JS 26

INTELLIGENT TOOLS AND METHODS

FOR RADIO SCIENCE

Inverse modeling for RF adaptive front-end electronics with bayesian machine learning

B. Censier[1], S. Bosse[1]

[1]Nançay radio astronomy observatory, UAR704, route de Souesmes 18330 Nançay, FRANCE, (benjamin.censier@obs-nancay.fr)

Keywords: Radioastronomy, RF electronics, energy, machine learning, inverse modeling

Abstract

Following our previous work on machine learning based front-end modeling, an inverse modeling method is presented. Using a specific neural network setup called Conditional Variational AutoEncoder (CVAE), the inverse relation defining optimal input parameters for a given performances goal can be defined through a probabilistic bayesian approach. This overcomes the non-uniqueness problem in a robust way, and allows exploration of the full set of solutions within a given requirements range. It therefore makes fine tuning of an adaptive front-end based on variable components possible, allowing various set of specifications to be fulfilled with a single circuit.

1 Introduction

Radio astronomy requirements for front end electronics is covering a large range with respect to e.g. bandwidth, noise level or dynamics, depending on the scientific subject at hand and the type of source under study. The advent of large, general purpose instruments moreover tends to gather those different requirements in a single setup. The growing use of phased arrays is all the more pushing this tendency, with performances possibly varying with respect to pointing direction and the massive use of front end electronics on large antenna fields posing the crucial question of energy consumption. The latter point is or particular interest, being a crucial bottleneck to large phased arrays adoption at Ghz frequencies [1,2]. Acknowledging this diagnostic, the development of adaptive front-end could optimize contemporary instruments further, and unlock innovative instrumentation possibilities, allowing a great flexibility with respect to the balance between the diversity of targets, the performances of the electronics chain, and the associated energy cost. This could be based on circuits including variable components, an already feasible hardware setup but that also requires an efficient and accurate modeling of their behavior with respects to input parameters. The ultimate feature is an inverse modeling allowing to get input parameters satisfying a given set of output performance requirements. Here we present a method based on machine learning allowing for such an inverse modeling, tested on a simulated prototype of adaptive front-end. This is a first step proving the feasibility of the concept, ultimately aiming for full measurements-based characterization of front-end electronics.

2 Adaptive front-end

The simulated adaptive front-end is a prototype circuit designed for the Smart Aperture Array Integrated Receivers (SAAIR) initiative developed at Nançay observatory. It includes 4 variable components (variable impedances and active voltage/current sources), their values being the 4 input parameters. Its performances are tracked by 5 features simulated with the ADS software as a function of frequency in the 400-1500 MHz range, namely 4 S-parameters and a noise figure. The latter are the output of the model.

3 Inverse modeling method

In a previous work [3], we already explored the modeling possibilities with neural networks, including direct modeling for a compact and handy representation of a front-end, and a first attempt at inverse modeling. The latter required a preparation step using the direct model and a classical optimization method for defining the training dataset on which the inverse modeling was based. More fundamentally, this method bypassed the so called “non unicity” problem traditionally encountered in inverse modeling. The trained network could not propose the whole range of possible solutions, outputting a single point in the input parameters phase space. As a result, the method though functional was sub-optimal and didn’t leverage some additional capacities in statistical data modeling offered by bayesian machine learning.

Here we present a second, more robust and efficient implementation using Conditional Variational AutoEncoder (CVAE). To present this concept, let us decompose it in three incremental steps.

3.1 Autoencoder

The autoencoder concept, while designed in the 1980s, has been promoted by a seminal paper in 2006 [4]. It is sketched in Figure 1, where **x** is a vector input of arbitrary dimension, and **x’** is an output vector. The encoder and decoder modules are neural networks, with possibly several hidden layers, and a set of weights set by convergence of a training process. The encoder module role is to transform the input vector into another vector described in an abstract basis, the so called “latent space”, with an arbitrary number of dimensions set by design. If the dimensionality of this latent space is less than the one of the input vector, this amounts to a compression process in an abstract space. Unsurprisingly, the role of the decoder is to decode that latent vector back to the original one, and the decoder module is also a neural network to be trained. Following the same example as above, that could be a decompression process, ideally back to the original uncompressed vector without losses. The whole point of the training phase, while presenting a set of input training vectors [**x**] at input, it to converge towards a set of weights for encoder and decoder that will allow the best possible reconstruction of **x** in **x’**.

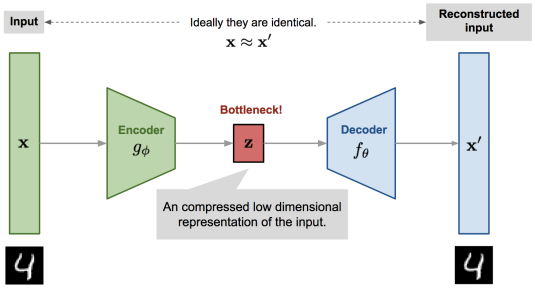

Figure 1: Sketch of an autoencoder architecture. (source :https://lilianweng.github.io/posts/2018-08-12-vae/)

3.2 Variational Autoencoder (VAE)

The Variational AutoEncoder (VAE) uses the same encode/decode architecture, but the latent space is now a probabilistic one : instead of mapping the input vector x on a fixed z latent vector, it is mapped on a distribution, allowing an implementation of the methods of variational bayesian model [5]. Its is sketched in figure 2, where the **x** input vector passing through the encoder results in an output mean vector **μ** and standard deviation vector **σ**, defining a multivariate gaussian distribution **ϵ** from wich the latent vector **z** is drawn. This **z** vector thus becomes a random vector jointly distributed with the input vector **x**. Grossly speaking, the goal of the process is to encode the arbitrary distribution of **x** input vectors used during training in a multivariate gaussian distribution in the latent space. Through the constraint “**x** = **x’**” driving the training process, the decoder part becomes a generative module, able to generate a set of **x’** output vectors based on a set of gaussian distributed **z** vector, with the **x’** vectors following closely the arbitrary distribution of the initial **x** input training vectors.

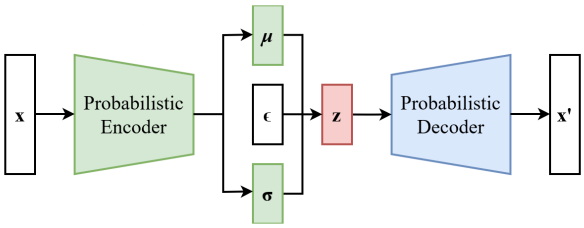

Figure 2: Sketch of a variational autoencoder (VAE) architecture. The encoder outputs vector parameters μ and σ that define the multivariate distribution ϵ, from which the latent vector z is drawn. (source : https://en.wikipedia.org/wiki/Variational_autoencoder)

3.3 Conditional Variational AutoEncoder (CVAE)

The CVAE structure is close to the VAE one, except for an additional **y** vector input to both the encoder and the decoder. This is the concept we used for inverse modeling [7], and we can finally directly linked this abstract description to our front-end modeling problem. Here the **x** vector contains the 4 input parameters setting the values of each variable component of the circuit. The **y** vector contains the simulated performances features (Scattering and noise parameters). In the CVAE concept, **y** acts as a label for each **x** input vector presented during the training phase. As a result, the latent space is still encoding the probability distribution of **x** input parameters, but with a separate sub distribution for each **y** vector. Therefore, the trained decoder module still acts as a generative one, able to output a set of **x'** closely matching the arbitrary distribution of **x** vectors based on a gaussian distributed vector **z**, but now only outputs the sub distribution corresponding to the given **y** label. Transposed to our concrete front-end modeling case, the trained decoder is able to output a set of component parameters that closely matched the distribution maximizing the probability of obtaining the performances **y**. In practice, one gives the desired performances as an input of the decoder, and scans the phase space of possible component values by providing N **z** vectors drawn from a multivariate standard normal distribution (μ=0, σ=1). The decoder outputs a corresponding set of component values, and a histogram of those values reveals the ones that maximize the probability of getting the desired performances.

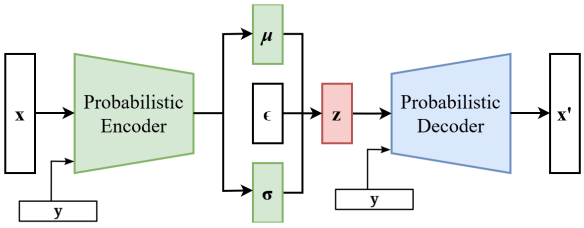

Figure 3: sketch of a Conditional Variational AutoEncoder (CVAE). An additional input vector y is added to both the encoder and the decoder. This vector acts as a label associated to each x input during training. As with VAE, the trained decoder acts as a generative module following the statistical distribution of x training inputs, but the latent space encodes a specific sub-distribution for each y label.

4 Implementation

The CVAE is implemented using the tensorflow python library. Both encoder and decoder are built with 4 hidden layers of 120 neurons each, consecutive layers being fully connected. The latent space has 20 dimensions, allowing for a detailed description of the involved distributions. Encoder input is made of a 4 dimensions vector containing the component values, each one coded on 4 bits (0-15), and a 6 dimensions vector containing the maximum values in frequency of S11, S22, S12 and NF the noise figure, the minimum value of the S21 forward gain, and the (max-min) value in frequency of S21. This allows for requirements limiting the amount of reflected power at input and output, ensuring a minimum gain, and restraining the gain variation over the bandwidth. As explained above, the decoder uses the same vector structure, with performance parameters given at its input together with the latent vector, and a 4 dimension output corresponding to the electronic components values.

The training data set is built from simulated data covering the full 16^4=65536 set of possible components values combination.

The training phase lasted for 4 hours on a regular DELL precision laptop, using the sum of squared differences between input of the encoder and output of the decoder as the loss function to be minimized, and the Adam (Adaptive Moment Estimation) optimizer for weight convergence.

Once trained, the decoder part is used as a standalone generative module. The desired performance requirements are chosen, and defined as fixed inputs, while N latent vector inputs are drawn from a multivariate standard normal distribution. N is on the order of 100 000 sampled vectors, allowing to output a cloud of points in the 4D phase space of component values. The performance requirements input may also cover a range of requirements instead of a single set.This 4D space is then binned on a regular 16x16x16x16 grid, and a histogram is generated to define a probability distribution for reaching the requirements. One decoder inference operation takes on the order of tens of microsecond, so that the overall scanning of the phase space amount to few seconds.

5 Results

Results are illustrated in figure 4 and figure 5, showing the output of the decoder for the aforementioned scanning of phase space, as a points cloud in different slices of the 4D component values space, and as a 2D histogram on those slices. Figure 4 shows a typical unambiguous optimal point for reaching a narrowly defined performance goal, with a unique set of components values (x0,x1,x2,x4) satisfying the chosen requirements. The spread around the unique solution is gaussian to a good approximation and the standard deviation is far smaller than the unit difference between 2 values.

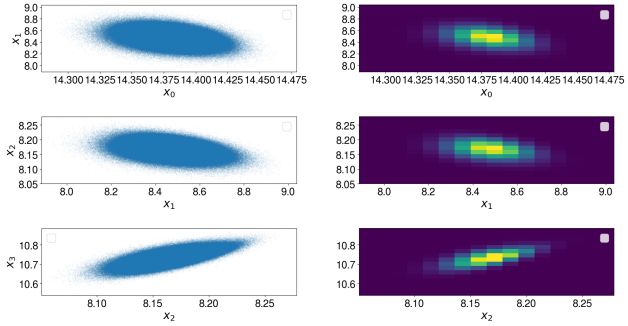

Figure 4: x0, x1, x2, x3 are the 4 components values. Left : Output of the decoder for 100 000 sampled latent vectors, in 3 slices of the 4D space. Right : corresponding 2D histogram amounting to the probabilty distribution for reaching the performances goal.

In figure 5, a more ambiguous case is illustrated. Here the the input requirements on scattering and noise parameters are also scanned on a given range of acceptable values. As a result, the distribution of possible component values satisfying those requirements has a complex shape, with a significant spread along both the x0 and x1 axis. Thresholding that distribution allows for defining a set of operating points all compatible with the desired range of performances (contour in the scatter plot, and red dots in the 2D histogram), while finding the maximum of the distribution allows for the definition of an optimal operating point with respect to the probability of fulfilling the requirements (x sign in the 2D histogram).

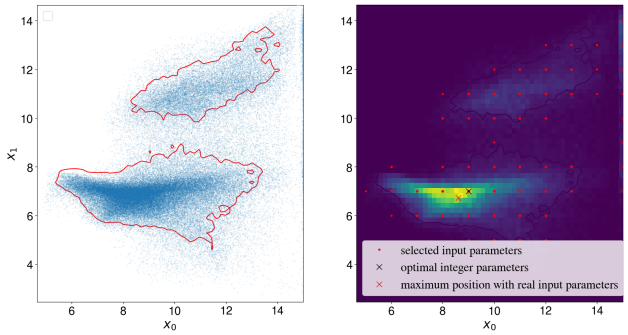

Figure 5: Output of the decoder for a 100 000 latent vectors scan, together with a scan on a range of performance requirements, on a typical case showing complex patterns with arbitrary distribution, and an ensemble of possible solutions for components values optimal settings. Left : decoder outputs in a (x0,x1) slice of the 4D space (dots) and thresholding contour (red line). Right : 2D histogram with color coded values. The red dots are the operating points selected by thresholding, the 'x' symbols show the optimal values (real and rounded) in terms of probability maximization.

Those results have been validated by comparison with a classical optimization method, which showed compatible results in simplest cases, several more complex configurations where the CVAE managed to output a range of solutions where the classical method provides only one, and a few cases where CVAE managed to find an optimal solution when the classical method failed to reach optimality.

6 Conclusions and perspectives

The use of CVAE for dealing with inverse modeling of an adaptive circuit parametrized by a few components values has been demonstrated. Showing a robust behaviour with respect to complex cases, and allowing the definition of a fast inference inverse model in a single training step, this opens the way to full measurements based characterization of a circuit, which will be an important step towards a functional, accurate and easily usable adaptive front-end. Further work will aimed at precising its validity domain and improving the efficiency of the training phase. The former point will benefit from the addition of new requirements like compression point, not included in this study. This will allow to lift some degeneracies in the output, and it is a very important parameter with respect to energy consumption optimization. Concerning training optimization, an adaptive meshing of the input parameters space could allow to further speed up the process without precision loss due to the extensive interpolation ability of neural networks, and complexify the modeling with more input parameters as well as performance specifications, while keeping the required computing resources in a reasonable range. Input phase space restriction could also benefit from screening implausible operating points with respect to e.g. DC bias voltage of transistors and other common sense rules of electronics. An optimization of neural networks layers topology and latent space dimensionality could also further refine the implementation, e.g. by leveraging selection through genetic algorithms [8]. We ultimately plan to apply that method on measured data, evaluating a reconfigurable frontend system on an automatized test bench using a network analyzer. This will allow to validate the method on experimental data, and to build a functional inverse model without any prior hypothesis on the system, thus closely matching its actual behaviour.

A salient feature of this inverse modeling method is its great flexibility, and the numerous possible applications. Besides adaptive circuit, it can be seen as a general method for circuit design, building on a fixed topology with free parameters to be optimized. Some recent works [6] also show machine learning based methods may optimize the circuit topology itself, and a junction between the 2 approaches could be a very powerful assets to RF design.

References

[1] B. Censier and S. Bosse, “Future developments and energy management: an analog point of view”, SF2A 2021, *Proceedings of the annual meeting of the French Society of Astronomy & Astrophysics*

[2] B. Censier and S. Bosse, “Prospects for energy management on modern radiotelescopes”, *2021 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, Honolulu, HI, USA, 2021, pp. 234-234, doi: 10.1109/ICEAA52647.2021.9539723.

[3] B. Censier and S. Bosse, “Front-end adaptive electronic modeling with neural networks for radioastronomy”, *2020 XXXIIIrd General Assembly and Scientific Symposium of the International Union of Radio Science*, Rome, Italy, 2020, pp. 1-4, doi: 10.23919/URSIGASS49373.2020.9232198.

[4] G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, *Science*, **313**, 504-507 (2006). DOI:10.1126/science.1127647

[5] Fox, C.W., Roberts, S.J. “A tutorial on variational Bayesian inference”, *Artif Intell Rev*, 38, 85–95 (2012). doi:10.1007/s10462-011-9236-8

[6] Karahan, E.A., Liu, Z., Gupta, A. *et al.* “Deep-learning enabled generalized inverse design of multi-port radio-frequency and sub-terahertz passives and integrated circuits”. *Nat Commun* **15**, 10734 (2024). doi:10.1038/s41467-024-54178-1

[7] Wallace Anderson McAliley and Yaoguo Li, “Stochastic inversion of geophysical data by a conditional variational autoencoder”, *GEOPHYSICS* 2024 89:1, WA219-WA232

[8] Abuelenin, S., Elmougy, S., Habeeb, F., “Optimizing Deep Learning Based on Deep Auto Encoder and Genetic Algorithm”, *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics 2017*. AISI 2017. Advances in Intelligent Systems and Computing, vol 639, doi:10.1007/978-3-319-64861-3_62