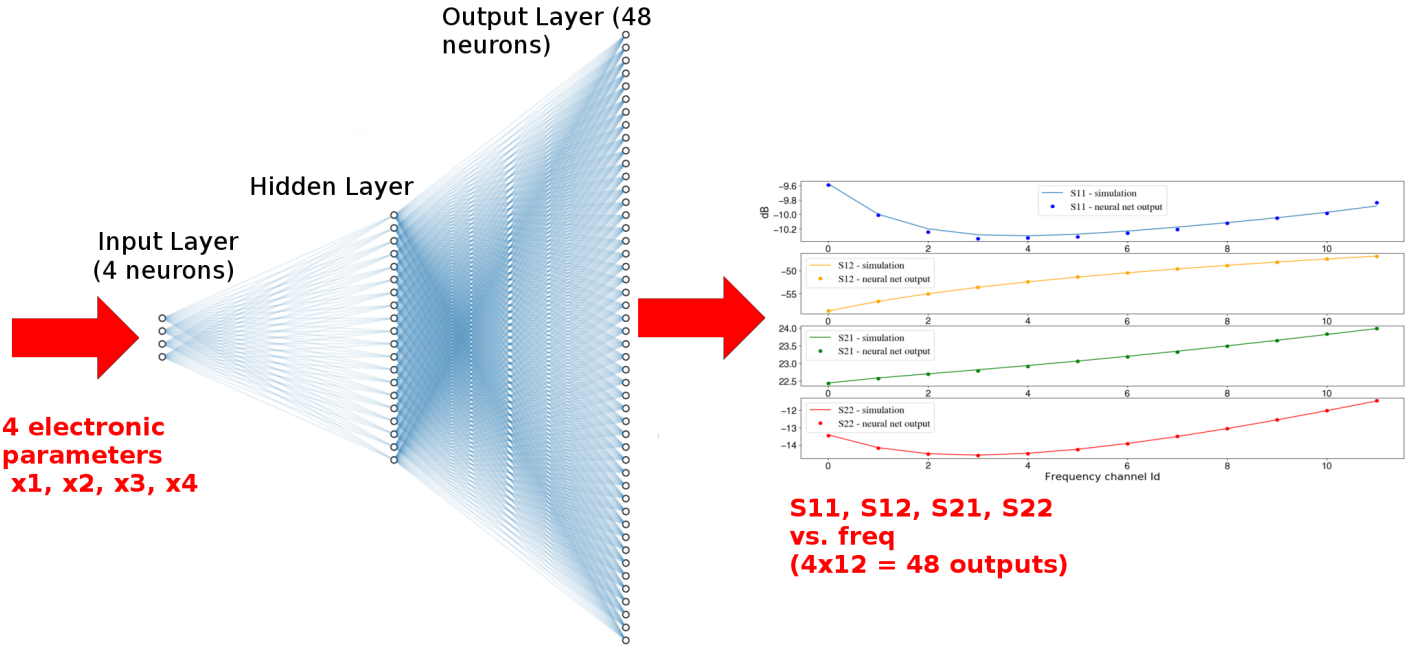


Figure 4: Sketch of the neural network used for direct modeling, with one input layer (4 neurons for 4 input parameters), one hidden layer (here 20 neurons) and one output layer (48 neurons for the 4 S-parameters in 12 frequency channels). Each layer is "fully connected", meaning that each neurons of one layer is linked to every neurons of an adjacent layer.

### 2.4.2 Network topology optimization

As already stated, the method for designing an optimal neural network is often based on trial/errors. In particular, for a given problem a given number of neurons can be optimal while another close number is sub-optimal or even bad with respect to accuracy.

For this direct modeling, we will restrict to one-hidden layer network that should be sufficient for a first estimation.

The error between simulated data and neural outputs have been measured for several different numbers of neurons in the hidden layers, ranging from 5 neurons to 30 neurons, with a test at 500 neurons to evaluate the possible benefits of a large number. The error monitoring during the training process may be seen in figure 5, showing the variety of complex behaviours that are possible during this optimization step. The error is more or less noisy, it usually is obviously overall monotonic and decreasing, but local patterns may be complex (momentary increase and then decrease, sudden discontinuous decrease, steady increase and or very noisy behaviour after a long training..)

The resulting error after 1000 training cycles can be seen in figure 6, as a function of the number of neurons in the hidden layer. There is obviously an optimum around 20 neurons, and we'll use that number in the following. One can note the relatively low progress made between 30 and 500 neurons. It seems to indicate that adding neurons is not useful above the optimal point. A few tests with several hidden layers were made, but no significant benefits have been noted neither.

More precisely, while the universal approximation theorem (see section 1.2) implies that more neurons theoretically means more precision, the desired precision may need:

- an absurdly large number of neurons given the available computer resources
- an adapted learning scheme, including the nature of training data and choice of an efficient optimization method for the learning.

Those 2 points may thus be overcome in later works by the use of another framework, allowing more detailed settings of the neural network and training process parameters (see section 2.3)

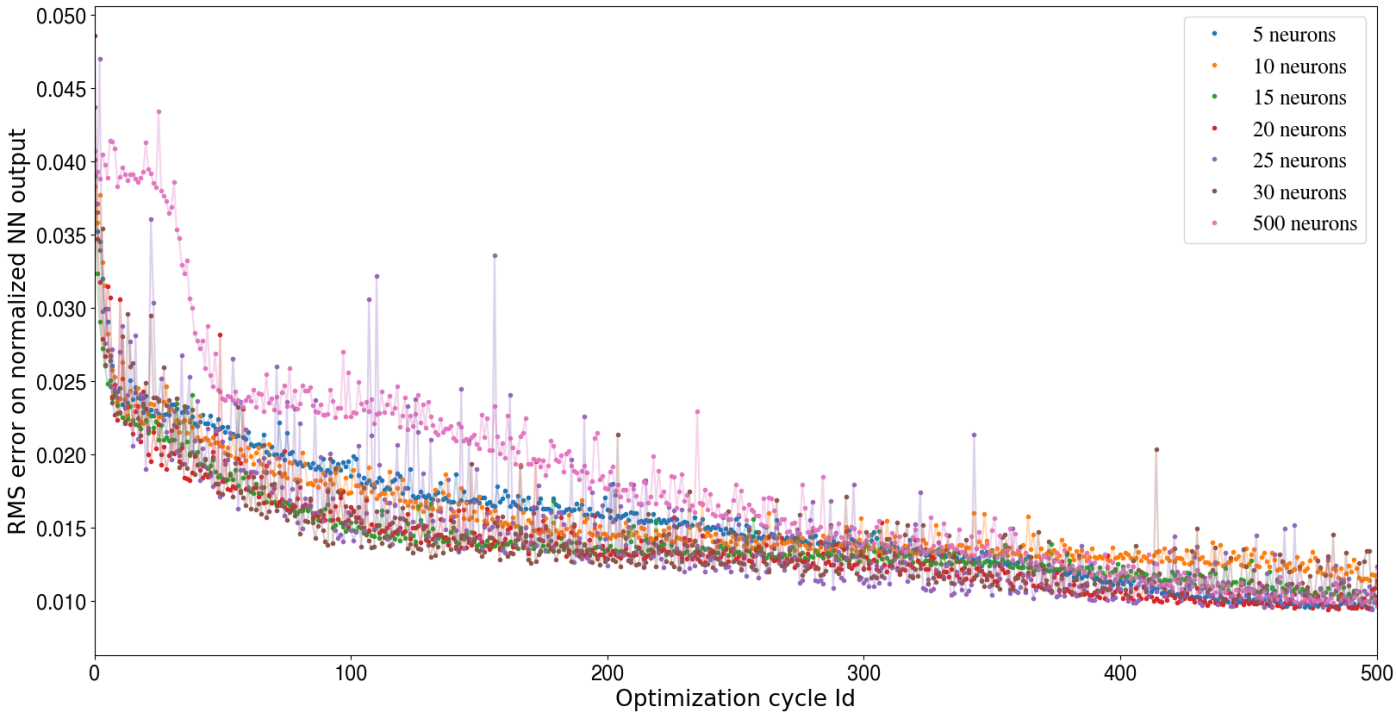


Figure 5: RMS error during training, for one hidden layer with 5, 10, 15, 20, 25, 30 and 500 neurons (color coded, see legend).That error is evaluated on normalized outputs and must be renormalized to represent a dB error.

### 2.4.3 Training

The training reaches a minimum error after some hundreds of cycles, it is nevertheless systematically performed over 1000 cycles but the accuracy gain above the $\approx 500^{th}$ cycle is small. See figure 7, the error on validation data set closely follows the error on training data set, only 4% above it at worst. The difference bewtween both dB errors is thus at the 0.01 dB scale and overfitting can be considered as negligible.

### 2.4.4 Results

Performances of neural network have to be carefuly analyzed, since output errors can exhibit a very complex statistical behaviour. In particular, it's possible to fit perfectly a model for 99% of the parameters space, but show large errors in the remaining 1%. In term of probability distribution, the error could look gaussian, but with more extreme events than a gaussian.

For each S-parameters, the min/max value are thus given in addition to a RMS and a median error. The difference between RMS and median error is a good indicator for the presence of rare but important errors.

Error distribution for the 4 S-parameters can be seen in figure 8, some numbers characterizing it are presented in table 1.

| (dB) | min error | max error | RMS | median |
|---|---|---|---|---|
| S11 | $2.10^{-7}$ | 3.92 | 0.51 | 0.26 |
| S12 | $1.4.10^{-6}$ | 0.43 | 0.08 | 0.05 |
| S21 | $1.1.10^{-6}$ | 0.63 | 0.1 | 0.05 |
| S22 | $3.8.10^{-7}$ | 0.93 | 0.06 | 0.04 |

Table 1: Errors between simulation data and neural network output in dB. The median error is a RMS error computed with a median instead of a classical average.

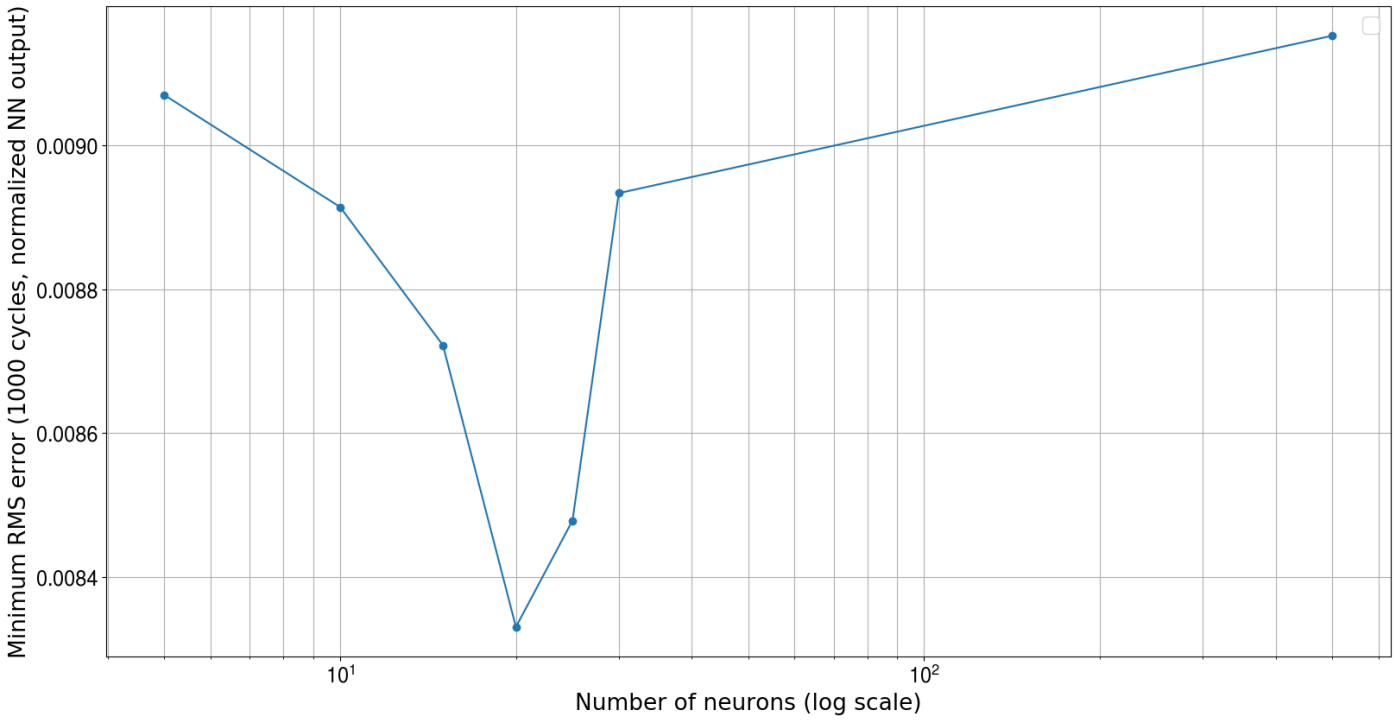


Figure 6: RMS error between simulated data and neural network outputs after 1000 training cycles. That error is evaluated on normalized outputs and must be renormalized to represent a dB error.

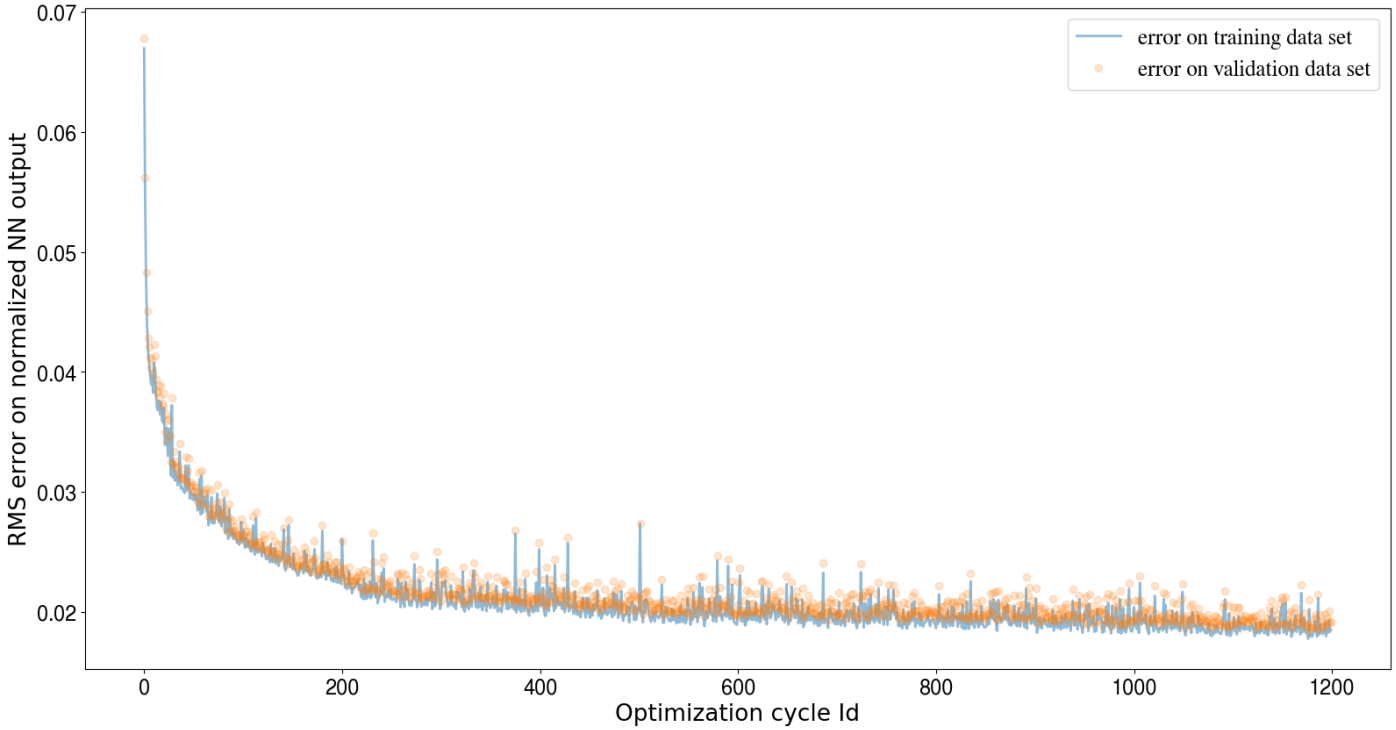


Figure 7: RMS error on training and validation data set during training of a 20 neurons/1 hidden layer network. The error on validation data is 4% above the error on training data at worst, the overfitting is thus negligible.

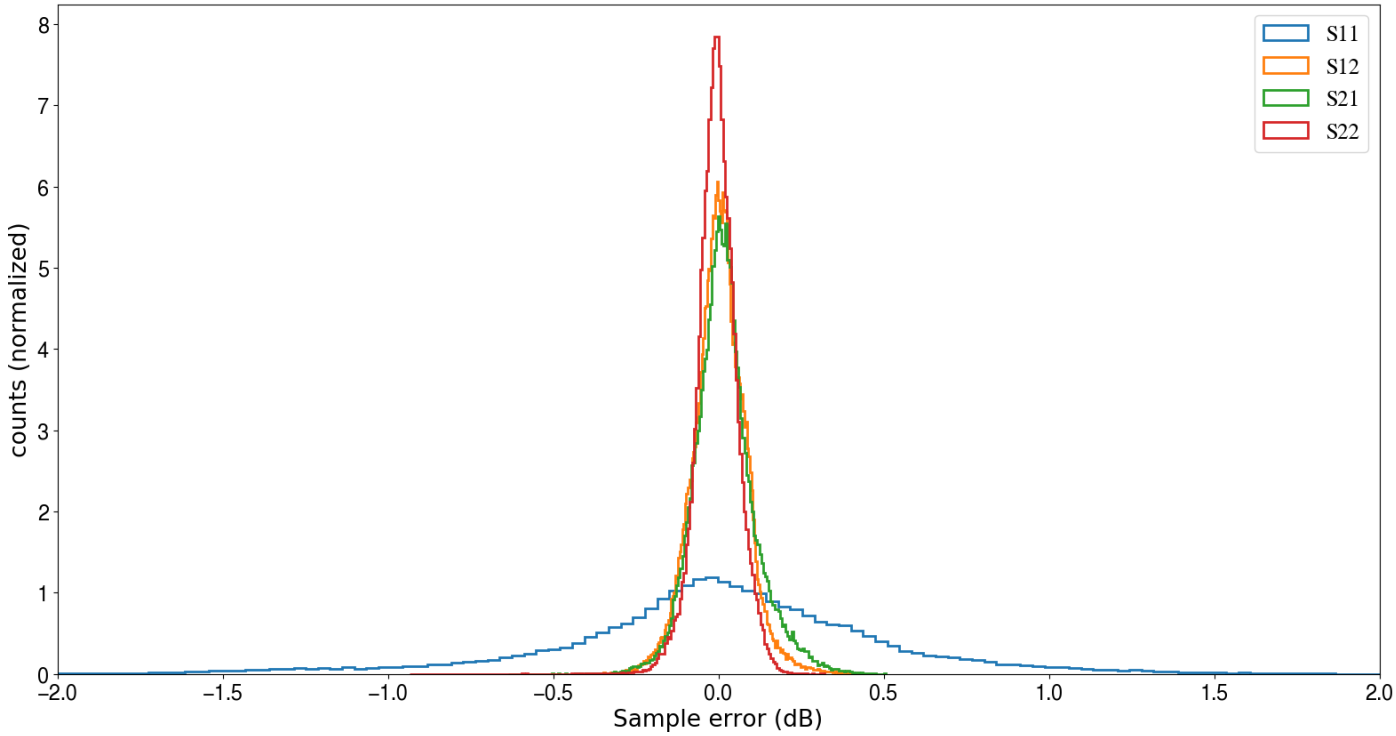


Figure 8: Histogram of errors (simulation-model) on all validation data, for the 4 S-parameters

They show an overal average error of 0.1 dB (representing a 1% error on linearly scaled data) or less for S21, S12 and S22, 0.5 dB (about 6%) for S11. As expected from the training data statistics (see section 2.1), S11 errors are above the 3 other S-parameters due to the neural output smoothing the noisy-like behaviour of S11 vs. x2. Given the different source of errors from such a simulation, 0.5 dB is a good performance, and 0.1 dB is compatible with the precision expected from measurements as training data.

Minimum errors are very low, while some maximum errors are close or above the dB level. The systematic difference between RMS and median errors indicate there are some quite rare data samples that show errors above the gaussian level.

There are 25% of all the S11 samples that shows an error above 0.5 dB (which would be 31% for a gaussian distribution above $1\sigma$), 7% above 1 dB (4.5% for a gaussian above $2\sigma$), 1.5% above 1.5 dB (0.2% for a gaussian above $3\sigma$ level)

The noisy-like behaviour is stronger at highest frequencies, as illustrated in figures 9 and 10. We see the error distribution has more outliers at high frequencies, and the 0.5 dB figure for S11 is an average one: RMS error vary monotically from $\approx$ 0.3 dB to 0.7 dB w.r.t. frequency.

On S21, we have 24% of errors above $1\sigma$ (i.e. 0.1 dB), 5% above $2\sigma$ (0.2dB) and 0.9% above $3\sigma$ (0.3 dB).

It should be noted that this one-layer version with only 20 neurons is holding 1060 free parameters, the neural networks net to be optimized during training. In reference, a crude linear interpolation of the simulation data would require one parameter per data point in the S(x1,x2,x3,x4,f) space, which makes a total number of 673920 points. The resulting neural network can thus be seen as a model "compressing" the raw simulation data to a given accuracy. It should be easily improved with a bigger network and adapted training methods.

Under its current (unoptimized) form, the network takes about one microsecond on an average laptop to output the 4 S-parameters corresponding to a given x1,x2,x3,x4 input.

## 2.5 Inverse modeling

### 2.5.1 The non uniqueness problem

If direct modeling is finding the function f that describes $y = f(x)$, with x as input and y as output, inverse modeling is finding a function g that describes $x = g(y)$.

The situation is nevertheless very different for inverse modeling, practically speaking there is often several x values as input that will lead to the same value y at output. In this case, g can not be put under functionnal form: this is the non uniqueness problem, and it's the major problem arising in inverse modeling.

A naive method would be to train a neural network with y values as inputs, and x values as outputs, so that the network learn to reproduce the function $g$. But if g is not bijective (several possible x for one y value), the training will not converge to acceptable errors.

In the case of direct modeling $y = f(x)$, several y output values for one x input value is most generally simply due to noise, and the network will act as an efficient noise averaging filter. In the case of inverse modeling, multiple values are not due to noise but an effective complex behaviour of the system under study. The network will compute some kind of average over all possible x values, distorting the information we want to access. Some more advanced possibilities using

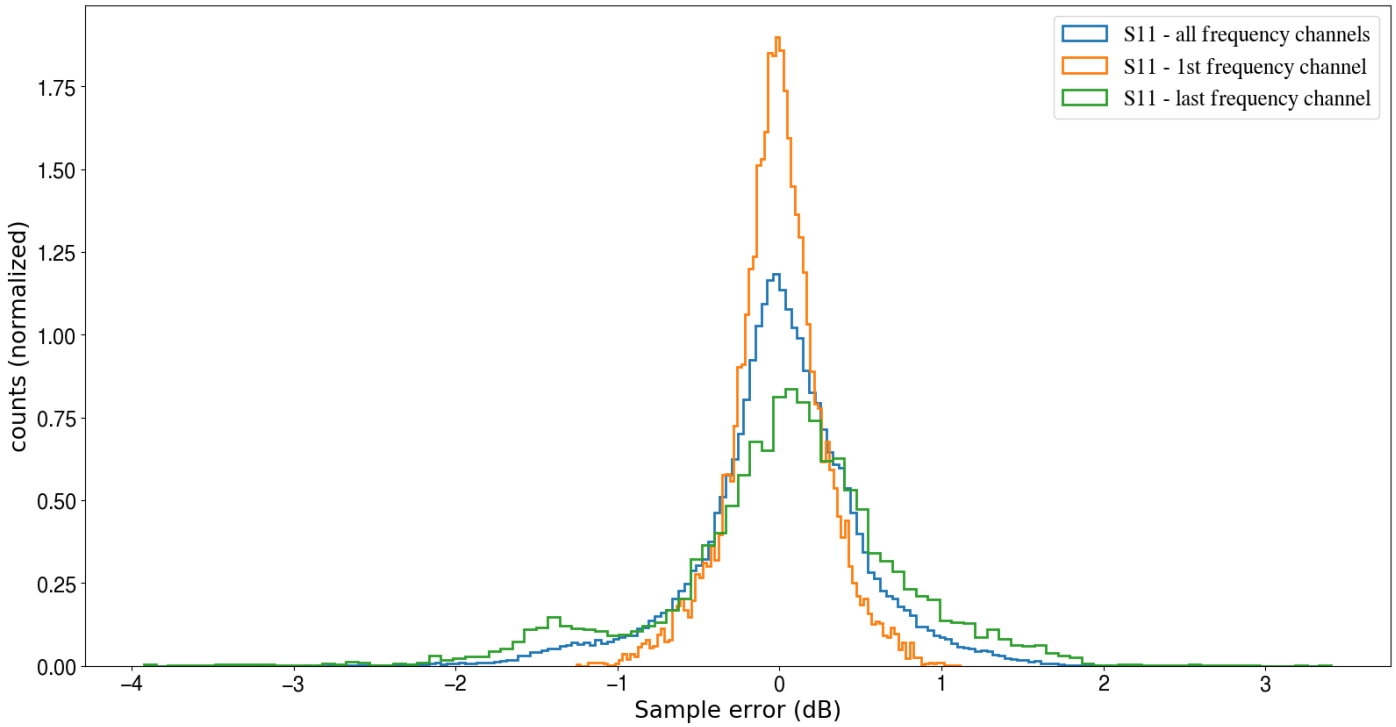


Figure 9: Histogram of S11 errors (model-simulation), for the first and the last frequency channel.

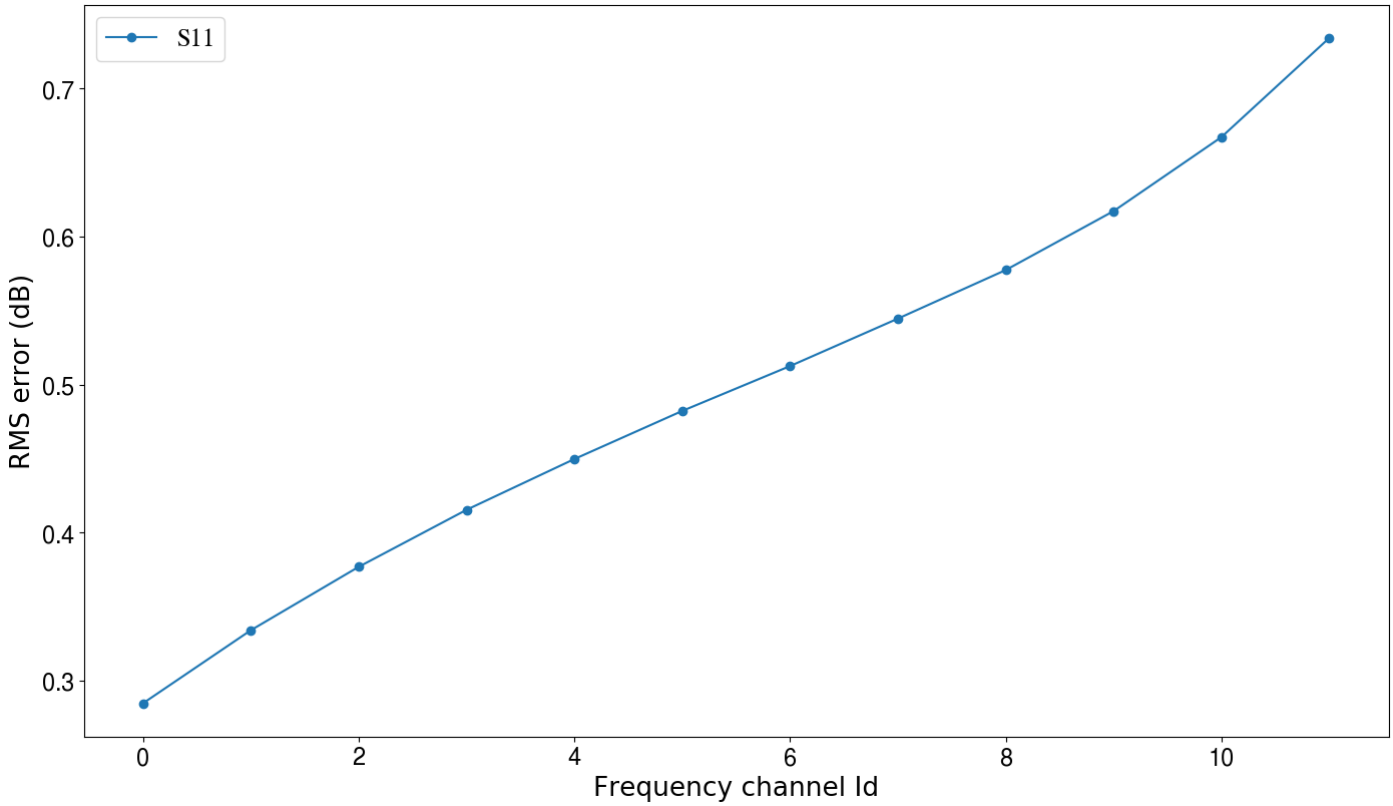


Figure 10: RMS error on S11 vs. frequency channel.

statistics tools and dedicated types of network exist (see e.g. [11] and [12], here we will stick to more basic methods for a first evaluation.

### 2.5.2 Hybrid method

The naive training method mentioned in previous section (simply inversing inputs and outputs during training) has been tried, and as expected the error did not converge to sufficiently low levels. The non-uniqueness problem must be at play here (see section 2.5.1), preventing the convergence. In addition, we don't need a system giving the 4 x1, x2, x3, x4 parameters that will match exactly some given S vs frequency relationship, we rather want to put *constraints* on desired specifications, like (S11 < -10 dB & S22 < -10 dB & S21 > 20 dB).

A dedicated method has thus been created, with a first step consisting in creating an optimization function, and a second step where a neural network will be trained to follow that function.

We want constraints on maximum allowed return losses (S11 and S22 < some maximum values), desired amplfier gain and flatness of this gain w.r.t. frequency ($\Delta$ S21 < some maximum values, and S21 close to a given goal S210), and maximum allowed noise (Noise figure NF < some maximum value NFmax).

For each set of constraints, we have to find the x1, x2, x3, x4 set of electronic parameters that will best achieve those constraints.

The classical method used in optimization is the creation of a loss function that is minimum when the constraints are best met. The search for optimal settings is thus equivalent to searching for a minimum in the loss function.

Since we have access to all the possible output values, we can compute this loss function for any set of input parameters and any set of constraints.

The loss function is defined as follow:

$$10 \times \delta S22max + 1000 \times \delta S11max + 10 \times \delta S21 + 100 \times \delta NFmax \tag{2}$$

Where:

- $\delta S22$ is the squared sum of differences between S22(x1,x2,x3,x4) and the constraint S22max over all frequency channels. This corresponds to the constraint $S22 < S22max$, where S22max is an input specification.
- $\delta S11$ is the squared sum of differences between S11(x1,x2,x3,x4) and the constraint S11max over all frequency channels. This corresponds to the constraint $S11 < S11max$, where S11max is an input specification.
- $\delta S21$ is the squared sum of differences between S21(x1,x2,x3,x4) and the desired gain S210. Only differences greater than the input specifications $\Delta$ S21max are considered. This corresponds to the constraint $S21 = S210 \pm \Delta S21max$, where S210 and $\Delta$ S21max are input specifications.
- $\delta NFmax$ is the squared sum of differences between NF(x1,x2,x3,x4) and the constraint on noise figure NFmax. This corresponds to the constraint $NF < NFmax$, where Nfmax is an input specification.

The sum of all these terms adds up the different constraints together like a logical and, and each of those constraints are punderated by a value in order to prioritize those constraints (here for example the constraint on S11, with a punderation of 1000, is given more importance than the constraint on S22 with a punderation of 10).

The method for generating training data is the following:

- set a grid over the 5-D space covering input specifications S11max, S22max, $\Delta$ S21max, S210 and NFmax
- for each point of that grid, compute the loss function over the whole x1,x2,x3,x4 range
- find the minimum value, and the corresponding $x1_{opt}$,$x2_{opt}$,$x3_{opt}$,$x4_{opt}$ parameters that are optimal with respect to the given constraints.

A training set composed of S11max, S22max, $\Delta$ S21max, S210 and NFmax values as inputs, and $x1_{opt}$,$x2_{opt}$,$x3_{opt}$,$x4_{opt}$ values as outputs is then built.

### 2.5.3 I/O and Network topology

We have 5 input neurons corresponding to 5 constraints on S-parameters and noise, 4 outputs corresponding to optimal electronic parameters $x1_{opt}$,$x2_{opt}$,$x3_{opt}$,$x4_{opt}$ (see figure 11).

Here the 4 outputs will more or less vary in the same [0,15] value range, hence output normalization is not mandatory (see section 2.2).

Each of these 4 outputs is a real number, while the simulated electronic parameters are integers coded on 4 bits.

The outputs are simply rounded to the closest value, thus an output error of less than 0.5 will give the correct output after rounding.

For single hidden layer networks that have been tested, the error during training did not converge to sufficiently low values. Adding layers is systematically improving the accuracy, sufficient performances have been reached with 4 layers of 30 neurons (see figure 4).

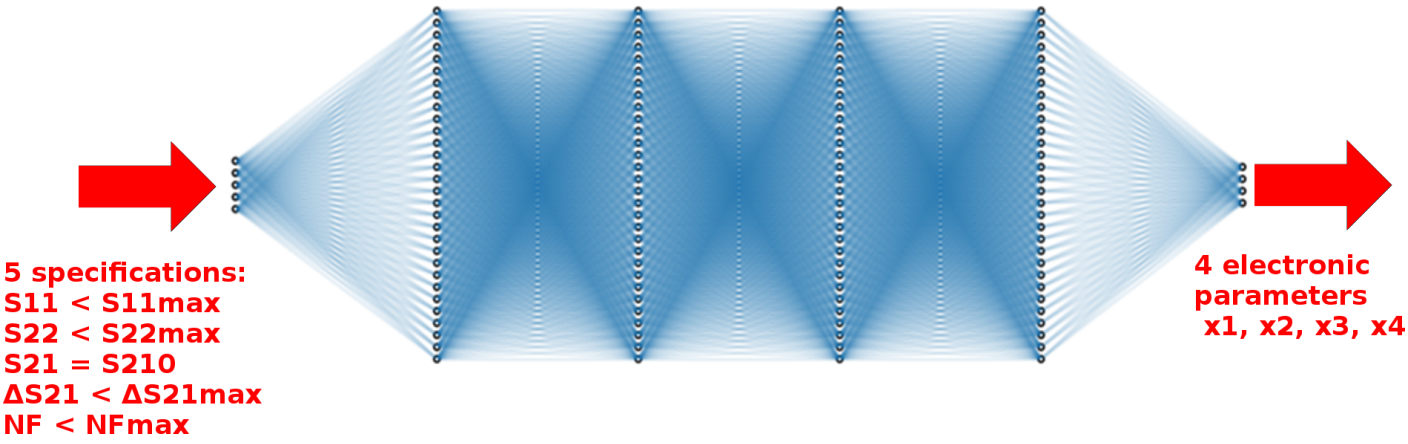


Figure 11: Sketch of the neural network used for inverse modeling, with one input layer (5 neurons for 5 constraints on maximum return loss (S11max and S22max), desired gain value and maximum variation in the frequency domain (S210 and $\Delta$ S21max), and maximum noise (NFmax)), four hidden layers (30 neurons each) and one output layer (4 neurons to output electronic parameters x1, x2, x3 and x4).

### 2.5.4 Results

A dataset is computed for input values -20 dB $<$ S11max, S22max $<$ -10 dB, 0 dB $<$ $\Delta$ S21max $<$ 5 dB, 15 dB $<$ S210 $<$ 30 dB and 0 dB $<$ NF $<$ 6 dB, on a 5x5x5x5x5 = 3125 points grid.

10 % of those data are used for validation, the other 90% for training.

The accuracy reaches a sufficient level after around 300 to 400 cycles during training.

Once trained, the rounded output values of the neural network reproduces exactly the validation data with no errors.

The un-rounded outputs compared to theoretical integer outputs in the validation data show a RMS error of only 0.03, with a maximum error of 0.24.

## 3 Conclusions

Direct and inverse modeling of a complex electronics for the S-AAIR project has been achieved with neural networks.

Concerning direct modeling, a single hidden layer network with 20 neurons seems near optimal, with residual errors on the order of 0.1 dB RMS for S21, S22 and S12, 0.5 dB for S11. The bigger error on S11 is explained by noise-like behaviour in the simulated data, the network acting as a filter w.r.t. to that noise-like feature. The error distribution is slightly non gaussian, on S11 some dB scale errors exist on isolated points in the data sample. While this is already compatible with respect to expected uncertainties from simulations, and sufficient for several RF applications, the relatively low neurons number used imply there should be margin for significant improvement. During this study using a basic python library, the increase of neurons number did not improve performances significantly though, a more advanced framework is needed, allowing to finely tune several network parameters like activation function and optimization method used during training.

Inverse modeling has been first tested using a direct but too naive method, which is inversing inputs and outputs during training. While this could be sufficient for some problems, the non uniqueness of electronic parameters solutions for a given set of constraints prevent the convergence during training. This a well known problem in inverse modeling, that can be address using more specific neural networks. For this first study we hold on basic tools, an hybrid method has thus been designed in which a loss function is first computed on a range of parameters, then the minimum optimal value is looked for, so that a training set exempt of the non uniqueness problem is created. Contrary to direct modeling, single layer network is not sufficient and increasing the number of layers systematically improves the performances. We reach sufficient accuracy with a 4 hidden layers/30 neurons each neural network, with no error between validation data and neural outputs after rounding.

## 4 Perspectives

*Improve direct modeling accuracy*:

- use of finely tunable framework like tensorflow, allowing to adapt activation function, optimization algorithm, loss function used during training.. This is particularly important in the case of large number of free parameters to be optimized (meaning large number of neurons links), including multi layers network.

- In particular some more exotic activation functions, and associated learning methods could be relevant, for example wavelets are particularly suited for discontinuities modeling [13] [14]
- In the present study, the optimal network architecture has been done mostly "by hand", with successive trials on a small number of topologies. That process could be automatized and be more efficient. In order to find the optimal, possibly multi layer achitecture, a genetic method seems particularly suited.
- The ultimate error goal would be around 0.1 dB, which is the approximative accuracy expected from measurements. That criterion is already satisfied on the majority of the data.

*Inverse modeling*:
We have roughly 3 types of possible method for implementing inverse modeling:

- direct model used for optimization: a network is trained for direct modeling, then used in a classical optimization loop. One could also use differential modeling, for example train a network to directly model the jacobian function associated to the optimization problem to be solved. See e.g. [6], in which the neural network is used to speed up optimization by replacing repeated circuit simulations. It is shown to be faster and/or more accurate than traditional methods (table lookup, polynomial models) in addition to being able of handling high-dimensional and highly nonlinear problems (as also illustrated by our present study).
- Hybrid methods: the method presented in section 2.5 is in this group. The idea is to first solve the optimization inverse problem to generate a training dataset, then train a neural network to model this dataset.
- inverse modeling based on training only: while the naive approach of simply inversing inputs and outputs during training is not sufficient, there are several other possibilities for training on an inverse problem directly on data. and without the intermediary steps of the hybrid method. That's an interesting possibility since it would allow straightforward and continuous adaptation of the inverse modeling based on measurements (be it during dedicated measurements periods, or even during the telescope operations). Some methods have been reviewed, among them let us cite the possible use of multivalued outputs [11] and invertible neural networks [12]. Both are solving the non uniqueness problem, the first one by using multiple possible output values, the second by estimating a likelihood over possible solutions. They both imply a more dedicated network, with specific methods of training and/or specific network topologies.

*Training data, inputs/outputs*:

- the data used here for training are not covering all the simulated parameters. The direct model could be trained to output noise parameters in addition to S-parameters, allowing a full characterization of the electronic system. Also some additional input electronic parameters could be added. While increasing the input dimensionality and thus the complexity, it could allow to better model the data. It could also mitigate the non uniqueness problem encountered in inverse modeling by solving some degeneracies in the inputs vs. outputs relationship.
- Simulations as well as measurements as a source of training data can be very time consuming. Since neural networks can be seen as universal interpolation systems, fewer training data could be sufficient to reach the same level of accuracy. By using smart methods in the selection of the most important training data (for example iterative methods sampling the parameter space adaptively), the training data volume could be significantly reduced. It would be anyway useful to define the minimum amount of data needed for reaching a given accuracy.

*Network optimization*: there are several possibilities to optimize the speed of neural networks (which are already very much quicker than simulations), as well as reduce their complexity (essentially the number of weights used in a compression process). While these are not limiting factors at the present stage of the study, that could be helpful in the case of real-time implementation of a neural network directly on an electronic board. This could be achieved on several different platforms (GPU, FPGA, or dedicated neural chips)

# References


[1] Kurt Hornik, Maxwell Stinchcombe, Halbert White, *Multilayer feedforward networks are universal approximators*, Neural Networks, Volume 2, Issue 5, 1989, pp. 359-366

[2] Francois Chollet, *Deep learning with Python*, Manning publications (2017)

[3] Robert Hecht-Nielsen, *Theory of the Backpropagation Neural Network*, Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989, Neural Networks for Perception, Academic Press, 1992, Pages 65-93, ISBN 9780127412528

[4] E.M. Johansson , F.U. Dowla, and D.M. Goodman, *Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method*, International Journal of Neural Systems, 291-301, 1991

[5] Y. LeCun, K. Kavukcuoglu and C. Farabet, *Convolutional networks and applications in vision*, 2010 IEEE International Symposium on Circuits and Systems (ISCAS), Paris, 2010, pp. 253-256.

[6] A. H. Zaabab, Qi-Jun Zhang and M. Nakhla, *A neural network modeling approach to circuit optimization and statistical design*, in IEEE Transactions on Microwave Theory and Techniques, vol. 43, no. 6, pp. 1349-1358, June 1995.

[7] Wang, F. , Devabhaktuni, V. K., Xi, C. and Zhang, Q., *Neural network structures and training algorithms for RF and microwave applications*, Int. J. RF and Microwave Comp. Aid. Eng., 9, 216-240 (1999)

[8] Richard McKeon, *Neural Networks for Electronics Hobbyists*, Apress (2018) ISBN 978-1-4842-3506-5

[9] J. E. Rayas-Sanchez, *EM-based optimization of microwave circuits using artificial neural networks: the state-of-the-art*, in IEEE Transactions on Microwave Theory and Techniques, vol. 52, no. 1, pp. 420-435, Jan. 2004.

[10] Calik, N, Belen, MA, Mahouti, P. Deep learning base modified MLP model for precise scattering parameter prediction of capacitive feed antenna. Int J Numer Model. 2019;e2682.

[11] Zhang, Chao Jin, Jing Na, Weicong Zhang, Qi-Jun Yu, Ming, *Multivalued Neural Network Inverse Modeling and Applications to Microwave Filters*, IEEE Transactions on Microwave Theory and Techniques. PP. 1-17 (2018)

[12] L. Ardizzone, J. Kruse, C. Rother and U. Köthe, *Analyzing Inverse Problems with Invertible Neural Networks*, International Conference on Learning Representations, 2019

[13] Antonios K. Alexandridis, Achilleas D. Zapranis, *Wavelet neural networks: A practical guide*, Neural Networks, Volume 42, 2013, pp 1-27,

[14] Zainuddin, Zarita Ong, Pauline. *Function approximation using artificial neural networks*, WSEAS Transactions on Mathematics. 7. (2008)

# List of Figures

1 S11 vs. S21 scatter plot (top right panel) for all data points including all frequency channels, before and after selection. Upper left and lower right panels shows histogram of resp. S11 and S21 values. . . . . 4
2 S11 vs. x2, for several values of x1 (color coded, see legend). The noise like behaviour may be due to unresolved oscillations. It is characterized by a $\approx$ 0.5 dB dispersion and may thus limit the accuracy of modeling. . . . . 5
3 S11 vs. x1, for several values of x2 (color coded, see legend). To be compared to figure 2, here the curves stay smooth without the noise-like behaviour seen w.r.t. x2. . . . . 5
4 Sketch of the neural network used for direct modeling, with one input layer (4 neurons for 4 input parameters), one hidden layer (here 20 neurons) and one output layer (48 neurons for the 4 S-parameters in 12 frequency channels). Each layer is "fully connected", meaning that each neurons of one layer is linked to every neurons of an adjacent layer. . . . . 6
5 RMS error during training, for one hidden layer with 5, 10, 15, 20, 25, 30 and 500 neurons (color coded, see legend).That error is evaluated on normalized outputs and must be renormalized to represent a dB error. . 7
6 RMS error between simulated data and neural network outputs after 1000 training cycles. That error is evaluated on normalized outputs and must be renormalized to represent a dB error. . . . . 8
7 RMS error on training and validation data set during training of a 20 neurons/1 hidden layer network. The error on validation data is 4% above the error on training data at worst, the overfitting is thus negligible. . 8
8 Histogram of errors (simulation-model) on all validation data, for the 4 S-parameters . . . . . 9
9 Histogram of S11 errors (model-simulation), for the first and the last frequency channel. . . . . 10
10 RMS error on S11 vs. frequency channel. . . . . 10
11 Sketch of the neural network used for inverse modeling, with one input layer (5 neurons for 5 constraints on maximum return loss (S11max and S22max), desired gain value and maximum variation in the frequency domain (S210 and $\Delta$ S21max), and maximum noise (NFmax)), four hidden layers (30 neurons each) and one output layer (4 neurons to output electronic parameters x1, x2, x3 and x4). . . . . 12