2015 IEEE International Symposium on Circuits and Systems (ISCAS)
DOI: 10.1109/ISCAS.2015.7168698

RF-LNA Circuit Synthesis Using an Array of Artificial Neural Networks with Constrained Inputs

Etienne Dumesnil
Dept. of Computer Science
University of Quebec at Montreal
Montreal, Canada
dumesnil.etienne@courrier.uqam.ca

Frederic Nabki
Dept. of Computer Science
University of Quebec at Montreal
Montreal, Canada
nabki.frederic@uqam.ca

Mounir Boukadoum
Dept. of Computer Science
University of Quebec at Montreal
Montreal, Canada
boukadoum.mounir@uqam.ca

***Abstract*—We describe a method for circuit synthesis that determines the parameter values by using a set of artificial neural networks (ANNs) that learn in sequence. Each ANN is optimized to output only one design parameter, and the latter constrains the learning/recall of its successor(s). Two competing ANN architectures are considered, the multilayer perceptron (MLP) and the radial basis functions (RBF) network, and each one has its internal parameters tuned by a genetic algorithm. The method was tested on the design of a radio-frequency, low-noise amplifier (RF-LNA) with ten design parameters to set, and it yielded one-hundred percent success rate in specifying the parameter values at five percent tolerance.**

Keywords—Synthesis, artificial neural network, genetic algorithm, multilayer perceptron, radial basis function, radiofrequency low noise amplifier.

I. INTRODUCTION

The relationship between design parameters and circuit performances is usually understood as either a direct or an inverse problem. The direct problem sees circuit performance as a function of design parameters, while the inverse problem relates to the often more difficult task of finding design parameters that meet given performance specifications, [1]. In the case of electronic circuits, the direct problem refers to circuit analysis, and the inverse problem refers to circuit synthesis. For nonlinear systems, the latter problem is currently very difficult to solve by formal techniques [2]. In this context, using ANNs may constitute a better solution approach, since they use a black-box methodology that relies on analogies instead of precise models. Still, there seems to be no reports of ANNs capable of providing all the design parameters to meet a set of performance requirements. This may be due to two facts: 1) There is often more than one circuit topology that satisfies a set of performance criteria; 2) in the most mainstream ANNs (MLPs and RBFs), the possible interrelations between outputs are ignored. As a result, although the ANN may learn to correctly memorize a set of input-output pairs and generalize from them, the knowledge it develops is context-free and ignores the possible constraints imposed by the design parameters on one another. Because of this, the ANN may end up providing inappropriate design parameters when supplied with new performance data.

This paper proposes a solution to the problem that still makes use of the common MLP and RBF models, but within a modified ANN architecture. The idea is to find the design parameters in sequence, each one constraining the determination of the next one(s). First, an ANN is trained to correctly specify a first design parameter. It takes the set of desired performances as input and has only the chosen design parameter as output. A genetic algorithm (GA) [8] is used to characterize this ANN: it selects the architecture to use (MLP or RBF), and determines its internal parameters (e.g., how many neurons will be in the hidden layer) and which design parameter should be output. Once an ANN implementation is found that can successfully generalize the input-output associations it learned during a training phase to new input-output pairs, the GA stops and the ANN is kept to specify part of the sought design. The same process is repeated for a second ANN that, in addition to the performance criteria, takes also the output of the first ANN as input. Thus, this ANN has the task of specifying a second design parameter as a function of the performance criteria and of the first design parameter. Once again, a GA is used for tuning the ANN and here also, once an implementation has successfully generalized the associations learned to new input-output pairs, the resulting ANN is kept. Notice that, during the selection of the second ANN, the first ANN remains unchanged. We now have two ANN instances that provide two of the design parameters. We move to a third ANN that also takes as input the set of performance criteria, along with the outputs from the first and second ANNs. Accordingly, the third design parameter will be a function of the performance criteria and of the two previously specified design parameters. The third ANN is also tuned by a GA, in the same way as the two previous ones. The process of adding ANNs goes on until all the design parameters have been found. Fig. 1 shows the block diagram of the overall process.

The method was tested on the circuit topology of a radio-frequency low noise amplifier (RF-LNA). The usual synthesis process for this circuit is expert-based and time consuming, being based mainly on simulations and design iterations. We will show that the method proposed here requires a few hours of computer processing to determine the correct sequence of ANNs for the given circuit topology, and from there, less than a second to compute the whole set of design parameters for new sets of performance criteria.

In what follows, the RF-LNA topology is presented first, along with the method for generating the set of designs used to train the ANNs. Next, the MLP and RBF ANN models are described, followed by an explanation of the GA used for tuning them. Then, the methodology proposed for the RF-LNA synthesis is detailed, followed by the experimental validation results and concluding remarks.

© 2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

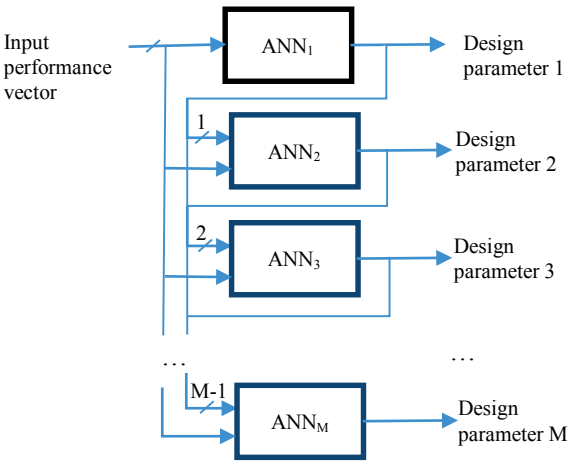

Fig. 1. Block diagram of the proposed synthesis methodology

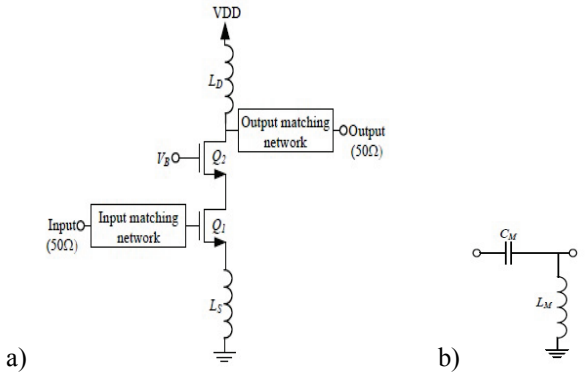

Fig 2. Used LNA topography (a) and matching network (b)

II. METHODOLOGY

A. Radiofrequency Low Noise Amplifier

The circuit topology for realizing the RF-LNA is the same as in [3] and [4], and it is presented in Fig. 2. It consists of a cascode common-source stage with degeneration and inductive load. The cascode configuration was selected for its design simplicity (easier matching, straightforward stability, etc.) and widespread use. All of its inductances, including those in the matching networks, were assumed to be non-ideal with Q-factors of 10, which is consistent with current CMOS fabrication processes. To simplify the design effort further and reduce the number of parameters, only the L-shaped matching network shown in Fig. 2b is considered for 50 Ω source and load impedance matching. Its component values are output by the ANN.

The performance (P) specifications and design (D) parameters considered in this work are listed in Table I, along with their ranges of values after random generation during the validation experiments. Notice that the drain inductance (L_D in Fig. 2) is not mentioned, and it was fixed at 0.578 nH.

B. Neural Architecture

Two common ANN architectures are considered, the multilayer perceptron (MLP) [5] and the radial basis functions (RBF) model [6]. Both consist of three layers of neurons in sequence: an input layer, a hidden layer and an output layer.

Table I. Performance and design variables considered.

Var. #	Variable description	Perf. (P) or Des. (D)	Min. value	Max. value
1	Bandwidth (MHz)	P	387.0	882.3
2	1-dB compression Point (dB)	P	-21.04	-13.03
3	Center Frequency (GHz)	P	2.07	4.76
4	IIP3 (dB)	P	-11.58	2.64
5	Noise Figure (dB)	P	1.386	3.33
6	S21 (dB)	P	9.11	15.29
7	Input capacitance (fF)	D	238.2	1528.2
8	Input Inductance (nH)	D	1.811	10.610
9	Output capacitance (fF)	D	260.0	2902.6
10	Output Inductance (nH)	D	4.913	14.123
11	Transistor 1 length – Q_1 (nm)	D	160	510
12	Transistor 1 width – Q_1 (µm)	D	60	560
13	Transistor 2 length – Q_2 (nm)	D	200	610
14	Transistor 2 width – Q_2 (µm)	D	90	500
15	Source Inductance – L_S (nH)	D	1.707	8.671
16	Bias voltage - V_B (mV)	D	300	600

1) *MLP neural network:* In the input layer, each neuron simply holds the value it receives. In the hidden layer, the output of a neuron j is given by

$$z_j = f\left(\sum_{i=1}^{M} \omega_i x_i\right) \quad (1)$$

where M is the number of afferent neurons, x_i is the output of the i^{th} input layer neuron, ω_i is a weight to be determined and $f(\,)$ is an nonlinear output function, often the logistic sigmoid $f_{logsig} = \frac{1}{1+e^n}$ or the hyperbolic tangent sigmoid $f_{tansig} = \frac{2}{1+e^{-2n}} - 1$. Within a given MLP, all the hidden layer neurons use the same transfer function.

In the output layer, the neural outputs take also the form defined in equ. (1), but the identity function is usually chosen for the output function (linear output). As a result, the output of a neuron k is given by

$$y_k = \sum_{j=1}^{N} \omega_j z_j \quad (2)$$

where N is the number of neurons in the hidden layer, z_j is the output of the j^{th} hidden layer neuron and ω_j is also a weight to be determined.

2) *RBF neural network:* The input layer is the same as for the MLP. In the hidden layer, the output of a neuron is

$$z_j = \varphi_j(\vec{x}) = \varphi\left(\left\|\vec{x} - \vec{x}_j\right\|\right), \qquad j = 1,2,\dots,N \quad (3)$$

where N is the number of hidden layer neurons, $\vec{x}$ is the vector of outputs from the input layer neurons, $\vec{x}_j$ the coordinate vector of the j^{th} hidden layer neuron vector and φ_j is a radial basis function, typically the Gaussian $\varphi_j(\vec{x}) = e^{\left(-\frac{1}{2\sigma_j^2}\|\vec{x}-\vec{x}_j\|^2\right)}$, where σ_j expresses the basin of attraction of the j^{th} hidden layer neuron. In the output layer, the output of a neuron is the same as for the MLP (equ. (2)).

Here, both the MLP and the RBF take as input the set of performance specifications, plus additional inputs acting as

constraints (see section II.C), and have their weights determined by the error back-propagation algorithm [5], which tries to minimize the mean squared error between the desired and the predicted outputs. During training, known input-target pairs are provided as examples; then, other known input-target pairs are used to test the generalization capabilities of the selected ANN.

C. Genetic algorithm

Since the combinations of architectural parameters to define an ANN is susceptible to combinatorial explosion and no formal method exists for ANN definition, a GA was designed to select i) the best ANN type (i.e. MLP or RBF), ii) its operation parameters and iii) the design parameter to output. For all ANNs, the steps for creating a population of ANNs to evolve in search of the most performing individual were as follows: first, a *k*-bit binary chromosome structure is defined, where each bit or group of bits indicates the ANN architecture to use, the value of an ANN parameter, or which design parameter the ANN will output. Next, *N* instances of the chromosome structure are created randomly, forming an initial population. Then, a fitness function evaluates how well the ANN specified by each created chromosome can generalize what it learned from a training set to new designs (see section III). The fitness value used is the difference between the desired design parameter value and the actual output from the ANN.

As the population evolves, half of it is selected for reproduction at each iteration (generation). This is accomplished with the roulette wheel algorithm [8], where the probability of a chromosome to be selected depends on its fitness value. Here, it was computed by first elevating the fitness value to the fourth power in order to increase the bias in favor of the fittest chromosomes (by increasing their area in the roulette even more). The first two fittest survivors then have a pair of children together, as do the next two and so on. Each child chromosome is the result of a crossover between its parents. Two crossover points are randomly selected (e.g., 11th and 22nd bits). Then, for the first child, the bits preceding the first crossover point and following the second one come from one parent, and the remaining bits come from the other parent. The second child has the substitution order reversed. Finally, each child is susceptible to mutation with a certain probability (0.001 in this work), which means that each bit could be inverted. The *N* exemplars that reach this stage (N/2 parents and N/2 offsprings) form the next generation.

The preceding selection, reproduction and mutation process is repeated continuously, with each cycle resulting in a new generation, until the success rate of the best ANN in a generation reaches a threshold (100 % here), at which point the GA iterations stop. Success is defined as the correct prediction of the target LNA component values for new designs (i.e. correct generalization) with an error of five percent or less. Then, the winning ANN is saved for future use.

After specifying the ANN to output the first design parameter, the GA starts over to tune a new ANN for the second design parameter. The tuning process is the same, except for the fact that the second ANN takes the output of its predecessor as additional input. Once the second ANN is found, it is also saved for future use. For the third ANN, the outputs of both its predecessors are used as additional inputs, and it is also saved once defined. A GA-tuned ANN is added for each new design parameter to output and, each time, the added inputs bring contextual constraints to the learning process to find it. In the end, there are as many ANNs as there are design parameters to determine, each ANN tuned by a GA.

Table II. MLP chromosome.

Bit #	Role in ANN	Range of values
1	Type of ANN	0 (MLP)
2-5	Design parameter to output	One of #7 to #16 in Table I
6-12	Nb. of stimuli used for training	100 to 227
13-17	Nb. of hidden layer neurons	2 to 62
18	Output function of hidden layer	logsig or tansig
19-21	Nb. of epochs used for training	25 to 200
22-24	Learning rate	10^{-5} to 10^{2}
25-27	Decrease in learning rate	10^{-8} to 10^{-1}
28-30	Increase in learning rate	10 to 10^{8}

Table III. RBF chromosome

Bit #	Role in ANN	Range of values
1	Type of ANN	1 (RBF)
2-5	Design parameter to output	# 7 to 16 (see Table I)
6-12	Nb. of stimuli used for training	100 to 227
13-22	Width of Gaussian RBF (σ)	0.005 to 5.12
23-30	Max. Nb. of hidden layer neurons	1 to 128

III. VALIDATION

A. Experimental Set-up

To build the training and test sets for the ANNs, two-hundred-thirty-five valid LNA designs corresponding to the circuit of Fig. 1a were randomly specified in CMOS 0.18 μm technology from TSMC, and simulated with the spectreRF circuit simulator, each design requiring several simulations and iterations to complete. The design methodology was performed by first selecting the value for L_D to achieve a certain frequency of operation. Then, the size and bias of Q_1 were chosen to generate different design specifications. The size of Q_2 was also varied for each design in order to generate information about the impact of Q_2 on specifications. In addition, L_S was varied to reach different linearity specifications. For each design, the input and output were matched by using the network topology shown in Fig. 1b. Table 1 summarizes the design variables and the ranges of values that were considered as mentioned previously.

The obtained designs were normalized to the range [0, 1] before use for training and testing ANNs. Then, they were split into a training set and a testing set for each ANN instance, with the respective sizes set by the GA. Upon GA exit, the ANN in the chromosome having achieved the best component value prediction performance during testing was selected. An output value was considered to be correct when the difference with its corresponding target value was within 0.05. All the experiments were realized with the Matlab NN Toolbox.

Generation 0 of a GA consisted of 128 30-bit chromosomes. Half of the population corresponded to MLPs and the other to RBFs. The values of the remaining 29 bits in a chromosome

were attributed randomly based on their definitions in Table II or Table III. Each ANN specified by a chromosome used the inputs defined previously, and one design parameter was selected as output at the end of the GA run. As already mentioned, the chromosomes stopped evolving when one ANN reached 95 % or more prediction accuracy of one RF-LNA component value on its test set. A test set was an input-target pair that was not seen by the ANN during training.

A total of ten GAs were generated in the same manner, each one selecting a different design parameter to output by its winning ANN, with the ANN taking as input all the performance criteria in addition to the outputs from the previous winning ANNs.

B. Results

The experiment was run twenty times on a computer station equipped with an Intel Core i7 2.67 GHz processor and 9 GB of RAM, and running under Windows 7 Professional. The proposed method successfully found the ten design parameters on all twenty simulations. The mean duration for a simulation was 5.375 h, with a standard deviation of 2.81 h, and once a correct ANN sequence was found, the mean time to find the correct set of design parameters given any set of performance criteria was 290 ms ± 27 ms. Table IV presents the mean number of generations it took the GA to select a successful ANN for each of the ten ANN levels.

Table IV. Number of generations in the GA to reach generalized learning.

ANN level	Number of generations required (mean ; standard deviation)
1	1.70 ; 0.86
2	1.20 ; 0.41
3	1.05 ; 0.22
4	1.10 ; 0.45
5	1.30 ; 0.57
6	1.15 ; 0.37
7	1.15 ; 0.37
8	1.60 ; 0.99
9	1.65 ; 1.18
10	3.65 ; 3.22

IV. DISCUSSION AND CONCLUSION

Although the described method was tested only with a specific design, it is generic in nature and extends beyond the RF-LNA design that is described here. As stated in Introduction, the design cycle for a RF-LNA is usually expert-based and time consuming. Moreover, the process must be started over for each new set of performance criteria. On the other hand, the method proposed here only needs a couple of hours to generate a sequence of ANNs capable of correctly specifying the design parameters for any performance criteria required for a given topology. Once started, the algorithm does not need user supervision, and once an adequate sequence of ANNs has been found, it takes less than a second to obtain a design meeting a different set of performance criteria.

Another finding to emphasize is the low number of generations required to find a relevant ANN by the GA. It took less than two generations on average for the first nine ANNs and less than four for the tenth one in each series. The mean time of over five hours to complete learning is attributed mostly to the fact that both the ANNs and the GA were simulated on a sequential computer, while both methods are parallel processing algorithms. Running them on adequate parallel computing machinery such as graphics processing units (GPU) would probably cut the learning time drastically. In any case, the response time of the system after completing learning is less than one second. This makes the proposed design methodology very fast in comparison to the current alternatives, once the final ANN-based model is specified.

A second experiment was conducted, which replicated the one presented with one exception: the GA also selected which of the possible inputs should define the inputs of the ANN. The reason for conducting the experiment came from the literature suggesting that redundancy in an ANN's input may impede its performance [7]. The idea of dealing with this issue by letting the GA select which inputs to include in the ANN definition was suggested in [4] with promising results, and it was thus considered here. However, our simulations showed that the method did not decrease the learning time, which remained approximately the same as before. Moreover, the mean number of generations required by the GA before finding a solution in each ANN level was significantly higher than in the first experiment. Therefore, the simplicity gained from reducing the number of inputs was counterbalanced by the time lost from augmenting the number of generations for learning. This suggests that, instead of gaining little in learning time by fine tuning the ANN and GA algorithms, more efforts should be put on physical implementation in parallel processing systems. It then becomes possible to gain substantially in learning time, even with the more basic algorithms.

ACKNOWLEDGMENT

This work was possible thanks to financial support from NSERC and ReSMiQ.

REFERENCES

[1] M. V. Korovkin, V. L. Chechurin, M. Hayakawa, Inverse Problems in Electric Circuits and Electromagnetics. Springer, 2007.

[2] K. Gallagher and M. Sambridge, "Genetic algorithms: a powerful tool for large-scale nonliner optimization problems", Computers & Geosciences, vol. 20, no. 7/8, pp. 1229-1236, 1994.

[3] M. Boukadoum, F. Nabki and W. Ajib, "Towards the neural network-based design of nadiofrequency now-noise amplifiers", proc. ISCAS 2012, Seoul (S. Korea), pp. 2741-2744, May 2012.

[4] E. Dumesnil, F. Nabki and M. Boukadoum, "RF-LNA Circuit Synthesis by Genetic Algorithm-Specified Artificial Neural Network", (submitted)

[5] S. Haykin, Neural Networks: a Comprehensive Foundation (2nd edition). Upper Saddle River, NJ: Prentice Hall, 1999.

[6] M. D. Buhman, Radial Basis Functions: Theory and Implementations. Cambridge University, 2003.

[7] J. Mohamad-Saleh and B. S. Hoyle, "Improved neural network performance using principal component analysis on matlab", International Journal of The Computer, the Internet and Management, vol. 16, no. 2, pp. 1-8, May-August, 2008.

[8] M. M. Mitchell, An Introduction to Genetic Algorithms, Cambridge, Ma: MIT Press, 1998.