URSI GASS 2020, Rome, Italy, 29 August - 5 September 2020

Front-end adaptive electronic modeling with neural networks for radioastronomy

B. Censier*(1) and S. Bosse(1)
(1) Nançay radioastronomy station (USN Nançay), Observatoire de Paris, CNRS/INSU, F-18330 Nançay, France

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 include execution speed without loss of accuracy, and the possibility of modeling a system without any prior hypothesis based on measurements only, or any other source of information. This is a crucial step for designing a fully adaptive front-end electronics, with a single electronic board capable of adapting to several different observational and instrumental constraints like signal over noise, bandpass, dynamic range, power consumption, and several other possible features. In this first study, simulated data are used to evaluate the performances of such machine learning methods. The direct model is shown to reach less than 0.1 dB RMS error with respect to the simulated data, which is sufficiently accurate to be compatible with measurements-based modeling without distorsion. The inverse model is constructed by training a network on a pre-computed optimization function. It is able to output optimal electronics parameters satisfying a set of performances constraints without errors compared to the original function. Those first tests confirm neural networks are valid tools for complex RF electronic modeling, even with modest computing resources running basic networks topologies, and thanks to the availability of powerful associated algorithmy. There is thus several perpectives for improving and extending those first results.

1 Introduction

Within the framework of the S-AAIR project [1], which aims at designing an adaptative front-end electronics for radioastronomy phased arrays, a fast and accurate modeling method is required. The circuit has to adapt its performances to each type of observation through controlable input parameters (e.g. impedances and active sources). It is particularly important for generalist telescopes, for which several different types of observations require different performances goals, but may be useful for any set of constraints specific to given scientific goals. This implies one is able to compute the optimal input parameters with respect to the desired performances, fastly enough to be compatible with a regular update of the performances. This is the goal of *inverse modeling*, while *direct modeling* allows predicting output performances given a set of input parameters. The system can be considered as a *complex* one, with high non-linearity and dimensionality, and without any closed-form expression. Both types of modeling thus often imply the use of simulation and optimization methods that are heavy on computing resources and time consuming, with an accuracy limited by systematic effects due to approximations and/or ill-adapted empirical models. This can also be done using direct measurements of the performances with a dedicated instrumentation, which has the advantage of being hypothesis-free, qualifying the system as it really is. On the other hand the information provided by measurements are statistical by nature and have to be interpreted or at least filtered in some way for an accurate model to be constructed. Finally both measurements and simulations are requiring a trade-off to be made between time and accuracy: one may have accuracy at the cost of time spent, or may save time at the cost of accuracy. Neural network seem to be particularly adapted to this problem, being able to model non-linear systems with high input/output dimensionnality, with a much faster execution speed than simulations or measurements without any loss of accuracy, and with the possibility of aggregating all sort of differents input data to construct the model, be it simulations, empirical/analytical models, or measurements. Moreover the neural network may be seen as a kind of universal fitting machine, able to filter out complex noises fundamentally based on least-square fit without prior hypothesis, and is thus particularly adapted to noisy data. This article deals with both direct and inverse modeling with neural networks, based on the simulation of a S-AAIR frontend electronics.

[1]Smart Aperture Array Integrated Receiver

2 Neural networks: motivation and goal

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", which is introducing non linearities in the modeling). An artificial neural network is a set of interconnected neurons. In the present study, the *feed-forward* topology is used with one input layer, one output layer, and one or more hidden layers of neurons in between. The neural network is set to follow a measured and/or an-

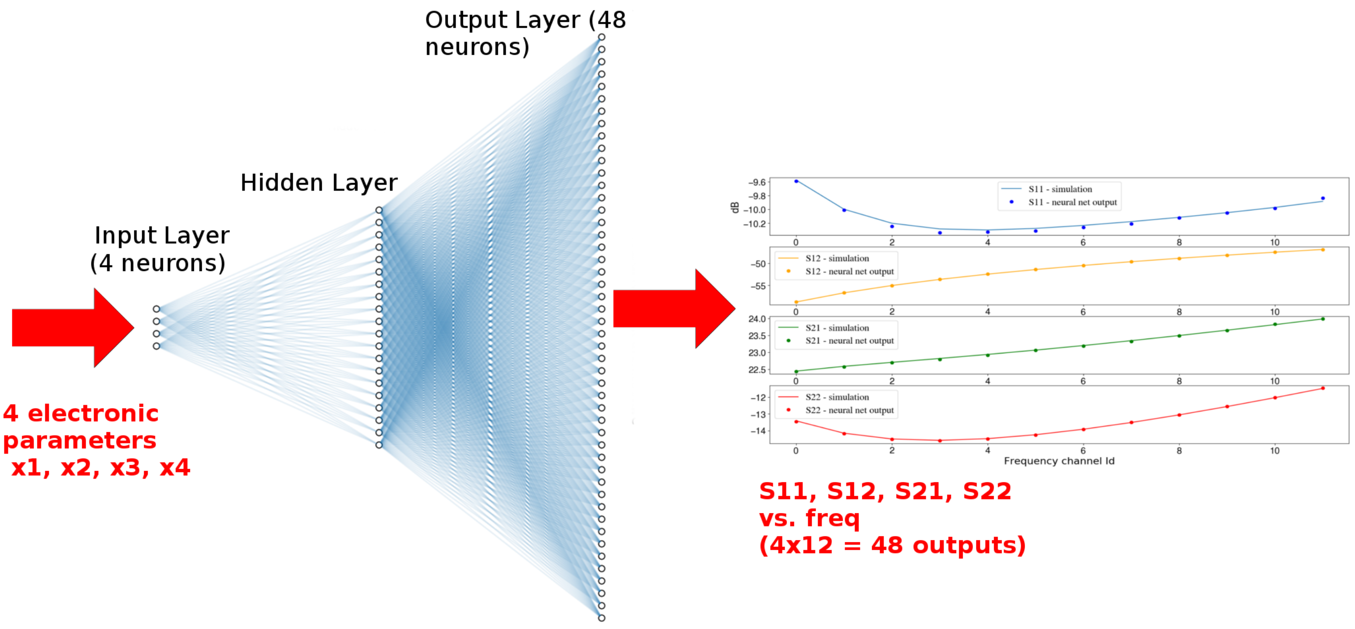

Figure 1. Sketch of a neural network used for direct modeling, with one input layer (4 neurons for 4 input parameters), some hidden layers (20 neurons on a single layer on the sketch for the sake of clarity) and one output layer (48 neurons for the 4 S-parameters in 12 frequency channels). Each layer is "fully connected", meaning that each neurons of one layer is linked to every neurons of an adjacent layer. The actual network used for direct modeling has 5 hidden layers with 60 neurons each.

alytical 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, with weights being adjusted iteratively during training by e.g. a gradient-based algorithm. As already stated, there is a trade-off to be made between detailed physical modeling that may be accurate but slow and heavy on resources, or an 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. Indeed 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. This is a rigourously demonstrated nevertheless purely theoretical result. In practice one has to take care about the *learnability* of a given problem: does the optimization algorithm exist and will it converge in a reasonable time ? 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 slow and/or difficult convergence requiring dedicated algorithm. We fix two reference orders of magnitude for the accuracy goal. We expect a 10% relative error on simulation, and a 1% relative error on measurement. This translates to between 0.5 dB and 0.1 dB difference between input data and neural outputs. 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. The basic goal is to build a fast direct model, allowing to sum up all the informations about the system behaviour in a functionnal block. 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. For inverse modeling, this direct model may in turn be used in a classical optimization loop, in this work however a network is directly trained with specifications as inputs and optimal electronics parameters as output. Two different python libraries have been evaluated: *pybrain* which is a basic and easy to use machine learning library, and *tensorflow* which is a more sophisticated library with far more fine tuning possibilities, a wide user base and the access to several algorithms optimized for multilayer networks.

3 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. We first restrict the data

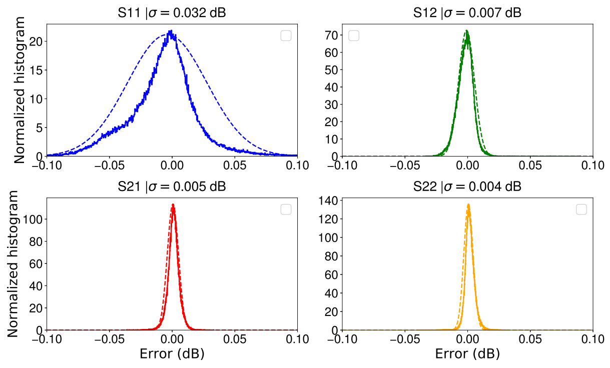

Figure 2. Histogram of errors in dB, with error = (simulation data(dB) - neural model(dB)) computed on each output sample, for each of the 4 S-parameters and over the full input parameters and frequency range. The dashed line in each histogram is a gaussian distribution with the same standard deviation as the measured error distribution for reference. This standard deviation σ is showed on top of each panel.

to useful operating points, with S21 > 0 dB, S11 < -10 dB, and S22 < -10 dB. Based on that filtering, we keep 14040 samples from the original 65536 simulated samples.

4 Direct modeling

The neural network for direct modeling has four input neurons corresponding to the 4 electronic parameters x1, x2, x3, x4. Each of the 48 output neurons will represent a given frequency channel for one of the 4 s-parameters (see figure 1). Following a classical training scheme, 90% of the simulated data are used for training and the remaining 10% is used for validation, in particular monitoring of possible overfitting. A typical training session is completed in several minutes to hours on a regular laptop, depending on the desired accuracy, the volume of the training set, and the number of neurons. Once trained, the accuracy is evaluated by computing an error which is the difference between the actual simulated data, and the neuron network outputs in dB. A first try with only one 60 neurons hidden layer with the *pybrain* library already allows to reach a root mean square (RMS) error of 0.5 dB on S11, while the other 3 S-parameters errors are around 0.1 dB RMS. The error distribution nevertheless shows a non negligible probability of isolated samples showing dB scale errors. We thus increased the number of hidden layers, which increases the number of free parameters with additional neurons. The definitive results presented in figure 2 are obtained on a 5 hidden layers/60 neurons each network using the *tensorflow* library. The RMS errors are of the order of hundredth of dB or less, with a distribution close to or even well under a corresponding gaussian with the same standard deviation (see figure 2 dashed lines). The neural network model is thus able to model data with greater accuracy than the $\approx$ 0.1 dB noise expected with measurements. The $\approx$0.5 dB accuracy obtained with a single layer network is reached in about 10 minutes on a classical laptop. The more accurate multi-layer version requires about 2 hours to get close to the 0.01 dB RMS error level. This illustrates the cost of using bigger networks in terms of training time, while the execution time once trained remains very low. If regular and frequent update of the model are needed, there is a thus a possible trade-off to be made between the accuracy needed and the time spent on regular training session for adaptive model update.

5 Inverse modeling

Compared to direct modeling, inverse modeling's main difficulty is the non uniqueness of solutions: there can be several set of input parameters satisfying the desired constraints. Moreover, what we need is not an inverse function that gives input parameters that will lead to the *exact performances* we asked for. We rather need the system to find an *optimal* solution to a set of performance constraints, typically defined with low and high bounds. There are several ways of overcoming those problems. In later development stages, we expect to use the ability of advanced neural networks topologies to compute this optimization feature during training. For this first step we trained the network on a pre computed optimization function. The classical method used for optimization is the creation of a loss function that is minimum when the constraints are best met. Since we have access to a whole simulated datacube, we can compute this loss function for any set of input parameters and any set of constraints. This loss function is defined as follows:

$10 \times \delta S22max + 1000 \times \delta S11max + 10 \times \delta S21 + 100 \times \delta NFmax$

Where:

- $\delta S22$ is the squared sum of differences between S22(x1,x2,x3,x4) and the constraint S22max over all frequency channels.
- $\delta S11$ is the squared sum of differences between S11(x1,x2,x3,x4) and the constraint S11max over all frequency channels.
- $\delta S21$ is the squared sum of differences between S21(x1,x2,x3,x4) and the desired gain S210. Only differences greater than the input specifications Δ S21max are considered.
- $\delta NFmax$ is the squared sum of differences between NF(x1,x2,x3,x4) and the constraint on noise figure NFmax.

The sum of all these terms adds up the different constraints together, and each of those constraints are punderated 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

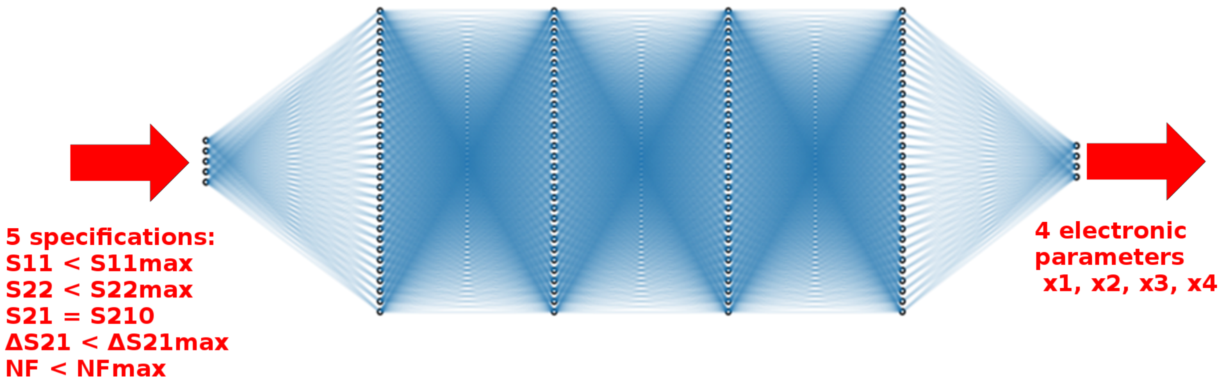

Figure 3. Sketch of the neural network used for inverse modeling, with one input layer (5 neurons for 5 constraints on maximum return loss (S11max and S22max), desired gain value and maximum variation in the frequency domain (S210 and Δ S21max), and maximum noise (NFmax)), four hidden layers (30 neurons each) and one output layer (4 neurons to output electronic parameters x1, x2, x3 and x4)

of 10). The method for generating training data is the following:

- set a grid over the 5-D space covering input specifications S11max, S22max, Δ 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, Δ S21max, S210 and NFmax values as inputs, and $x1_{opt}$,$x2_{opt}$,$x3_{opt}$,$x4_{opt}$ values as outputs is then built. The network has 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 3). 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. Sufficient performances have been reached with 4 layers of 30 neurons using the *pybrain* library (see figure 3). Once trained, the neural network reproduces exactly the optimization function behaviour over validation data.

6 Conclusion

Neural networks have been showed to be an interesting tool for both direct and inverse modeling. A system with controllable features is modelled with sufficient precision to be compatible with both measurements and simulation-based modeling. Later applications could include the use of measurement-based functional blocks for both reducing simulation time and using an effective and realistic model. Concerning inverse modeling, the ability of neural networks to be used as inverse solvers on typical engineering constrained problems has also been illustrated. The network obtained represents a huge time advantage compared to similar optimization in simulation software like ADS. A network indeed has a typical milliseconds execution time once trained, while a single point could take hours to be computed with a classical optimization scheme using simulations. Nevertheless, the network still relies on those simulations to be computed over the whole input parameters range. Further works will study the possibility for computing this optimization step directly during training using more sophisticated neural network topologies and/or training scheme (see e.g. [2]) It may also be noted that the hardware and software involved in this study is of average performance and easily available (average laptop without graphical processor parallelization) For an adaptive electronics, real-time implementations seems possible on existing dedicated hardware like GPU or FPGA based solutions, or the more recent developments around neural processing hardware [3].

References

[1] Kurt Hornik, Maxwell Stinchcombe, Halbert White, Neural Networks, Volume 2, Issue 5, 1989, pp. 359-366

[2] Zhang Chao, Jin Jing, Na Weicong, Zhang Qi-Jun, Yu Ming, IEEE Transactions on Microwave Theory and Techniques. PP. 1-17 (2018)

[3] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, Tianqi Tang, Engineering, 2020 (In press) ISSN 2095-8099