2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)
DOI: 10.1109/ICECS.2014.7050096

RF-LNA Circuit Synthesis by Genetic Algorithm-Specified Artificial Neural Network

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—**A genetic algorithm (GA) was used to determine the optimal architecture and input parameters of a feed-forward artificial neural network (ANN), the purpose of which was to synthesize a radio-frequency, low noise amplifier (RF-LNA) circuit. The parameters (chromosomes) processed by the GA included: i) the LNA performance specifications and design constraints; ii) the type of ANN to use –multi-layer perceptron (MLP) or radial-basis function (RBF) network –; iii) the ANN parameters to set. For two different sets of design parameters, the input/output matching network components and transistor geometries, the GA found ANN solutions capable of predicting their values with success rates above 99 %.**

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

I. Introduction

Contrarily to circuit analysis where finding circuit performance given a set of components, their interconnections and their values is generally a perfectly determined problem, circuit synthesis, which solves the inverse problem, is usually open-ended. At first, it may appear that any solution that yields the desired performance is acceptable, but this reasoning ignores implementation and design time constraints. In real-life, they are important factors in the overall design process, and they may force iteration of the design cycle to obtain satisfactory results. For instance, the current design of radiofrequency, low-noise amplifiers (RF-LNA) routinely requires hours, if not days of fine-tuning by an expert. One way to address the design faster and with fewer resources is to exploit the generalization capability of an ANN to solve new design problems, once they have been trained with older designs that have been optimized [1]. However, if the problem is underdetermined, there is the risk that the ANN provides solutions which, although acceptable formally, are not compatible with the desired target design. One way to prevent this is to constrain the ANN synthesis by adding design parameters as inputs, along with the required performance data. However, simply increasing the number of inputs in this manner may create redundant information, with a possible detrimental effect on ANN performance. Typically, principal component analysis (PCA) has been used to address this issue by building orthogonal inputs [4]. Unfortunately, the meanings of the components obtained through PCA are often ambiguous and thus provide little insight regarding the variables at play.

In the literature, very few reports exist of ANNs capable of circuit synthesis, and none use design constraints [e.g. 2, 3]. Moreover, they address a very small set of design parameters, with a maximum of three component values reported so far [2].

In this work, we propose to use a GA to select which inputs – performance criteria and design constraints – should be fed to the ANN. GAs are known to be good at dealing with nonlinear problems, particularly in the context of optimization [5]. Moreover, they offer the advantage over PCA of preserving the meaning of the dimensions they select. We also use the GA to select between a MLP and a RBF neural architecture and determine its parameters for the problem at hand. We will show that this combined GA-ANN approach can efficiently solve the RF-LNA synthesis problem that we considered.

In summary, this work describes a methodology to generate an ANN capable of RF-LNA circuit synthesis in an optimal or close to optimal way (in terms of speed and accuracy of design). More precisely, the ANN must find the correct component values to meet set performance specifications and design constraints. In what follows, the RF-LNA circuit that we used as example is portrayed first. Next, the ANN types to be parameterized by the GA are described. Then, the genetic algorithm to select the fittest ANN with its parameters is explained. Finally, the experimental validation results are presented, along with concluding remarks.

II. Methodology

A. Radiofrequency Low Noise Amplifier

The circuit topology realizing the RF-LNA is the same as in [1] and is presented in Fig. 1. It consists of a cascode common-source stage with degeneration and inductive load. The cascode configuration was selected for its design simplicity (easier matching, 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. 1b is considered for 50 Ω source and load impedance matching. Its component values are output by the ANN.

Table I lists the design (D) and performance (P) variables that we used in this work, along with their ranges of values that were randomly generated during the validation experiments. Notice that the drain inductance (L_D in Fig. 1) is not mentioned, as it was fixed at 0.578 nH for all designs.

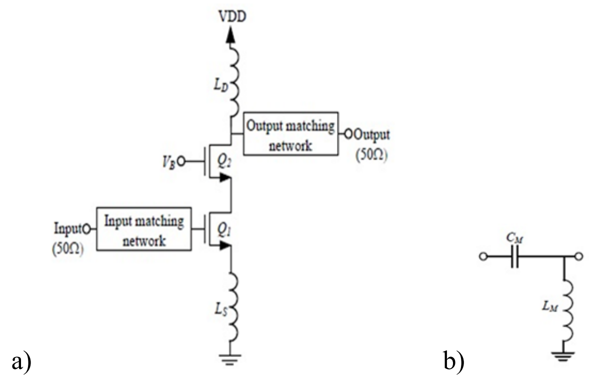

Fig 1. Used LNA topography (a) and matching network (b)

B. Neural Architecture

Two types of ANN architectures are considered, the multilayer perceptron (MLP) [6] and the radial basis functions (RBF) model [7]. Both consist of three layers of neurons in sequence: an input layer, a hidden layer and an output layer.

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

$$z_j = f_\theta\left(\sum_{i=1}^{M} \omega_i x_i\right) \qquad (1)$$

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

In the output layer, the output of a neuron is the summation of its inputs and is thus given by

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

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 a weight that is also set by the learning algorithm.

2) RBF neural network: The input layer is the same as for the MLP. In the hidden layer, the output of the j^{th} neuron is given by

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

where N is the number of neurons in the hidden, $\boldsymbol{x}$ is the vector in M dimensional space representing the input layer and φ is a basis function, typically a Gaussian. Thus for the j^{th} hidden layer neuron, we have $\varphi_j(\boldsymbol{x}) = e^{\left(-\frac{1}{2\sigma_j^2}\|x-x_j\|^2\right)}$, where $\boldsymbol{x}_j$ and σ_j^2 are respectively the center vector and the width of the corresponding Gaussian basis function, both to be determined by a learning algorithm. In the output layer, the output of a neuron is the same as for the MLP and is thus given by (2).

Both the MLP and the RBF can be trained to find their weights with the error back-propagation algorithm, which tries to minimize the mean square error between the desired and the predicted outputs. During training, input-target pairs are given as examples; then, a set of new input-target pairs is used to verify if the ANN is capable of generalizing what it learned.

3) Genetic algorithm: Since the number of combinations of ANN parameters is susceptible to combinatorail explosion, a GA was designed to select the best ANN type (i.e. MLP or RBF), its optimal parameters and its best predictive inputs.

The steps for creating a population of ANNs to evolve were as follows: first, a k-bit binary chromosome structure is defined, where each bit may indicate the presence/absence of an input variable, the type of ANN to be used, or the value of an ANN parameter. Then, N instances of the chromosome structure are randomly created, forming the *generation 0* population, in which a fitness function measures how well the ANN specified by each individual can generalize what it learns form a training set to new designs (see Validation).

Half of the population is selected for reproduction, where the probability for an exemplar to be selected is proportional to its fitness. The two fittest survivors then have a pair of children together, as do the next two and so on. Each child is the result of a crossover process between its parents' chromosomes. 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 suscep1tible to mutation with a certain probability (0.001 in this work), which means that each bit could be inverted. The population of N exemplars that reached this stage is called *generation 1*. Then, the selection, reproduction and mutation cycle take place over and over again, each cycle resulting in a new generation. The GA stops when the success rate of the best ANN in a generation reaches a threshold (99 % here). Success was defined as the correct prediction of the LNA component values for new designs (i.e. correct generalization).

Table I. Performance and design variables considered.

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

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 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 values for L_D to achieve a certain frequency of operation. Then, the size and biasing for Q_1 were chosen to generate different specification designs. The size of Q_2 was also varied for each design generated in order to create designs with information on 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. As mentioned previously, Table 1 summarizes the design variables and the ranges of values that were considered for the designs.

The obtained designs were first 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 the GA consisted of 128 37-bit chromosomes. Half of the population corresponded to MLPs and the other to RBFs. The values of the remaining 36 bits were attributed randomly; they are defined in Table II and Table III. As mentioned before, the GA stopped when a chromosome reached a success rate above 99 % for predicting the LNA component values on its test set. A test set was an input-target pair that was not seen by the ANN during training.

Initially, the ANN outputs corresponded to four design variables: the *input capacitance* and *input inductance* of the LNA input matching network and the *output capacitance* and *output inductance* of its output matching network. These correspond to variables 7 to 10 in Table I. The inputs of the ANNs were selected by the GA. Twelve bits were dedicated to this characterization: six bits to establish which of the performance variables (# 1 to 6 in Table I) should be included and six more bits to decide which design constraints (# 11 to 16 in Table I) were to be added. Even though the problem at hand is one of synthesis, the design constraints were presented to the GA as possible inputs because of the underdetrmination of the synthesis problem as explained previously. Without them, it is possible that an ANN will produce an output that does not correspond to the tested targets while still representing an adequate solution. Giving the GA the opportunity to consider the remaining six design variables should help in constraining the problem and thus reduce the level of under determination.

We tested the methodology's genericity by reusing it for a different set of output variables. They became the transistor geometries: *transistor 1 length*, *transistor 1 width*, *transistor 2 length* and *transistor 2 width* (variables 11 to 14 in Table I), and the values of the input and output matching networks inductances and capacitances (variables 7 to 11 in Table I) became constraining inputs.

Table II. MLP chromosome.

Bit #	Role in ANN	Possible values in ANN
1	Type of ANN	MLP (= 0)
2-13	Presence/absence of each input	Table I for list of inputs
14-20	Qt. of stimuli used for training	100 to 227
21-24	Qt. of nodes in hidden layer	2 to 17
25	Output function of hidden layer	logsig or tansig
26-28	Nb. of epochs used for training	25 x k, k = 1,…,8
29-31	Learning rate	10^{-5} to 10^{2}
32-34	Decrease in learning rate	10^{-8} to 10^{-1}
35-37	Increase in learning rate	10 to 10^{8}

Table III. RBF chromosome

Bit #	Role in ANN	Possible values in ANN
1	Type of ANN	RBF (= 1)
2-13	Presence/absence of each input	Table I for list of inputs
14-20	Qt. of stimuli used for training	100 to 227
21-24	Width of Gaussian RBF (σ)	0.1 x k, k = 1,…,16
25-32	Max. qt. of hidden neurons	1 to 256
33-37	Hid. neur. added between inputs	1 to 32

B. Results

For experiment one, Table IV describes the first chromosome to reach a success rate above 99 % for its test set. It corresponds to a RBF and the GA took four generations to converge to it. To verify the stability of generalization capability of this RBF further, we realized 100 additional simulations where we randomly varying which data served for training and which were used for testing. The mean accuracy of the RBF for the test data over these simulations was 99.22 %, with 100 % for the best simulation and 95 % for the worst.

The time needed by the GA to complete a cycle was approximately 11 minutes, for a total convergence time of 44 minutes. Then, the one hundred simulations applied to the winning RBF took a total of approximately 3 minutes to run. Thus, the ANN specification took less than one hour to complete. Then, using it for any new input vector within the ranges of those presented in Table I required 62 ms to yield the desired outputs. The previous time values were obtained on a computer with Windows 7, an Intel Core i7 2.66 GHz processor and 4 GB of RAM.

For the second experiment, the first chromosome to reach a success rate above 99 % for the test set corresponded to a MLP. Eight generations were necessary to obtain the chromosome, which is described in Table V. Again, we verified the ANN generalization stability over 100 additional simulations, obtaining a mean accuracy on the test data of 95.23 %, with 100 % on the best simulation and 88.46 % on the worst. This time, the GA took 88 minutes to converge. The one hundred validation simulations took a total of 10 minutes and applying the MLP to one new input vector required 62 ms to yield the output.

The results for the matching networks as outputs and the transistors geometries as outputs are summarized in Table VI.

IV. Discussion and Conclusion

The work used a GA to set the parameters of an ANN for successfully predicting the component values for a RF-LNA given certain performance criteria and design constraints. The obtained validation results have shown that, for two different sets of target variables, impedance matching networks and transistor geometries, the GA was successful at determining the required ANN architecture.

Since the ultimate objective of the approach is to speed up the LNA design cycle, the time required for running the GA and validating the stability of the winner ANN's generalization capability is important. Our results show that the time required for getting and validating an appropriate ANN using a GA represents an improvement over current methods. Moreover, using this ANN for subsequent applications is extremely time efficient (approximately 60 ms response time).

We investigated the model further by comparing the performance of the best chromosome obtained to that of the chromosome produced by a GA that would only consider performance data as input. We suggested previously that the possible underdetermined character of the design problem may lead to solutions inconsistent with the input-target test pairs fed to the ANN. The results, presented in Table VI, show that ignoring design constraints reduces the chromosome performance stability, both when the outputs are the matching networks and when they consisted in the transistors geometries.

We also compared the performance of the best chromosome obtained to that of the chromosome produced by a GA that uses all possible inputs (i.e. all performance criteria and all design constraints in Table I). The results in Table VI show that forcing all inputs on the ANN also reduces the stability of the chromosome's performances, but in a less important manner than ignoring the design constraints.

Only two ANN architectures were considered in this work. Among other kinds of ANNs, one has recently shown encouraging signs for circuit synthesis applications: the bidirectional associative memories (BAM) [8]. When set up for heteroassociative operation, it appears to have the potential to reconstruct the exemplar that led to a category label [9]. This process is similar to finding a circuit component values given a set of performance specifications. Thus, applying a GA to this type of ANN may be a promising avenue in circuit synthesis (work planned). Work is also needed to automatically set the design constraints instead of specifying them manually. This could be done by auxiliary ANNs that supply the needed information based on the training data (work in progress).

Acknowledgment

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

Table IV. Description of the best chromosome with matching networks components values as outputs (after four generations).

Bit #	Role in ANN	Actual value in ANN
1	Type of ANN	RBF
2-13	Inputs present (see Table 3)	2, 3, 5, 11, 14, 15, 16
14-20	Qt. of stimuli used for training	224
21-24	Width of Gaussian RBF (σ)	0.3
25-32	Max. qt. of hidden neurons	111
33-37	Hid. neur. added between inputs	21

Table V. Description of the best chromosome for transistors geometries values as outputs (after eight generations).

Bit #	Role in ANN	Actual value in ANN
1	Type of ANN	MLP
2-13	Inputs present (see Table 3)	3, 4, 5, 7, 8, 15, 16
14-20	Qt. of stimuli used for training	222
21-24	Qt. of nodes in hidden layer	8
25	Output function of hidden layer	tansig
26-28	Nb. of epochs used for training	150
29-31	Learning rate	10^{-2}
32-34	Decrease in learning rate	10^{-3}
35-37	Increase in learning rate	10^{2}

References

[1] 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.

[2] M. Rahnama, Y. M. Gilmanek and A. M. Kordalivand, "Ultra wide-band LNA using RFCMOS technology and tunability band with neural network", Control and System Graduate Research Colloquium (ICSGRC), pp.75-79, June 2010.

[3] C. Pandit, A. Patnaik and S. N. Sinha "Neural network based CAD models for analysis and design of fin-lines for mm-wave applications", Applied Electromagnetics Conference, pp.1-4, December 2007.

[4] 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.

[5] 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.

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

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

[8] B. Kosko, "Bidirectional associative memories", IEEE Transactions on Systems, Man and Cybernetics , vol. 18, pp. 49-60, 1988.

[9] S. Chartier and M. Boukadoum, "A bidirectional heteroassociative memory for binary and grey-levelpatterns", IEEE Transactions on Neural Networks, vol. 17, no. 2, pp. 385-396, 2006.

Table VI. Testing (generalization) performance of the best chromosome found during the GA and in the following 100 ANN simulations.

Outputs (see Table I)	Inputs (see Table I)	During GA		100 simulations		
		Nb. of generations for success rate > 99 %	Best success rate	Mean success rate	Maximum success rate	Minimum success rate
Matching networks (7 to 10)	GA among 1 to 6 and 11 to 16	4	100.00 %	99.22 %	100.00 %	95.00 %
	GA among 1 to 6	6	100.00 %	88.67 %	100.00 %	75.00 %
	1 to 6 and 11 to 16 imposed	14	100.00 %	98.50 %	100.00 %	91.67 %
Transistors geometries (11 to 14)	GA among 1 to 10 and 15 to 16	8	100.00 %	95.23 %	100.00 %	88.46 %
	GA among 1 to 6	11	100.00 %	86.23 %	100.00 %	71.76 %
	1 to 10 and 15 to 16 imposed	16	100.00 %	92.47 %	100.00 %	80.19 %