Front-end adaptive electronic modeling with neural networks for radioastronomy

B. Censier

USN Nançay (Nançay radioastronomy station, France)/CNRS

URSI2020 (07/2020)

Context

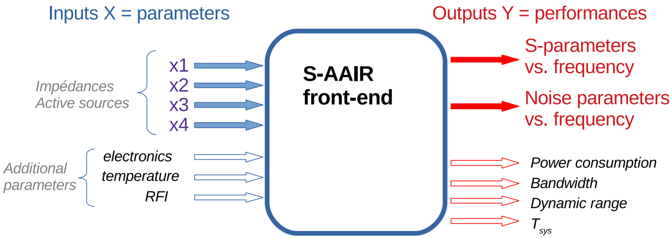
- S-AAIR project, Nançay microelectronic team

(Smart Aperture Array Integrated Receiver)

- Goals:
  - adapt front-end performances to each type of observation
  - ultimate goal : trade-off beween required performances and energy consumption
  - especially interesting for generalist telescopes with very wide range of constraints depending on observation
- Hardware:
  - adptative front-end via controllable impedances and active sources

Direct and inverse modeling

- direct modeling: predict performances given input parameters
- inverse modeling: predict optimal input parameters for a given set of performances contraints



Constraints and Requirements

- Compute the optimal input parameters with respect to the desired performances
  - fast enough for regular updates
  - with 1% / 0.1 dB max error

  $\Rightarrow$ needs fast and precise modeling

  $\Rightarrow$ on a complex system (non linearity and dimensionality)

  $\Rightarrow$ able to cope with both measurements and simulation data

$\Rightarrow$ Neural networks

Neural networks in short

- machine learning paradigm: the model is "learnt" based on inputs data that the network is feed with
- one neuron = linear combination of inputs with weights + thresholding by a non linear function
- neural network: set of interconnected neurons
- training step:
  - weights are adjusted by an optimization method over the whole network
  - optimization runs until a sufficient convergence between network inputs and test data is obtained
- here use of feedforward networks: one input layer, one output layers, several "hidden" layers in between, 2 adjacent layers fully connected

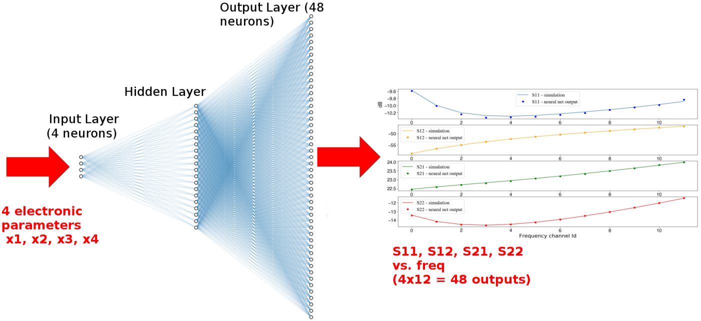
Data and framework

- direct model:
  - 4 inputs electronic parameters on 4 bits (0-15)
  - S-parameters vs. frequency (12 subbands) as performance output
- inverse model:
  - 5 constraints on S-parameters as input
  - 4 optimal parameters w.r.t constraints as output
- this first study based on ADS simulation of a 2-stage S-AAIR front-end
  - 65536 (2**16) simulated samples for 4 parameters on 4 bits
  - 14040 samples after selection S21>0dB, S11<-10dB and S22<-10dB
- eventually hypothesis-free modeling based on measurements only

Data and framework

- every results with python scripts on a regular laptop
- 2 libraries tested:
  - pybrain: first tests, simple and not maintained anymore, easy to set up but lack of fine grain tuning
  - tensorflow (2.0): large user base, most used in state of the art machine learning, more complex to use but extended possibilities
  - pybrain was for prototyping, tensorflow revealed much more efficient (speed + convergence with multilayers (i.e. deep learning))

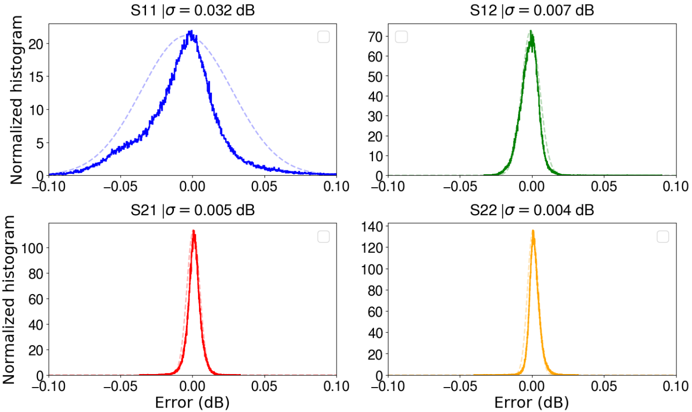
Direct modeling: network



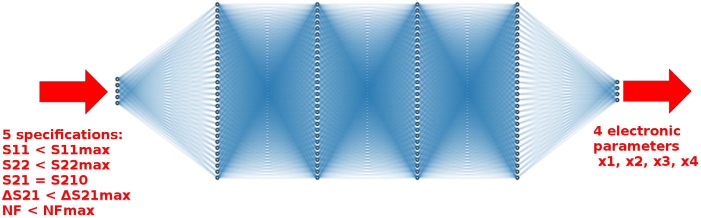
- 4 input neurons = 4 input electronic parameters
- 48 output neurons = 4 S-parameters in 12 subbands each (500-1500MHz)

Direct modeling: results

- first tests: single 60 neurons hidden layer with pybrain
  - tens of minutes training
  - RMS error < 0.1dB for S21,S22,S12, <0.5dB for S11
- second tests: 5 hidden layers with 60 neurons each on tensorflow
  - few hours training
  - RMS error < 0.1 dB for all S parameters



Inverse modeling: network



- 5 input neurons for 5 constraints on S-parameters
- 4 output neurons for 4 optimal parameters w.r.t. input constraints

Inverse modeling: method

- inverse modeling main difficulty: non unicity of solutions (several possible outputs for a given inputs)
- naive method: exchange electronic parameters (become outputs) and S-parameters constraints (become inputs) during training
  - does not converge, or towards a model where multiple solutions are averaged and thus inaccurate
- our method:
  - precompute a loss function representing the set of constraints (weighted by importance): $Loss(x1, x2, x3, x4) = 10 \times \delta S22max + 1000 \times \delta S11max + 10 \times \delta S21 + 100 \times \delta NFmax$
  - precompute the minimum value of the loss function for all possible constraints
  - train the neural network on the precomputed dataset

Inverse modeling: results

- Already reach a satisfying system using pybrain
- 4 hidden layers, 30 neurons each
- optimal parameters between 0 and 15 are reproduced without error when compared to the precomputed optimization function

Conclusion and perspectives

- Neural network are a valid tools for such complex systems with crucial modeling needs
- already feasible on personal computing resources
- training step from minutes to hours depending on the accuracy needed and the training data volume
- execution time once trained is dramatically lower compared to traditional simulation/optimization schemes (few micro seconds vs. few hours)

Conclusion and perspectives

- inverse modeling:
  - non unicity problem can be bypassed using precomputed optimization
  - but optimal solutions should be computed directly during training
    - interesting tool for inverse problems in general
  - need to test other topologies that could cope with non unicity:
    - statistical (variationnal autoencoder), invertible (invertible networks). . .
- further tests on measurements data only
- neural network design is essentially based on trial/error process

(optimal number of layers, neurons, inter connections. . .)

- this design could itself be automatized using genetic-like algorithm to explore possible topologies and select the most adapted ones.