POLITECNICO DI TORINO

SCUOLA DI DOTTORATO
Dottorato in Dispositivi Elettronici – XVIII ciclo

Tesi di Dottorato

Nonlinear RF/Microwave device and system modeling using neural networks

Georgina S. Stegmayer Machado

Tutore
prof. Carlo Naldi

Coordinatore del corso di dottorato
prof. Carlo Naldi

Dicembre 2005

Summary

The aim of this PhD thesis is to highlight the relevance of the neural network paradigm application as a very useful tool inside the new behavioral approach for modeling RF/Microwave devices or (sub)systems. This thesis proposes a model that has the capability to learn and predict the dynamic behavior of nonlinear devices based on a Time-Delayed Neural Network (TDNN) model. This model can be trained with input/output device measurements and a very good accuracy can be obtained in the device simulation easily and rapidly. The TDNN model has been validated with transistor and Power Amplifier(PA) time-domain measurements. The properties of this kind of model make it specially suitable for new wireless communications components modeling and is of particular interest for the industry, because it allows fast device model development and characterization.

Furthermore, a new method has been found for the identification of Volterra kernels in the time domain, suitable for representation of a nonlinear electronic device. This new algorithm allows extraction of the Volterra kernels directly from the neural network model parameters, and the resulting model represents a very good approximation of the nonlinear behavior, also for strong nonlinearities, with a limited number of kernels.

In summary, the purposes of this thesis are, on the one hand, the proposal of a new behavioral model neural network-based and a simple approach for the obtention of its corresponding Volterra series, which can be applied even in the case of a nonlinearity that depends on more than one variable, and that allows to obtain a general model independent of the physical circuit under study. On the other hand, a prototype of a software toolbox that has been developed as result of this research work, with an intuitive interface for creation and use of neural network-based models for RF/Microwave devices and that could be implemented inside a commercial RF simulator.

Acknowledgments

The process of researching and writing this PhD thesis has been mainly conducted at Dipartimento Elettronica - Politecnico di Torino (Italy) and partially with the group GIDSATD of Universidad Tecnológica Nacional - Facultad Regional Santa Fe (UTN-FRSF, Argentina).

First of all I want to gratefully acknowledge the person who has believed in me and has given me the exceptional opportunity of doing my PhD in Europe, my tutor Prof. Carlo Naldi. I want to thank also Prof. Giovanni Ghione for his helpful guidance and Prof. Marco Pirola for his patience, for offering me advise and support, as well as a sincere friendship. I cannot forget to mention Prof. Giancarlo Orengo, a friend from Università Tor Vergata di Roma, for the fruitful discussions and collaboration research activities that we had together. Very special thanks go to Dr. Omar Chiotti and Dra. Ma. Rosa Galli, from UTN-FRSF, for inspiring me in the pursuit of a research career and for giving me the tools that lead me here today.

There are many people who have been a great company during all these years of work. My friends at Politecnico di Torino: Jorge, Daniel, Valeria, Vittorio, Alessandro and Luca. My friends at UTN-FRSF: Ma. Laura, Luciana, Mariel, Mariela and Pablo; and the student who helped me with the implementation part of this thesis, Mariano. I cannot forget to mention my apartment-mates Anna, Chiara B., Chiara P., Elisabetta and Michela, because they made me feel "home" while I was in Turin.

I want to particularly thank my parents and my family for their support all these years, because without them I would not be here today.

Finally, and above all, I dedicate this thesis to Federico (my husband, my friend, my advisor, my partner, my all) who has encouraged me and supported me with his love, all these years, always.

Turin, December 2005

Table of contents

Chapter 1

Introduction

Radio-Frequency (RF) and Microwave Engineering are growing rapidly in importance, stimulated in particular by the exceptional world-wide growth in digital mobile communication systems and wireless applications.

Modern communications systems composed by RF/Microwave electronic devices or sub-systems generally have nonlinear behavior, rich in high-frequency dynamics. For analysis and design purposes, they have to be replaced by realistic models, which must particularly take into account the nonlinearity, which may influence the whole systems performance. The wireless communications market is commercially very important and therefore there is a big interest in finding good suitable models to use in the RF system design process.

Models have to fulfill two main requirements: first, they should be built in an efficient way and should represent a device or system with a sufficient accuracy. Second, they could be developed to replace existing too much detailed (and therefore, costly to build and hard to simulate) models.

The first requirement holds particularly for the semiconductor industry, subject to continuous technological improvements which lead to new or significantly modified technologies. In the second case, the objective is to improve model efficiency without sacrificing too much model accuracy. Moreover the model should be predictive, that is to say, it should not only closely match the device behavior, but also yield a response for situations not specified in the dataset near to the real behavior of the element studied.

This PhD thesis proposes a new model which tries to fulfill both these requirements. The proposed model can be applied for the modeling of electronic components of a communication system (devices or sub-systems), reproducing accurately their nonlinear and dynamic behavior using a neural network-based approach. Moreover, this thesis proposes a prototype of a software toolbox for the creation and use of the proposed model and later implementation inside a commercial circuits simulator.

The structure of this PhD thesis is the following:

Chapter 2 Introduces existing and new modeling techniques for RF/Microwave nonlinear devices analysis.

Chapter 3 Discusses in detail the approach followed in this thesis, behavioral modeling, and the different techniques that can be used for building such models, i.e. neural networks.

Chapter 4 Presents the novel behavioral model neural network-based for nonlinear electronic device modeling and how it can be used to obtain a Volterra series representation.

Chapter 5 Presents the results obtained from the application of the proposed model for the modeling of electronic devices and circuits.

Chapter 6 Shows examples of the application of the proposed software toolbox prototype to support the creation and use of behavioral models neural network-based for electronic devices simulation.

Chapter 7 Finalizes this thesis with the conclusions an analysis of the results.

Appendix A The appendix explains in detail the procedure that allows calculating the Volterra kernels from the parameters of a neural network model and shows the codification of the algorithm.

A list of references and an index complete the presented work.
Results of this PhD thesis are the following publications and awards:

Congresses Proceedings

1. G. Orengo, P. Colantonio, A. Serino, F. Giannini, G. Stegmayer, M. Pirola and G. Ghione, "Time-Domain Neural Network Characterization for Dynamic Behavioral Models of Power Amplifiers", European Microwave Week (GaAs Simposium), Paris, France, October (2005).

2. G. Stegmayer and O. Chiotti, "Identification of frequency-domain Volterra model using Neural Networks", 15th International Conference on Artificial Neural Networks (ICANN), CD-ROM proceedings, ISBN 3-540-28752-3, Warsaw, Poland, 12-15 September (2005).

3. G. Stegmayer, "Comparison of Volterra models extracted from a Neural Network for nonlinear systems modeling", 15th International Conference on Artificial Neural Networks (ICANN), CD-ROM proceedings, ISBN 3-540-28752-3, Warsaw, Poland, 12-15 September (2005).

4. G. Stegmayer and O. Chiotti, "Modeling New Generation Wireless Communications devices with Neural Networks", VII Argentine Symposium on Artificial Intelligence (ASAI), CD-ROM proceedings, ISSN 1666-1141, pp. 137-147, Rosario, Argentina, 29-30 August (2005).

5. G. Stegmayer, "Neural-based Identification for Nonlinear Dynamic Systems", IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA), ISBN 0-7803-9026-1, Giardini Naxos, Italy, 20-22 July (2005).

6. G. Stegmayer, O. Chiotti and G. Orengo, "A Neural Network that helps building a Nonlinear Dynamical model of a Power Amplifier", 13^{th} European Symposium on Artificial Neural Networks (ESANN), ISBN 2-930307-05-6, pp. 539-544, Bruges, Belgium, April (2005).

7. G. Orengo, P. Colantonio, A. Serino, F. Giannini, G. Stegmayer, M. Pirola and G. Ghione, "Time-Delay Neural Networks for Dynamic Behavioral Models of Power Amplifiers", TARGET Workshop on RF Power Amplifiers, ISBN, pp. 87-90, Orvieto, Italy, April (2005).

8. G. Stegmayer, M. Pirola, G. Ghione, C. Naldi, G. Orengo and P. Colantonio, "Neural Network based Volterra back-box model for RF devices", Workshop on Integrated Nonlinear Microwave and Millimetre-wave circuits (INMMIC), ISBN 88-88748-34-2, pp. 121-124, Rome, Italy, November (2004).

9. G. Stegmayer and O. Chiotti, "Novel Neural Network application to nonlinear electronic devices modeling: building a Volterra series model", VI Argentine Symposium on Artificial Intelligence (ASAI), CD-ROM proceedings, ISSN 1666-1079, Cordoba, Argentina, 20-24 September (2004).

10. G. Stegmayer and O. Chiotti, "The Volterra representation of an electronic device using the Neural Network parameters", XXX Latin-American Conference on Informatics (CLEI), CD-ROM proceedings, ISBN 9972-9876-2-0, pp. 266-272, Arequipa, Peru, 27-30 September (2004).

11. G. Stegmayer, "Volterra series and Neural Networks to model an electronic device nonlinear behavior", IEEE International Joint Conference on Neural Networks (IJCNN), vol. 4, ISBN 0-7803-8359-1, pp. 2907-2910, Budapest, Hungary, July (2004).

12. G. Stegmayer, M. Pirola, G. Ghione, G. Orengo and P. Colantonio, "Volterra back-box model of electron devices nonlinear behavior based on Neural Networks parameters", 8^{th} WSEAS International Conference on Circuits, Systems, Communications and Computers (CSCC'04), CD-ROM proceedings,

ISBN 960-8052-99-8, Athens, Greece, 12-15 July (2004).

13. G. Stegmayer, M. Pirola, G. Orengo and O. Chiotti, "Towards a Volterra series representation from a Neural Network model", WSEAS/IASME International Conference on Neural Networks and Applications (NNA), CD-ROM proceedings, ISBN 960-8052-96-3, Udine, Italy, 25-27 March (2004).

Journals

1. G. Orengo, P. Colantonio, A. Serino, F. Giannini, G. Stegmayer, M. Pirola and G. Ghione, "Neural Networks and Volterra-series for Time-domain PA Behavioral models", *International Journal of RF and Microwave CAD Engineering*, ISSN 1096-4290, Ed. John Wiley & Sons Inc., in press (2006).
2. G. Stegmayer and O. Chiotti, "Identification of frequency-domain Volterra model using Neural Networks", *Lecture Notes on Computer Science 3697*, ISSN 0302-9743, Ed. Springer-Verlag, pp. 465-471 (2005).
3. G. Stegmayer, "Comparison of Volterra models extracted from a Neural Network for nonlinear systems modeling", *Lecture Notes on Computer Science 3697*, ISSN 0302-9743, Ed. Springer-Verlag, pp. 457-463 (2005).
4. G. Stegmayer and O. Chiotti, "Novel Neural Network application to nonlinear electronic devices modeling: building a Volterra series model", *Revista Iberoamericana de Inteligencia Artificial*, ISSN 1137-3601, Ed. AEPIA, in press (2005).
5. G. Stegmayer, M. Pirola, G. Ghione, G. Orengo and P. Colantonio, "Volterra black-box model of electron devices nonlinear behavior based on Neural Networks parameters", *WSEAS Transactions on Circuits and Systems*, WSEAS Press, ISSN 1109-2734, vol. 3, no. 5, pp. 1140-1144, July (2004).
6. G. Stegmayer, M. Pirola, G. Orengo and O. Chiotti, "Towards a Volterra series representation from a Neural Network model", *WSEAS Transactions on Systems*, WSEAS Press, ISSN 1109-2777, vol. 3, no. 2, pp. 432-437, April (2004).

Awards

- On September 2005 this PhD thesis has won a prize given by Banco RIO S.A. and Portal Universia (ARG) in a competition of research projects to impulse the regional development in Argentina (http://www.universia.com.ar/becasypremio). The prize will be used to fully develop the software toolbox proposed in this thesis.

- 2nd place on the poster competition at "Escuelas de Verano en Inteligencia Computacional" (EVIC 2004) - IEEE Computational Intelligence Society, Santiago de Chile, Chile, December 2004. Jury: Dr. B. Widrow, Dr. Y. Zurada, Dr. B. Eberhardt.
- 2nd place (silver medal) on the competition "Primer Torneo Argentino de Redes Neuronales Aplicadas" (TAR I) - IEEE Capitulo Argentino, Buenos Aires, Argentina, August 2004. Jury: Ing. R. Veiga, Dr. B. Zanutto, Dr. M. Mayosky.

Chapter 2

Modeling approaches

This chapter introduces existing classical and new modeling approaches in the RF/Microwave field. It underlines and exemplifies the differences between old and new modeling techniques, at both device and system-level.

2.1 Introduction

No general theory is available for modeling. The classical (and generally followed) approach for modeling an electronic device is building a representation for it in the form of a circuit derived from a physical model, whose internal relationships (i.e. current-voltage, charge-voltage) and elements topology (connections between them) have to be determined according to the formalisms of the Electronics Circuit Theory, as illustrates figure 2.1.

This procedure is based on the known physical behavior of the modelled element that dictates the model topology [68]. The physical circuit is measured and the measured parameters are mathematical quantities which have to be related to the circuit model through equations. For this reason, not only the model must be tailored to the device, but also the extraction of its parameters strongly depends on the device or system under study. Parameter extraction is the procedure of comparing the values of some parameters of interest obtained from the circuit model, against those actually measured in the physical circuit. If they do not compare satisfactorily, a more adequate model must be found.

A different approach, that claims to be able to represent any general device or system, is referred to as a behavioral modeling. A behavioral model tries to estimate both the functional form of the relations between variables and the numerical parameters in those functions.

Neural Networks are nowadays being used as black-box behavioral models, because they usually do not make assumptions regarding the data they represent

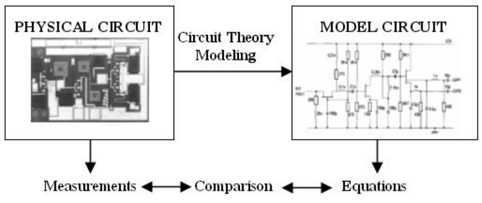


Figure 2.1. Classical modeling approach.

[77][93]. As an analytical behavioral model, the Volterra series expansion approach has been proposed since many years for nonlinear electronic devices modeling [85][95]. However, due to the difficulties and complexity in identifying the terms of the Volterra series through experimental data, this approach has not been effectively exploited within commercially available circuit simulators.

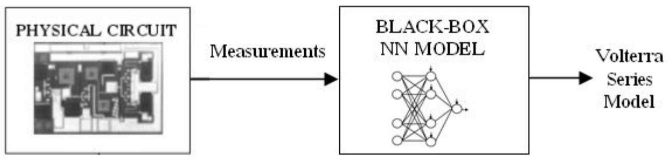


Figure 2.2. Novel behavioral neural network-based (NN) approach for modeling.

The objective of this thesis is to propose a solution to this last problem in the Volterra series building, and at the same time to present a general behavioral model, independent of the physical circuit modelled and of the number of controlling variables, receiving only simple and standard device measurements used to train a neural network-based model (a black-box model), able to generate the corresponding Volterra series model, as shows schematically figure 2.2.

In the next sections, both physical-circuit and behavioral modeling approaches are briefly described. In the second part of the chapter, some examples of these techniques are given.

2.2 Physical and circuit approach

In the traditional approach to the design of RF/microwave devices, they are modeled by electrical models characterized through standard or on-chip measurements performed on manufactured prototypes. Some steps must be followed when developing the model [10]. First of all, the important physical variables and phenomena must be identified through a careful low-level analysis of the device physics. Then, the next step consists in formulating the relevant physical equations which relate the internal physical variables to each other and to the external terminal currents and voltages. These (generally nonlinear) equations must be solved.

Physics-based device modeling relates the physical parameters set β of a given semiconductor device (e.g. $\beta =$ activated doping profile, actual gate length, gate depth, surface or substrate state density, etc.) to its electrical parameters set γ (i.e., $\gamma =$ frequency-dependent S-parameters, DC characteristics, transconductance, junction capacitances, noise parameters, etc.). Circuit analysis finally provides the link between the electrical device parameters and the corresponding circuit performance within the framework of an integrated CAD[1] environment (i.e. a circuits simulator) [21].

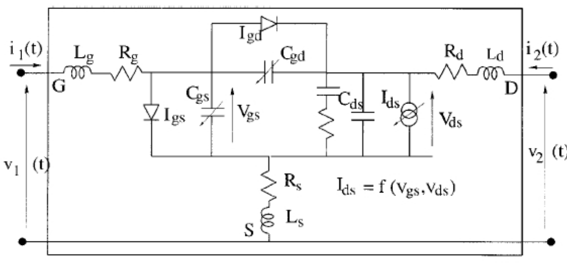


Figure 2.3. Two-port equivalent-circuit model.

To obtain a suitable compact model for circuit simulation, available physical knowledge has to be used to obtain a numerically well-behaved circuit model. The relationship of the circuit with the underlying device physics and physical structure is a key point when building such models. Assuming that a satisfactory mathematical model of a device D is obtained, the final step consists in synthesizing a circuit model, which uses both lumped[2] and distributed[3] components. A circuit model is often called an equivalent-circuit of D. Depending on the application (e.g. amplitude and frequency of operation) a given device may have many distinct models. An example of an equivalent-circuit model for a two-port[4] device is shown in figure 2.3.

Lumped equivalent electrical circuits offer the advantage of being computationally efficient and accurate, but at the expense of very complex model parameter extraction carried through numerical fitting and optimization, for which no physical insight is available [33]. The physics-based design approach makes direct use of physical and geometrical input data as the variables to be tuned during circuit

[1]Abbreviation for Computer Aided Design".

[2]A discrete component, that is self-contained, i.e. capacitors, resistors, inductors.

[3]An electrical property that is spread throughout a circuit or device, rather than being concentrated at one point, as in a discrete component, i.e. a transmission line.

[4]In a circuit, device, or system, a port is a point at which energy or signals can be introduced or extracted.

optimization. Accurate physical modeling of semiconductor devices cannot usually be achieved through computationally efficient analytical, or quasi-analytical, models. Instead, a numerical implementation of the models is often needed to accurately foresee the device performances in DC[5], AC[6] small signal and large-signal operation [5].

A model able to provide complete device performance prediction (DC characteristics, bias-dependent small-signal AC parameters, large-signal response, noise, temperature dependence) in terms of physical parameters alone, must be based on fundamental semiconductor equations. Accurate and general-purpose algorithms for the solution of these equations require the numerical discretization of sets of partial differential equations over a two or three-dimensional domain. As a consequence, numerical physics-based models are computationally intensive and therefore unsuitable for direct inclusion into CAD tools for circuit analysis and optimization. Moreover, an excellent agreement shown for a specific device does not guarantee that the physical model is able to accurately reproduce the variations in the electrical characteristics caused by variations in the physical input parameters [21].

A major disadvantage of the physical modeling approach is that it usually takes a lot of time to develop a good model for a new device. A quite detailed description of the device internal structure is needed, and then the construction of the equivalent-circuit components and its topology is a complex and time-consuming task, which not only requires experience but also is a difficult trial-and-error process.

It is less justifiable nowadays when new devices and technologies are constantly introduced into the market, which have to be characterized in a reasonable amount of time and with an acceptable accuracy. This is particularly true in the semiconductor industry, where continuous technological improvements lead to the appearance of new devices built with new technologies. As a result of rapid changes in semiconductor technology, development of models to represent the new transistor has become a continuous activity. These are besides the major reasons to explore alternative modeling techniques, such as the behavioral approach introduced in the following lines.

2.3 Behavioral approach

Behavioral modeling has appeared as an alternative to the traditional circuit-based approach. A behavioral model relies on directly measured data of a device D or system S, allowing to completely forget a circuit model topology. The only concern in this kind of model is the description of the element behavior that is observed at

[5]Abbreviation for Direct Current.
[6]Abbreviation for Alternating Current.

its accessible terminals or ports, that is to say, the model is simply built as a black-box from input/output measurement data or simulations. A block representation of such behavioral model for a two-port element is depicted in figure 2.4.

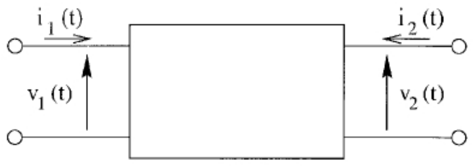


Figure 2.4. Two-port black-box model.

Behavioral models are particularly suited for nonlinear devices and systems modeling, whose internal operation, mechanism and composition are not well understood, but could be measured. The difficult and fundamental problem with this approach is to determine the model structure that best describes the real device or system under study. In other words, the key question that has to be answered for building a behavioral model is: from input/output measurements only, is it possible to identify a unique model topology or structure to represent the device behavior?

The answer to this question is clearly no for a linear element, because i.e. given any linear one-port device D, there exist many equivalent one-port models having distinct circuit topologies but the same behavior, which are not distinguishable from input/output measurements alone. However, this lack of uniqueness does not hold in the presence of nonlinearities. The same complexity which makes nonlinear systems difficult to represent, analyze and design, also allows much more information about the internal structure to be extracted from input/output measurements [11].

Behavioral models are considered as a black-box in the sense that no knowledge of the internal structure is required for model building, and the modeling information is completely included in the device external response [95]. The main advantage of this measurement-based modeling approach is that a low-order model of a complex circuit or system can be derived, without prior knowledge of its internal topology. A behavioral model is a simplified model of the essential behavior of an element at a given level of abstraction hierarchy (e.g. device or sub-system level), for use in simulation at the next highest level (e.g. system level) [65]. At a device-level, it is often possible to obtain discrete behavioral data from measurements or device simulations, such as a list of applied voltages and corresponding device currents, bias conditions, etc. At a system-level, power levels and input/output waveforms can also be measured or estimated. Behavioral models can be derivable from both simulations and/or measurements. Measurement based model extraction is typically used for model building, while simulation based modeling is typically used for reduction of model complexity [80].

A very important advantage of this technique is that one can in principle obtain a model of any required accuracy by providing a sufficient amount of (sufficiently accurate) discrete data. Considering for example a circuit model for an integrated circuit[7] (IC), the simulator can be used to stimulate the IC model and the responses used to derive a behavioral model for simulation at the next higher level of abstraction. Alternatively, behavioral models can be derived from real measurements made on the component, which can therefore lead to more accurate models than starting from lower level, possibly inaccurate equivalent-circuit simulations. This feature also helps the designers to make behavioral models from real components for which a circuit model may not be available.

2.4 Device-level modeling

Given a device D, it is physically impossible to measure its generally infinite collection *F(D)* of possible admissible voltage(v)-current(i) signal pairs. An alternative approach is to develop a device model *M(D)* made of some well-defined set of elements, so that each admissible voltage-current signal pair associated with *M(D)* represents a good approximation to a corresponding measurement ($v(t)$,$i(t)$) from D.

In general, the model *M(D)* contains several parameters which are determined by measurements on the given D. If not, a mathematical model can be developed, consisting of one or more equations, each of whose solution under a given excitation gives a good approximation to a corresponding pair *(v(t), i(t))* measured from D. These equations are generally nonlinear and may include a combination of algebraic equations, ordinary differential equations, partial differential equations or integral equations [10].

To incorporate into the model effects like signal propagation delay, *M(D)* may be constructed from several so-called quasistatic (sub)models. A quasistatic model consists of functions describing the static behavior of a model. Time is not an explicit variable in any of these functions, it only affects the model behavior via the time dependence of the input variables of the model. In an equivalent-circuit representation, obtained from the equations of the model, the Kirchhoff current law (KCL) relates the behavior of different topologically neighboring quasistatic models, by requiring that the sum of the terminal currents towards a shared circuit node should be zero in order to conserve charge [25]. It is through the corresponding differential algebraic equations that truly dynamic effects like delays are accounted for. Non-input, non-output circuit nodes are called internal nodes, and a model

[7]A circuit whose components (transistors, resistors, capacitors, inductors) and connecting wires are made by processing distinct areas of a chip of semiconductor material, such as silicon.

or circuit containing internal nodes can represent truly dynamic or non-quasistatic behavior.

A non-quasistatic model is simply a model that can represent the non instantaneous responses that quasistatic models cannot capture by themselves. A set of interconnected quasistatic models then constitutes a non-quasistatic model through the KCL equations. Essentially, a non-quasistatic model may be viewed as a small circuit by itself, but the internal structure of this circuit need no longer correspond to the physical structure of the device it represents, because the main purpose of the non-quasistatic model is to accurately represent the electrical behavior, not the underlying physical structure [52].

A comparison of examples at this level of analysis, using the circuit-based approach and behavioral modeling, are shown in the next subsection.

2.4.1 Example: transistor modeling

In the RF/Microwave field, the most commonly used transistor[8] nowadays inside integrated-circuits (ICs) as the basis for most digital electronic devices is the field-effect transistor (FET). It is a semiconductor amplifying device, which relies on an electric field to control the shape and the conductivity of a channel in a semiconductor material. Based on its input voltages, it allows a precise amount of current to flow through it. The terminals in a FET are called gate, drain and source. The voltage applied between the gate and source terminals modulates the current between source and drain. The FET is actually used as low-noise or power discrete element, as well as active device inside microwave ICs. An example of an equivalent-circuit model for a FET together with the physical meaning of the model topology are shown in figure 2.5.

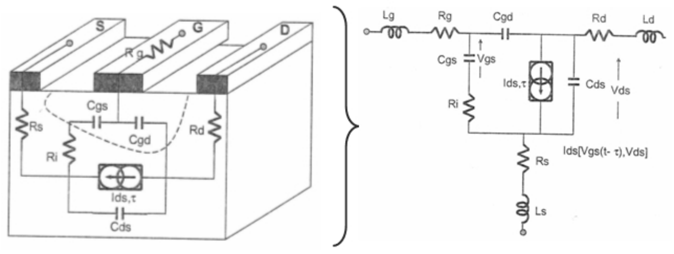


Figure 2.5. Physical meaning of an equivalent-circuit model for a FET transistor.

The well-known Curtice-Ettenberg model (traditionally called the Curtice model)[12][13], proposed almost two decades ago for modeling FET transistors, has been extensively used for all kinds of microwave circuit designs. The Curtice model consists

[8]Solid-state semiconductor device which can be used for amplification, switching, signal modulation, etc.

of an equivalent-circuit for the device that includes linear elements, nonlinear capacitances, diodes and a nonlinear current source representing the channel current. The channel current (or drain current) is given by equation 2.1, where the coefficients A_i can be evaluated from measurement data, V_{gs} is internal gate voltage, V_{ds} is drain voltage, V_{ds0} is a particular bias voltage (usually the one where the A_i polynomial coefficients were determined), β, γ and l are constants, t is gate-to-drain time delay and V_1 is the input voltage which includes the phenomena of pinch-off[9] voltage increase with drain-source voltage. It can be noted that $V_1 = V_{gs}$ at DC when $V_{ds} = V_{ds0}$.

$$I_{ds}(V_{gs},V_{ds}) = \left(A_0 + A_1V_1 + A_2V_1^2 + A_3V_1^3\right)\tanh(\gamma V_{ds})(1 + lV_{ds}) \tag{2.1}$$

where

$$V_1 = V_{gs}(t)\left(1 + \beta(V_{ds0} - V_{ds})\right) \tag{2.2}$$

Ignoring equation 2.2, by assuming that $\beta = 0$, equation 2.1 can be put in the following form:

$$I_{ds} = f_g(V_{gs})f_d(V_{ds}) \tag{2.3}$$

where

$$f_g(V_{gs}) = A_0 + A_1V_1 + A_2V_1^2 + A_3V_1^3 \tag{2.4}$$

and

$$f_d(V_{ds}) = \tanh(\gamma V_{ds})(1 + lV_{ds}) \tag{2.5}$$

Equation 2.3 is a convenient expression which allows to separate the effects of the gate (equation 2.4) and drain (equation 2.5) I-V characteristics and to determine them easily: $f_g(V_{gs})$ is a cubic polynomial, which should be relatively easy to fit to a set of measured data and can be found from a plot of I_{ds} at fixed V_{ds}; and $f_d(V_{ds})$ is a hyperbolic tangent function, used because any FET's drain I-V characteristic looks like a hyperbolic tangent curve when $V_{ds} > 0$ and can be calculated at fixed V_{gs}. The remaining term, $1 + lV_{ds}$ accounts for DC drain-to-source resistance. The corresponding equivalent circuit and I-V (current vs. voltage) curves of the Curtice model can be seen in figure 2.6 and 2.7, respectively.

The Curtice model, in one form or another, has been implemented in virtually all circuit simulators. Still, there are some problems when using the model. Designers frequently encounter convergence difficulties and errors. As Dr. Curtice himself has recently said : "... it is a simple model for a very complex device ..." [49]. This

[9]Region on the characteristic curve of a FET in which the gate bias causes the depletion region to extend completely across the channel.

model works reasonably well for small-signal levels, but fails to follow the device behavior satisfactorily where saturation[10] and cut-off[11] occur.

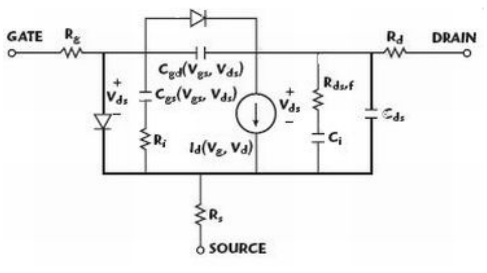


Figure 2.6. The Curtice model: equivalent- circuit.

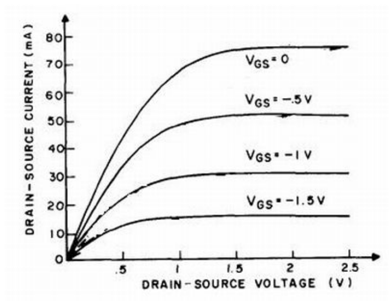


Figure 2.7. The Curtice model: I-V curves.

An example of a physics-based model for a FET is presented in [21]. The simulated and measured I-V curves for the device are shown in the right part of the figure 2.8. This model, to be able to provide complete device performance prediction (DC characteristics, bias-dependent small-signal AC parameters, large-signal response, noise, temperature dependence) in terms of physical parameters alone, is based on fundamental semiconductor equations. For the correct comparison between measured and computed scattering [S] parameters[12], a set of external parasitics was added to the model, according to the equivalent circuit shown in the left part of figure 2.8. The circuit is largely redundant, since some elements are actually negligible or can be merged; it has however been chosen because it allows a physical estimate of all reactive parameters. The model is completely based on a deep knowledge about the physical insights of the device modelled, i.e. the substrate residual donor and

[10]On a voltage-current conduction curve, condition in which a further increase in voltage produces no appreciable increase in current.

[11]The point on the characteristic curve of an amplifying device, at which the output current drops to zero under no-signal conditions.

[12]Set of parameters that characterize a device, they relate its inputs and outputs in a linear way.

acceptor concentrations phenomena (happening at the very physical low-level) are included into the model, as well as the external parasitics that were approximated on the basis of electromagnetic models.

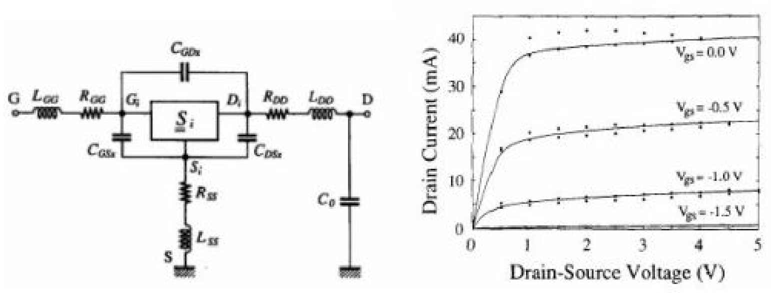


Figure 2.8. Small-signal equivalent circuit and I-V curves for a FET.

As an alternative to these traditional models, new curve-fitting techniques have been recently proposed for behavioral modeling of nonlinear FET devices used in amplifiers, generating characteristic amplifier data as function of bias current level [76]. In the cited work, the traditional small-signal Curtice model is used as a starting point and then modified to obtain a behavioral large-signal model. It is built as combination of several functions as the result of polynomial fitting. The coefficients of the functions are actually measurable parameters from the device, such as power, voltage, current, gain, noise figure, among others. The behavioral model is based on the following device parameters: I_{ds} is drain current, V_{gs} is gate voltage and V_{po} is pinch-off voltage. Its final form is shown in equation 2.6.

$$\begin{aligned} I_{ds} &= \frac{I_{ds}}{4}\left\{\log_{10}\left[1+10^{\left(\frac{4V_{gs}-3V_{po}}{|V_{po}|}\right)}\right]+\log_{10}\left[1+10^{\left(\frac{4V_{gs}-V_{po}}{|V_{po}|}\right)}\right]-\right. \\ &\left.\frac{4}{5}\log_{10}\left[1+10^{\left(\frac{10V_{gs}}{|V_{po}|}\right)}\right]\right\} \end{aligned} \qquad (2.6)$$

A comparison between this behavioral model and the traditional Curtice model is shown in figure 2.9. As can be concluded from the comparison, the new behavioral model follows more accurately the real behavior of the current in a FET than the Curtice model, specially taking into account the saturation phenomena.

Recently, neural networks (NNs) models, as the one shown in figure A.1, are being preferred over polynomials for behavioral models building, because of their speed in implementation and accuracy. By profiting from their capability to learn the circuit behavior based on simulated or measured records of input-output parameters (inputs and outputs to the neural model), they are used in nonlinear modeling and design of many circuits [94].

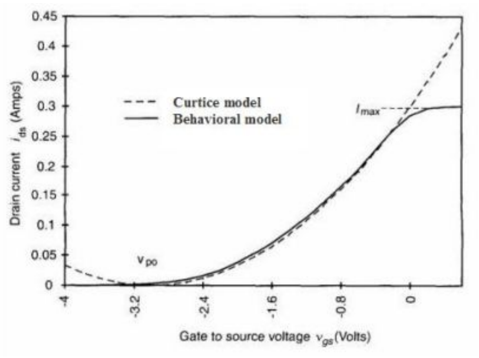


Figure 2.9. Comparison between the traditional Curtice model and a new behavioral model.

In [17][91] a neural network for modeling of nonlinear RF/Microwave devices or circuits in continuous time-domain is proposed. The papers illustrate the modeling of nonlinear effects on a FET operating at high frequencies. The model can be trained directly from input-output large-signal data (in this case, spectrum data) irrespective of internal details of the circuit. However, a circuit representation of the proposed neural model is introduced in order to incorporate it into circuit simulators for high-level design. The neural model still needs the circuit representation to be simulated and to generate the data that will train it. This equivalent-circuit approach to behavioral modeling has its limitations. The identification of model parameters is done through virtual measurements, i.e. from transistor-level models of the devices, and the physical effects taken into account in the model are decided a-priori, when the circuit is built [9]. Therefore the disadvantages and problems previously mentioned, associated to an equivalent-circuit topology generation, are yet present in this work. Another neural-based proposal appears in [27], where NNs are used for modeling small-signal and large-signal FETs, learning also the [S] parameters and the derivatives of the model. However, this approach, again, is not totally black-box because the parameters of the model are found by using small-signal and large-signal equivalent-circuits.

Due to the mentioned advantages of NN-based behavioral models, this approach has been followed in this PhD thesis. Chapter 3 is devoted to explain behavioral modeling in more detail. But, due to the mentioned weaknesses of existing models, this thesis proposes a new behavioral model based on NNs, trying to fullfil the existing gap between accurate, fast models and available models.

For the model proposed in this thesis, the equivalent-circuit is no longer useful because directly device measurements are used for model training, which assures the representation of the real behavior of the device. Of course, the accuracy and good quality of the measurements used have a direct influence on the model accuracy. The dynamic and nonlinear phenomena in the device behavior are taken

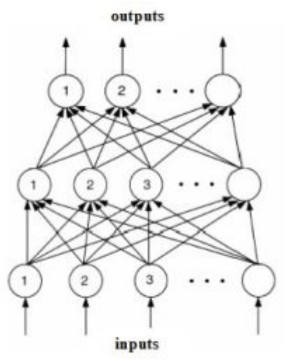


Figure 2.10. Typical neural network model.

into account inside the NN model topology itself. And finally, concerning the model implementation inside a simulator, a circuit will not be used but it will be done as a completely black-box model. The details of the proposed model are explained in chapter 4.

2.5 System-level modeling

The design of telecommunication systems is a hierarchical process, involving the use of a large set of simulation tools and relying on appropriate modeling of system elements. There is a continuous push to improve the accuracy obtained from equivalent-circuit simulations, as well as to increase the capabilities for simulating very large circuits, containing many thousands of devices at a system-level abstraction. These are conflicting requirements, because higher accuracy tends to require more complicated models for the circuit components, while higher simulation speed favors the selection of simplified, but less accurate, models. The latter holds despite the general speed increase of available computer hardware on which run the circuit simulation software [52].

When dealing with semiconductor circuits and devices, the system under study is typically a continuous, but highly nonlinear, multidimensional dynamic system. Moreover, due to the adoption of increasingly complex signal modulation techniques in the newest wireless communication systems standards - i.e. single-multiple carrier modulation; second generation (2G) GSM and CDMA; third generation (3G) CDMA2000 and UMTS - extra nonlinear behavior is added to the system, which directly influences the system performance.

Microwave and wireless communications systems are too complex to allow a complete simulation of its nonlinearities at a detailed transistor level of description. This problem is a significant productivity bottleneck for design engineers. New models have to be developed to deal with this growing complexity, to improve the accuracy and efficiency with which the behavior of these systems can be analyzed, predicted and simulated.

Currently, the most widely used RF/Microwave system design methodology, according to [34], is composed of two stages, graphically shown in figure 2.11:

1. Top-down design stage: the target system is split-up into sub-systems or circuits. The specific system is replaced by a hierarchical system-level model composed of a behavioral circuit model based on input-output relations. The behavior of this hierarchical model might be analyzed by system simulators to improve the circuit models specifications.

2. Bottom-up design stage: component models which constitute a circuit are extracted from simulations or, more directly, from measurements. The electrical behavior of the circuits that compose the overall system model are then analyzed and optimized to meet as closely as possible the system specifications previously established.

In practice the designer goes through several iterations of the methodology until he/she arrives at a stable system model that adequately covers the requirements. Going up in the hierarchy implies the use of specific simulation tools acting on one hand on specific models, from low-level physical models to high-level behavioral models, and on the other hand on specific signals, from one carrier to complex modulations.

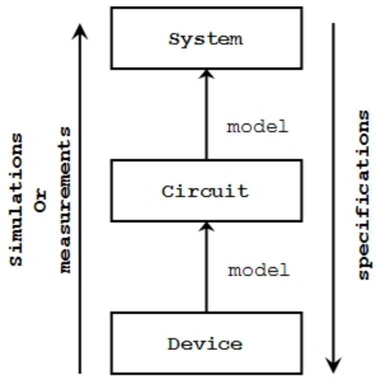


Figure 2.11. Two stage system design process.

For circuit-level simulation purposes, a microwave circuit such as an amplifier, oscillator or mixer is always described as an architecture composed of linear and nonlinear device models. At system-level simulation, no physical insight is available for the circuit which must only be described by a nonlinear input-output mapping, allowing the handling of telecommunication systems composed of a large number of circuits acting on complex signals.

System-level behavioral modeling consists of constructing a black-box analytic function that admits responses alike to those obtained at the output of a real device or a subsystem driven by the same input signal. Such models are an important help for designers of communication systems, in particular, transmitters and power amplifiers (PAs) since they provide them with greatly reduced complexity and time-consuming design and optimization procedures [46]. Examples of this level of analysis using the circuit-based approach and behavioral modeling are shown in the next subsection.

2.5.1 Example: power amplifier modeling

A major building block of modern RF circuits are amplifiers. An amplifier works by increasing the magnitude of an applied signal. Amplifiers can be divided into two big groups: linear amplifiers, which produce an output signal directly proportional to the input signal, and power amplifiers (PAs) which have the same function as the first ones, but their objective is to obtain maximum output power [25]. Due to that, the active devices that compose the PAs arrive at the limit of the linearity (class A) or even work in the nonlinear regime (class B and C) and generate harmonics[13] and intermodulation (IM) products[14].

In a PA the small-signal power output P_{out} is directly related to power input P_{in} by amplifier small-signal or operative gain G_{ss} ($P_{out}[dB] = G_{ss}P_{in}$). This is a linear relationship. In practice, however, power output increases linearly until compression occurs, and finally saturation is obtained. The output of a PA is a signal at frequency f_0 (called the fundamental frequency) amplified at the output power level P_{out} plus all the harmonics nf_0 of the fundamental frequency. This happens when a one-tone signal is introduced into the amplifier, as illustrates the left part of figure 2.12.

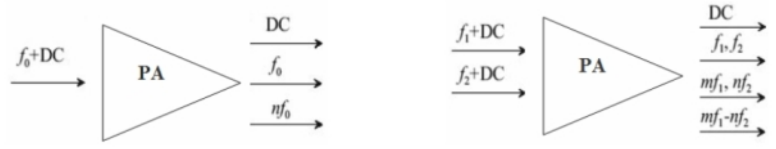


Figure 2.12. Power Amplifier with one-tone (left) or two-tone (right) input signal: harmonics generation.

A common representation for a PA, that helps visualizing its input-output behavior, is the so-called P_{in}/P_{out} curve, which shows the power of the output at the

[13] Components whose frequencies are multiples of the fundamental frequency f_0.

[14] Undesired modulation of one signal by another, caused by nonlinear processing of the signals.

fundamental frequency and its harmonics (in a logarithmic scale) as function of the input power. A typical example of this curve is shown in figure 2.13. Fundamental output power as a function of input power has to exhibit a linear range where output increases 1 dB for every 1 dB of input power increase. However, as input power increases, gain compression begins to occur and output power begins to saturate. In the figure, two important PA parameters are marked, the 1 dB compression point and the third-order intercept point or third harmonic output intercept $IP3$. These parameters have been traditionally used to describe the nonlinearity of a RF/Microwave PA.

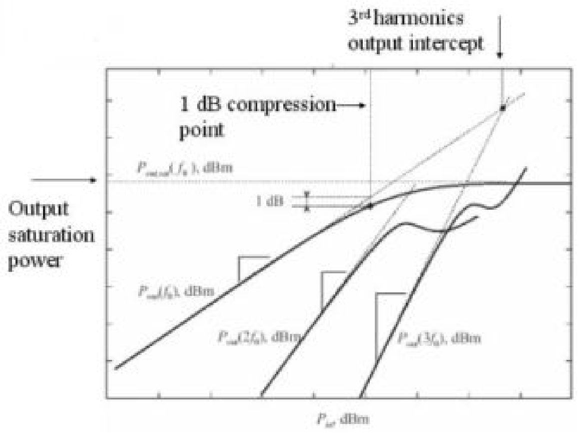


Figure 2.13. Amplifier power output vs. power input traces ($P_{in} - P_{out}$ curve).

The 1 dB compression point is the input power to which the output power is 1 dB below the corresponding output power of the small-signal gain G_{ss}. The $IP3$ point is usually determined by introducing two tones into an amplifier, at frequencies f_1 and f_2 as shows the right part of figure 2.12, and increasing their amplitudes equally, driving the amplifier into compression. Frequency difference $f_1 - f_2$ must be less than the amplifier active bandwidth such that both tones are equally amplified. When the two tones are increased in power input through the linear range and push into compression, the amplifier nonlinear behavior generates intermodulation (IM) at frequencies $2f_1 - f_2$ and $2f_2 - f_1$, as depicted in figure 2.14. Amplitude of the two tones f_1 and f_2 in the amplifier output increases 1 dB for every dB increase in power input through the linear range. In the ideal amplifier, amplitude of the IM at frequencies $2f_1 - f_2$ and $2f_2 - f_1$ increases 3 dB for every 1 dB increase in power input.

Graphically, as figure 2.15 shows, the IP_3 point can be seen by projecting third-order intermodulation sideband amplitude growth at either $2f_1 - f_2$ or $2f_2 - f_1$ in a P_{in}/P_{out} curve at a 3 dB per 1 dB slope and projecting linear signal f_1 and f_2 output amplitude growth at 1 dB per dB slope to a point of intersection. That point

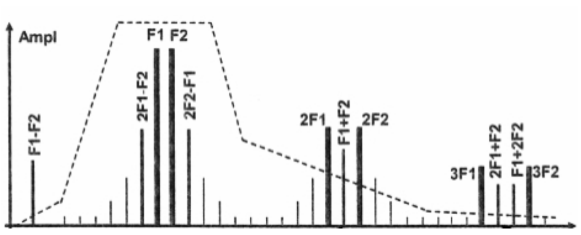


Figure 2.14. Intermodulation (IM) distortion phenomena.

is defined to be the amplifier output $IP3$ and is valued at whatever output power level the intersection occurs.

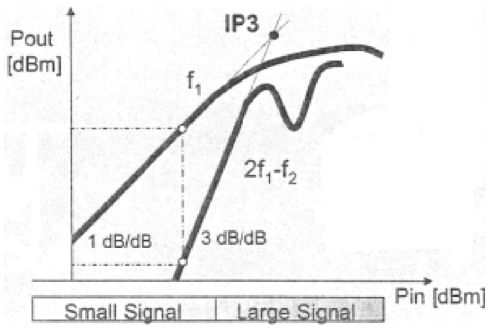


Figure 2.15. Classical two-tone test in an amplifier.

An important task when designing an amplifier is the choice of its active component (the transistor). A common choice for PAs is a FET build on GaAs (Gallium Arsenium), used as high-gain and high-power device. Figure 2.16 shows a common amplifier schematics configuration. For maximum gain design, matching networks are used to transform the generator/load impedances (Z_G,Z_L) into the optimum impedances (Z_{in},Z_{out}) seen from the active device (optimum according to i.e. power). A schematics of an equivalent-circuit model for a PA of this kind appears in 2.17.

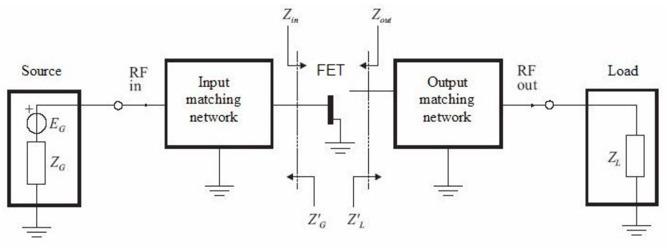


Figure 2.16. Schematics configuration of an amplifier with matching networks.

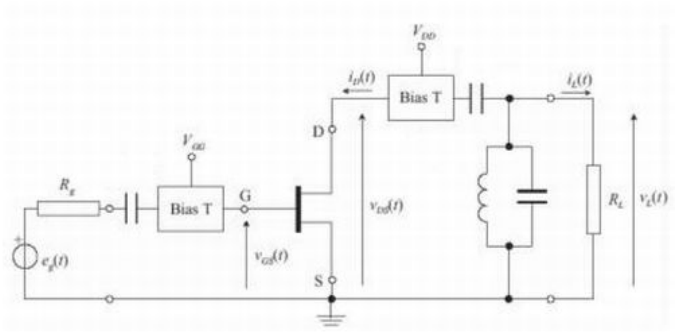


Figure 2.17. Schematics of a Power Amplifier equivalent circuit model.

Concerning behavioral models for PAs, they can be classified into three categories depending on whether the model considers the existence of memory effects or not: memoryless nonlinear systems, quasi-memoryless nonlinear systems, and nonlinear systems with memory [43].

A memoryless nonlinear system is represented in figure 2.18, where the system is defined by equation 2.7. The response r of the system at time t is a function of the input signal s at time t only, no past values of the signal have incidence on the response.

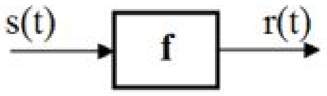


Figure 2.18. Block representation of a memoryless nonlinear system.

$$r(t) = f\left(s(t)\right) \tag{2.7}$$

If the model of equation 2.7 is expanded in a power series around 0, the result is equation 2.8. A linear PA can be modeled with this simple power series. The higher order terms of the series describe a gradual reduction in gain on the amplifier from its nominal value until the gain reaches zero. For the quasi-memoryless nonlinear system, the PA block is often represented by functions measured by sweeping the power of a single-tone in the center frequency of the band of the RF PA.

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots \tag{2.8}$$

An example of a memoryless behavioral model for a nonlinear PA is proposed in [76] by constructing a graph having two asymptotes. Amplifier fundamental power output as a function of power input traces a curve along the linear for small signals, then transitions onto saturated power output as power input increases. The asymptotes are connected by a right-hand-function having an asymptotic intercept at $P_{in} = P_{sat} - G_{ss}$. The equation describing power output as function of power input is shown in equation 2.9.

$$P_{out} = P_{in} + G_{ss} - K \log_{10} \left[1 + 10^{\left(\frac{P_{in} + G_{ss} - P_{sat}}{K} \right)} \right] \tag{2.9}$$

The first two terms of the formula describe the amplifier linear range and the third term describes the amplifier compression depth. The parameter K is defined to be the compression coefficient. It determines how softly or sharply compression occurs. It can be defined by measuring power output $P_{out}(K)$ when power input is set at $P_{in} = (P_{sat} - G_{ss})$. Saturated power output and small signal gain are in terms of dBm and dB respectively.

Concerning systems with memory, a block representation of such a system is shown in figure 2.19, whose input-output relationship is given by equation 2.10. The formula expresses the nonlinear and dynamic nature of the system. In fact, on the one hand products of the signal $(y(t)^2)$ contribute to the response and therefore the system is nonlinear; on the other hand past values of the signal contribute to the output also $(s(t-\tau))$, which implies that the system is dynamic.

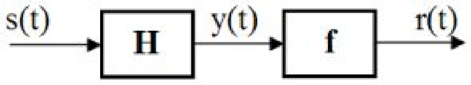


Figure 2.19. Block representation of a dynamic nonlinear system. A cascade of a linear system and a memoryless nonlinearity to create a Volterra series model.

$$\begin{aligned} r(t) &= a_1 y(t) +_2 y(t)^2 + \cdots \\ &= a_1 \int_{-\infty}^{+\infty} h(\tau) s(t-\tau) d\tau + a_2 \left[\int_{-\infty}^{+\infty} h(\tau) s(t-\tau) d\tau \right]^2 + \cdots \end{aligned} \tag{2.10}$$

This model is called Volterra series model in the continuous time-domain, and its $h(\tau)$ coefficients are called Volterra kernels. The Volterra series has been traditionally used for the modeling of dynamic systems and its analysis is well suited to the simulation of nonlinear microwave devices and circuits, in particular in the weakly nonlinear regime where a few number of kernels are able to capture the nonlinearity in the device behavior. The Volterra kernels allow the analysis of device characteristics of great concern for the microwave designer, such as harmonic generation and intermodulation[15] (IM) phenomena in the case of a FET transistor [47]. How to build a Volterra series model is shown in the next chapter, dedicated to explain behavioral modeling in detail.

Generally, behavioral modeling of PAs is widely accomplished on the basis of quasi-memoryless or quasi-static models, which are enough for narrowband modulated signals. However, modern high-capacity radio links involve large signals bandwidth, and therefore PA models should also take into account the amplifier distortion due to the large amplitude and bandwidth on the input signal [3]. Some dynamic effects show up only in the presence of nonlinear regimes. This is the case of the so-called long-term memory effects. This nonlinear dynamics cannot be modelled by any memoryless nonlinear model [63].

Some recent works propose behavioral models to treat memory effects in nonlinear PAs using polynomial models. The work of [2] compares polynomials having different types of delays between samples in the input space (uniform and non-uniform delay taps). In [44] the objective is to model the memory effects observed in PAs when they are excited by a two-tone RF signal. It tries to show that a PA behavioral model based on a memory polynomial can improve the accuracy in predicting output nonlinear signals by modeling memory effects. However, the polynomial method has two disadvantages: possible convergence problems when extrapolating the data; and the order of the polynomial which constrains the model applicability. Although polynomial are easily extracted, they are restricted to mild nonlinearities.

The proposal of [81] is to use describing functions in the frequency-domain, to simulate the behavior of a PA under large-signal one-tone excitation at the input, with any arbitrary impedances present at the output (fundamental and all harmonics). The model parameters are extracted based upon a relatively small set of measurements performed with a vectorial nonlinear network analyzer (VNA). The drawback of the method is that the model is only valid for one-tone excitation, with a frequency corresponding to the frequency used to extract the model.

The work of [95] states that analytical Volterra series models can be successfully applied to RF PAs behavioral modeling, but their high complexity tends to limit their applications to weakly nonlinear systems. To model a PA with strong nonlinearities and long memory effects, the general Volterra model involves a great number of coefficients. It is proposed a pruning algorithm to remove the series coefficients which are very small, or not sensitive to the output error. In [18] a new analytical procedure is proposed for the extraction of the Volterra model from complex envelope representations of the input and output signal in time-domain. In both cases, the solutions proposed are analytical equations or procedures.

Finally, concerning neural-network based behavioral models of PAs, the model

[15]The (usually undesired) modulation of one signal by another, caused by nonlinear processing of the signals.

of [91] is of the recursive type and represents the amplifier voltage/current dynamic relationship. The model deals with the time derivatives of the input and output variables. The measurements are extracted from circuit-level simulated waveforms. In [69] and [87] nonlinear device models are proposed whose multidimensional functions are approximated by general polynomials or neural networks, involving two-port input voltages, output currents, and their continuous time derivatives. To train the models, simulations data from a nonlinear circuit-level simulator, for an appropriate set of tone frequencies and amplitudes, are used. In [20] recursive nets are used to model a bandpass amplifier, although using one recursive network for the transient and another for the steady-state regimes. The model parameters are extracted from circuit-level transient and sinusoidal steady-state simulation data. In [1] the neural model is based on a complex-valued network which receives time-delays inputs with unity and non-uniform time delay taps between instantaneous and previous input signal of the PA.

These neural networks-based models share some common weakness. Most of them are designed to reflect a one-variable input dependency only, which sometimes can be restrictive to a particular device or system and not of general application. Most of them also use simulation data to train the network, produced with an equivalent-circuit model implemented inside a circuit simulator, neural model which afterwards cannot be used inside the simulator itself. Some of the models propose the use of non-standard network topologies, which need much knowledge of the neural network theory to be able to build the model and train it, using new specialized and complex learning algorithms, all of which adds complexity to the task of model building and lead to a long training time and elevated computation resources.

In this thesis, a neural network-based behavioral model is proposed in chapter 4, which tries to overcome the mentioned disadvantages of existing models. The proposed model can be used for modeling one or multiple input/output dependency. Measurement data, obtained with an accurate time-domain characterization procedure developed at Politecnico di Torino as result of a PhD thesis [75], designed for nonlinear and dynamic characterization, are used to train the neural model. Once trained, it will be shown how it can be implemented as a full black-box model inside a commercial circuit simulator, or even as an equivalent Volterra series model.

2.6 Summary

This chapter began with an introduction to the traditional and new modeling techniques that can be applied to the modeling of RF/Microwave electronic devices or systems, namely physical-circuit and behavioral approaches, respectively. Then, the second part of the chapter presents some examples of these techniques for device-level modeling. The last part of the chapter made this analysis at a system-level.

Chapter 3

Behavioral modeling

In this chapter behavioral modeling is explained. The first part of the chapter introduces this new approach. After that, the three main techniques for building behavioral models are described: Volterra series, curve-fitting techniques and artificial neural networks. The neural network paradigm is explained in much more detail because this is the technique followed in this thesis. The last part of the chapter explains the reasons of this choice showing its advantages over the other methods.

3.1 Introduction

What is behavioral modeling?

Behavioral modeling is the science of accurately expressing measured behavior (linear or nonlinear) of an object [76].

The behavioral modeling objective is to formulate a single closed-form equation representing some measured parameters, which might be function of one or multiple independent variables that control the element behavior, simultaneously. The main advantage of this measurement-based approach is that a low-order model of a complex circuit or system can be derived, without prior knowledge of its internal functioning, circuitry or topology. This new modeling approach is particularly useful when the observed object internal behavior is the result of a nonlinear process, or the physics of the object is not completely known.

A behavioral model is a simplified model of the essential behavior of an element, at a given level of abstraction hierarchy, for use in simulation at the next highest level. The simplification means that the model will execute more quickly, and use much less computer memory, than if an entire complex system or sub-system were simulated at the lowest detailed level [65]. This kind of model can be applied in device-level as well as in system-level modeling.

A behavioral model of a device could be, i.e. a set of ports characteristics equations (or the equivalent-circuit of such equations) obtained from external (possibly virtual) measurements. A behavioral model of a system is a black-box which produces a response to inputs as closest as possible the original system would do. Behavioral models have the required numerical efficiency and, when properly used, yield responses close to the response of a detailed device-level model [9].

A typical RF/Microwave design and modeling hierarchy is presented in figure 3.1. At the bottom appears the device represented through a detailed physical model and at the top there is a complicated module, sub-system or system. A top-down design methodology propagates specifications down the hierarchy. Bottom-up verification, instead, validates overall system performance based on the performance of lower level components and their configuration. At the bottom, the device is described by the detailed semiconductor physics inside a device model; this is abstracted to a circuit-level model that describes the terminal behavior through equivalent-circuit or phenomenological equations. The circuit model is used to design efficiently using circuit simulators, but it is too complex to use in a system level simulation at the top of the hierarchy. To bridge the circuit and system environment, a reduced order behavioral model of the circuit can be used [86].

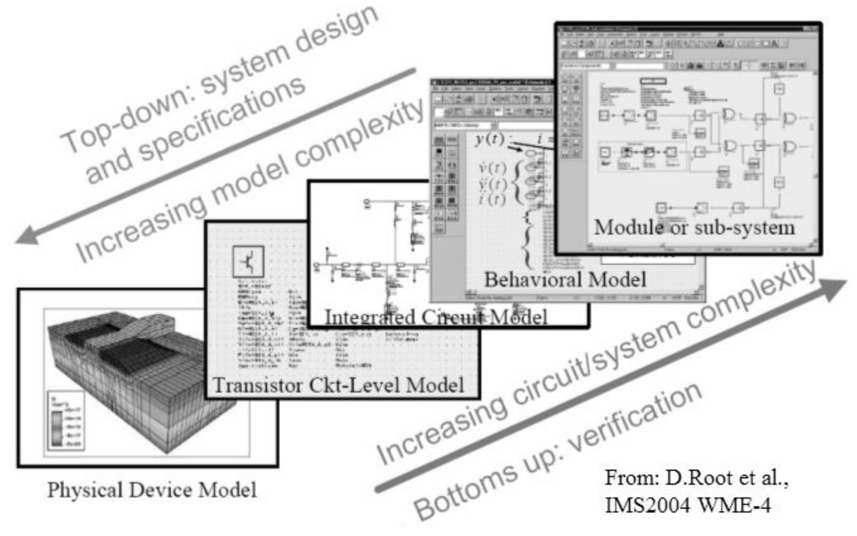

Top-down: system design and specifications
Increasing model complexity
Module or sub-system
Behavioral Model
Integrated Circuit Model
Transistor Ckt-Level Model
Physical Device Model
Increasing circuit/system complexity
Bottoms up: verification
From: D.Root et al., IMS2004 WME-4

Figure 3.1. Typical design and modeling hierarchy [86].

Behavioral models can be derivable from both simulations and (virtual) measurements. Considering, i.e. a circuit model developed from primitive components such as transistors, resistors, capacitors and inductors, a simulator can be used to stimulate the model, and the responses can be used to derive a behavioral model for simulation at the next higher level of abstraction. Alternatively, behavioral models can be derived from real measurements made on the component. Accurate measurements can therefore lead to more accurate behavioral models than starting from lower level, possibly inaccurate, models. This is, in fact, one of the major advantages of this approach. It allows designers to model real components whose equivalent-circuit model or analytical equations may not be available.

From a mathematical point of view, it can be said that a behavioral model relates output waveforms $y(t)$ to input waveforms $x(t)$. This fact can be written in functional notation as in equation 3.1. This is an example of a static relationship. The output is an instantaneous function of the input waveform and it depends only on the values of the actual input signal.

$$y(t) = F[x(t)] \tag{3.1}$$

Instead, a model which incorporates the notion of memory is shown in equation 3.2, which states that the output is not an instantaneous function only of the input signal, but it depends also on the time-derivatives of the input. This concept will be called memory. The function $F[.]$ which relates inputs with outputs may be linear or nonlinear.

Since the behavioral model has to be evaluated on a digital computer, it is convenient to adopt a discrete time representation of the variables, such as shown in equation 3.3. It assumes that time is a succession of (generally uniform) time samples of a convenient sampling period. In order to model the consequences of the memory phenomena within devices behavior, dynamic modeling is required. In fact, it is the incorporation of long-term memory beyond otherwise static nonlinear behavioral models what distinguishes the newest behavioral models, i.e. for PAs modeling [63].

$$y(t) = F[x(t),\frac{\partial x(t)}{\partial t},\frac{\partial^2 x(t)}{\partial t^2},\cdots,\frac{\partial^n x(t)}{\partial t^n}] \tag{3.2}$$

$$y(t) = F[x(t),x(t-1),x(t-2),\cdots,x(t-n)] \tag{3.3}$$

For the building of behavioral models (in other words, for choosing $F[.]$) three main techniques can be mentioned: Volterra series, curve-fitting or polynomial techniques and artificial neural networks. They are explained in the next sections.

3.2 The Volterra series model

If the function $F[.]$ represents a single-input, single-output (SISO) non-linear dynamical system, $F[.]$ can be represented exactly by a converging finite series of the form of equation 3.4. If the time in equation 3.4 is discrete, then the series assumes the form of equation 3.5.

This equation is known as the Volterra series expansion or power series with memory, because the actual output of the system depends on the actual and also on the past values of the inputs [7]. The series is formed with the sum of terms which have some particular coefficients h named *kernels* [68].

$$\begin{aligned} y(t) \;=\; & h_0 + \int_0^\infty h_1(\tau)x(t-\tau)d\tau + \\ & \int_0^\infty \int_0^\infty h_2(\tau_1,\tau_2)x(t-\tau_1)x(t-\tau_2)d\tau_1 d\tau_2 + \\ & \int_0^\infty \int_0^\infty \int_0^\infty h_3(\tau_1,\tau_2,\tau_3)x(t-\tau_1)x(t-\tau_2)x(t-\tau_3)d\tau_1 d\tau_2 d\tau_3 + ... + \\ & \int_0^\infty ... \int_0^\infty h_n(\tau_1,...,\tau_n)x(t-\tau_1)...x(t-\tau_n)d\tau_1...d\tau_n + ... \end{aligned} \tag{3.4}$$

The Volterra approach characterizes a system as a mapping between two function spaces, which represent the input and output spaces of that system. The Volterra series representation is an extension of the Taylor series representation to cover dynamic systems. It is based on a Taylor-series expansion of the device nonlinearity around a fixed bias point or around a time-varying signal. In 1959 V. Volterra [82] showed that such a series is capable of representing any analytic, time-invariant system. The Volterra series approximation produces the optimal approximation near the point where it is expanded. Therefore, it is known for its good modeling properties of small-signal (or mildly nonlinear) regimes [18].

$$\begin{aligned} y(t) \;=\; & h_0 + \sum_0^\infty h_1(k)x(t-k) + \\ & \sum_0^\infty \sum_0^\infty h_2(k_1,k_2)x(t-k_1)x(t-k_2) + \\ & \sum_0^\infty \sum_0^\infty \sum_0^\infty h_3(k_1,k_2,k_3)x(t-k_1)x(t-k_2)x(t-k_3) + ... + \\ & \sum_0^\infty ... \sum_0^\infty h_n(k_1,...,k_n)x(t-k_1)...x(t-k_n) + ... \end{aligned} \tag{3.5}$$

The Volterra Series and its kernels can be described in the time-domain or in the frequency-domain, changing from one representation to the other with the Fourier Transform. The Fourier Transform of the kernel functions yields the transfer functions of the nonlinear system under study. The transform of h_1 gives the linear transfer function H_1, the transform of h_2 gives the quadratic transfer function H_2 and in general, the transforms of the higher order kernels give the higher order transfer functions of the system. The Volterra series definition in the frequency domain is shown in equation 3.6.

$$\begin{aligned}Y(f) \;=\; & H_1(f)X(f) + \\ & \sum_{f_1}^{\infty}\sum_{f_2}^{\infty} H_2(f_1,f_2)X(f_1)X(f_2) + \\ & \sum_{f_1}^{\infty}\sum_{f_2}^{\infty}\sum_{f_3}^{\infty} H_3(f_1,f_2,f_3)X(f_1)X(f_2)X(f_3) + \cdots + \\ & \sum_{f_1}^{\infty}\cdots\sum_{f_n}^{\infty} H_n(f_1,\cdots,f_n)X(f_1)\cdots X(f_n) + \cdots \qquad (3.6)\end{aligned}$$

A schematic representation of a Volterra system can be seen in figure 3.2. The Volterra model can also be easily extended to a function depending on two or more input variables [24][62].

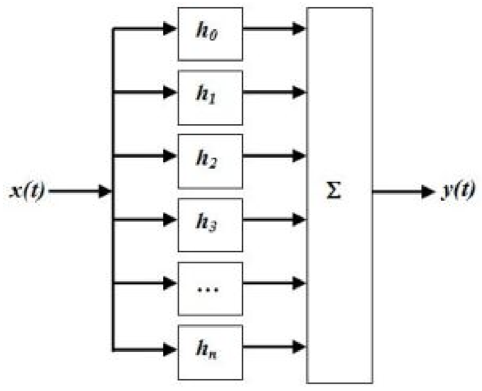


Figure 3.2. Schematic representation of a system characterized with the Volterra series.

Every nonlinear device having a Volterra series expansion has a set of symmetric Volterra kernels $h_1,h_2,h_3,\cdots,h_n$ about a DC operating point. The number of terms in the kernels of the series increases exponentially with the order of the kernel. This is the most difficult problem with the Volterra series approach and imposes some restrictions on its application to many practical systems, which are restricted to use second order models because of the difficulty in kernels expression [26].

The Volterra expansion approach has been proposed since many years for non-linear electronic devices modeling [47] and for weakly nonlinear devices it has been traditionally used as a behavioral black-box model [10]. The kernels allow a deeper understanding of the device or system under analysis; i.e. information about intermodulation (IM) phenomena in a FET transistor can be inferred from the first order kernels, among others [48][54][55][78][79]. An effective method of analyzing nonlinear RF and microwave intermodulation in PA is through the use of Volterra transfer functions [83][84].

However, due to the difficulties and complexity in identifying the higher order kernels through experimental data, this approach is not effectively used within commercially available circuit simulators. Heavy characterization efforts are needed to extract the kernels. There are some approaches which can be used for kernels analytical expression when their order is previously known [4][22][85]. The use of alternative methods for kernels identification [29][35][40][57][92] have proved to be a complex and time-consuming task.

Moreover, at microwave frequencies, suitable instrumentation for the measurement of the kernels is still lacking. The first published trial [6] for measuring second order transfer functions consisted basically in generating a multi-tone probing signal made of a sum of many sinusoids (typically more than fifteen) whose frequencies are generated by a simple algorithm which guarantees that no two pairs of input frequencies give rise to the same intermodulation frequency. To measure the second-order transfer function of a nonlinear device or system, the probing signal is applied to the device at several amplitude levels and its output sampled and collected [19]. It is quite difficult to separately detect the individual contributions of each Volterra operator from the global system response [45].

In the Biology field, Wray & Green [89][90] have outlined a method for extracting the Volterra kernels from the weights and bias values of a neural network model [88]. Based on this idea, there have been several proposals for kernels calculation with different, often non standard, neural networks topologies [16][30][74][50][41][59][61], also in the electronics field [31][33][23][38]. However, all of these approaches use a discrete temporal behavior by means of multi-delayed samples of a unique input variable. In the case of the voltages/current relationship in a transistor (i.e. $I_{ds}(V_{gs},V_{ds})$), such models are not useful because the multi-variable dependence cannot be modeled. Also the fact that most of these approaches propose the use of non-standard neural models and learning algorithms, makes difficult their use, and they do not offer a simple analytical solution derived from the network which could be implemented inside a commercial circuits simulator.

However, chapter 4 of this thesis proposes a simple and straightforward procedure to extract the Volterra series directly from a behavioral model built based on a standard neural network. The procedure can be applied even in the case of a nonlinearity that depends on more than one variable. A general model can be obtained, independently of the physical circuit under study. The behavioral model proposed is easy to load inside a circuit simulator, which simplifies and speeds up its practical implementation. This fact can be a useful chance when Volterra series analysis is implemented with a commercial RF CAD tool.

3.3 Curve-fitting techniques

In general, the process of generating a behavioral model, meaning transforming measured data into equations or finding a suitable $F[.]$ for equation 3.1, relies on curve-fitting techniques. These have the ability to generate equations that provide acceptably accurate fits to data that cannot otherwise be represented by a single equation. Many of the common linear, logarithmic, power function, exponential, polynomial and splines available techniques are useful where the data trace is well-defined over a defined range or where behavior of an object is known to follow a specific mathematical model. These methods are listed in the following lines.

Linear regression is used to represent data that traces a straight line as a function of an independent variable. It can be used to generate equations for asymptotes to curved traces of data. Logarithmic regression can be used only with positive values and the equation is difficult to accurately fit to a large number of data points unless the behavior is truly related to a natural logarithmic function.

Power functions are useful in modeling objects that are known to obey a power law, i.e FET transistors that under small-signal operation exhibit a quadratic behavior. However the power function is limited as a model for most nonlinear objects when extreme conditions are encountered, such as cut-off in FETs, when there is no current flow for all negative input signal values less than cut-off. They also saturate at some large positive input signal value. These two effects cannot be effectively modeled with one power series. Instead, FET pinch-off phenomena can be modeled by an exponential function; unfortunately it will fail to also represent the abrupt nonlinearity caused by saturation.

If $F[.]$ is replaced by a multi-dimensional polynomial approximation, the nonlinear system behavior can be approximated by a series of multi-linear terms, such as proposed in [2][70][71]. Polynomials are sometimes acceptable where measured data is well-behaved and quadratic or cubic expressions are enough. The polynomials expansions can be produced using McLaurin's or Taylor's procedure. However, interpolation errors will always exist between known data values, when higher order polynomials are used to capture sharp inflections.

Spline techniques have been developed to minimize curve-fit errors between measured data points where the data trace is a curve having rapid changes in the derivative orders. It uses polynomials equations to represent a small domain of data over a limited region of the independent variables. The polynomials are forced to have the same values and the same derivatives at the transition points or junctions between independent variable regions. The resulting curve fits are acceptably accurate, but as results of the application of this method an elevate number of equations are obtained.

In conclusion, problems arise when the object complex internal parameters cause the data to exhibit sharp inflections or discontinuities, and common curve-fitting techniques become useless. That is why new curve-fitting techniques have been recently proposed [76], using additive polynomial functions to replace $F[.]$, where the functions coefficients are actually identified from measurable device parameters, such as power, voltage, current, gain, noise figure, among others.

The new behavioral models use as a starting point a classical small-signal model and then modified it to obtain a large-signal model, which results a combination of right-hand and left-hand functions as the results of polynomial fitting. An example has been presented for transistor modeling in the previous chapter.

These new methods work well through sharp inflections because they use summations of logarithmic functions which are almost infinite series. However, these techniques have the disadvantage of requiring much knowledge of the device under study, to be able to identify the important parameters to measure and insert them into the model, and often as result of the process, the equations obtained are quite complex to generate and understand.

Moreover, polynomials have inherent local approximating properties instead of global approximation, which is a potential disadvantage for modeling strongly nonlinear systems. Polynomials show much degradation outside the zone of characterization and therefore they cannot extrapolate beyond the zone where the system was tested during parameter extraction. Polynomial order constraints the model applicability. Furthermore, as an analytical approach, it can be a slow and time-consuming task and may need the definition and calculus of a large amount of terms.

3.4 Artificial neural networks

What is an artificial neural network (NN)?

A neural network is a massively parallel distributed processor that has a natural propensity for storing knowledge and making it available for use. It resembles the brain in the following aspects [32]:

- Knowledge is acquired by the network through a learning process.
- Connection strengths between neurons, known as synaptic weights, are used to store the knowledge.

There is no universally accepted definition of what artificial NNs are, or what they should be. They can be loosely defined as large sets of interconnected simple units which execute in parallel to perform a common global task. These units usually undergo a learning process which automatically updates network parameters in response to a possibly evolving input environment. The units are often highly simplified models of the biological neurons found in the animal brain [32].

Neural networks are information processing systems inspired by the ability of human brain to learn from observations and to generalize by abstraction. A neural network can be thought of as an input/output system with many simple processors, each having a small amount of local memory. These units are connected by communication channels carrying data. The fact that NNs can learn totally different things led to their use in diverse fields such as pattern recognition, speech processing, control, medical applications, among others. They can be much faster than original detailed models, more accurate and flexible than empirical models, and easier to develop when a new device/new technology is introduced.

Neural models have certain common characteristics. They are given a set of inputs $x = (x_1, \cdots, x_n) \in \Re^n$ and their corresponding set of outputs $y = (y_1, \cdots, y_m) \in \Re^m$ for a certain process (behavior). The assumption of a NN model is that the process that produces the output response is given by some unknown mathematical relationship $y = G(x)$ for some unknown function G. This function may be nonlinear and very complicated and one cannot expect to compute it exactly. That is why a candidate activation function AF (for G) is chosen and the approximation is performed using a given set of examples, that is to say, some inputs x and their associated correct outputs y. The data are used to feed the NN model, which contains a set of processing elements (called neurons) and connections between them (called synaptic weights). Each neuron has an activation function AF, which process the incoming information from other neurons. Neural models may be considered as a particular choice of classes of functions $AF(x,w)$ where w are the parameters and specific procedures for training the network [64]. Training is similar to an optimization process where internal parameters of the neural model are adjusted, to fit the training data. Since a neural model is trained directly from data, the model can be developed even if the original problem formulae do not exist.

Nowadays, the modeling and simulation of nonlinear elements inside a wireless communication system using behavioral models based on the NN paradigm, is a field of increasing interest [52][77]. The NN approach for electronic device modeling has received increasing attention, especially in the recent years, for a large variety of industrial applications [53][94], since model tailoring to the device under study only needs a training procedure based on experimental data or measurements of the physical circuit. Lately, NNs have been recognized as a useful tool in the high-frequency CAD area, showing promising strength in addressing growing challenges in physics-based circuit design as well [93].

Neural nets can be trained to learn physics based behavior from component data, and the trained network can be used in high-level circuit and systems design, for both passive and active modeling, allowing fast optimization [34][39]. A trained neural model can be used online during microwave design stage providing fast model evaluation replacing original slow simulators. For example, large-signal transistor device modeling [72] can be automated exploiting NN learning from DC and small-signal data [27][28].

For behavioral models building, NNs are being preferred over polynomials and curve-fitting techniques, due to their asymptotic properties and because they give very smooth results for approximating discrete measured and simulated data, also when interpolating points.

They are becoming really attractive for RF black-box modeling due to their flexibility. In fact, they can be directly trained on the device non-linear measurements, without the need neither of the equivalent circuit topology, nor the corresponding non-linear component constitutive descriptions.

There are many different models of NNs. Most of them have a training rule which indicates how and what to learn from a set of examples. The different types of networks models and their learning rules are explained in the next subsections.

3.4.1 Representation and types

In engineering and physics domains, algebraic and differential equations are used to describe the behavior and functioning properties of real systems and to create mathematical models to represent them. To do this, accurate knowledge of the system dynamics is required, and the use of estimation techniques and numerical calculations to emulate the real system operation. The complexity of the problem itself may introduce uncertainties, making the modeling non-realistic or inaccurate. Therefore, in practice, approximate analysis is used and linearity assumptions are usually made.

Artificial neural networks implement algorithms that attempt to achieve a neurological related performance, such as learning from experience, making generalizations from similar situations and judging states where poor results were achieved in the past. Nowadays, NNs are being applied to a lot of real world, industrial problems, from functional prediction and system modeling (where physical processes are not well understood or are highly complex), to pattern recognition engines and robust classifiers, with the ability to generalize while making decisions about imprecise input data [53].

In many artificial intelligence (AI) applications, the knowledge needed to solve a problem may be incomplete, because the source of the knowledge is unknown, or the environment may be changing and cannot be anticipated at design time. There exist engineering problems for which finding the perfect solution requires a practically impossible amount of resources and an acceptable solution would be fine. NNs can give good solutions for such classes of problems.

Choosing a neural network solution for an immediate application (e.g. choosing the model topology, learning method, data flow, etc.) can become itself an engineering problem. A choice must be made among several options, depending on the application. The main features of a NN can be classified according to the following characteristics:

Topology of the network: Feedforward, recurrent or self-organized. This classification is shown in figure 3.3, including examples of each category. A network is feedforward if the flow of data is from inputs to outputs, without feedback. Generally this topology has distinct layers such as input, hidden and output. There are no connections among the neurons within the same layer. A recurrent network distinguishes itself from the feedforward topologies in that it has at least one feedback loop. Self-organizing nets map high dimensional data onto a lower representation space, therefore, after the learning process, the resultant map topology reflects the unknowns clusters or internal groups that exist inside the data.

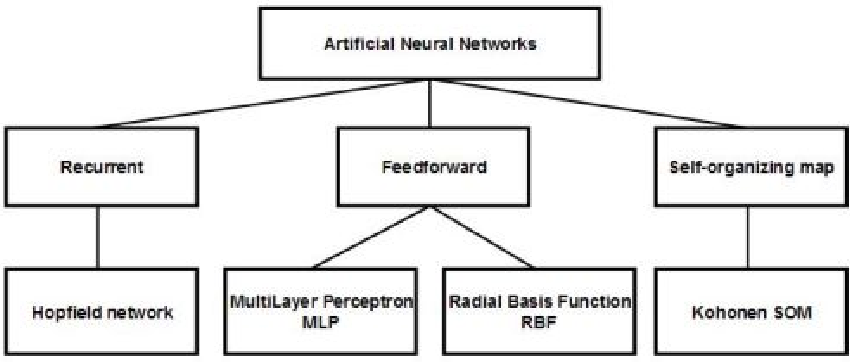

Artificial Neural Networks
Recurrent
Feedforward
Self-organizing map
Hopfield network
MultiLayer Perceptron MLP
Radial Basis Function RBF
Kohonen SOM

Figure 3.3. Classification of neural network models according to network topology.

Data flow: Recurrent or non-recurrent. In a recurrent model, the outputs can propagate in both directions, forwards and backwards. In a non-recurrent or feedforward model, the outputs always propagate from the inputs to the outputs.

Types of input values: Binary, bipolar or continuous. Neurons can be defined to process different kinds of signals. The most common types are binary (restricted to either 0 or 1), bipolar (either -1 or +1) and continuous (continuous real numbers in a certain range).

Forms of activation: Linear, step, sigmoid, radial, etc. Activation functions define the way neurons behave inside the network structure and, therefore, the kind of relationship that occurs between input and output signals.

Inside the so-called *feedforward* network models, there are two main types of models: the multi-layer perceptron (MLP) and the radial basis function (RBF).

The MLP model consists of a finite number of units called neurons or perceptrons, where each unit of each layer is connected to each unit of the subsequent/previous layer. These connections are called links or synapses and they only exist between neurons in subsequent layers but connections among neurons within a layer do not exist, nor are there any direct connections across layers. The signal flow is unidirectional, from the inputs to the outputs, from one layer to the next one, thus the term feedforward. The first layer is called input layer, it is followed by intermediate groups of neurons called hidden layers, and the result of the network is obtained in the output layer.

The number of neurons in the hidden layer is usually determined by trial-and-error, to find the simplest network that gives acceptable performance. Since the numbers of inputs and outputs are fixed (the independent and dependent variables of the problem, respectively) the number of hidden neurons determines the number of weights that must be optimized during the training process to obtain the best function approximation. The MLP network model can be seen in figure 3.4.

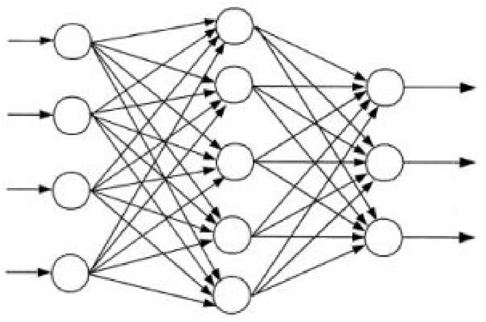


Figure 3.4. Multi-layer Perceptron (MLP) neural network.

The rules for the MLP model are the following:

- The j^{th} neuron of the k^{th} layer receives as input each x_i of the previous layer. Each value x_i is then multiplied by a corresponding constant, called weight w_{ji}, and then all the values are summed.

- A shift θ_j (called threshold o bias) is applied to the above sum, and over the results an activation function σ is applied, resulting in the output of the j^{th} neuron of the k^{th} layer. The MLP neuron model is shown in equation 3.7, where $i = 1 \cdots N$, $j = 1 \cdots M$, and graphically in figure 3.5.

$$y_j = \sigma \left(\sum_{i=1}^{N} w_{ji} x_i \pm \theta_j \right) \tag{3.7}$$

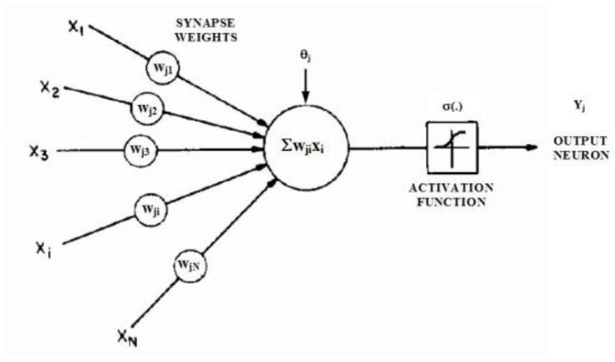


Figure 3.5. Neuron model.

Some common choices for σ can be one of the following functions:

- The logistic sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.8}$$

- The hyperbolic tangent

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.9}$$

- The linear function

$$y = x \tag{3.10}$$

A priori, the number of layers and neurons in each layer is chosen, as well as the type of activation functions for the neurons. Then the values of the weights w_{ji} and bias θ_j are initialized. These values are chosen so that the model behaves well on some set (named training set) of inputs and corresponding outputs. The process of determining the weights and thresholds is called learning or training. In the MLP model the basic learning algorithm is called backpropagation, which uses gradient descend to minimize the cost function of the NN model (the error E) generally defined as the mean square error (MSE) between the desired output (target behavior to approximate) and the actual network output. During learning, the MSE propagates backwards through the network, hence the term backpropagation. For output units the error depends on the difference between the actual and desired output. For hidden neurons the error is computed using the unit own activation function and the weighted error of all its output units in the layer above. Thus the error is propagated backwards across the layers of the network model and the weights are changed according to the following equations:

$$w_{ji}^{new} = w_{ji}^{old} + \varDelta w_{ji}^{old} \tag{3.11}$$

$$\varDelta w_{ji}^{old} = -\eta \nabla E \tag{3.12}$$

$$\nabla E = \frac{\delta E}{\delta w_{ji}} \tag{3.13}$$

$$E = \sqrt{\frac{1}{P} \sum \|(Y - T)\|^2} \tag{3.14}$$

where E is the error mismatch (MSE) between the actual network outputs Y and the target outputs T over all the training samples P and η is the learning rate. This parameter is critical in the speed of convergence of the backpropagation algorithm. In general, it is desirable to have fast learning, but not so fast as to cause instability of learning iterations. Starting with a large learning coefficient and reducing it as the learning process proceeds, results in both fast learning and stable iterations.

Selection of training data plays a vital role in the performance of a supervised NN. The number of training examples used to train an NN is sometimes critical to the success of the training process. If the number of training examples is not sufficient, then the network cannot correctly learn the actual inputoutput relation of the system. If the number of training examples is too large, then the network training time could be too long.

A major result regarding MLP models is the so-called *universal approximation theorem* [14] which states that given enough neurons in the hidden layer, an MLP neural network can approximate any continuous bounded function to any specified accuracy; in other words, there always exists a three-layer MLP neural network which can approximate any arbitrary nonlinear continuous multidimensional function to any desired accuracy, provided that the model has enough neuronal units [37]. That is why this model has been more extensively studied and used during these years. It is worth noting, however, that the theorem does not say that a single-layer network is optimum in the sense of learning time or ease of implementation; moreover, the theorem does not give indications about the required number of hidden units necessary in order to achieve the desired degree of accuracy [8].

Until very recently, NNs for modeling were applied almost exclusively to static functions of the input variables alone. Although this approach has been largely exploited for DC simulation, their application to dynamic system is a rather new research field. Recently have appeared models taking into account the dynamic phenomena, i.e. for power amplifier(PA) modeling [60][91] which propose the use of a special type of MLP model, the time-delayed neural networks (TDNN). They have been successfully applied for solving the temporal processing problems in signal processing, speech recognition, system identification and control [46]. They are suited for dynamic systems representation because the continuous time system derivatives are approximated inside the model by time-delays of the input/output variables in the discrete time.

A TDNN is based on the feedforward MLP neural network with the addition of tapped delay lines (Z^{-1}) which generate delayed samples of the input variables. They are used to add the history of the input signals to the model, needed for memory effects modeling. The TDNN entries include not only the current value of the input signal, but also its previous values, as shows equation 3.15 and the figure 3.6. The memory depth of the element or system analyzed is reflected on the length of the taps. The strategy followed to set the system memory is the same adopted for the design of numerical filters, where the input tap number is dictated by the bandwidth accuracy required.

$$y_j = \sigma \left(\sum_{i=0}^{N} w_{j,i} x(t-i) \pm \theta_j \right) \tag{3.15}$$

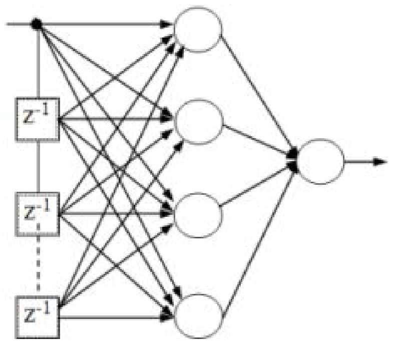


Figure 3.6. Time-Delayed neural network (TDNN) model.

Another feedforward model is the Radial Basis Function (RBF). Here the activation of a hidden unit is determined by the distance between the input vector and a prototype vector (the center of the neuron). The Euclidean distance between the input signal X and the center of the t^{th} neuron c_j is first evaluated, then an activation function, also known as a basis function, is applied. This way, an unknown relationship can be approximated as a linear combination of a set of basis functions.

The most common form of basis function used is the Gaussian function, which has the form of equation 3.16, where $X = [x_1, \cdots, x_n]$ represents n inputs to the network, h_j is the output of the j^{th} hidden neuron and c_j is its centroid. The Euclidean norm is denoted by $\|.\|$. A centroid is a fixed point in n-dimensional space, and must be appropriately chosen from the input domain. The Gaussian function is radially symmetric around c_j and has variance determined by α. This parameter α controls how spread the curve is. Using other radially symmetric functions besides the Gaussian distribution generalizes the construct [15].

$$h_j = G(X) = e^{-\frac{\|X-c_j\|^2}{2\alpha^2}} \tag{3.16}$$

Figure 3.7 shows the basic structure of a RBF network. The input nodes pass the input values to the hidden layer directly, the first layer connections are not weighted. Thus, each hidden node receives each input value unaltered. The hidden nodes are the radial units. The second layer of connections is weighted and the output nodes are simple summations. In other words, the first layer performs a fixed non-linear transformation which maps the input n-dimensional space onto a new space, and the output layer implements a linear combiner on this new space. This network does not extend to more layers.

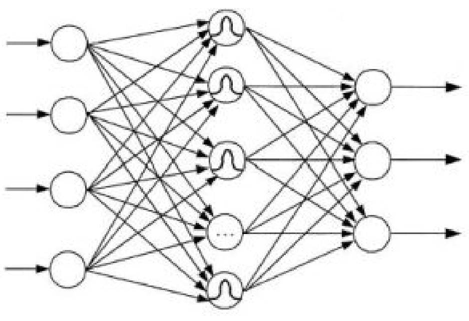


Figure 3.7. Radial Basis Function (RBF) neural network.

The learning algorithm for the centroids of the hidden neurons is usually a clustering algorithm such as K-means. The simplest approach is to set the center of the hidden neurons in randomly chosen patterns of the training set. Learning of the weights from the hidden layer to the output layer can be achieved using a gradient descent method, such as backpropagation.

A different category of neural networks are the **recurrent** neural networks (RNN). A network is called recurrent when the inputs to its neurons come from external inputs, as well as from the internal neurons themselves, consisting on both feedforward and feedback connections between layers and neurons. The Hopfield model is the most popular type of RNN [36]. It can be used as associative memory and can also be applied to optimization problems. The basic idea of the Hopfield network is that it can store a set of exemplar patterns as multiple stable states. Given a new input pattern, which may be partial or noisy, the network can converge to one of the exemplar patterns nearest to the input pattern.

As shown in figure 3.8 a Hopfield network is unstructured, its units are not organized into layers. The network is recurrent and fully interconnected (every neuron in the network is connected to every other neuron and the connections have the same weight). Each input/output takes a discrete bipolar value of either -1 or +1.

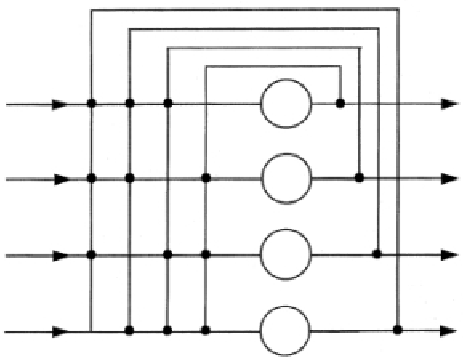


Figure 3.8. Recurrent neural network (RNN): the Hopfield model.

This network is trained using unsupervised learning. The weights of the connections inside the net are set so that each example pattern is represented by a stable state. Given a new input pattern, the network will settle into a previously learned stable state, closest to the input pattern.

When an input pattern is presented to the network, it changes its overall state by asynchronous iteration, updating the states of its units and computing their activation functions. Usually the activation function is simple the weighted sum of the incoming connections. Through the iterations the model converges to an overall stable state, where the states of the internal units do not change anymore. When a Hopfield network is in a stable state, its energy measure (the potential for change within the network) is minimized.

In general this kind of network model reproduces the stored training sample that is closest to the current input pattern. Therefore, the net can be used as associative storage of partial patterns and recognition of noisy patterns. If the training data are typical cases of different classes, a Hopfield net can be used for classification.

A third group of NN models are the **self-organizing maps** or SOMs. In the brain there are many cases in which aspects of sensory inputs are represented in two dimensional areas of the cortex. These topographic mappings within the brain are the inspiration for the SOMs. The importance of global organization is recognized on top of individual functionality of the units.

These networks appeared with the Kohonen map [42], which is an associative network and is presented in figure 3.9. Their units are organized in a two-dimensional array and they are connected so that there are excitatory connections between nearby units and inhibitory connections between distant units. This network uses unsupervised learning. When an input pattern is presented to the net, the unit with the highest response to that pattern is located and its weights are updated, as well as the neighboring units weights. This way the formation of clusters is encouraged

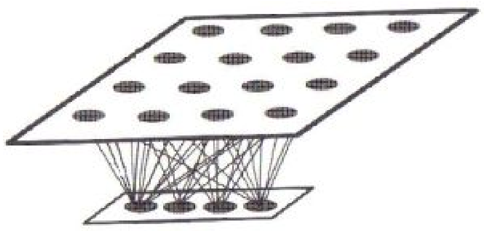


Figure 3.9. Self-organizing map (SOM): the Kohonen model.

so that nearby units respond similarly to inputs. Usually the system will start organizing into large regions and focusing into smaller regions during training. Natural applications of Kohonen maps are pattern recognition (speech, text, pictures, etc.), feature detections and clustering.

3.4.2 Learning algorithms and training considerations

Training a neural network means adapting its connections so that the network exhibits the desired computational behavior for all input patterns. It refers to optimization of the degree to which some desired behavior, the target behavior, is represented. The process usually involves modifying the weights, but sometimes it also involves modifying the actual topology of the network, i.e., adding or deleting connections from the network.

The terms learning and optimization are nowadays used interchangeably, although the term learning is normally used only in conjunction with NNs, because, historically, learning used to refer to behavioral changes occurring through synaptic and other adaptations within biological neural networks [52].

There are basically two categories of learning algorithms:

Supervised learning The correct results (target values or desired outputs) are known and are given to the NN during training so that the network can adjust its weights to match its outputs to the target values. Before the learning algorithms are applied to update the weights, all the weights are initialized randomly. The network, using its parameters, produces its own outputs. The network outputs are compared with the correct targets and the differences are used to modify the weights, as shown in figure 3.10. After training, a NN is tested in the following way: it is given only input values, not target values, and it is analyzed how close the network comes to the correct target values.

Unsupervised learning Involves no target values; the network tries to auto-associate information from the inputs with an intrinsic reduction of data dimension. This is the role of the training algorithms. It is also known as self-organized

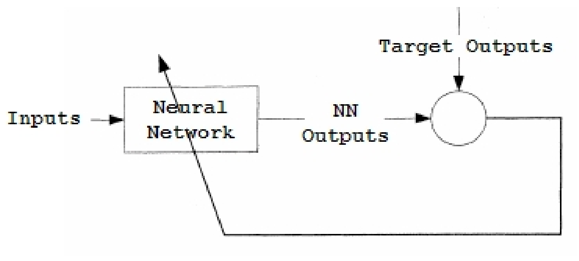


Figure 3.10. Supervised learning.

learning, because the network develops some classification rules by extracting information from the inputs presented to the network. This is illustrated in figure 3.11. By using the correlation of the input vectors, the learning rule changes the network weights to group the input vectors into clusters. Similar input vectors will produce similar network outputs since they belong to the same cluster.

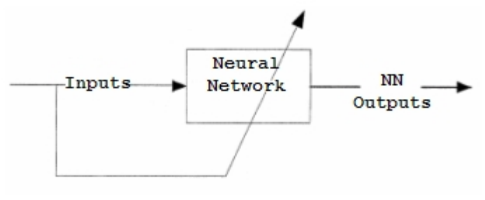


Figure 3.11. Unsupervised learning.

Considerations such as determining the input and output variables, choosing the size of the training data set, initializing the network weights, choosing the learning rate for the training algorithm and selecting the stopping criteria, are important for several network topologies. However, there is no generic formula that can be used to choose these parameter values. The initial weights play a significant role in the convergence of the training method. Without a priori information about the final weights, it is a common practice to initialize all weights randomly with small absolute values, e.g., in the range [-0.5;+0.5]. Another option is to use the Nguyen-Widrow algorithm [58] which chooses initial values for the weights according to the training data and the number of hidden neurons.

For some applications, training time is a critical variable. For others, the training can be performed off-line and more training data are preferred over using insufficient training data to achieve greater network accuracy. Generally, rather than focusing on volume, it is better to concentrate on the quality and representational nature of the data set. A good training set should contain routine, unusual and boundary-condition cases.

Design considerations regarding a neural model include determining the number of input and output nodes to be used, the number of hidden layers in the network and the number of hidden nodes used in each hidden layer. The number of input/output nodes is typically taken to be the same as the number of independent/dependent variables of the problem, respectively.

In the past, there was a general practice of increasing the number of hidden layers to improve training performance. Keeping the number of layers at three and adjusting the number of processing elements in the hidden layer, can achieve the same goal. A trial-and-error approach is usually used to determine the number of hidden layer processing elements, starting with a low number of hidden units and increasing this number as learning problems occur.

The good generalization property of a network means the ability of a trained network to correctly predict a response to a set of inputs that has not seen before. It says that a trained network must perform well on a new dataset distinct from the one used for training. A very small value of the training error does not necessarily imply that a good model is obtained and that it can generalize good to new inputs. Even excessive training on the learning phase can make performance to decrease. In some cases, the network can be trained to fit the target data extremely well, but performs poorly on other data of a similar class. If this happens it is said that the network has memorized the target data and generalizes poorly. This is a symptom of the over-fitting phenomena.

During learning, the training error will usually decrease initially. However, as training continues, the error can start to increase. This happens because as training continues, weights become more tuned to particular features of the training data, therefore the weights are becoming more precisely. This phenomena is called over-fitting, which is when true error starts to increase during training, due to too much training time.

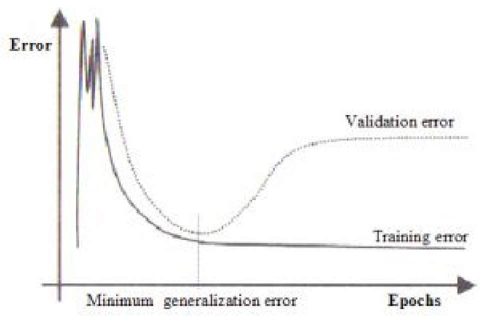


Figure 3.12. The over-fitting phenomena during neural network training.

Techniques for improving the generalization of a neural network model include regularization (normalization of the dataset) and early-stopping [73]. Early-stopping techniques divide the total data available for training into two data sets: the training set and the validation set. During training, the error on the validation set is monitored, and training is stopped when validation set error starts to increase. If the error on the validation set begins to rise, even though the error on the training data set continues to fall, then this is an indication of a loss of generalization, and is an indicator to stop training the network. If there is a succession of training epoch in which performance improves only for the training data and not for the validation data, over-fitting has occurred. At this point (called minimum generalization point) the training is stopped. This is shown in figure 3.12.

3.4.3 Advantages

To justify the choice made in this thesis regarding the neural network approach, the following list shows some of the potential benefits that may be obtained from using behavioral models based on neural networks.

- Using NNs, much more compact models (also easier to generate and use) are obtained than with physical or circuit models. They can be directly trained on the device non-linear measurements, without the need neither of the equivalent circuit topology, nor the corresponding non linear component constitutive descriptions.

- Comparing the NN approach with the polynomial one, contrary to the inherent local approximating properties of polynomials, neural networks can behave as global approximates, a potential advantage for modeling strongly nonlinear systems. NNs are, in principle, better in extrapolating beyond the zone where the system was tested during parameter extraction.

- The generalization ability of a NN model of producing results from inputs not previously presented during the training phase, by accepting an input and producing a plausible response, makes such a model robust against noisy data.

- Behavioral models resulting from the use of the NN paradigm may replace sub-circuits in simulations that would otherwise be too time consuming to perform with an analogue circuit simulator.

- Neural networks can be used to provide a general link from measurements or device simulations to circuit simulation. The discrete set of measurements or device simulations can be used as the target data set for a neural network. The NN then tries to learn the desired behavior. If this succeeds, the neural network can subsequently be used as a neural behavioral model in a circuit simulator.

- An efficient link, via neural models, between device simulation and circuit simulation would allow for the anticipation of consequences of technological choices to circuit performance. For example, neural models could contribute to a reduction of the design cycle when new semiconductor device technologies appear.
- Feedforward neural networks can be defined in such a way that stability can be guaranteed, as well as the networks having a unique behavior for a given set of (time-dependent) inputs. This implies that the corresponding neural models have unique solutions in both DC and transient analysis when they are applied in circuit simulation. This property can help the nonlinear solver of a circuit simulator to converge.
- Associated with the NNs models, a software toolbox could automatically generate the models in the appropriate syntax of a set of supported simulators. The automatic generation of models can help to ensure mutually consistent model implementations for the various supported simulators. In fact, one chapter of this thesis is dedicated to present a prototype of such a software toolbox.

3.5 Summary

In this chapter, the behavioral modeling approach has been introduced, and the three main techniques to build a behavioral model have been presented. The neural network paradigm has been explained in more detail, because this is the approach followed in this thesis. At the end of the chapter appear the advantages of using the neural network approach.

Chapter 4

Neural network-based behavioral model

In this chapter, the neural network-based model proposed in this thesis for behavioral modeling of electronic devices is presented. The proposed procedure for Volterra series extraction from the neural model is also explained.

4.1 Introduction

Electronic devices having nonlinear characteristics are, often, difficult to model accurately. Generally it is easier to perform measurements of the device parameters than to develop an analytical model of its behavior. As neural networks (NNs) can be used to learn a system dynamics from input/output data only, we have developed a NN model which reproduces the nonlinear behavior of an electronic device, using laboratory measurements or simulation data.

However, electronic devices nonlinear analysis requires an analytical model described as a closed-form function, that allows to draw conclusions about the device, such as the Volterra series model. The Volterra model was presented in Chapter 3 and it was explained that the terms of the series, named Volterra kernels, contain important information for the electronic engineer such as the model derivatives, but they are difficult to calculate, estimate or measure.

In this chapter will be shown how the neural model and the analytical Volterra series model of a device are totally equivalent, which is the relationship that exists between the kernels and the network parameters, and how the kernels can be obtained from the combination of the networks weights and bias values. Therefore, this chapter explains how it is possible to build an analytical expression for a nonlinear dynamic device or system (the Volterra series model) with parameters of a standard NN, properly trained with measurements or simulation data. Two cases study are presented for the application of the novel approach: at device-level analysis, a Field-effect transistor (FET); and at system-level analysis, a Power Amplifier (PA).

4.2 Model building

The procedure proposed starts with the definition of a neural network model. The type of network that is used is a Time-delayed neural network (TDNN) because, as was explained in Chapter 3, this type of network can represent a system which has a dynamic behavior. The TDNN model is based on a Multi-layer perceptron net (MLP) and the general topology of the model, for two input variables and one output, can be seen in figure 4.1.

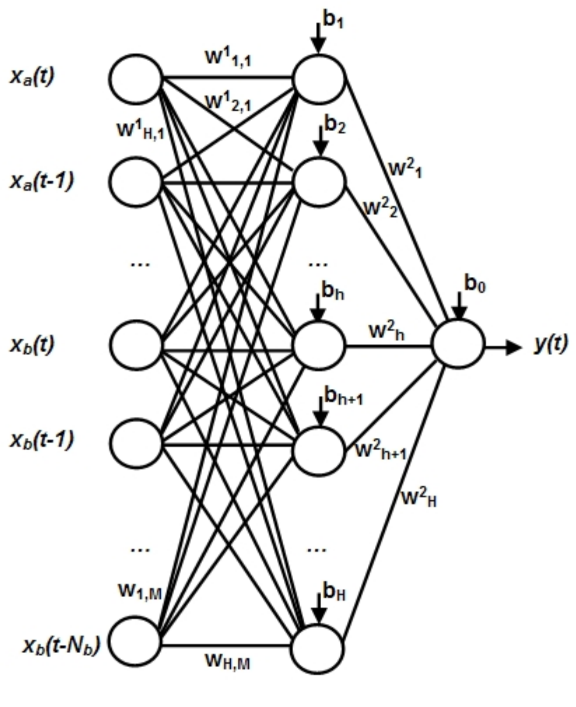


Figure 4.1. General Time-Delayed neural network (TDNN) model able to represent an electronic device/system.

The model has one input layer, one hidden layer and an output layer with a single output. The input layer has as inputs the samples of the independent variables, together with their time-delayed values. The general model here presented shows a two-input variable dependence (x_a and x_b) of one output variable (y), but the model can be easily extended to more or less input/output variables.

Each variable can have a delay tap between 0 and N. If we call N_a the maximum number of delays for variable x_a and N_b the maximum number of delays for variable x_b, then the total number of input neurons is M ($M = (N_a + 1) + (N_b + 1)$).

In the hidden layer, the number h of hidden neurons varies between 1 and H. The hidden units have a nonlinear activation function. In general, the hyperbolic tangent (tanh) will be used because this function is usually chosen in the electronics field for the type of devices we want to model. If necessary, the number of neurons in the hidden layer can be changed to improve the network accuracy. The hidden neurons receive the sum of the weighted inputs plus a corresponding bias value for each neuron b_h. All neurons having bias values gives more degrees of freedom to the learning algorithm, which means more parameters that can be optimized to improve the model accuracy.

The weights between the layers are described with the usual perceptron notation introduced in Chapter 3, where the sub-indexes indicate the sense of the connection, the origin and the destination of the connection weight, e.g. $w_{j,i}$ means that the weight w relates the destination neuron j with the origin neuron i. However, a modification in the notation was introduced, adding a super-index to the weights, to more easily identify to which layer it belongs. Therefore, w^1 means that the weight w belongs to the connection between the inputs and the first hidden layer, and w^2 means that the weight w belongs to the connection between the first hidden layer and the second one (in our particular case, the second hidden layer happens to be the output layer). In this second group of connections weights, the sub-index which indicates the destination neuron has been eliminated because there is only one possible destination neuron (the output).

This unique output neuron has a linear activation function which acts as a normalization neuron (this is usual choice in MLP models). Therefore, the output of the proposed neural network model is calculated as the sum of the weighted outputs of the hidden neurons plus the corresponding output neuron bias (b_0), yielding equation A.1, where $vi = [a,b]$ is the variable index.

$$y(t) = b_0 + \sum_{h=1}^{H} w_h^2 \tanh\left(b_h + \sum_{i=0}^{N_{vi}} w_{h,i+1}^1 x_{vi}(t-i)\right) \tag{4.1}$$

Once the TDNN model has been defined, it is trained with device/system time-domain measurements. The input and output waveforms are expressed in terms of their discrete samples in the time-domain. To improve the network accuracy and speed up learning, the inputs are normalized to the domain of the hidden neurons nonlinear activation functions (in our case, for the hyperbolic tangent tanh, the interval should be $[-1;+1]$). The formula used to obtain normalization in this interval is $x^{norm} = \frac{2(x-min\{x\})}{(max\{x\}-min\{x\})} - 1$.

During training, the network parameters are optimized using a backpropagation algorithm. The Levenberg-Marquardt algorithm [51], which has been chosen due to its good performances and speed in execution. This method is a second-order backpropagation algorithm because it uses the Hessian matrix[1] in the calculations. To evaluate the TDNN learning accuracy, the mean square error (mse) is calculated using equation 4.2, being P the number of input/output pairs in the training set, y_T the output target and y_{NN} the network output.

$$mse = \frac{1}{P}\sum_{k=1}^{P} E(k)^2 = \frac{1}{P}\sum_{k=1}^{P}\left(y_T(k) - y_{NN}(k)\right)^2 \tag{4.2}$$

The good generality property of a neural network says that it must perform well on a new dataset distinct from the one used for training. Even a excessive number of epochs or iterations on the learning phase could make performance to decrease, causing the over-fitting phenomenon.

That is why, to avoid it, the total amount of data available from measurement/simulation is divided into training and validation subsets, all equally spaced. We have used the "early-stopping" technique [73], where if there is a succession of training epoch in which accuracy improves only for the training data and not for the validation data, over-fitting has occurred and the learning is terminated.

The initialization of the network is an important issue for training the TDNN with the back-propagation algorithm, in particular in what respect speed of execution, that is why the initial weights and biases of the model are calculated using the Nguyen-Widrow initial conditions [58], instead of a purely random initialization.

4.3 Volterra identification method

In this section, the found procedure for generating the Volterra series models using the weights and bias values of the TDNN proposed model is briefly explained. A more detailed explanation of the procedure can be found on the Appendix A.

The procedure, which can be applied to a function depending on one, two or even more input variables, begins by expanding the output of the TDNN model (equation A.1) as a Taylor series around the bias values of the hidden nodes. Therefore, the hidden nodes activation functions derivatives regarding the bias values have to be calculated (in this case, the tanh derivatives). This yields equation A.2, where d means derivative order.

$$y(t) = b_0 + \sum_{h=1}^{H} w_h^2 \sum_{d=0}^{+\infty}\left[\frac{\frac{\partial^d \tanh}{\partial x^d}\big|_{x=b_h}}{d!}\left(\sum_{i=0}^{N_{vi}} w_{h,i+1}^1 x_{vi}(t-i)\right)^d\right] \tag{4.3}$$

[1]Matrix of second derivatives of the error matrix E with respect to the weights w. It contains information about how the error gradient changes in different directions in weight space.

Developing equation A.2 and grouping common terms according to the derivatives yields equation A.6, which is the final form of the network model output. Equation A.7 shows an instance of this formula for the proposed TDNN with two input variables x_a and x_b, when $d = 3$.

$$y(t) = \sum_{d=0}^{+\infty} \left[\sum_{h=1}^{H} w_h^2 \sum_{i=0}^{N_{vi}} \cdots \sum_{j=0}^{N_{vi}} w_{h,i+1}^1 \cdots w_{h,j+1}^1 \left(\frac{\frac{\partial^d \tanh}{\partial x^d} \,|_{x=b_h}}{d!} \right) \right] (x_{vi}(t-i) \cdots x_{vi}(t-j))^d \tag{4.4}$$

$$\begin{aligned} y(t) \;=\; & b_0 + \sum_{h=1}^{H} w_h^2 \tanh(b_h) + \\ & \left[\sum_{h=1}^{H} w_h^2 w_{h,1}^1 \left(\frac{\partial \tanh}{\partial x} \,|_{x=b_h} \right) \right] x_a(t) + \\ & \left[\sum_{h=1}^{H} w_h^2 w_{h,h}^1 \left(\frac{\partial \tanh}{\partial x} \,|_{x=b_h} \right) \right] x_b(t) + \\ & \left[\sum_{h_1=1}^{H} \sum_{h_2=1}^{H} w_{h_1}^2 w_{h2,1}^1 w_{h_2,1}^1 \left(\frac{\frac{\partial^2 \tanh}{\partial x^2} \,|_{x=b_h}}{2!} \right) \right] x_a^2(t) + \\ & \left[\sum_{h_1=1}^{H} \sum_{h_2=1}^{H} w_{h_1}^2 w_{h_2,h}^1 w_{h_2,h}^1 \left(\frac{\frac{\partial^2 \tanh}{\partial x^2} \,|_{x=b_h}}{2!} \right) \right] x_b^2(t) + \\ & \left[\sum_{h=1}^{H} w_h^2 w_{h,1}^1 w_{h,h}^1 \left(\frac{\frac{\partial^2 \tanh}{\partial x^2} \,|_{x=b_h}}{2!} \right) \right] x_a(t) x_b(t) + \cdots + \\ & \left[\sum_{h_1=1}^{H} \sum_{h_2=1}^{H} w_{h_1}^2 w_{h_2,1}^1 w_{h_2,1}^1 w_{h_2,1}^1 \left(\frac{\frac{\partial^3 \tanh}{\partial x^3} \,|_{x=b_h}}{3!} \right) \right] x_a^3(t) + \\ & \left[\sum_{h_1=1}^{H} \sum_{h_2=1}^{H} w_{h_1}^2 w_{h_2,h}^1 w_{h_2,h}^1 w_{h_2,h}^1 \left(\frac{\frac{\partial^3 \tanh}{\partial x^3} \,|_{x=b_h}}{3!} \right) \right] x_b^3(t) + \\ & \left[\sum_{h_1=1}^{H} \sum_{h_2=1}^{H} \sum_{h_3=1}^{H} w_{h_1}^2 w_{h_2,1}^1 w_{h_2,1}^1 w_{h_3,h}^1 \left(\frac{\frac{\partial^3 \tanh}{\partial x^3} \,|_{x=b_h}}{3!} \right) \right] x_a^2(t) x_b(t) + \\ & \left[\sum_{h_1=1}^{H} \sum_{h_2=1}^{H} \sum_{h_3=1}^{H} w_{h_1}^2 w_{h_2,1}^1 w_{h_3,h}^1 w_{h_3,h}^1 \left(\frac{\frac{\partial^3 \tanh}{\partial x^3} \,|_{x=b_h}}{3!} \right) \right] x_a(t) x_b^2(t) + \cdots \end{aligned} \tag{4.5}$$

If this last equation is compared to the double Volterra series model (representation for two input variables) [62][56] partially reproduced in equation A.8, the Volterra kernels h_n can be easily recognized as the terms between brackets in the TDNN output expression.

$$\begin{aligned} y(t) &= h_0 + \sum_{k=0}^{+\infty} h_1(k)x_1(t-k) + \\ &\sum_{k=0}^{+\infty} h_1(k)x_2(t-k) + \\ &\sum_{k=0}^{+\infty} h_2(k,k)x_1(t-k)^2 + \\ &\sum_{k=0}^{+\infty} h_2(k,k)x_2(t-k)^2 + \\ &\sum_{k=0}^{+\infty} h_3(k,k,k)x_1(t-k)^3 + \\ &\sum_{k=0}^{+\infty} h_3(k,k,k)x_2(t-k)^3 + \\ &\sum_{k=0}^{+\infty} h_n(k,\cdots,k)x_1(t-k)^n + \\ &\sum_{k=0}^{+\infty} h_n(k,\cdots,k)x_2(t-k)^n + \cdots \end{aligned} \tag{4.6}$$

From this comparison some formulas have been derived which allow the calculation of any Volterra kernel order using the weights and hidden neurons bias values of a TDNN. These, are general formulas that allow calculating any kernel order from a network with any topology (any number of input neurons, any number of hidden neurons and any activation function $af(x)$).

$$h_0 = b_0 + \sum_{h=1}^{H} w_h^2 af(b_h) \tag{4.7}$$

$$h_1(k) = \sum_{h=1}^{H} w_h^2 w_{h,k+1}^1 \left(\frac{\partial af}{\partial x} \mid_{x=b_h} \right) \tag{4.8}$$

$$h_2(k_1,k_2) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 \frac{\left(\frac{\partial^2 af}{\partial x^2} \mid_{x=b_h} \right)}{2!} \tag{4.9}$$

$$h_3(k_1,k_2,k_3) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 w_{h,k_3+1}^1 \frac{\left(\frac{\partial^3 af}{\partial x^3} \mid_{x=b_h} \right)}{3!} \tag{4.10}$$

$$h_n(k_1,\cdots,k_n) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 \cdots w_{h,k_n+1}^1 \frac{\left(\frac{\partial^n af}{\partial x^n} \mid_{x=b_h} \right)}{n!} \tag{4.11}$$

More than one kernel can be used to describe a given system. A kernel can be rewritten in several ways simply by reordering the variables. This feature can lead to difficulties when system properties are described in terms of properties of the kernel. Therefore, it becomes important in many situations to impose uniqueness by working with a special, restricted form for the kernel. One such form is the symmetric kernel which satisfies equation 4.12, where $\pi(.)$ denotes any permutation of the integers $1,\cdots,n$. It can be shown that without loss of generality the kernel of a homogeneous system can be assumed to be symmetric [67].

$$h_{sym}\left(k_1,\cdots,k_n\right) = h_{sym}\left(k_{\pi(1)},\cdots,k_{\pi(n)}\right) \tag{4.12}$$

In fact any given kernel, say $h(k_1,\cdots,k_n)$, can be replaced by a symmetric kernel simply by setting the kernel as shows equation 4.13, where the indicated summation is over all n! permutations of the integers 1 through n.

$$h_{sym}\left(k_1,\cdots,k_n\right) = \frac{1}{n!}\sum_{\pi(.)} h_{sym}\left(k_{\pi(1)},\cdots,k_{\pi(n)}\right) \tag{4.13}$$

The following group of equations can be used when the hidden neurons activation function is the hyperbolic tangent. In the formulas, the hyperbolic tangent derivatives have been developed, where $\tanh = \tanh(b_h)$.

$$h_0 = b_0 + \sum_{h=1}^{H} w_h^2 \tanh \tag{4.14}$$

$$h_1(k) = \sum_{h=1}^{H} w_h^2 w_{h,k+1}^1 \left(1 - \tanh^2\right) \tag{4.15}$$

$$h_2(k_1,k_2) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 \frac{\left(-2\tanh + 2\tanh^3\right)}{2!} \tag{4.16}$$

$$h_3(k_1,k_2,k_3) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 w_{h,k_3+1}^1 \frac{\left(-2 + 8\tanh^2 - 6\tanh^4\right)}{3!} \tag{4.17}$$

$$h_n(k_1,\cdots,k_n) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 \cdots w_{h,k_n+1}^1 \frac{\left(\frac{\partial^n \tanh}{\partial x^n}\,|_{x=b_h}\right)}{n!} \tag{4.18}$$

However, for some particular cases (i.e. PA nonlinearities), we have found that there is a simpler way of calculating the kernels, avoiding the need for applying a Taylor expansion to the TDNN model and the calculation of derivatives of the activation functions. The proposal is to use polynomial activation functions in the hidden neurons, in cases where the system order is previously known, because that increases the speed and the simplicity in the Volterra series extraction. In this case, the procedure to follow is simply to develop and to distribute the polynomials.

In general, a polynomial as activation function in the TDNN model should look like equation 4.19, where P means polynomial order.

$$y(t) = b_0 + \sum_{h=1}^{H} w_h^2 \left(b_h + \sum_{i=0}^{N_{vi}} w_{h,i+1}^1 x_{vi}(t-i) \right)^P \tag{4.19}$$

The procedure, then, for finding the Volterra kernels, becomes simply distributing the polynomials and after common factoring, the kernels are immediately identifiable, as shown in equation A.4.

$$\begin{aligned} y(t) &= b_0 + \sum_{h=1}^{H} w_h^2 (b_h)^P + \\ & \left[\sum_{h=1}^{H} w_h^2 \sum_{i=0}^{N_{vi}} w_{h,i+1}^1 \left(P b_h^{P-1} \right) \right] x_{vi}(t-i) + \\ & \left[\sum_{h=1}^{H} w_h^2 \sum_{i=0}^{N_{vi}} \sum_{j=0}^{N_{vi}} w_{h,i+1}^1 w_{h,j+1}^1 \left(\frac{P(P-1) b_h^{P-2}}{2!} \right) \right] x_{vi}(t-i) x_{vi}(t-j) + \\ & \left[\sum_{h=1}^{H} w_h^2 \sum_{i=0}^{N_{vi}} \sum_{j=0}^{N_{vi}} \sum_{k=0}^{N_{vi}} w_{h,i+1}^1 w_{h,j+1}^1 w_{h,k+1}^1 \left(\frac{P(P-1)(P-2) b_h^{P-3}}{3!} \right) \right] \\ & x_{vi}(t-i) x_{vi}(t-j) x_{vi}(t-k) + \cdots \end{aligned} \tag{4.20}$$

The new simpler formulas to calculate the kernels using polynomial activation functions are the following:

$$h_0 = b_0 + \sum_{h=1}^{H} w_h^2 (b_h)^P \tag{4.21}$$

$$h_1(k) = \sum_{h=1}^{H} w_h^2 w_{h,k+1}^1 \left(P b_h^{P-1} \right) \tag{4.22}$$

$$h_2(k_1,k_2) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 \left(\frac{P(P-1) b_h^{P-2}}{2!} \right) \tag{4.23}$$

$$h_3(k_1,k_2,k_3) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 w_{h,k_3+1}^1 \left(\frac{P(P-1)(P-2) b_h^{P-3}}{3!} \right) \tag{4.24}$$

$$h_n(k_1,\cdots,k_n) = \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 \cdots w_{h,k_n+1}^1 \left(\frac{P \cdots (P-(n-1)) b_h^{P-n}}{n!} \right) \tag{4.25}$$

Once the TDNN model has been trained with time-domain measurements and the desired accuracy has been obtained, an algorithm has been developed and implemented inside a software toolbox prototype which allows the creation and use of the proposed neural models and the building of the corresponding Volterra model. The software toolbox, which is explained in detail in chapter 5, produces a file which contains the TDNN/Volterra models and their parameters, to be implemented inside a circuits simulator as a full black-box model. The models can be used interchangeably, they both represent the same behavior, but in a different format. As the simulation and validation results of chapter 6 show, they give exactly the same response.

In the next section, applications of the proposed models, as well as their detailed implementation inside a simulator, will be presented.

4.4 Applications

4.4.1 Transistor model

Inside the traditional equivalent-circuit Curtice model [12] of a FET, most of the nonlinear behavior of the device is due to the drain to source current I_{ds}, that depends on the gate and drain voltages V_{gs} and V_{ds}. Curtice proposed an equation for modeling the current behavior (presented in chapter 2) which generates the curves that can be seen in figure 4.2 when different input voltage combinations are considered. These are the typical I/V curves of a Curtice FET device model in DC. In the figure, the device bias point $I_{ds0} = (V_{ds0}, V_{gs0})$ has been marked.

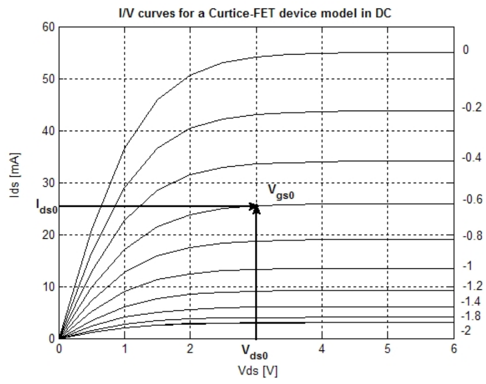


Figure 4.2. Typical I/V curves for a FET.

This drain current nonlinear behavior in DC can also be described with a more compact model, which could also provide more information about the device under test (DUT), i.e. intermodulation phenomena. This compact model is the Volterra series model, shown in equation 4.26 including up to the third order terms. This equation represents a static relationship, the instantaneous behavior of the drain current in DC. Therefore, in this case, the Volterra series turns to be a simple Taylor expansion.

$$\begin{aligned} I_{ds} \;=\; & I_{ds}(V_{gs0},V_{ds0}) + \\ & G_{m1}.V_{gs} + G_{ds}.V_{gs} + \\ & G_{m2}.V_{gs}^2 + G_{ds2}.V_{gs}^2 + G_{md}.V_{ds}.V_{gs} + \\ & G_{m3}.V_{gs}^3 + G_{ds3}.V_{gs}^3 + G_{m2d}.V_{ds}^2.V_{gs} + G_{md2}.V_{ds}.V_{gs}^2 \end{aligned} \tag{4.26}$$

With this model, the intermodulation distortion (IMD) phenomena can be easily identified, as shows figure 4.3.

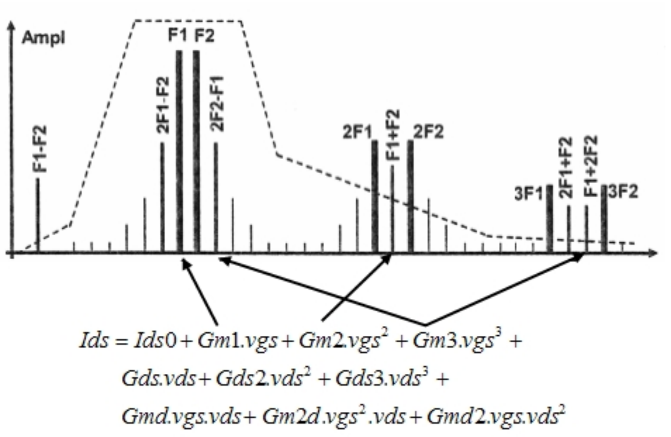

Ampl
F1 F2
F1-F2
2F1-F2
2F2-F1
2F1
F1+F2
2F2
3F1
2F1+F2
F1+2F2
3F2
Ids = Ids0 + Gm1.vgs + Gm2.vgs² + Gm3.vgs³ +
Gds.vds + Gds2.vds² + Gds3.vds³ +
Gmd.vgs.vds + Gm2d.vgs².vds + Gmd2.vgs.vds²

Figure 4.3. Intermodulation distortion (IMD) phenomena identified by a Volterra model.

Once the model is built, the values of the Volterra kernels have to be identified. If an analytical model for I_{ds} is available for the DUT, such as the Curtice formula for a FET, the kernels can be determined analytically. A Taylor expansion is applied to the original formula at the bias point and the kernels are calculated as the first order (G_{m1} and G_{ds}), second order (G_{m2}, G_{ds2} and G_{md}) and third order (G_{m3}, G_{ds3}, G_{m2d}, G_{md2}) derivatives of the current with respect to the voltages. For example, the kernel G_{m1}, which is a figure of the transconductance of the device[2] would be calculated as the first derivative of the drain current with respect to the gate voltage. Similarly, the kernel G_{ds} or the output conductance[3] would be obtained as the first derivative of the drain current with respect to the drain voltage.

[2]Expression of the performance of the transistor; e.g. the larger the transconductance the greater the gain it is capable of delivering, when all other factors are held constant.

[3]The internal conductance of a circuit or device, as seen at the output terminals.

$$G_{m1} = \frac{\partial I_{ds}}{\partial V_{gs}} |_{V_{gs0},V_{ds0}} \tag{4.27}$$

$$G_{ds} = \frac{\partial I_{ds}}{\partial V_{ds}} |_{V_{gs0},V_{ds0}} \tag{4.28}$$

However, if an analytical model for the DUT is not available, the formulas 4.7 to 4.11 proposed in this chapter can be used, which extract the Volterra kernels from the TDNN model. To validate the proposal, the neural network model of figure 4.4 is used to model the drain current nonlinearity and to obtain the corresponding Volterra kernels in DC. The inputs to the model are the instantaneous gate and drain voltages ($V_{gs}(t)$ and $V_{ds}(t)$) and the output is the instantaneous drain-source current $I_{ds}(t)$, sampled on the time-domain. Using the procedure mentioned in the first part of this chapter, the Volterra series model is found and compared with the analytical Curtice model for a FET transistor. The results are presented in chapter 6.

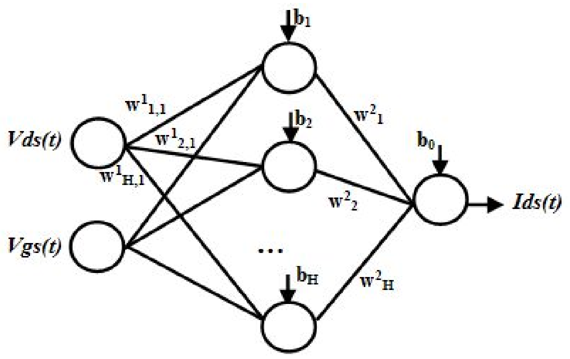


Figure 4.4. Neural network model for instantaneous drain current in a transistor.

If memory effects have to be considered into the model, the TDNN model has the form of figure 4.5, where not only the instant values of the input variables are considered, but also their past values. This model has also been validated with measurements on a real DUT, i.e. the characterization of a transistor for a PA design project, where the device works on different classes according to the bias. These results also appear on chapter 6.

As chapter 5 will illustrate, the model generation, training and use, and the associated Volterra model obtention, have been codified inside a software toolbox prototype, with a user-friendly interface which helps the electronic designer in the task of behavioral model definition.

When the model has to be simulated, together with other traditional models inside a circuits-based simulator, the toolbox generates a file containing the models

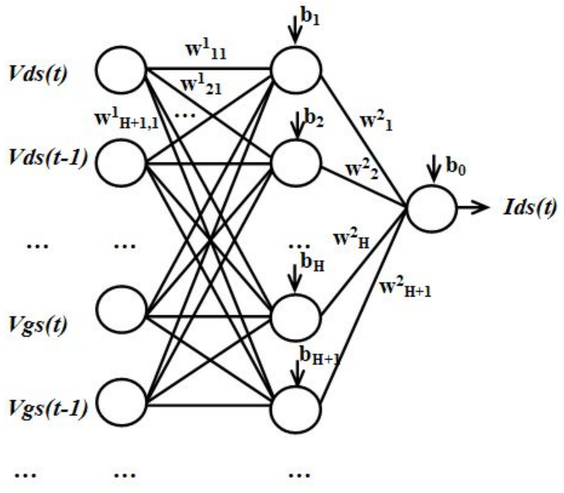


Figure 4.5. TDNN transistor model with memory.

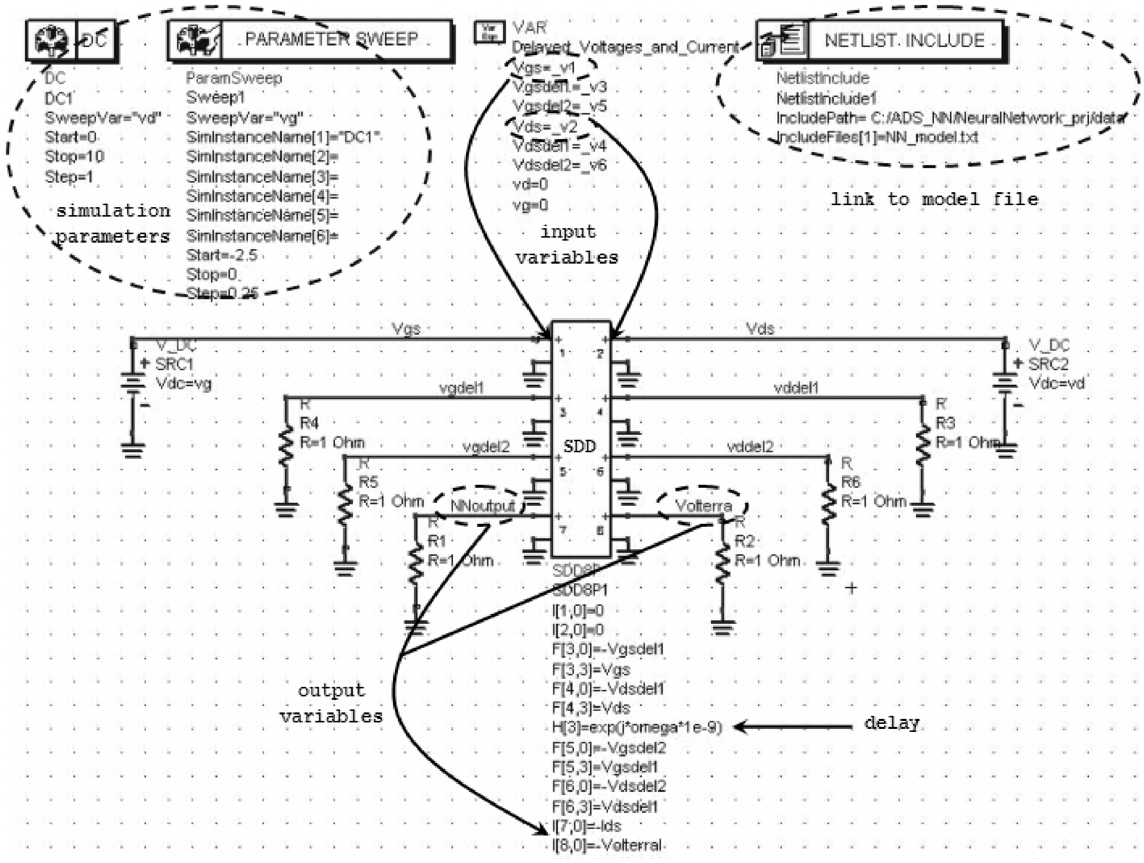
definition and parameters. This way the model can be loaded into a user-defined model inside a simulator. For example, in the simulator Agilent ADS©, an element named symbolically defined device (*SDD*) can be linked to a file containing the model definition. The ports of the *SDD* element are defined to be the model variables. Generators can be associated to the model as well, to simulate the input signals. Once the model has been loaded, DC, AC and transient simulations as well as harmonic-balance analysis can be made.

The following lines explain the implementation of the proposed behavioral model inside Agilent ADS© as a full black-box model, using a *SDD* element.

Figure 4.6 shows an implementation inside the circuits simulator of a general TDNN/Volterra model for a transistor (two possible inputs with a maximum of two delayed samples each). The figure marks the elements inside the schematics that are needed for model implementation.

The *SDD* element allows the definition of a black-box model, which receives input stimulus through some of its ports and produces outputs through some other ports. In the shown example, the generic *SDD* defined has eight ports: two input ports (port 1 and 2), four internal ports with the delays of the input variables, and two output ports (ports 7 and 8). Input variables declaration is marked in the figure and it associates port 1 of the *SDD* with the input variable, in this example for a transistor, the gate voltage V_{gs}); and port 2 with V_{ds} (drain voltage). In the schematics this is performed with the following variables declaration:

$$Vgs = \_v1 \tag{4.29}$$
$$Vds = \_v2 \tag{4.30}$$



```
DC
DC1
SweepVar="vd"
Start=0
Stop=10
Step=1

ParamSweep
Sweep1
SweepVar="vg"
SimInstanceName[1]="DC1"
SimInstanceName[2]=
SimInstanceName[3]=
SimInstanceName[4]=
SimInstanceName[5]=
SimInstanceName[6]=
Start=-2.5
Stop=0
Step=0.25

VAR
Delayed_Voltages_and_Current
Vgs=_v1
Vgsdel1=_v3
Vgsdel2=_v5
Vds=_v2
Vdsdel1=_v4
Vdsdel2=_v6
vd=0
vg=0

NetlistInclude
NetlistInclude1
IncludePath= C:/ADS_NN/NeuralNetwork_prj/data
IncludeFiles[1]=NN_model.txt

SDD8P
SDD8P1
I[1,0]=0
I[2,0]=0
F[3,0]=-Vgsdel1
F[3,3]=Vgs
F[4,0]=-Vdsdel1
F[4,3]=Vds
H[3]=exp(j*omega*1e-9)
F[5,0]=-Vgsdel2
F[5,3]=Vgsdel1
F[6,0]=-Vdsdel2
F[6,3]=Vdsdel1
I[7,0]=-Ids
I[8,0]=-Volterral
```

Figure 4.6. Black-box implementation inside a circuits simulator of the TDNN/Volterra model for transistor DC analysis.

Inside the *SDD* the following statements indicate that those ports will not absorb current (they are inputs):

$$
\begin{aligned}
I[1,0] &= 0 \qquad (4.31)\\
I[2,0] &= 0 \qquad (4.32)
\end{aligned}
$$

Depending on the kind of simulations that will be done with the model, the input ports can be terminated with resistances in the case of DC analysis or they can be replaced with sources in the case of AC and Harmonic Balance simulations. In the figure, as the model will be tested in DC simulation, the *DC* and *PARAMETER SWEEP* elements appear in the schematics, which set the sweeps that will be performed on the input variables.

When modeling the dynamic behavior, delayed versions of the input variables are needed to feed the TDNN. In Agilent ADS© delay in time can be implemented inside a *SDD* element assigning the delay Fourier Transform Operator ($e^{j\omega\tau}$) as a weighting function (H) at the port where the delay has to be imposed.

This will be exemplified for the first delayed sample of the $V_{gs}(t)$ variable, $V_{gs}(t-1)$, which will be called $Vgsdel1$ in the model. First of all, it is given a name and associated with a *SDD* port, i.e. port 3:

$$Vgsdel1 \;=\; \_v3 \tag{4.33}$$

Inside the *SDD* element, the variable is declared to be an output (generates a signal) which will be a delayed version (according to some defined delay τ) of the V_{gs} variable. This is done with the following formulas:

$$\begin{aligned} F[3,0] &= -Vgsdel1 && (4.34)\\ F[3,3] &= Vgs && (4.35)\\ H[3] &= e^{j\omega\tau} && (4.36) \end{aligned}$$

which represent $Vgsdel1 = Vgs * H = Vgs * e^{j\omega\tau}$. The formulas have to be repeated for each delayed variable which must be defined.

The output variables are given a name and defined to appear at ports 7 and 8 with the lines:

$$\begin{aligned} I[7,0] &= -Ids && (4.37)\\ I[8,0] &= -Volterra1 && (4.38) \end{aligned}$$

These output variables have the names `NNoutput` and `Volterra` inside the simulator, i.e. for plotting. They produce their responses according to the formulas of the behavioral TDNN/Volterra model contained in the model file and loaded into the schematics.

The *NETLIST INCLUDE* element provides the link between the model file and the *SDD* of the schematics. Its parameters *IncludePath* and *IncludeFile* indicate the location and the name of the file containing the model. In the example, the file that contains the model is named `NN_model.txt` and it contains the definition of the output variables `Ids` and `Volterra1` as result of a neural model. The model parameters are loaded and used inside the simulator, therefore when the input variables are mentioned in the model file, their names must be the same names defined for them in the schematics.

For example, the `Ids` variable and parameters definition, which could be find inside the file for a TDNN model with M=0, H=2 and tanh activation function, would be like follows:

```
b0=45.6198;
w1=-0.0069;
w11=24.2747;
w12=23.4633;
b1=-11.1024;
w2=45.5063;
w21=1.2322;
w22=1.5255;
b2=-8.7518;
Ids=b0+w1*tanh(w11*Vds+w12*Vgs+b1)+w2*tanh(w21*Vds+w22*Vgs+b2);
```

In the case of the `Volterra1` output variable, inside the same file, it would be:

```
h0=0.006;
h1(0)=0.4149;
h1(1)=0.0806;
h2(0,0)=0.0033;
h2(1,1)=-0.0015;
h2(0,1)=0.0001;
h3(0,0,0)=0.0007;
h3(1,1,1)=-0.0001;
h3(0,0,1)=0.0002;
h3(1,1,0)=-0.0002;
Volterra1=h0+h1(0)*Vds+h1(1)*Vgs+h2(0,0)*Vds^2+h2(1,1)*Vgs^2+
2*h2(0,1)*Vds*Vgs+h3(0,0,0)*Vds^3+h3(1,1,1)*Vgs^3+
3*h3(0,0,1)*Vds^2*Vgs+3*h3(1,1,0)*Vds*Vgs^2;
```

For harmonic balance analysis, the implementation of the TDNN/model inside the simulator should look like figure 4.7.

4.4.2 Power Amplifier model

When characterizing a power amplifier (PA) in function on input/output powers, the TDNN model of figure 4.8 can be used.

The training set for the model is formed with input/output sinusoidal time-domain waveforms power samples (P_{in}/P_{out}) and their delayed replies, according to a certain tap delay (T_d) which must be determined. When using a two-tone PA characterization, it could be calculated from the bandwidth (BW) according to:

$$T_d = \frac{1}{2 * BW} \tag{4.39}$$

$$BW = harm * fund \tag{4.40}$$

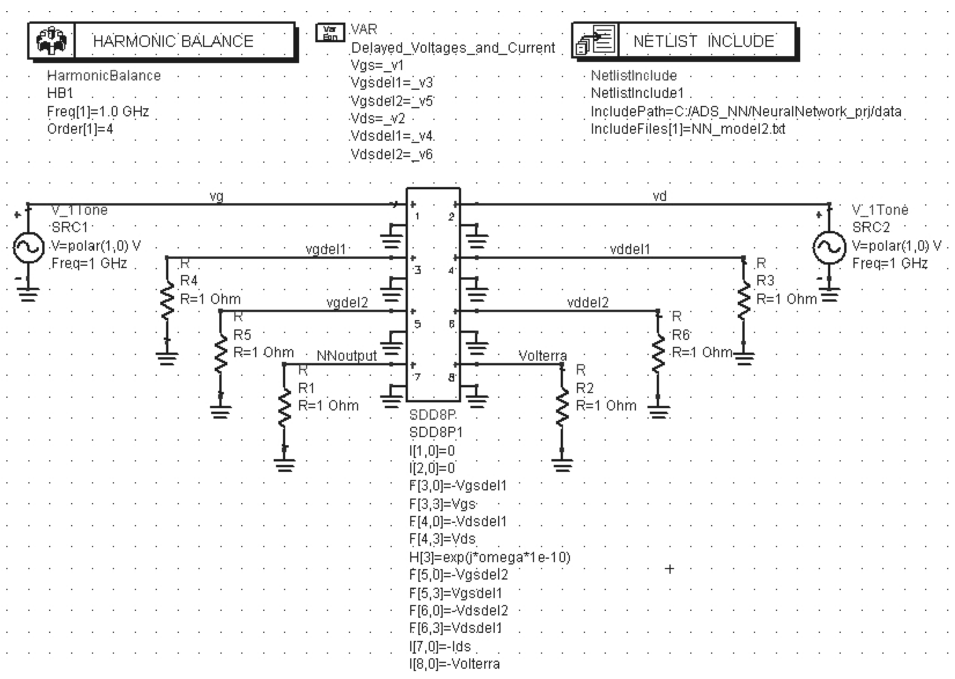


Figure 4.7. TDNN/Volterra model black-box implementation for harmonic balance analysis.

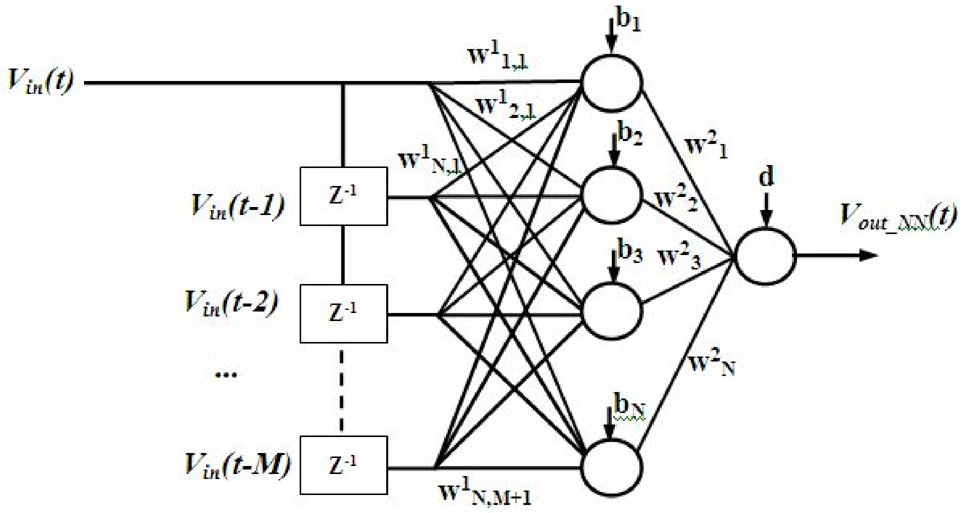


Figure 4.8. TDNN model for a PA.

$$fund = \frac{f_1 + f_2}{2} \tag{4.41}$$

where *fund* is the middle of the input tones f_1 and f_2, and *harm* is the harmonics order to be considered in the model. In this case the sampling period (T_s) should be:

$$T_s = nT_d; n \geq 1 \tag{4.42}$$

The input-output training data sets are built with cascading samples obtained from each source power level. Input data vectors from different input levels should be first joined together; the resulting vector should be copied and delayed as many times, to represent the network input, as necessary to take into account memory effects. The TDNN model must be trained with all the data, simultaneously. This allows to obtain a nonlinear model able to characterize, at the same time, the low and high nonlinear distortion, for all the input power levels of interest.

When the order of the system being modeled is known a priori, this information can be used to improve the model accuracy. As will be shown in the results chapter, when the system is of order 3 and therefor cubic activation functions are used for the TDNN model, the results are much better than using the hyperbolic tangent.

Concerning implementation of a general behavioral TDNN/Volterra model for PA simulations based on $Pin/Pout$ measurements inside a circuits simulator, it is presented in figure 4.9.

The same considerations for the implementation presented before for the TDNN model of a transistor, hold also for this PA model. The only observation to be made is that in this case there only one input variable `Vin` at port 1 of the *SDD*, which can have delays versions of itself, calculated exactly as was explained before. In this example the file that contains the model definition is named `NNamplifier.txt` and the output variables of the model, defined inside the model file, are `Vout` and `Volterra`.

The schematics shows also a *HARMONIC BALANCE* element which is setup to perform simulations at 1 GHz including 4 harmonics for the PA model, and therefore the input port 1 has a sinusoidal source connected to it instead of a resistance as in the DC case.

4.5 Summary

This chapter began with an explanation of the neural network-based model proposed in this thesis for behavioral modeling of electronic devices and systems. The proposed procedure for Volterra series extraction from the neural model, as well as model implementation inside a commercial circuits simulator have been explained. Finally, examples of the application of the model to transistors and power amplifier modeling have been shown.

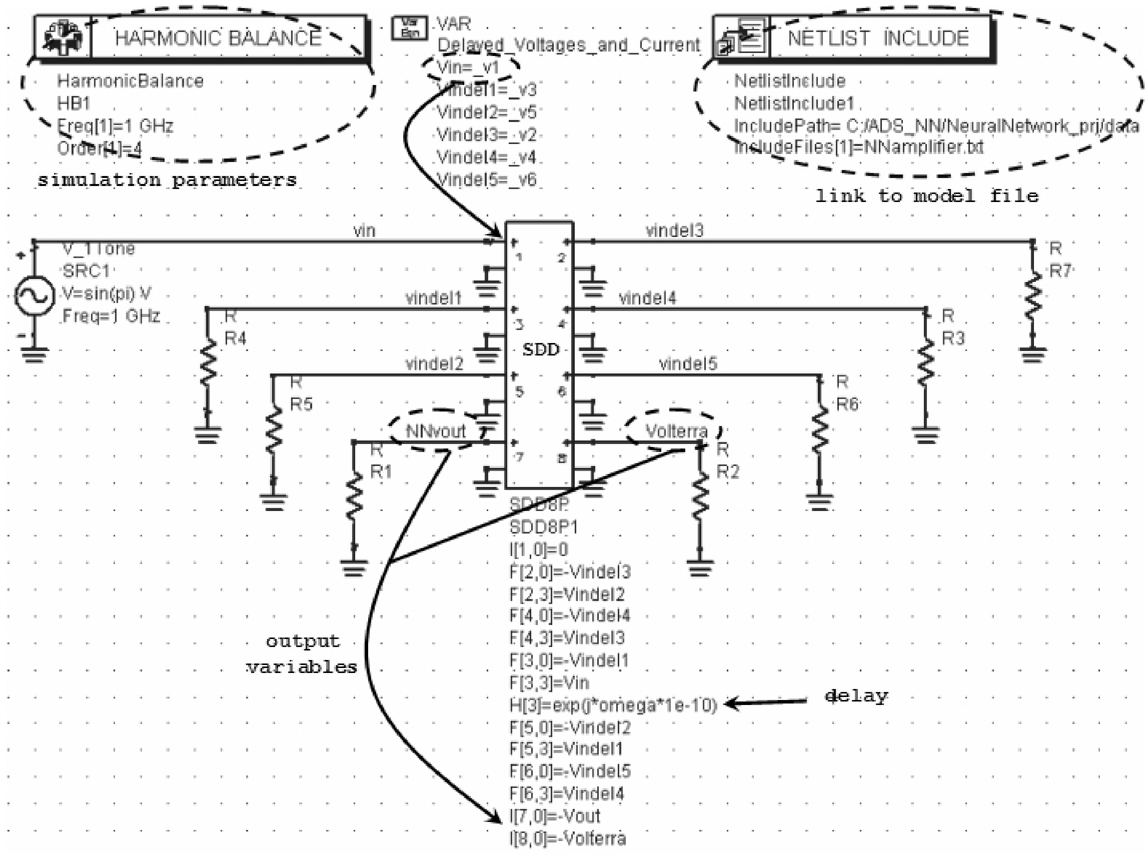


Figure 4.9. Black-box implementation inside a circuits simulator of the TDNN/Volterra model for a PA.

Chapter 5

Results

This chapter shows the results of validation of the proposed model against measurements. The results obtained from four modeling cases study are here reported.

5.1 Introduction

First of all, the proposed time-delayed neural network (TDNN) model was applied for the characterization of the static DC behavior of a FET transistor, compared to the analytical Curtice model simulations. For this device, the corresponding Volterra series model has been extracted as well, and the results from the comparison are reported.

Concerning transistor dynamic behavior modeling, results of a GaAs MESFET measurements used to train the TDNN model are shown, comparing a neural model without memory (no delayed variables) and a model with memory, highlighting the importance of the inclusion of memory into the TDNN. Finally, a GaN HEMT transistor has been also measured and characterized with the TDNN, when the device works in different classes according to the bias point. These measurements were performed at Politecnico di Torino.

Finally, regarding the power amplifiers (PAs) dynamic behavior modeling, measurements with one-tone and two-tones inputs have been made at Università Tor Vergata di Roma. These measurements were used to train the TDNN model for Pin/Pout waveforms characterization.

The strategy followed to set the TDNN memory is the same adopted for the design of numerical filters, where the input tap number is dictated by the bandwidth accuracy required, while the number of hidden neurons has been chosen to perform the best fitting between the input-output data (H=10). The TDNN is trained with a backpropagation algorithm, based on the Levenberg-Marquardt approach for network parameter optimization because of its execution speed and good accuracy, with a mean square error (MSE) level, in the worst cases of approximately around $1e^{-4}$, and in the best ones, an excellent accuracy of $1e^{-12}$, as shown in figure 5.1. This good approximation accuracy is confirmed by the obtained results.

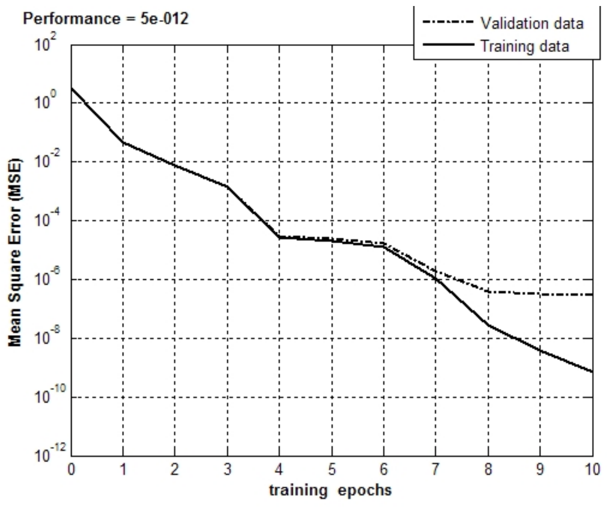


Figure 5.1. TDNN model performance.

Furthermore, the good generality property of a neural model, which says that it must perform well on a new dataset distinct from the one used for training, should be guarantee. That is why, to prevent the overfitting phenomena, the total amount of data available from measurements has been divided into training and validation subsets, and early-stopping technique has been applied. It determines that if there is a succession of training epoch in which performance improves only for the training data and not for the validation data, overfitting has occurred and the learning is terminated.

For all the cases, both neuronal and Volterra series based black-box model have been obtained. In all cases, the proposed behavioral models have been implemented and simulated inside a commercial circuits simulator, as was explained in chapter 4.

5.2 Transistor modeling

This section presents the results of comparison between the TDNN behavioral model - and its corresponding Volterra series model - and different measurements performed on a FET transistor.

In the first subsection, the DC behavior of a MESFET transistor at a certain bias point represented by the traditional Curtice formula is shown. This DC behavior will be modeled though a TDNN model and then its Volterra kernels extracted from the network.

The second subsection shows the results of modeling the dynamic behavior of a MESFET and a HEMT transistors in RF. First of all the importance of the inclusion of memory into the TDNN model will be highlighted. Then, a TDNN model capable of learning the behavior of a transistor able to work in different classes, according to the bias point, is reported.

5.2.1 DC behavior

W. Curtice proposed an equation - presented in chapter 2 - for modeling the current behavior of a FET transistor at a certain bias point.

This model generates the curves that can be seen in figure 5.2.1 when different intrinsic voltage combinations (or bias points) are considered for the device. In the figure, the following typical model parameter values have been used: $V_{gs} = [-2; 0]V$ step 0.2, $V_{ds} = [0; 5]V$ step 0.5, $A_0 = 0.0625$,$A_1 = 0.05$,$A_2 = 0.01$,$A_3 = 0.001$,$\beta = 0$, $\gamma = 0.3$. The bias point is $V_{ds} = 3V$.

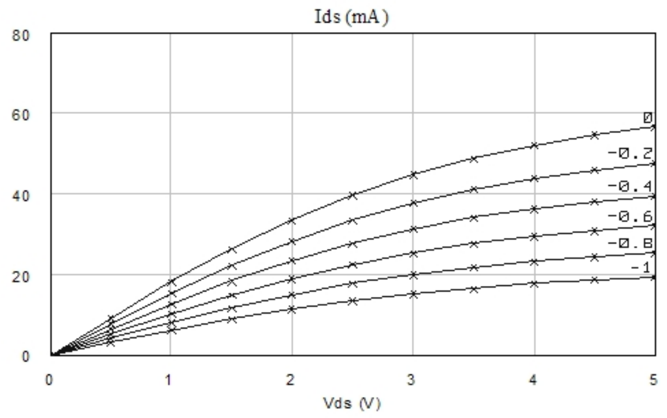


Figure 5.2. Typical I/V curves for a transistor in DC.

This device simulations have been used to train a simple TDNN model without memory, with only two input variables (V_{ds} and V_{gs}), 10 hidden neurons with hyperbolic tangent activation function and one output (I_{ds}). The network, shown in figure 5.3, has been trained with samples uniformly distributed; the training has been refined up to an average mean square error of $1e^{-10}$.

At the end of the training procedure, the Volterra kernels up to the 3^{rd} order have been extracted from the TDNN model and the Volterra series expansion corresponding to this device bias point has been built.

Figure 5.2.1 reports the Curtice model approximation of the drain current behavior in a FET transistor (dotted line) compared to the TDNN and Volterra models (full line). On the left, the figure shows the comparison between the TDNN model

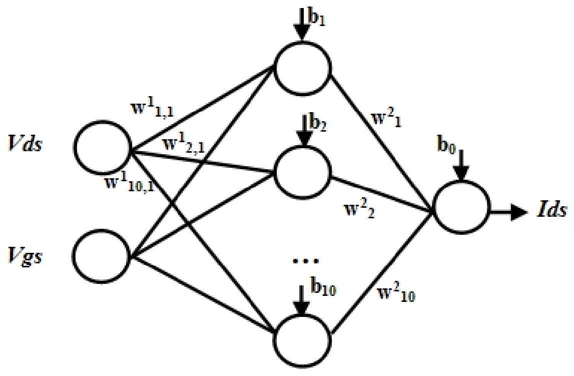


Figure 5.3. Neural network model for DC transistor modeling.

output and the original data of the problem. The approximation of the neural model is highly accurate. On the right, the figure compares the original data and the Volterra series approximation for this data. Considering that only up to the 3^{rd} order kernels have been included, the result can be qualified as very good. The same comparison is made in figure 5.2.1 concerning the current and the gate voltage.

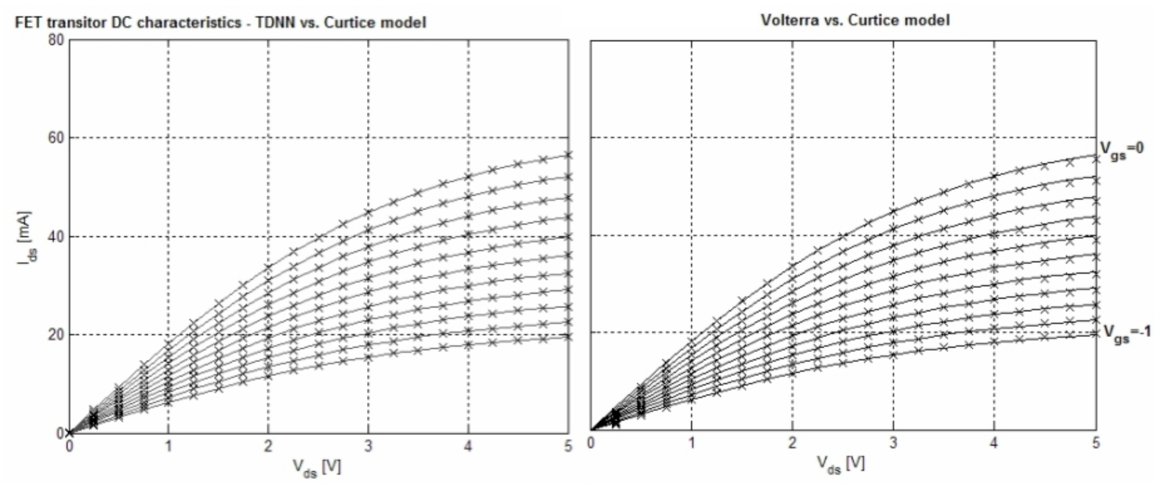


Figure 5.4. Comparison between the Curtice model (dotted line) and the TDNN model (left) and Volterra model (right) for modeling the DC characteristics of a FET transistor.

From the comparison it can be concluded that the TDNN and its corresponding Volterra model give almost the same results for the same inputs and therefore they could be used interchangeably; and that they can model the static characteristic of a FET transistor with very good accuracy.

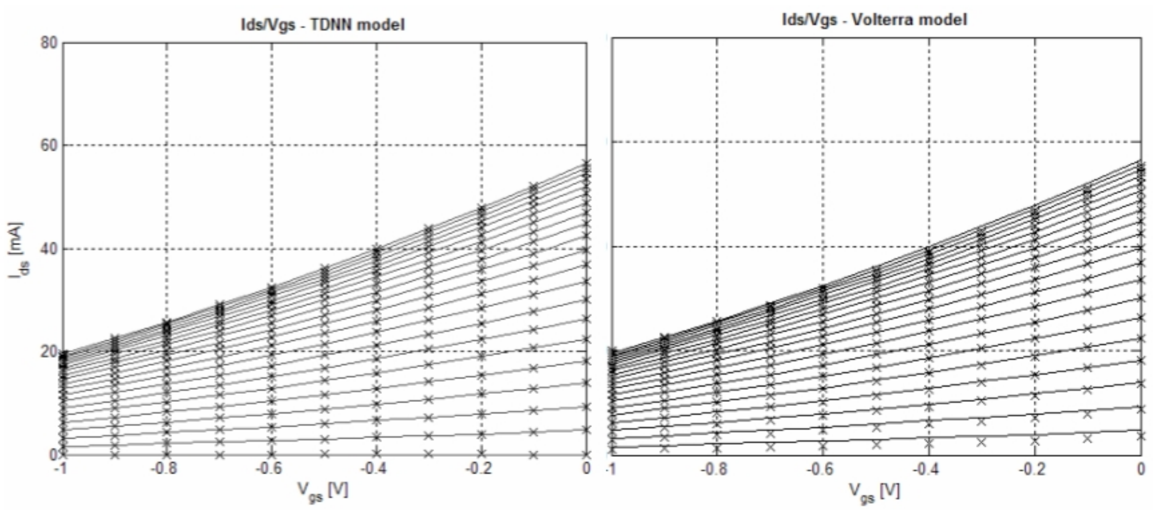


Figure 5.5. Comparison between the Curtice model (dotted line) and the TDNN model (left) and Volterra model (right) for modeling the I_{ds}/V_{gs} characteristics of a FET transistor.

5.2.2 RF behavior

Concerning the dynamic behavior of a transistor, time-domain measurements have been performed to obtain suitable training sets for a TDNN model taking into account the dynamic phenomena.

The first transistor which has been characterized was a medium power GaAs MESFET (10x100 mm) transistor. The device knee voltage is $V_k = 1.2$ V and its maximum drain current is $I_{max} = 180$ mA. A drain bias voltage of 5 V has been selected, while -2 V gate bias voltage has been chosen, corresponding to a drain current $I_{dc} = 70$ mA. Operating frequency (f_0) is 1 GHz and time-domain characterization is performed up to the fourth harmonic.

The load-source-pull test proposed in the doctoral thesis of [75] at Politecnico di Torino has been used for time-domain device characterization, and therefore, for generating the training and validation data sets necessary for neural model training.

The dedicated test-set for accurate device characterization used for measurements obtention is shown in figure 5.6. It provides static and pulsed DC characterization, scattering parameter measurements, real-time load/source-pull at fundamental and harmonic frequencies, and gate and drain time-domain RF waveforms. The measurements are carried out with a Microwave Transition Analyzer and a Vector Network Analyzer. A classical VNA calibration between port 1 and 2 (e.g. TRL, LRM) is completed with absolute power and harmonic phase calibration at port 3.

A time-domain investigation on the bias influence on the model has been performed, as shows figure 5.7, which reports drain current and voltage waveforms at

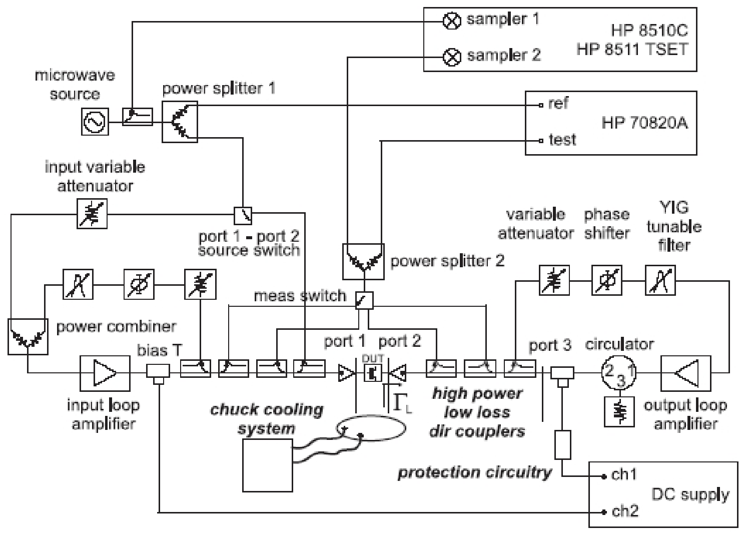


Figure 5.6. Simplified scheme of the load/source-pull system with time-domain waveform capabilities at Politecnico di Torino.

-1dB compression varying the gate voltage from -2 to -2.5 V.

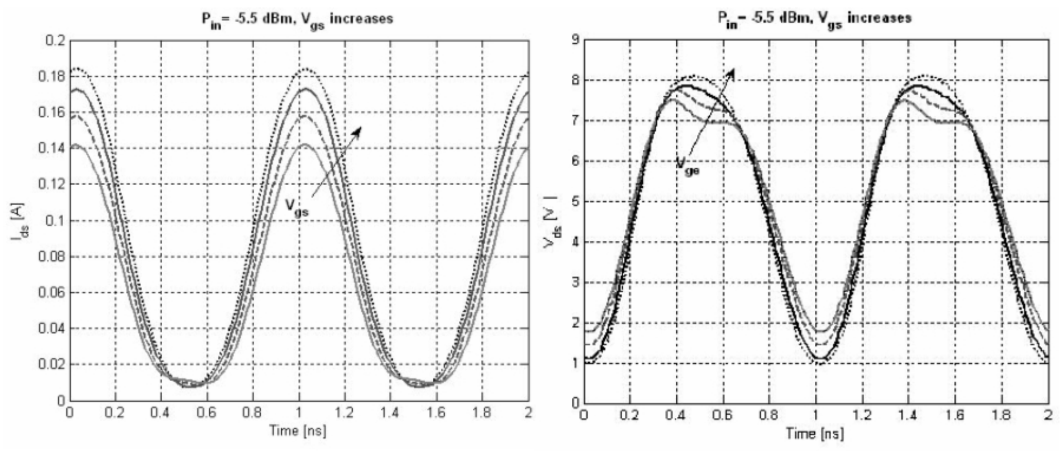


Figure 5.7. Drain current and voltage waveforms (-1dB compression) for V_{gs} = -1.75, -2, -2.25 and -2.5 V.

To model this behavior, first of all the static TDNN model used in the previous case for DC modeling has been proved. The results obtained concerning drain current modeling appear in figure 5.8 (no memory model - left). On the right part of this figure appear the results of using the model of figure 5.9, which includes memory M = 2. A clear improvement in the approximation can be noticed, in fact, the model with memory perfectly follows the behavior of the drain current.

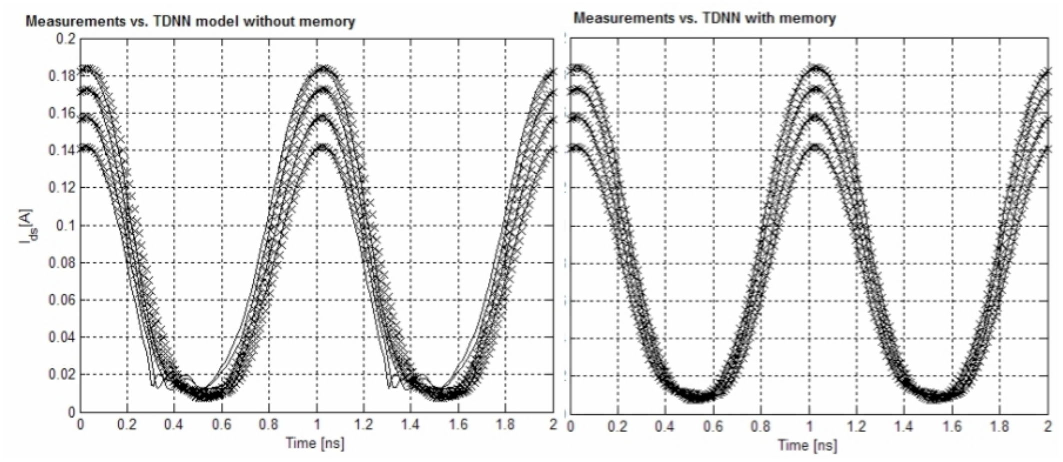


Figure 5.8. Neural model static (left) and dynamic (right) model approximations to drain current behavior.

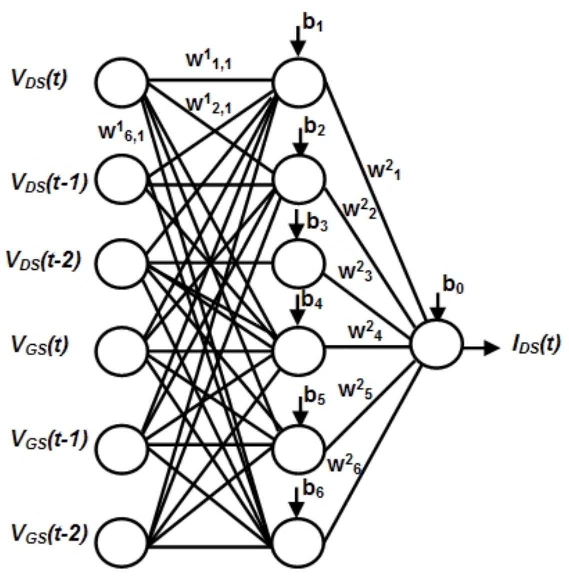


Figure 5.9. Neural network model considering memory.

Since the TDNN including memory is quoted as the best one, from this model the Volterra series representation has been built, which is shown in figure 6.9. A remark has to be done with respect to this figure. Compared to the TDNN output, the Volterra model has more error, but this is probably due the fact that the Volterra approximation is built including a limited number of kernels (in this case, up to the 3^{rd} order).

The TDNN dynamic behavioral model has been also applied to help PA design,

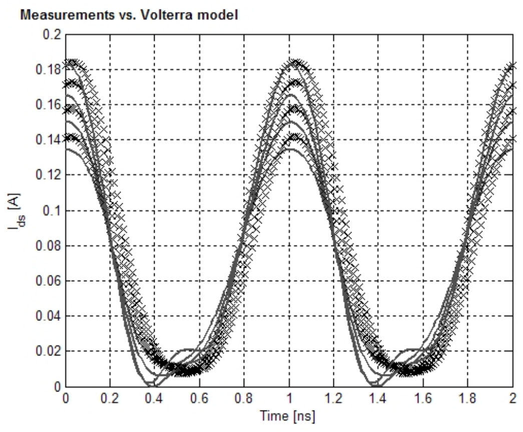


Figure 5.10. Measurements (dotted line) vs. Volterra model (full line) comparison for drain current modeling.

modeling an active device working in a wide range of classes. The model was capable of identifying the device response, through the training procedure, for a wide range of input power levels.

Complete characterization was performed for different classes of operation. Results are here reported for class A and B, at $V_{DS} = 30[V]$. Class A is biased at 50% I_{DSS}, and class B with $I_{DS} = 0$. A 1 mm total gate periphery GaN HEMT based on SiC with $I_{DSS} = 700[mA]$ has been measured at 1 GHz. Output loads at fundamental and harmonic frequencies are 50 Ohm. Only the first 4 harmonics are taken into account.

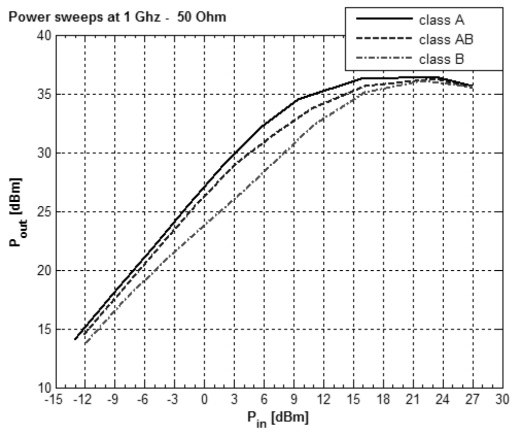


Figure 5.11. Power sweeps at 1GHz for class A, AB, and B bias points.

The power sweep ranges from -13 to +27 dBm, as shown in figure 5.11, that reports the device Pin/Pout at 1 GHz for class A, AB, and B. Samples have been collected on a 2 ns window. Figure **??** reports the drain current time-domain waveforms at increasing output power, the upper plots are relative to a class A bias condition, while the bottom ones refer to class B operations. Figure 5.12 reports the dynamic load lines (DLL) at 1 GHz for class A and B operation.

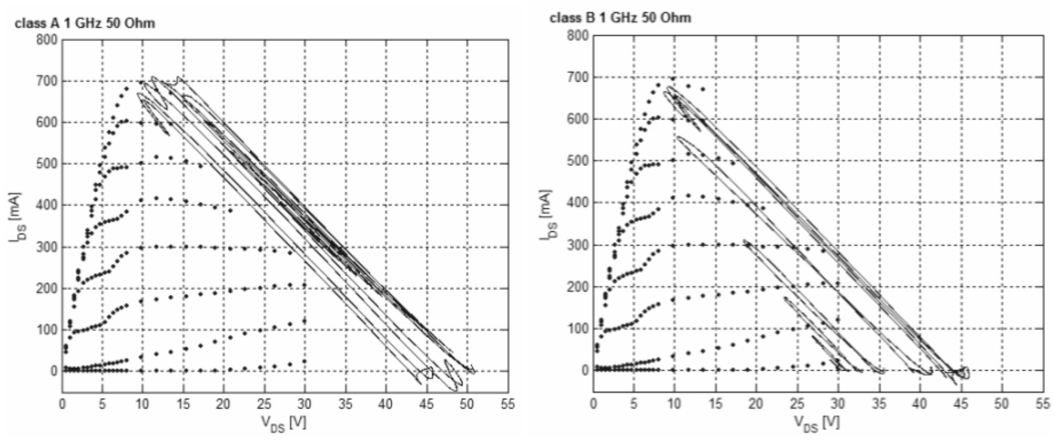


Figure 5.12. Dynamic load lines for 1GHz and 50 Ohm, in class A (left) and B (right) operation.

For this device, the dynamic TDNN model has been used (it was presented above in figure 5.9). The memory level is M = 2, meaning that two delay samples for each input variable are accounted for, and therefore, that a total of six inputs is obtained. The number of hidden neurons has been chosen H = 10, and the system outputs are the time samples of the drain current I_{DS}.

For training the TDNN model, input data vectors from all different input power levels have been first joined together, and the resulting vector has been copied and delayed as many times, to represent the network input, as necessary to account for memory. This is schematically depicted in figure 5.13. The TDNN model has been trained simultaneously for all the power levels chosen for training.

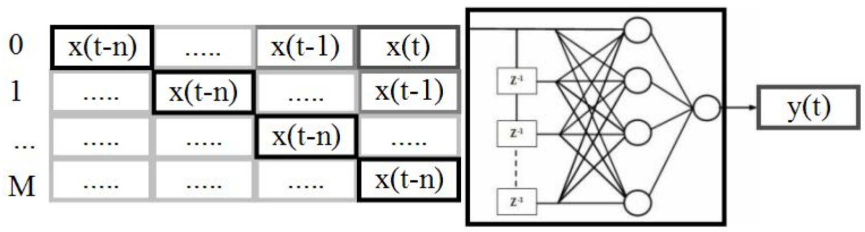


Figure 5.13. Training data for the TDNN model.

The simulation results reported in figure 5.14 show a rather good agreement between experimental and modelled data. The drain current time-domain waveforms are reported at increasing output power; the upper plots are relative to a class A bias condition, while the bottom ones refer to class B operations. The left figures report the measurements, while the right ones shows the TDNN simulation result.

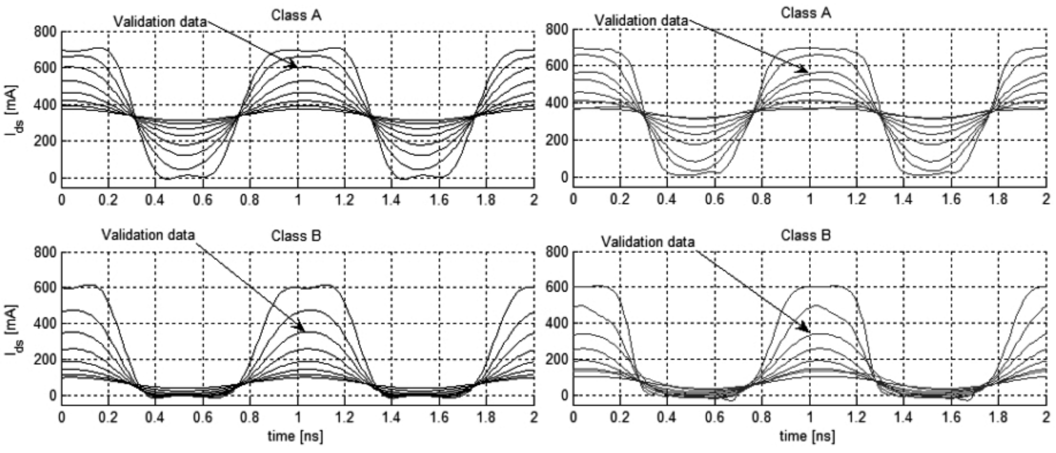


Figure 5.14. Time domain I_{DS} waveforms at 1 GHz at increasing power for class A (top), and B (bottom). Measurements (left), TDNN simulations (right). The pointed arrow waveform refers to the validation data subset.

The waveforms good agreement of the power levels used only for the model validation, and excluded from the training data, emphasized in the figure, is a key result to prove the model predictive capabilities.

5.3 Power amplifier modeling

This section presents the results of comparison between the TDNN behavioral model - and its corresponding Volterra series model - and one-tone and two-tone measurements performed on a power amplifier (PA).

The neural network used to model the amplifier has the architecture of figure 5.15. This network is trained with PA time-domain measurements. The input and output waveforms are expressed in terms of their samples in the time domain. M is the number of the input taps and represents the finite duration of the memory effect of the PA (in this case, M = 4). The tap delay Z^{-1} between successive samples must be a multiple or equal to the data sampling time T_s, and it is calculated as $T_s = M/F_s$, being F_s the data sampling frequency.

The measurements setup used at Università Tor Vergata di Roma is shown in figure 5.16. It has been used to generate the measurements for PA characterization. The PA is stimulated with tones generated by an HP83640A synthesized sweeper and an Anritsu MG3692 CW generator. Each power is swept from -30 to -8 dBm,

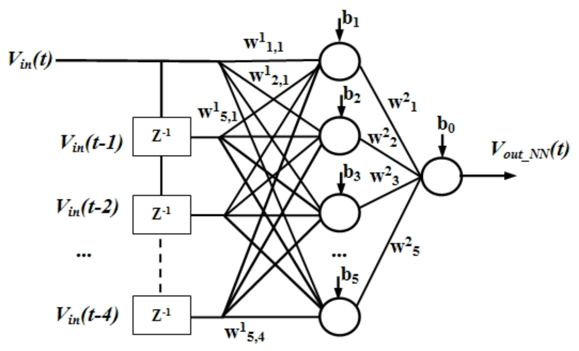


Figure 5.15. TDNN architecture for PA modeling.

that is 4 dB over the 1 dB compression point for the combined input power. The amplifier output has been connected to a Tek11801B Digital Sampling Oscilloscope (DSO), that has been triggered with the common 10 MHz RF reference from the generators; the equivalent time sampling algorithm for repetitive waveforms allows the instrument to exploit the full 50 GHz analog bandwidth of the testing probe. To compare the frequency response of the amplifier measurements and the simulation of the two behavioral models, an HP71000 Spectrum Analyzer (SA) has been connected to the amplifier output, and spectra have been measured for different input power levels.

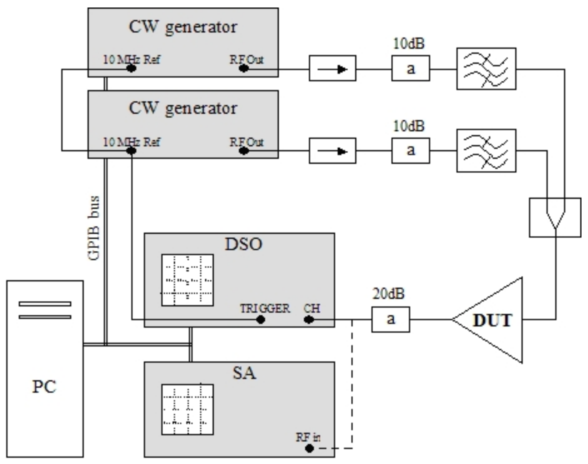


Figure 5.16. Measurement setup for PA characterization developed at Università Tor Vergata di Roma.

5.3.1 One-tone characterization

Measurements obtained from a Cernex 2266 PA, with 1-2 GHz bandwidth and 29 dB gain have been used to train a TDNN model of figure 5.15. The inputs to the model are the voltage signal V_{in} at the present instant, and four previous delayed samples, which makes a total of five inputs to the network. The input-output training data sets have been built with cascading samples obtained from eight source power levels, using all the data for training simultaneously.

Two type of networks have been trained: the first one with hyperbolic tangent activation function (equation 5.1), in the entire power range, with 3 dB power step, to perform a very large signal nonlinear model; the second one with third degree polynomial activation function (equation A.3), in a smaller power range, below 1 dB compression point. Both have been trained with five harmonics, eight input delays, and nine hidden neurons.

$$
\begin{aligned}
V_{out\_NNtanh} &= b_0 + \sum_{h=1}^{10} w_h^2 \tanh(b_h) + \\
&\left[\sum_{h=1}^{10} w_h^2 w_1^1 \left(1 - \tanh(b_h)^2\right) \right] V_{in}(t) + \\
&\left[\sum_{h_1=1}^{10} \sum_{h_2=1}^{10} w_{h_1}^2 (w_{h_1}^1)^3 \frac{\left(-1 + 4\tanh^2 - 3\tanh^4\right)}{3} \right] V_{in}(t)^3 + \cdots \qquad (5.1) \\
V_{out\_NNcubic} &= b_0 + \sum_{h=1}^{10} w_h^2 (b_h)^3 + \\
&\left[\sum_{h=1}^{10} w_h^2 w_1^1 3(b_h)^2 \right] V_{in}(t) + \\
&\left[\sum_{h_1=1}^{10} \sum_{h_2=1}^{10} w_{h_1}^2 (w_{h_1}^1)^3 \right] V_{in}(t)^3 + \cdots \qquad (5.2)
\end{aligned}
$$

Figure 5.17 reports simulation results after training the neural network with the original data (doted line) and building afterwards the Volterra series model (solid line) up to the 3^{rd} order approximation, from an neural network with hyperbolic tangent activation functions ($f(x) = \tanh(x)$) in the hidden layer (left part) and with cubic activation functions ($f(x) = x^3$) in the hidden layer (right part). More training results, obtained at 2 GHz, are shown in figures 5.18 and 5.19.

It has also been calculated the mean squared approximation error (MSAE) between the original device behavior V_{out} and each obtained Volterra series approximation V_{outV}, over the number of samples, using the formula of equation 5.3, being P the total number of samples (pairs input/output) used in the training set. The

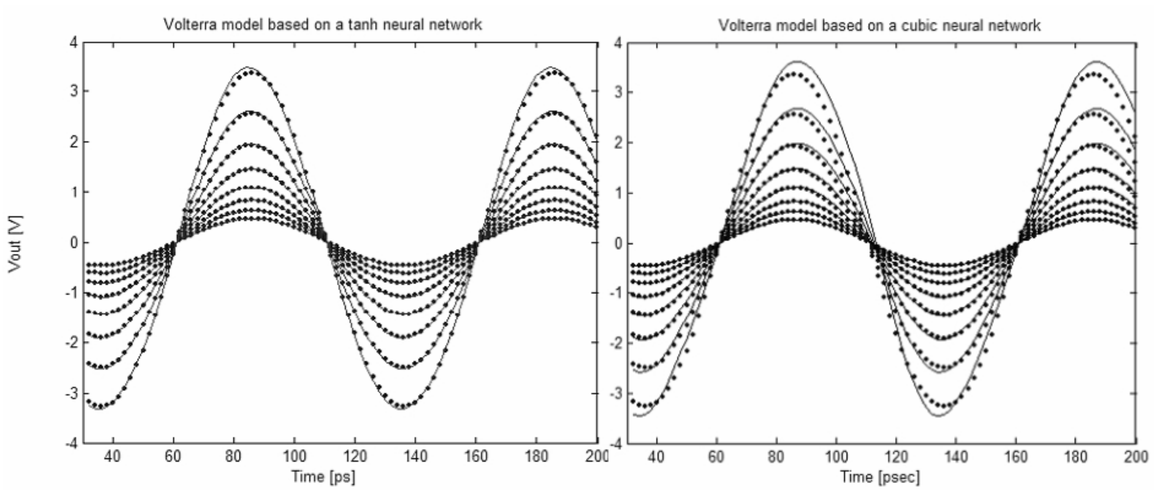


Figure 5.17. Comparison between measurements and Volterra series model.

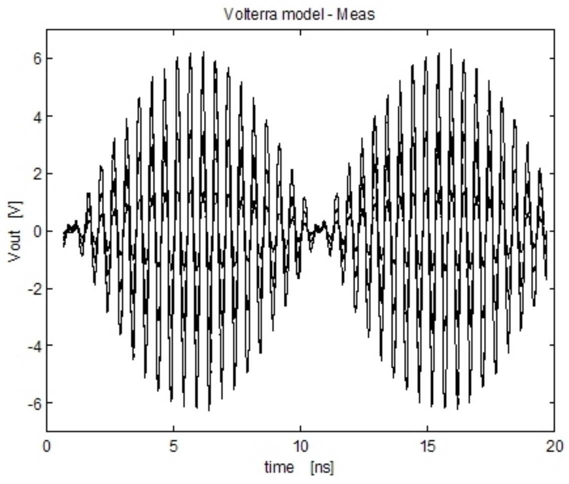


Figure 5.18. Volterra model simulation and amplifier measurement comparison for three input power levels (-20, -12, -7 dBm).

order of the MSAE error is around $1e^{-03}$, being the hyperbolic tangent the most accurate Volterra model. However, the cubic series can be more easily and quickly obtained.

$$MSAE = \frac{1}{P}\sum_{p=1}^{P}\left(V_{out}(p) - V_{out\_V}(p)\right)^2 \tag{5.3}$$

Finally and to conclude the analysis of the results, a natural question that could arise is why to build and use an approximation of the PA behavior (the Volterra model) that may have an error bigger than the TDNN model itself? The answer is

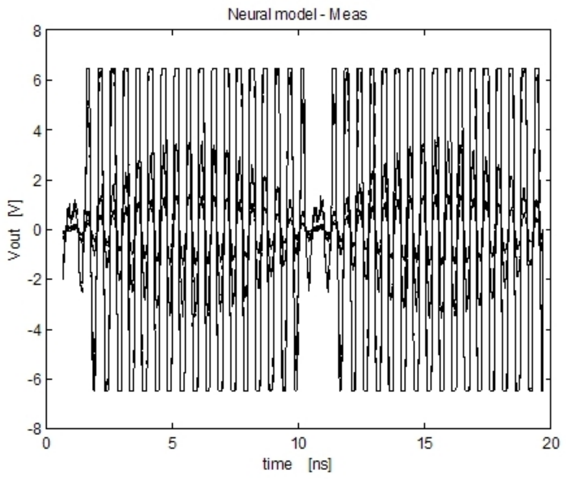


Figure 5.19. Hyperbolic tangent neural model simulation and amplifier measurement comparison for three input power levels (-20, -12, +1 dBm).

that, for simulation purposes, to be able to put the device model into a circuits simulator and to interconnect it with other devices models to make systems simulations, an analytical formula is most of the time required. The neural network is a valuable tool that saves time and complexity when building the analytical Volterra model, which is a particular suitable model traditionally used for nonlinear device behavior representation, and already available as a component (whose kernels, however, have to be assigned a value) inside most of today commercial simulators.

5.3.2 Two-tone characterization

In this case, measurements from a Cernex 2267 PA have been used, with a 2-6 GHz bandwidth, a 42 dB gain, and 1 dB compression at 32 dBm, stimulated with two tones at center frequency 4.1 GHz and frequency spacing 100 MHz, generated by an HP83640A synthesized sweeper and an Anritsu MG3692 CW generator, respectively.

Each power is swept from -30 to -8 dBm, that is 4 dB over the 1 dB compression point for the combined input power. The amplifier output has been connected to a Tek11801B Digital Sampling Oscilloscope (DSO), that has been triggered with the common 10 MHz RF reference from the generators; the equivalent time sampling algorithm for repetitive waveforms allows the instrument to exploit the full 50 GHz analog bandwidth of the testing probe. A characterization bandwidth of 22 GHz has been used to calculate the tap delay, in order to take into account fifth order harmonic distortion.

To compare the frequency response of the amplifier measurements and simulation of the two behavioral models, an HP71000 Spectrum Analyzer (SA) has been connected to the amplifier output, and spectra have been measured for different input power levels.

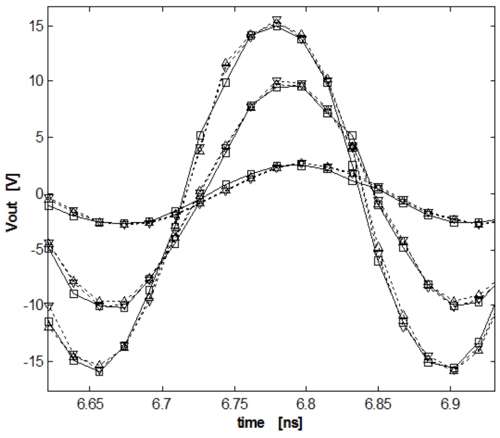


Figure 5.20. Time-domain measurements (solid line), TDNN (dotted ∇) and Volterra (dotted $\triangle$) model simulation comparison of amplifier voltage response, with two input tones at 4.05 and 4.15 GHz, for P_{in} = -25,-13,-5 dBm, respectively.

In this case the TDNN parameters have been setup at M=4 memory deep and H=5 hidden neurons. For larger memory duration the neural network tends to become unstable. Results are shown in figure 5.20. At the end of the training process, the measured and simulated time-domain waveforms of both models have been compared in figure 5.21 for three input levels.

To see the modeling performance in the frequency domain, a Fast Fourier Transform (FFT) has been applied to output waveforms either of TDNN and Volterra model simulation, and compared with spectra measurements made with an HP7000 Spectrum Analyzer. The results are presented in 5.22, where power amplitudes at fundamental, third-order and, only for the TDNN model, fifth-order intermodulation have been plotted against input power. Both models show a good behavior in the full power range. Concluding both models show well matched behavior both in the time and frequency-domain. In this case, better performance of the TDNN model is hardly discriminated.

To demonstrate the validity of the modeling approach for signals not used in the training phase, measurement data were obtained by stimulating the amplifier with a two-tone input signal at different frequencies, in particular 4.3 GHz center frequency, 20 MHz frequency spacing and the same power sweep.

Time-domain envelopes TDNN model simulations and measurements are shown in figure 5.23 for three input levels, whereas frequency-domain comparisons are presented in figure 5.24. As it can be seen, the third-order Volterra model is less

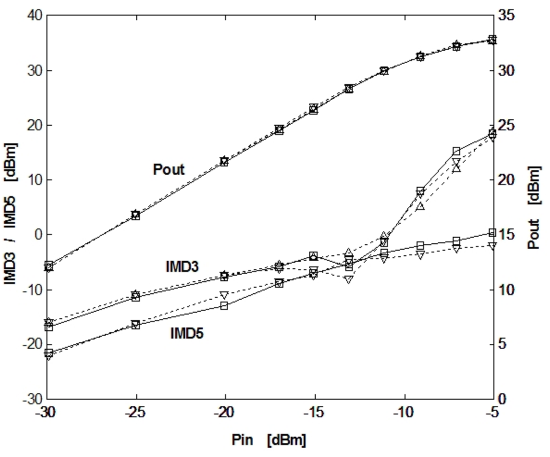


Figure 5.21. Comparison between measurements (solid line), TDNN (dotted ∇) and Volterra (dotted △) model simulations of amplifier response, with two input tones at 4.05 and 4.15 GHz, for P_{out} (@f_1=4.15 GHz), IMD3 (@$2f_2 - f_1$=4.25 GHz) and, only for the TDNN model, IMD5 (@$3f_2 - 2f_1$=4.35 GHz).

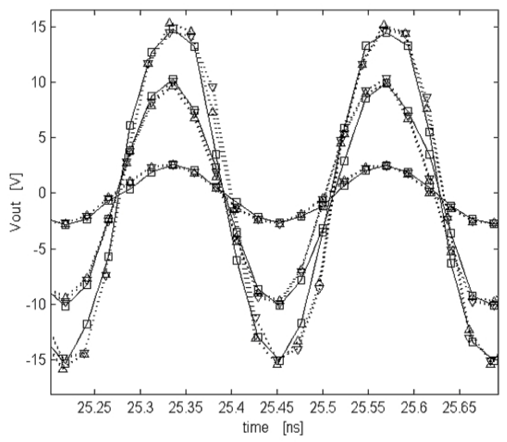


Figure 5.22. Time-domain measurements (solid line), TDNN (dotted ∇) and Volterra (dotted △) model simulation comparison of amplifier voltage response, with two input tones at 4.29 and 4.31 GHz, for P_{in} = -25,-13,-5 dBm, respectively.

accurate than the TDNN model around the 1 dB compression point, however still demonstrating good matching performance not only for low-to-medium distortion.

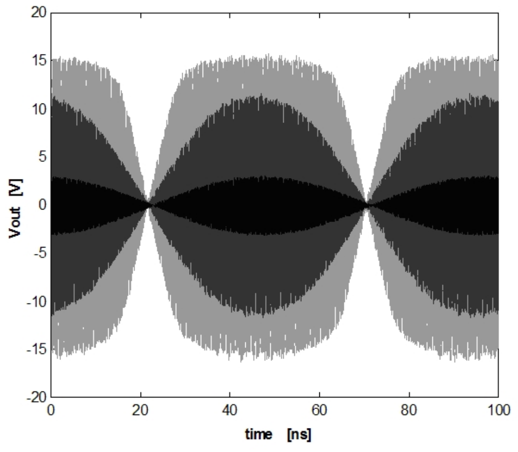


Figure 5.23. Time-domain envelopes of measurements, TDNN and Volterra model simulation of amplifier voltage response, with two input tones at 4.29 and 4.31 GHz, for P_{in} = -25,-13,-5 dBm, respectively.

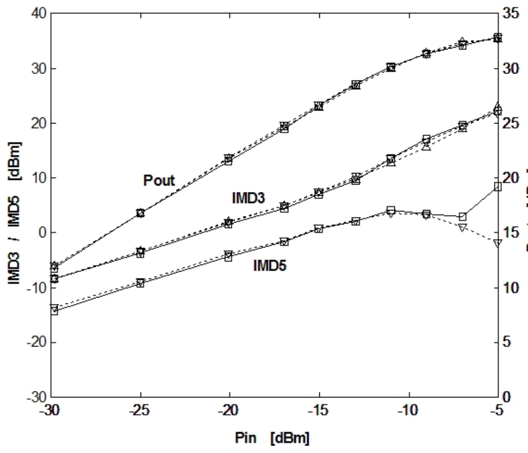


Figure 5.24. Comparison between measurements (solid line), TDNN (dotted ∇) and Volterra (dotted $\triangle$) model simulations of amplifier response, with two input tones at 4.29 and 4.31 GHz, for P_{out} (@f_1=4.31 GHz), IMD3 (@$2f_2 - f_1$=4.33 GHz) and, only for the TDNN model, IMD5 (@$3f_2 - 2f_1$=4.35 GHz).

5.4 Summary

This chapter showed the results of validation of the proposed model against measurements, for several different case studies. First of all, the proposed TDNN neural network model and corresponding Volterra series model were applied for the characterization of the static DC and dynamic RF behavior of FET transistors. Then, regarding power amplifiers dynamic behavior modeling, measurements with one-tone and two-tones inputs have been used to train the TDNN model for Pin/Pout waveforms characterization.

Chapter 6

Toolbox for behavioral model development

This chapter shows the software toolbox prototype that has been designed as result of this thesis to support the creation and use of behavioral models based on the neural network paradigm.

6.1 Introduction

Behavioral modeling based on neural networks (NNs) has been proved to be an important research area, of increasing interest for nonlinear devices and circuits modeling.

However, the process of neural model development is not trivial and involves many critical issues such as data generation, normalization, neural network topology definition, number of hidden neurons, learning rules, among others. As NN techniques are relatively new to the microwave community, it is often not easy for electronic engineers to make decisions regarding these issues.

Therefore, tools for automation of NN model development could be of great interest to RF/microwave engineers now, whose knowledge about neural network theory is rather limited.

There exist, actually, commercial and non-commercial products available on the market that allow for the creation and use of behavioral models, for example, for a communication system and components modeling. Nevertheless, most of them assume that the user is an expert in NN theory, because a rather deep understanding about how the NN paradigm works is needed to define a model. An electronic engineer who must design a neural model probably would not have a deep knowledge about the neural paradigm.

The mathematics software MatLab© has recently incorporated a Neural Network Toolbox for developing NN models, which can be applied to resolve a big variety of problems. The disadvantages of this toolbox, to support the task of electronic devices behavioral modeling, are many. First of all, a software licence must be bought. Once it is installed, to be able to exploit all its features it is necessary to know very well the toolbox specific commands. To be able to create really customized models, deep understanding on the toolbox is needed. Deep knowledge about neural models parameters, learning rules, etc., is required, which makes its use restrictive only to expert users. There is also a graphical interface for model creation and use, but it is poorly designed and not much user-friendly and could confuse the user.

Finally, another important point is that, if one has been able to create a neural model, it cannot be exported automatically to be used inside any available commercial circuits simulator. Therefore practically they cannot be used for real simulations, i.e. of a whole communications system.

There exist another commercial and non-commercial alternatives for NN models creations, such as:

- NeuralPlanner: http://www.tropheus.demon.co.uk/nplan.htm
- NeuroSolutions: http://www.nd.com/download.htm
- Lens Simulator: http://tedlab.mit.edu/ dr/Lens/
- PDP++: http://www.cnbc.cmu.edu/Resources/PDP++.html

among others, but all of them are too general purpose to be used in the electronics field, because they were designed for the generic creation of any kind of neural model, and they require an expert user as well.

Considering only tools designed specifically for behavioral or black-box modeling of electronic devices, *MπLog* can be mentioned (www.polito.it), which use however is specifically limited to analog-digital drivers modeling. Instead, NeuroModeler© (http://web.doe.carleton.ca/qjz.html) has been designed as a free tool, for the creation and use of behavioral models of electronic devices or systems, of any kind, which once defined can be afterwards exported into the circuits simulator Agilent ADS©. The good points of this toolbox is that it is specifically electronic-oriented and it allows the creation of behavioral models for a big variety of different problems, which can be saved and modified and loaded into a simulator. This is a very powerful tool which allows creation and use of many types of networks and it has implemented most of the neural training algorithms. Besides, it can generate a black-box model specific for the mentioned commercial simulator.

In this context, appears the need for a software toolbox easy to use for a RF/microwave engineer which must create and manage behavioral models based on NNs, but has not knowledge at all of the neural theory. Such a tool should have a good interface design really thought for en electronic engineer who must design a neural model for a device or system and who probably would not have deep knowledge about the neural paradigm. It should allow also be easily to implement inside a circuits simulator and efficient to execute.

That is why this thesis proposes a prototype of a software toolbox, which tries to fulfill these requirements. The prototype has been implemented as a MatLab© interface to initially improve the access to the Neural Network Toolbox© algorithms and commands. The continuation of this work should be the implementation of the prototype in a free programming language.

6.2 Toolbox features

The proposed toolbox, named `NeuroPOLI`, is a prototype of a RF/microwave oriented software tool, to help an electronic engineer to quickly develop neural models for communications systems components; at both device and circuit-levels; and for linear or nonlinear simulations. The software is written for typical electronic designers who are not neural network experts but would like to immediately get started with this new technology. The toolbox prototype has been implemented using the MatLab© GUI (Graphical User Interface).

Some of `NeuroPOLI` main features are:

- Model creation and editing: the toolbox allows for easily create a neural network model. The built-in default automatically defines a model structure and number of neurons, based on the file containing the input/output data or measurements. The user can choose from some templates to define customer neural models.

- Training: the toolbox automatically trains the neural model with the data, choosing the proper learning algorithm by default. In a future version of the prototype, the program will have a feature which, intelligently, will adjust the neural model size and training methods to achieve the required model accuracy. User can modify any training defaults.

- Learning methods: backpropagation methods of second order are supported (Levenberg-Marquardt algorithm).

- Activation functions: the functions that can be used as activation for the neurons are: sigmoid, hyperbolic tangent, linear, cubic. The supported neural models types are multilayer perceptron (MLP) and time-delayed neural models (TDNN).

- Test: the performance of neural model can be verified using an independent set of data. User can also evaluate interpolation and extrapolation capabilities of the model.

- Plot: the results of the training and simulation steps can be seen graphically.

- Export: the trained neural model can be exported to the own user-environment as a full-back-box model, be it a spreadsheet or a simulator.

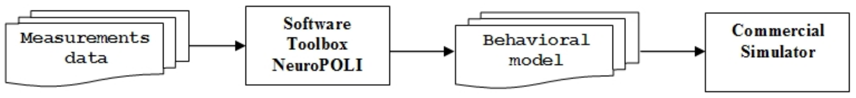


Figure 6.1. Schematics representation of `NeuroPOLI` functioning.

A block representation of the functioning of the proposed toolbox is shown in figure 6.1. A neural network model developed by the toolbox based on real device/circuit measurements/simulations can be exported as a behavioral model, to be used inside a commercial simulator for circuit simulation, design and optimization.

Figure 6.2 shows a flowchart with the model development process inside the proposed toolbox. The measurements data files generated from laboratory measurements are directly loaded into the toolbox. They have to follow a specific format, which allows for the automatic detection of number of input/output variables. Therefore, from the detection of the variables from the measurements, the NN model is automatically created. The designer can change the original networks design proposed by the toolbox and set some model parameters, such as number of hidden neurons and activation functions.

Once the model has been defined, it can be trained and/or simulated with the measurements, up to a user-defined accuracy (mean square error of the NN). If after training the desired accuracy has been reached, the model can be save (exported) as a text file which will include the NN model and its parameters values, and it can also be saved as a Volterra model extracted from the NN model. The toolbox includes an algorithm which automatically implements the formulas that have been presented in chapter 4 for the Volterra kernels extraction and Volterra series model generation.

The NN/Volterra models are saved into files, which afterwards are loaded as a device or circuit behavioral model inside a commercial circuit simulator, i.e. using a netlist file suitable to implement the model in a symbolically-defined-device (SDD), as was shown in the previous chapter.

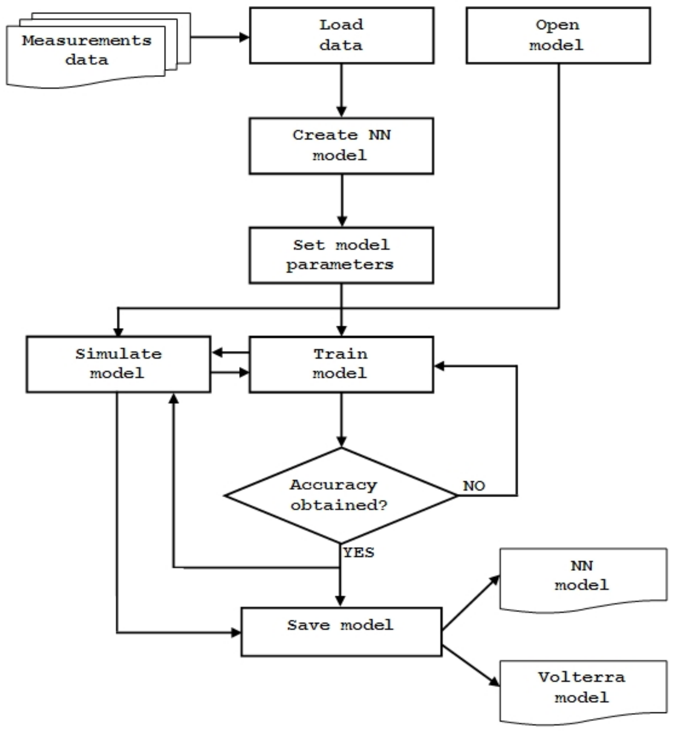


Figure 6.2. Flowchart of the proposed automatic behavioral model development toolbox.

6.3 Graphical User Interface

The graphical user interface provides support for model creation from input/output simulation or measurements data, as well as the visualization and control of the simulation itself. Snapshots of its windows are described in the subsequent paragraphs.

One interface of the prototype is presented in figure 6.3. The interface has been programmed taking into account the classical windows style, grouping into a menu the options which have to be available to most of the interfaces (windows), and activating/deactivating the menu options accordingly.

Figure 6.4 shows the main control screen of `NeuroPOLI`, which allows the creation and opening of behavioral neural network-based models. The menu options are displayed to show the toolbox features.

6.3.1 Model creation

Figure 6.5 and 6.6 are snapshots of the new and open interfaces, respectively. For the creation of a new model, as the model is created from measurements, the files containing them have to be localized (the input data and the corresponding output

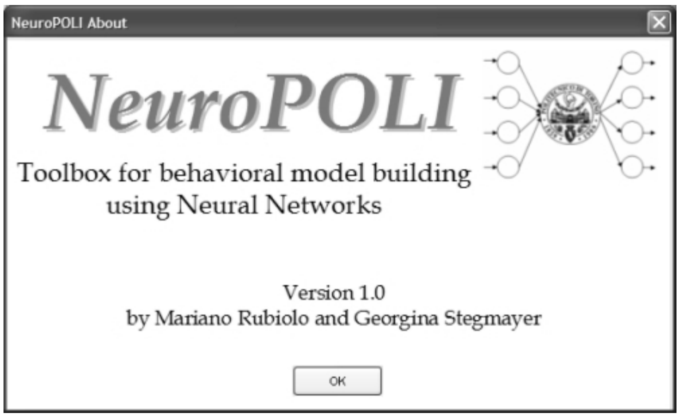

NeuroPOLI About
NeuroPOLI
Toolbox for behavioral model building using Neural Networks
Version 1.0
by Mariano Rubiolo and Georgina Stegmayer
OK

Figure 6.3. NeuroPOLI prototype: about interface.

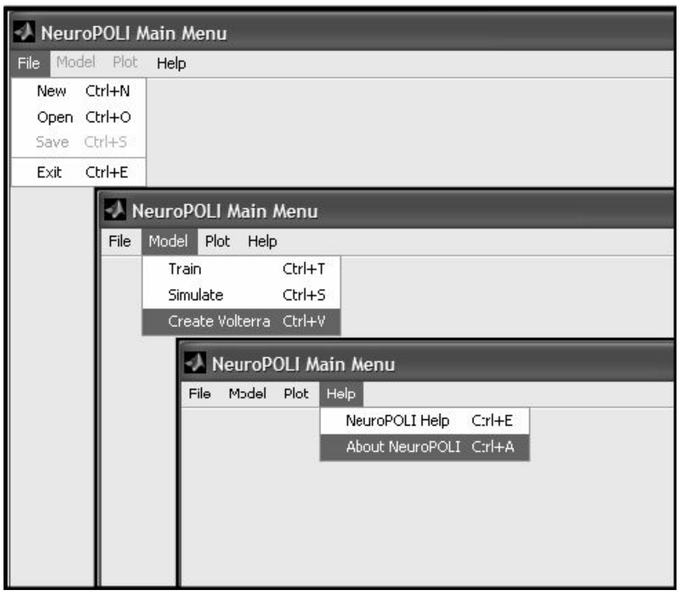


Figure 6.4. NeuroPOLI prototype: main screen and menu options.

data). Once these files have been loaded, the input/output variables are automatically detected and a neural model topology (number of hidden neurons, type of network model, activation function) is suggested, which the user can change. The neural model parameters (weights, bias, etc.) are automatically initialized as well. Once the model has been created, it can be saved into a file.

An existing model can be loaded into memory to work with it. The neural model and its associated information will be automatically detected (such as the model approximation error).

Once the model has been created, it can be trained to adjust and learn the

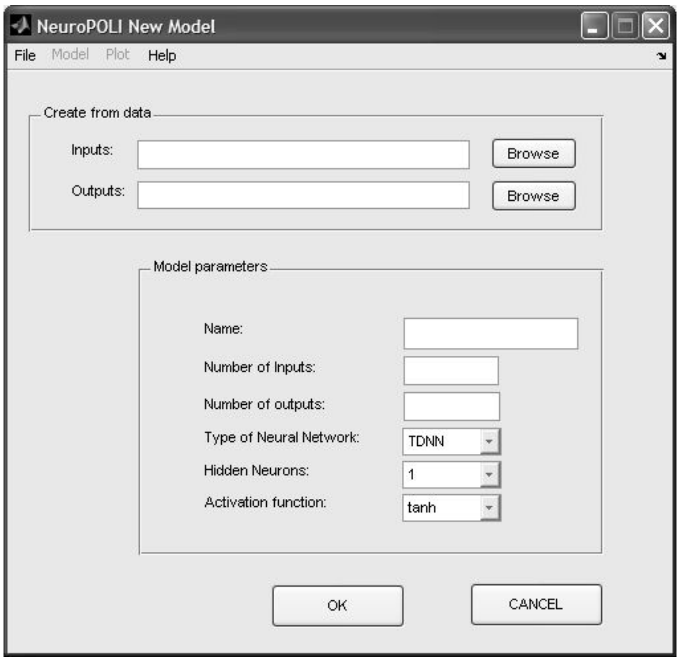


Figure 6.5. NeuroPOLI prototype: new model interface.

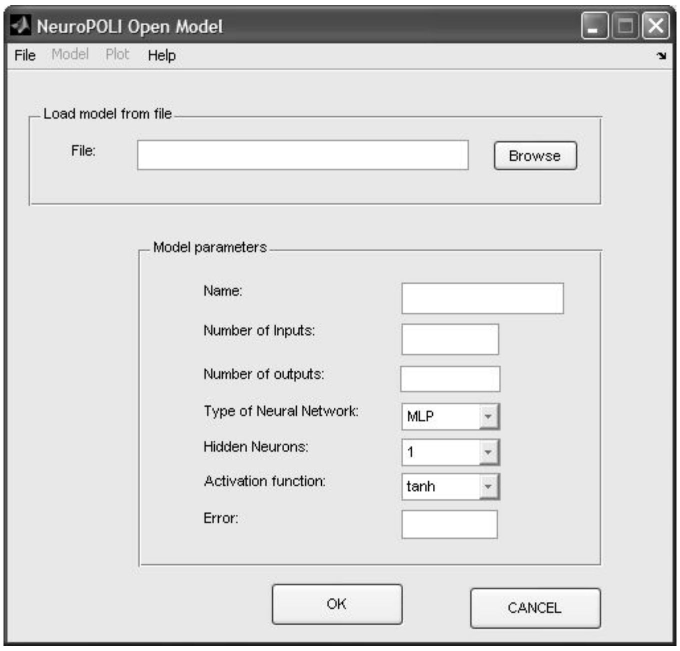
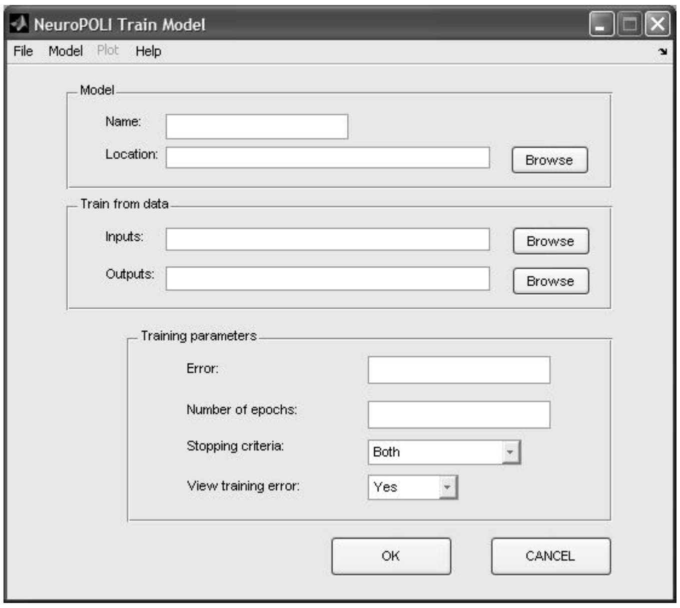


Figure 6.6. NeuroPOLI prototype: open model interface.

measurements data, using the train and/or simulate menu options.

6.3.2 Model training and simulation

The train and simulate windows of the prototype appear on figure 6.7 and 6.8, respectively. The train interface allows to set the training algorithm parameters such as the stopping criteria (number of epochs or error level), and hiding or not the performance plot.



Stopping criteria: Both

View training error: Yes

Figure 6.7. NeuroPOLI prototype: train model interface.

The simulate interface allows to simulate a model, that it to say to generate the model response to certain input data, and the network response can be compared to the measurements data using the plot menu option.

6.3.3 Volterra model generation

The Volterra series model associated to a neural model can be automatically generated by choosing the corresponding menu option. By default the Volterra series model is created into a `.txt` file and a model windows alerts the correct model creation. This file afterwards can be loaded as a black-box model into a commercial circuits simulator.

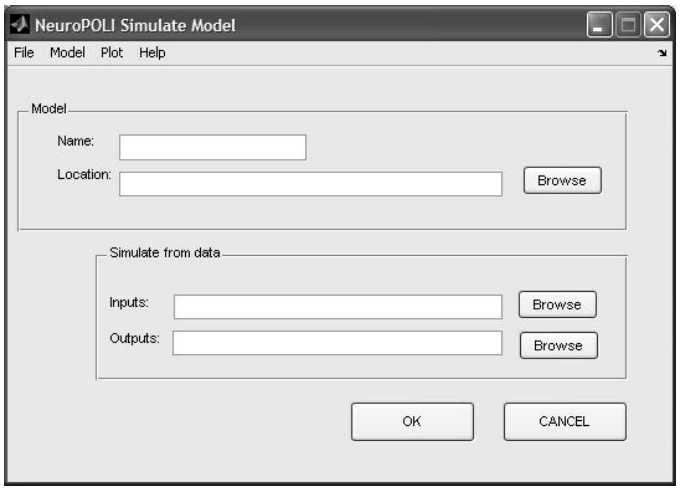


Figure 6.8. NeuroPOLI prototype: simulate model interface.

Volterra model created!

Figure 6.9. NeuroPOLI prototype: Volterra model generation interface.

6.3.4 Plotting

The plot menu option plots all the available data about the model and the measurements. The variable names are automatically extracted from the measurements data and correctly showed in the plots, even when some input variables may have associated delays.

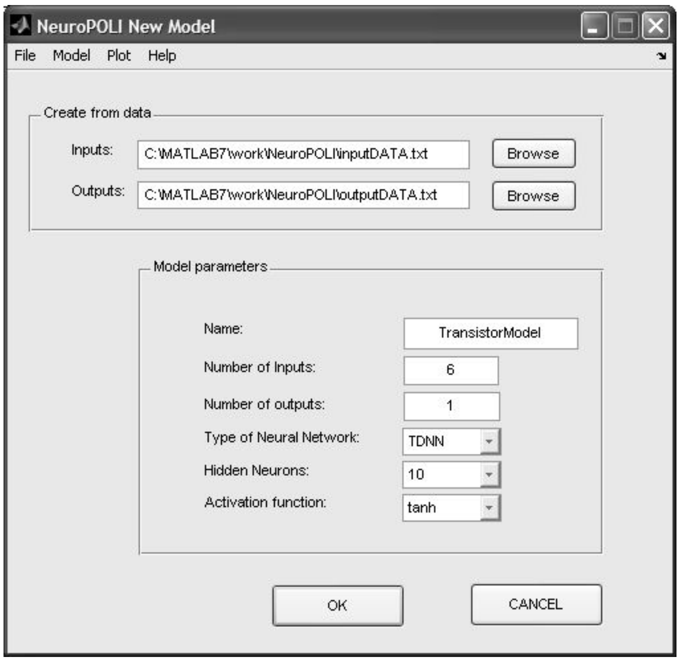
6.4 Application examples

In this section two examples ar presented. The first example demonstrates the use of the software toolbox use for creation of a transistor behavioral model. The second example shows a power amplifier model and simulation results.

6.4.1 Transistor modeling and simulation

In this example, input/output time-domain measurements of the parameters V_{ds}, V_{gs} and I_{ds} are used to create a neural network model.

From the main menu, the option new model is selected, which opens the windows shown in figure 6.10, where the files containing the measurements are selected. The files are `.txt` files which follow a certain template. The variables (and their delays if they are part of the model) must be defined with their corresponding names inside the test file. An automatic procedure extracts the name and the number of variables. The user can define the type of network that will be created. In this example, a TDNN model will be created.


NeuroPOLI New Model

Create from data

Inputs: C:\MATLAB7\work\NeuroPOLI\inputDATA.txt

Outputs: C:\MATLAB7\work\NeuroPOLI\outputDATA.txt

Model parameters

Name: TransistorModel

Number of Inputs: 6

Number of outputs: 1

Type of Neural Network: TDNN

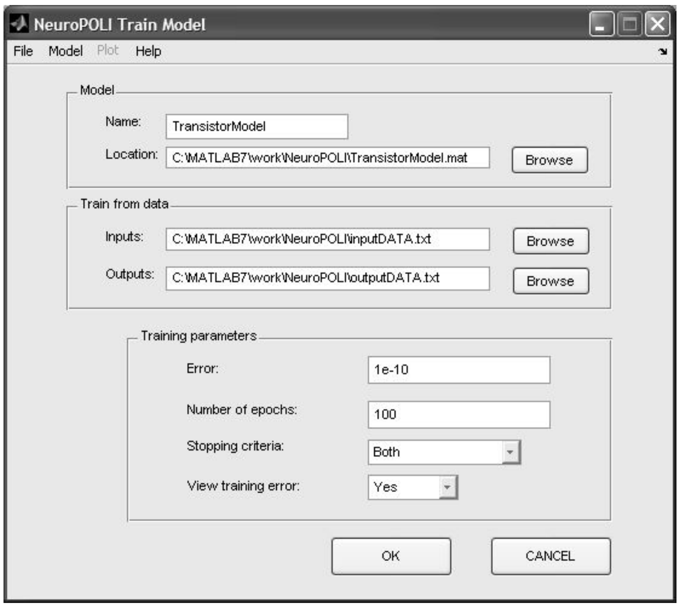
Hidden Neurons: 10

Activation function: tanh

Figure 6.10. NeuroPOLI prototype: new transistor model definition.

In the shown example, the number of inputs to the neural model is 6 and the output is 1 variable. Actually the input variables are only two (V_{ds} and V_{gs}) with two delays each, which makes a total of 6 inputs. The model is given a name (in the example, transistor) and it is saved.

Now the model and plot options are activated. The model can be trained with the measurements, as shows figure 6.11, and the results can be plot. If the plot option of the menu is chosen, the result is figure 6.12.



NeuroPOLI Train Model

File Model Plot Help

Model

Name: TransistorModel

Location: C:\MATLAB7\work\NeuroPOLI\TransistorModel.mat

Train from data

Inputs: C:\MATLAB7\work\NeuroPOLI\inputDATA.txt

Outputs: C:\MATLAB7\work\NeuroPOLI\outputDATA.txt

Training parameters

Error: 1e-10

Number of epochs: 100

Stopping criteria: Both

View training error: Yes

Figure 6.11. NeuroPOLI prototype: transistor model training.

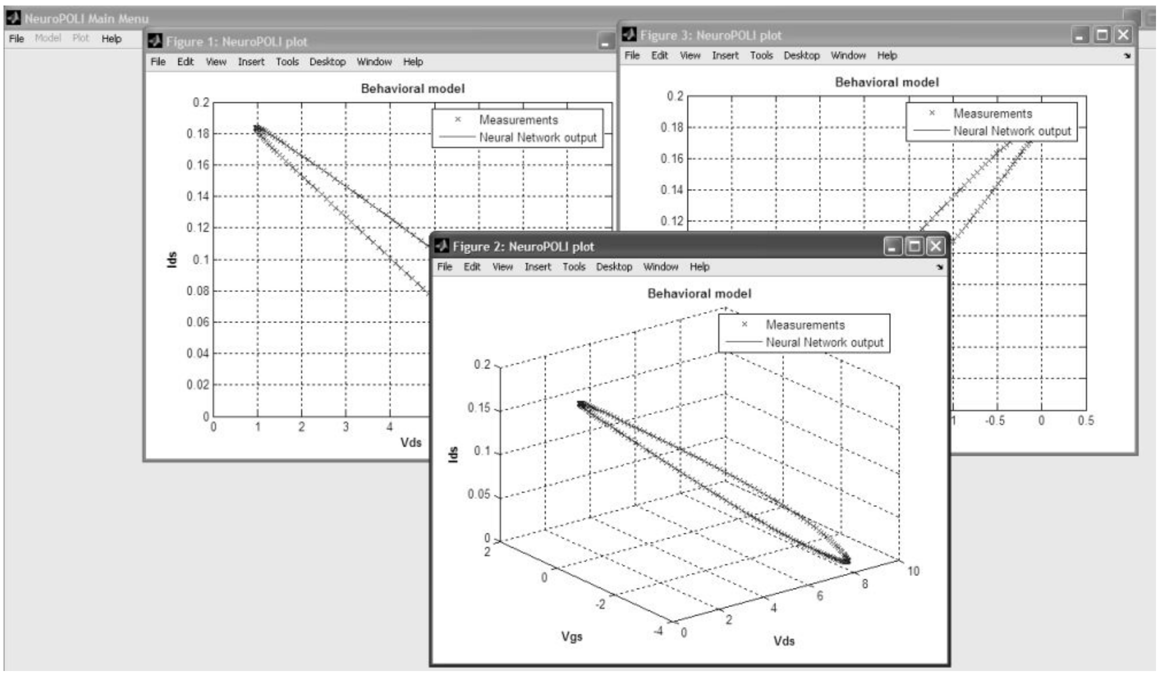
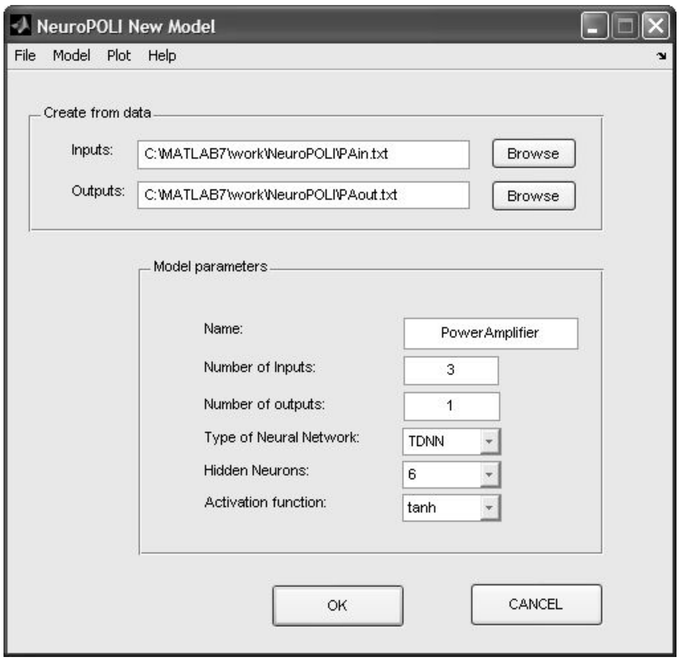


Figure 6.12. NeuroPOLI prototype: plots of the transistor model.

6.4.2 Power Amplifier modeling and simulation

In this example, Pin/Pout time-domain measurements of a power amplifier are used to create a neural network model. In this case the parameters chosen in the new model window appear in figure 6.13.


NeuroPOLI New Model

Create from data
Inputs: C:\MATLAB7\work\NeuroPOLI\PAin.txt
Outputs: C:\MATLAB7\work\NeuroPOLI\PAout.txt

Model parameters
Name: PowerAmplifier
Number of Inputs: 3
Number of outputs: 1
Type of Neural Network: TDNN
Hidden Neurons: 6
Activation function: tanh

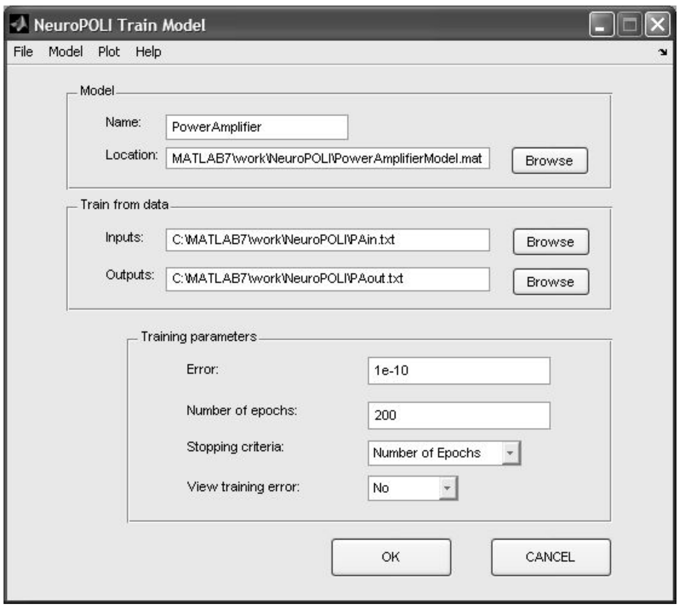
Figure 6.13. NeuroPOLI prototype: new power amplifier model definition.

In this case the model has only one input variable and two delayed samples, therefore the toolbox recognizes that the neural model should have 3 inputs.

This model is trained (figure 6.14) and then the results are plotted (figure 6.15).

6.5 Summary

This chapter explained the software toolbox prototype that has been designed as result of this thesis to support the creation and use of behavioral models based on the neural network paradigm. At the end of the chapter, examples of applications of the toolbox to electronic devices modeling have been shown.



Model

Name: PowerAmplifier

Location: MATLAB7\work\NeuroPOLI\PowerAmplifierModel.mat

Train from data

Inputs: C:\MATLAB7\work\NeuroPOLI\PAin.txt

Outputs: C:\MATLAB7\work\NeuroPOLI\PAout.txt

Training parameters

Error: 1e-10

Number of epochs: 200

Stopping criteria: Number of Epochs

View training error: No

Figure 6.14. NeuroPOLI prototype: power amplifier model training.

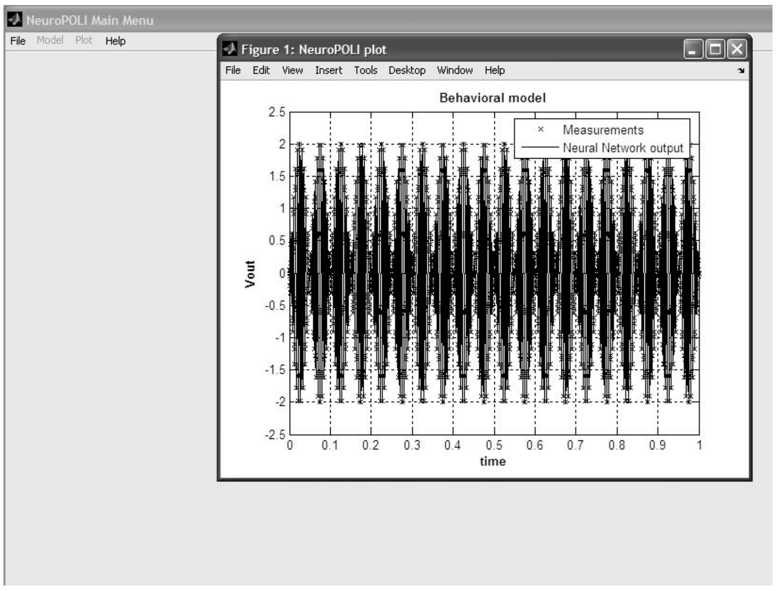


Figure 6.15. NeuroPOLI prototype: power amplifier model plot.

Chapter 7

Conclusions

This chapter presents the conclusions of this work.

7.1 Introduction

This thesis has been motivated by the increasing importance of Radio-Frequency (RF) and Microwave Engineering, as a consequence of the the exceptional world-wide growth in digital mobile communication systems and wireless applications.

Modern communications systems composed by RF/Microwave electronic devices or sub-systems generally have nonlinear behavior, rich in high-frequency dynamics. For analysis and design purposes, they have to be replaced by realistic models, which must particularly take into account this dinamicity and nonlinearity, which influence the whole systems performance. The wireless communications market is very important commercially and therefore there is a big interest in finding good suitable models to use in the RF system design process.

In recent years the measurement-based modeling technique has been identified as a particularly critical issue for simulation time reduction without loosing accuracy. Behavioral models have become the object of extensive research during the last few years as a black-box, technology independent tool to model off-the-shelf devices or sub-systems starting from conventional or ad-hoc measurements. In particular, a new technique that has received increasing attention for the development of behavioral models is the Neural Network approach.

This PhD thesis proposed a new behavioral model, which could be applied for the modeling of electronic components of a communication system (devices or sub-systems), reproducing accurately their nonlinear and dynamic behavior using a neural network-based approach. Measurements have demonstrated the validity of the modeling approach, both for static device characterization in DC transistors and strong nonlinearities and memory effect characterization in power amplifiers.

Moreover, an efficient procedure to extract Volterra kernels in the time-domain from the parameters of the neural network has been proposed, thus providing a simple way to construct very compact and accurate Volterra models. Even if neural models are often more accurate to characterize hard nonlinearities, Volterra-series behavioral models provide open information about the nonlinear behavior, and their implementation in circuit simulators is generally less time-consuming.

Due to need of an adequate support to the process of behavioral model generation and use, this thesis also proposed a prototype of a software toolbox for the creation and use of neural models and implementation inside a commercial RF CAD circuits simulator.

7.2 Main contributions

The main contributions of this thesis to the RF/Microwave field can be summarized as follows:

New behavioral model: A new compact and highly accurate behavioral model is proposed, to be applied for the modeling of devices or sub-systems of a communication system, reproducing their nonlinear behavior using a neural network-based approach.

Volterra series model generation: Automatically and straight-forward procedure for Volterra model generation and kernels extraction. The procedure can be applied also in the case of a nonlinearity that depends on more than one variable. It allows obtaining a full black-box model, independently of the physical circuit under study.

Toolbox for behavioral model development: A prototype of a software toolbox specially deigned for electronics engineers, which would like to use behavioral models and are not familiar with the neural networks theory.

Behavioral model implementation: A template for efficient implementation of the behavioral model inside a commercial circuits simulation has been provided as well, allowing fast simulation.

Appendix A

Volterra identification procedure

In this chapter, the procedure that allows obtaining the Volterra kernels from the parameters of a neural network model, procedure which has been first addressed in [89], is explained in detail. The second part of the appendix show the codification of the obtained formulas.

A.1 Algorithm

In the following lines, the procedure that allows calculatin the Volterra kernels from the parameters of a neural network model, will be shown. To make the explanation more clear, a simple example will be used.

Let us suppose a neural model which has to reflect a one input (x)/one output(y) dependence. The neural model reflects also a dynamical dependence on the input (past values). For example, it may take into account not only the actual value (at time t) of the input variable but also a delayed sample (at time t-1), as shows figure A.1. This network is a time-delayed neural network (TDNN) which receives two input values and produces one output. The hidden neurons activation function has been chosen to be the hyperbolic tangent. Instead, the output neuron activation function in linear. All the neurons have bias values, which allows having more degrees of freedom when making the model learn a nonlinear behavior.

Using the convention for naming the TDNN parameters (weights and bias) which has been presented in chapter 4, the output of the model should be equation A.1. If this equation is specialized for the TDNN model under consideration, with N=1 and H=2, the result is equation A.2.

$$y(t) = b_0 + \sum_{h=1}^{H} w_h^2 \tanh\left(b_h + \sum_{i=0}^{N} w_{h,i+1}^1 x(t-i)\right) \tag{A.1}$$

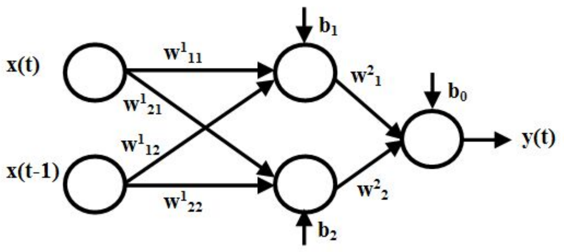


Figure A.1. Example of a simple TDNN model.

$$\begin{aligned} y(t) &= b_0 + \\ & w_1^2 \tanh\Big(b_1 + w_{1,1}^1 x(t) + w_{1,2}^1 x(t-1)\Big) + \\ & w_2^2 \tanh\Big(b_2 + w_{2,1}^1 x(t) + w_{2,2}^1 x(t-1)\Big). \end{aligned} \tag{A.2}$$

In [90] it has been proved that if a neuron activation function is considered in terms of its Taylor series equivalent (e.g. a polynomial), the neuron bias acts to change the output function. Therefore, if the hidden neurons in a network have different bias values, each hidden neuron can be considered as a different polynomial output function, $p_h(x)$, which can be written as equation A.3.

$$p_h(x) = a_{0h} + a_{1h}x + a_{2h}x^2 + a_{3h}x^3 + \cdots + a_{nh}x^n. \tag{A.3}$$

The TDNN model we are analyzing has two hidden neurons. Their corresponding output polynomials would be the following:

$$p_1(x) = a_{01} + a_{11}x + a_{21}x^2 + a_{31}x^3 + \cdots \tag{A.4}$$
$$p_2(x) = a_{02} + a_{12}x + a_{22}x^2 + a_{32}x^3 + \cdots \tag{A.5}$$

where $x = \sum_{i=0}^{N} w_{h,i+1}^1 x(t-i)$. Combining equation A.2 and A.4, replacing $\tanh(b_h + x)$ by $p_h(x)$, results in equation A.6.

$$\begin{aligned} y(t) &= b_0 + \\ & w_1^2 \Big[a_{01} + a_{11}\Big(w_{1,1}^1 x(t) + w_{1,2}^1 x(t-1)\Big) + a_{21}\Big(w_{1,1}^1 x(t) + w_{1,2}^1 x(t-1)\Big)^2 + \cdots\Big] + \\ & w_2^2 \Big[a_{02} + a_{12}\Big(w_{2,1}^1 x(t) + w_{2,2}^1 x(t-1)\Big) + a_{22}\Big(w_{2,1}^1 x(t) + w_{2,2}^1 x(t-1)\Big)^2 + \cdots\Big] \end{aligned} \tag{A.6}$$

Distributing and gathering like terms, we get equation A.7.

$$\begin{aligned} y(t) &= b_0 + w_1^2 a_{01} + w_2^2 a_{02} + \\ & w_1^2 a_{11}\left[w_{1,1}^1 x(t) + w_{1,2}^1 x(t-1)\right] + w_2^2 a_{12}\left(w_{2,1}^1 x(t) + w_{2,2}^1 x(t-1)\right) + \\ & w_1^2 a_{21}\left[w_{1,1}^1 x(t) + w_{1,2}^1 x(t-1)\right]^2 + w_2^2 a_{22}\left[w_{2,1}^1 x(t) + w_{2,2}^1 x(t-1)\right]^2 + \cdots \\ &= b_0 + w_1^2 a_{01} + w_2^2 a_{02} + \\ & \left[w_1^2 w_{1,1}^1 a_{11} + w_2^2 w_{2,1}^1 a_{12}\right] x(t) + \\ & \left[w_1^2 w_{1,2}^1 a_{11} + w_2^2 w_{2,2}^1 a_{12}\right] x(t-1) + \\ & \left[w_1^2 (w_{1,1}^1)^2 a_{21} + w_2^2 (w_{2,1}^1)^2 a_{22}\right] x(t)^2 + \\ & \left[w_1^2 (w_{1,2}^1)^2 a_{21} + w_2^2 (w_{2,2}^1)^2 a_{22}\right] x(t-1)^2 + \\ & \left[w_1^2 w_{1,1}^1 w_{1,2}^1 a_{21} + w_2^2 w_{2,1}^1 w_{2,2}^1 a_{22}\right] x(t)x(t-1) + \cdots \end{aligned} \tag{A.7}$$

If this last equation is compared to the Volterra series model - reproduced in equation A.8 - they are equivalent, and the Volterra kernels are identified as the terms between brackets.

$$\begin{aligned} y(t) &= h_0 + \\ & [h_1(k)]x(t) + \\ & [h_1(k)]x(t-1) + \\ & [h_2(k_1,k_1)]x(t)^2 + \\ & [h_2(k_2,k_2)]x(t-1)^2 + \\ & [h_2(k_1,k_2)]x(t)x(t-1) + \cdots \end{aligned} \tag{A.8}$$

The comparison shows that the TDNN model is equivalent to a finite memory discrete Volterra series and the Volterra kernels can be calculated using the following formulas.

$$h_0 = b_0 + \sum_{h=1}^{H} w_h^2 a_{0h} \tag{A.9}$$

$$h_1(k) = \sum_{h=1}^{H} w_h^2 w_{h,k}^1 a_{1h} \tag{A.10}$$

$$h_2(k_1,k_2) = \sum_{h=1}^{H} w_h^2 w_{h,k_1}^1 w_{h,k_2}^1 a_{2h} \tag{A.11}$$

$$h_n(k_1,\cdots,k_n) = \sum_{h=1}^{H} w_h^2 w_{h,k_1}^1 \cdots w_{h,k_n}^1 a_{nh} \tag{A.12}$$

Now, the hidden node polynomials coefficients a_{nh} have to be calculated. To do this we have to remember that we have used a Taylor expansion in the hidden units for replacing the activation function tanh. But, if as suggested in [90], this expansion is evaluated around the bias values of the hidden nodes (instead of around zero), then the polynomials coefficients can be calculated with the derivatives of the activation function, in this case tanh, with respect to the bias. Therefore, the formula for calculating the coefficient a_{nh}, of the d^{th} power, is given by,

$$a_{nh} = \frac{1}{d!}\left(\frac{\partial^d \tanh}{\partial x^d}\,|_{x=b_h}\right) \tag{A.13}$$

Putting all the pieces of the presented procedure together, we arrive at the formulas that have been presented in chapter 4 (reproduced below, where tanh means $\tanh(b_h)$), which allow calculating any kernel order from a TDNN model with any topology (any number of input neurons, any number of hidden neurons) and the hyperbolic tangent as activation function of the hidden nodes.

$$\begin{aligned}
h_0 &= b_0 + \sum_{h=1}^{H} w_h^2 \tanh & \text{(A.14)}\\
h_1(k) &= \sum_{h=1}^{H} w_h^2 w_{h,k+1}^1 \left(1-\tanh^2\right) & \text{(A.15)}\\
h_2(k_1,k_2) &= \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 \frac{\left(-2\tanh + 2\tanh^3\right)}{2!} & \text{(A.16)}\\
h_3(k_1,k_2,k_3) &= \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 w_{h,k_2+1}^1 w_{h,k_3+1}^1 \frac{\left(-2+8\tanh^2 - 6\tanh^4\right)}{3!} & \text{(A.17)}\\
h_n(k_1,\cdots,k_n) &= \sum_{h=1}^{H} w_h^2 w_{h,k_1+1}^1 \cdots w_{h,k_n+1}^1 \frac{\left(\frac{\partial^n \tanh}{\partial x^n}\,|_{x=b_h}\right)}{n!} & \text{(A.18)}
\end{aligned}$$

A.2 Coding

The code of the function that obtains a Volterra series model from a neural network model is presented in the following code lines. This functions is used when the Volterra models is calculated from a TDNN mdoel with tanh activation function. It implements the formulas that are presented in chapter 4.

```
------------------------------
function[VolterraSeries,H0,H1,H2,H3]=NN_Volterra(Input,Output,NN)
-------------------------
```

```
% Output parameters:
% VolterraSeries:  output Volterra series
% H0:  Volterra kernel of order 0
% H1:  matrix with the Volterra kernels of order 1
% H2:  matrix with the Volterra kernels of order 2
% H3:  matrix with the Volterra kernels of order 3
% Input parameters:
% Input:  input values
% Output:  output values
% NN: neural network model
-------------------------
% Extraction of parameters from the NN
H = size(NN.LW{1},1); % number of NN hidden neurons
W = NN.IW{1,1}; % input weights matrix
C = NN.LW{2,1}; % hidden neurons weights matrix
B = NN.b{1}; % hidden neurons bias matrix
D = NN.b{2}(1); % hidden neurons bias matrix
--------------------------
% Calculation of Volterra kernel of order 0
H0 = D;
for h = 1:H
H0 = h0 + C(h).*tanh(B(h));
end;
---------------------------
% Calculation of Volterra kernel of order 1
for m = 1:M, for h = 1:H,
H1(m) = H1(m) + C(h)*W(h,m)*(1 - tanh(B(h))^2);
end,end;
% H1 = [ h1(0), h1(1), h1(2), ...  ]
--------------------------
% Calculation of Volterra kernel of order 2
for m = 1:M, for k = 1:M, for h = 1:H,
H2(m,k) = H2(m,k) + C(h)*W(h,m)*W(h,k)*[(-2*tanh(B(h))+2*(tanh(B(h))^3))/2];
end,end,end;
% H2 = [ h2(0,0) h2(0,1) h2(0,2) ...
% h2(1,0) h2(1,1) h2(1,2) ...
% h2(2,0) h2(2,1) h2(2,2) ...  ]
---------------------------
% Calculation of Volterra kernel of order 3
for m = 1:M, for k = 1:M, for h = 1:H
if m == k
```

```
$H3(m,k) = H3(m,k) + C(h) * W(h,m)^2 * W(h,k) * [\frac{-2+8*tanh(B(h))^2-6*tanh(B(h))^4}{6}]$;
else
$H3(m,k) = H3(m,k) + 3 * C(h) * W(h,m)^2 * W(h,k) * [\frac{-2+8*tanh(B(h))^2-6*tanh(B(h))^4}{6}]$;
end,
end,end,end;
% H3 = [ h3(0,0,0) h3(0,0,1) h3(0,0,2) ...
% h3(1,1,0) h3(1,1,1) h3(1,1,2) ...
% h3(2,2,0) h3(2,2,1) h3(2,2,2) ...]
-----------------------------
% Volterra Analytical model
V_1=h0;
for m = 1:M,
$V\_1 = V\_1 + Vin(m, :). * H1(m)$;
end;
V_2=V_1;
for m = 1:M, for h = 1:M,
$V\_2 = V\_2 + Vin(m, :). * Vin(h, :). * H2(m,h)$
end,end;
V_3=V_2;
for m = 1:M, for h = 1:M,
$V\_3 = V\_3 + Vin(m, :). * Vin(m, :). * Vin(h, :). * H3(m,h)$;
end,end;
-----------------------------
VolterraSeries=V_3;
-----------------------------
```

A.3 Summary

This appendix began with a detailed explanation of the procedure that allows obtaining the Volterra kernels from a neural network and its parameters. Finally, the code that implements the presented formulas and that has been used to automatically calculate the Volterra kernels was shown.

List of figures

Bibliography

[1] A. Ahmed, E.R. Srinidhi and G. Kompa, "Efficient PA Modeling Using Neural Network and Measurement Setup for Memory Effect Characterization in the Power Device, In Proc. IMS, Amsterdam, USA, pp. 191-194, xxx (2005)

[2] A. Ahmed, M.O. Abdalla, E.S. Mengistu and G. Kompa, "Power Amplifier Modeling using memory polynomial with non-uniform delay taps", in Proc. IEEE 34th European Microwave Conference, Amsterdam, Holland, pp. 1457-1460 (2004)

[3] A. Alabadelah, T. Fernandez, A. Mediavilla, B. Nauwelaers, A. Santarelli, D. Schreurs, A. Tazn and P.A. Traverso, "Nonlinear Models of Microwave Power Devices and Circuits", In Proc. GaAs Symposium, Amsterdam, Holland, pp. 191-194, October (2004)

[4] A. Bauer and W. Schwarz, "Circuit analysis and optimization with automatically derived Volterra kernels", in Proc. IEEE Int. Symposium on Circuits and Systems, vol. 1, pp. 491-494 (2000)

[5] F. Bonani, S. Donati Guerrieri, F. Filicori, G. Ghione and M. Pirola, "Physics-based large-signal sensitivity analysis of microwave circuits using technological parametric sensitivity from multidimensional semiconductor device models", *IEEE Trans. Microwave Theory Tech.*, vol. 45, no. 5, pp. 846-855 (1997)

[6] S. Boyd, Y.S. Tang and L.O. Chua, "Measuring Volterra kernels", *IEEE Trans. Circuits and Systems*, vol. 30, no. 8, pp. 571-577 (1983)

[7] S. Boyd, L.O. Chua and C.A. Desder, "Analytical Foundations of Volterra Series", *IMA Journal of Mathematical Control & Information*, vol. 1, pp. 243-282 (1984)

[8] P. Burrascano, S. Fiori and M. Mongiardo, "A Review of Artificial Neural Networks Applications in Microwave Computer-Aided Design", *Int. Journal of RF and Microwave CAE*, vol. 9, no. 3, pp. 158-174 (1999)

[9] F.G. Canavero, I.A. Maio and I.S. Stievano, "Behavioral modeling of digital devices via black-box identification", in Proc. IEEE 4th Workshop on Signal Propagation on Interconnects, Magdeburg, , pp. xx-xx, May (2000)

[10] L.O. Chua, "Device Modeling via basic nonlinear circuit elements", *IEEE Trans. Circuits and Systems*, vol. 27, no. 11, pp. 1014-1044 (1980)

[11] L.O. Chua, "Nonlinear Circuits", *IEEE Trans. on Circuits and Systems*, vol.

31, no. 1, pp. 69-87 (1984)

[12] W.R. Curtice and M. Ettenberg, ”A Nonlinear GaAs FET Model for Use in the Design of Output Circuits for Power Amplifiers”, *IEEE Trans. Microwave Theory Tech.*, vol. 33, pp. 1383-1386 (1985)

[13] W.R. Curtice, ”GaAs MESFET Modeling and Nonlinear CAD”, *IEEE Trans. Microwave Theory Tech.*, vol. 36, no. 2, pp. 220-230 (1988)

[14] G. Cybenko, ”Approximation by superposition of sigmoidal functions”, *Math. Control, Sygnals, Syst.*, vol. 2, pp. 303-314 (1989)

[15] G. Cybenko, ”Neural Networks in Computational Science and Engineering”, *IEEE Computational Science and Eng.*, vol. 3, no. 1, pp. 36-42 (1996)

[16] G.W. Davis and M.L. Gasperi, ”ANN Modeling of Volterra systems”, in Proc. Int. Joint Conference on Neural Networks, vol. 2, pp. 727-734 (1991)

[17] M. Deo, J. Xu and Q.J. Zhang, ”A New Formulation of Dynamic Neural Network for Modeling of Nonlinear RF/Microwave Circuits”, in Proc. IEEE 33^{th} European Microwave Conference, Munich, Germany, pp. 1019-1022 (2003)

[18] J. Dooley, B. O’Brien and T. Brazil, ”Behavioral modeling of RF power amplifiers using modified Volterra series in the time domain”, High Frequency Postgraduate Student Colloquium, pp. 169-174 (2004)

[19] C. Evans, D. Rees, L. Jones and M. Weiss, ”Periodic signals for measuring nonlinear Volterra kernels”, *IEEE Trans. Instrumentation and Measurement*, vol. 45, no. 2, pp. 362-371 (1996)

[20] Y. Fang, M.C. Yagoub, F. Wang, and Q.J. Zhang, ”A new macromodeling approach for nonlinear microwave circuits based on recurrent neural networks”, *IEEE Trans. Microwave Theory Tech.*, vol. 48, no. 12, pp. 23352344 (2000)

[21] F. Filicori, G. Ghione and C.U. Naldi, ”Physics-based electron device modelling and computer-aided MMIC design”, *IEEE Trans. Microwave Theory Tech.*, vol. 40, no. 7, pp. 1333-1352 (1992)

[22] F. Filicori, G. Vannini and V.A. Monaco, ”A nonlinear integral model of electron devices for HB circuit analysis”, *IEEE Trans. Microwave Theory Tech.*, vol. 40, no. 7, pp. 1456-1465 (1992)

[23] J.A. Garcia, A. Tazon Fuente, A. Mediavilla, I. Santamaria, M. Lazaro, C. Pantaleon and J.C. Pedro, ”Modeling MESFETs and HEMTs Intermodulation Distortion behavior using a generalized Radial Basis Function Network”, *Int. Journal of RF and Microwave CAE, Special Number on Neural Networks*, no. 9, pp. 261-276 (1999)

[24] J.A. Garcia, M.L. De la Fuente, J.C. Pedro, N.B. Carvalho, Y. Newport, A. Mediavilla and A. Tazon, ”Time-varying Volterra-series analysis of spectral regrowth and noise power ratio in FET mixers”, *IEEE Trans. Microwave Theory Tech.*, vol. 49, no. 3, pp. 545-549 (2001)

[25] G. Ghione and M. Pirola. *Elettronica delle microonde*, Ed. Otto editore, Torino, IT (2002)

[26] G.B. Giannakis and E. Serpedin, ”A bibliography on nonlinear system identification”, pp. 1-71 (2000)

[27] F. Giannini, G. Leuzzi, G. Orengo and M. Albertini, ”Small-signal and Large-signal modeling of active devices using CAD-optimized Neural Networks”, *Int. Journal of RF and Microwave CAE*, Special number on Neural Networks, vol. 12, no. 1, pp. 71-78 (2002)

[28] F. Giannini, G. Leuzzi, G. Orengo and P. Colantonio, ”Modelling Power and Intermodulation Behaviour of Microwave Transistors with Unified Small-Signal Neural Network Models”, *Int. Journal of RF and Microwave CAE*, vol. 13, no. 4, pp. 476-484 (2003)

[29] G. Glentis, P. Koukoulas and N. Kalouptsidis, ”Efficient Algortihms for Volterra system identification”, *IEEE Trans. Signal Processing*, vol. 47, no. 11, pp. 3042-3057 (1999)

[30] G. Govind and P.A. Ramamoorthy, ”Multi-layered NN and Volterra series: the missing link”, in Proc. Int. Conference on Systems Engineering, pp. 633-636 (1990)

[31] N.Z. Hakim, J.J. Kaufman, G. Cerf and H.E. Meadows, ”Volterra-Wiener characterization of a Recurrent Neural Networks”, in Proc. IEEE Engineering in Medicine and Biology, vol. 13, no. 3, pp. 1397-1398 (1991)

[32] S. Haykin, ”Neural Networks”, Ed. MacMillan (1994)

[33] Y. Harkouss, J. Rousset, H. Chehade, E. Ngoya, D. Barataud and J.P. Teyssier, ”Modeling Microwave Devices and circuits for telecommunications systems design”, in Proc. IEEE World Congress on Computational Intelligence, vol. 1, pp. 128-133 (1998)

[34] Y. Harkouss, J. Rousset, H. Chehade, E. Ngoya, D. Barataud and J.P. Teyssier, ”The use of ANN in nonlinear microwave devices and circuits modeling: an application to telecommunication system design”, *Int. Journal of RF and Microwave CAE*, vol. 9, pp. 198-215 (1999)

[35] S. Hassouna and P. Cirault, ”Identification of Volterra kernels of nonlinear systems”, pp. 3502-3507 (2000)

[36] J. Hopfield, ”Neural Networks and Physical Systems with Emergent Collective Computational Abilities”, in Proc. National Academy of Sciences USA, vol. 79, pp. 2554-58 (1982).

[37] K. Hornik, M. Stinchcombe and H. White, ”Multilayer feedforward networks are universal approximators”, *Neural Networks*, vol. 2, pp. 359366, 1989.

[38] M. Iatrou, T.W. Berger and V.Z. Marmarelis, ”Modeling of nonlinear nonstationary dynamic systems with a novel class of Artificial Neural Networks”, *IEEE Trans. Neural Networks*, vol. 10, no. 2, pp. 327-339 (1999)

[39] J. Jargon, K.C. Gupta and D. DeGroot, ”Application of Artifical Neural Networks to RF and microwave measurements”, *Int. Journal of RF and Microwave CAE*, vol. 12, pp. 3-24 (2002)

[40] H. Kashiwagi, H. Harada and L. Rong, "Identification of Volterra kernels of nonlinear systems by separating overlapped kernel slices", in Proc. 41th Conference of the Society of Instrument and Control Engineers of Japan, pp. 707-712 (2002)

[41] W.G. Knecht, "Nonlinear noise filtering and beamforming using the perceptron and its Volterra approximation", *IEEE Trans. Speech and Audio Processing*, vol. 2, no. 1, pp. 55-62 (1994)

[42] T. Kohonen, "The self-organizing map", *Proceedings of the IEEE*, vol. 78, pp. 1464-1480 (1990)

[43] H. Ku, M. Mckinley and J.S. Kenney, "Extraction of Accurate Behavioral Models for Power Amplifiers with Memory Effects using Two-tone Measurements", in *IEEE-MTT-S Int. Microwave Symp. Dig.*, pp. 139-142 (2002)

[44] H. Ku and J.S. Kenney, "Behavioral Modeling of Nonlinear RF Power Amplifiers Considering Memory Effects", *IEEE Trans. Microwave Theory Tech.*, vol. 51, no. 12, pp. 2495-2504 (2003)

[45] N. Le Gallou, D. Barataud, H. Buret, J.M. Nebus and E. Ngoya, "A novel measurement method for the extraction of dynamic Volterra kernels of microwave power amplifiers", in Proc. European GaAs Applications Symposium, pp. 330-333 (2000)

[46] T. Liu, S. Boumaiza and F.M. Ghannouchi, "Dynamic Behavioral Modeling of 3G Power Amplifiers using real-valued Time-Delay Neural Networks", *IEEE Trans. Microwave Theory Tech.*, vol. 52, no. 3, pp. 1025-1033 (2004)

[47] S.A. Maas, *Nonlinear Microwave circuits.* Ed. Artech House, Boston, MA (1988)

[48] S. Maas and A. Crosmun, "Modeling the gate I/V characteristics of a GaAs MESFET for Volterra-series analysis", *IEEE Trans. Microwave Theory Tech.*, vol. 37, no. 7, pp. 1134-1136 (1989)

[49] S. Maas, "Fixing the Curtice FET Model", *IEEE Microwave Journal*, Ed. Artech House, pp. 311-314 (2002)

[50] V.Z. Marmarelis and X. Zhao, "Volterra models and Three Layers Perceptron", *IEEE Trans. Neural Networks*, vol. 8, no. 6, pp. 1421-1433 (1997)

[51] D.W. Marquardt, "An algorithm for least-squares estimation of non-linear parameters", *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431-441 (1963)

[52] P. Meijer, "Neural Networks applications in device and subcircuit modeling for circuit simulation", PhD Thesis, Proefschrift Technische Universiteit Eindhoven, Netherlands (2003)

[53] M.R.G. Meireles, P.E.M. Almeida and M.G. Simoes, "A comprehensive review for industrial applicability of Artificial Neural Networks", *IEEE Trans. Industrial Electronics*, vol. 50, no. 3, pp. 585-601 (2003)

[54] R.A. Minasian, "Intermodulation distortion analysis of MESFET amplifiers using the Volterra series representation", *IEEE Trans. Microwave Theory Tech.*, vol. 28, no. 1, pp. 1-8 (1980)

[55] D. Mirri, G. Iuculano, F. Filicori, G. Pasini, G. Vannini and G. Pellegrini Gualtieri, "A modified Volterra series approach for nonlinear dynamic systems modeling", *IEEE Trans. Circuits and Systems. Part I: Fundamental Theory and Applications*, vol. 49, no. 8, pp. 1118-1128 (2002)

[56] S-W. Nam and E.J. Powers, "Volterra series representation of time-frequency distributions", *IEEE Trans. Signal Processing*, vol. 51, no. 6, pp. 1532-1537 (2003)

[57] J. Nemeth, I. Kollar and J. Schoukens, "Identification of Volterra kernels using interpolation", *IEEE Trans. Instrumentation and Measurement*, vol. 51, no. 4, pp. 770-775 (2001)

[58] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights", in proc. IEEE Int. Joint Conf. Neural Networks, vol. 3, pp. 21-26 (1990)

[59] S. Osowski and T. Vu Quang, "Multilayer Neural Network structure as Volterra filter", in Proc. IEEE Int. Symposium on Circuits and Systems, pp. 253-256 (1994)

[60] J. O'Toole and T. Brazil, "Novel Neural Network-Bessel transform for behavioral modeling of a Power Amplifier", pp. 45-50 (2000)

[61] R. Parker and M. Tummala, "Identification of Volterra systems with a polynomial Neural Network", in Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing, pp. 561-564 (1992)

[62] J.C. Pedro, J.C. Madaleno and J.A. Garcia, "Theoretical basis for extraction of mildly nonlinear behavioral models", *Int. Journal of RF and Microwave CAE*, vol. 13, pp. 40-53 (2003)

[63] J.C. Pedro and S.A. Maas, "A comparative overview of microwave and wireless power-amplifier behavioral modeling approaches", *IEEE Trans. Microwave Theory Tech.*, vol. 53, no. 4, pp. 1150-1163 (2005)

[64] A. Pinkus, "Approximation Theory of the MLP model in neural networks", *Acta Numerica*, pp. 143-195 (1999)

[65] D.E. Root, J. Wood and N. Tufillaro, "New Techniques for non-linear behavioral modeling of microwave/RF ICs from simulation and nonlinear microwave measurements", pp. 85-90 (2003)

[66] D. E. Root and J. Wood. *Fundamentals of Nonlinear Behavioral Modeling for RF and Microwave design.* Ed. Artech House (2005)

[67] W. Rugh. *Nonlinear system theory. The Volterra/Wiener approach.* Johns Hopkins University Press (1981)

[68] M. Schetzen. *The Volterra and Wiener Theories of nonlinear systems.* Ed. John Wiley & sons, (1980)

[69] D. Schreurs, N. Tufillaro, J. Wood, D. Usikov, L. Barford and D.E. Root, "Development of time-domain behavioral nonlinear models for microwave devices and ICs from vectorial large-signal measurements and simulations", in Proc. European GaAs Applications Symposium, Paris, France, pp. 236-239 (2000).

[70] D. Schreurs, J. Wood, N. Tufillaro, L. Barford and D.E. Root, "Construction of behavioral models for microwave devices from time-domain large-signal measurements to speed up high-level design simulations", *Int. Journal of RF and Microwave CAE*, vol. 13, pp. 54-61 (2003)

[71] D. Schreurs, J. Wood, N. Tufillaro, D. Usikov, L. Barford and D.E. Root, "The construction and evaluation of behavioral models for microwave devices based on time-domain large-signal measurements", Int. Electron Devices Meeting, IEDM Technical Digest, pp. 819-822, (2000)

[72] D. Schreurs, J. Verspecht, E. Vandamme, N. Vellas, C. Gaquiere, M. Germain and G. Borghs, "ANN model for AlGaN/GaN HEMTs constructed from near-optimal-load large-signal measurements", in *IEEE-MTT-S Int. Microwave Symp. Dig.*, vol. 1, pp. 447-450 (2003)

[73] J. Sjoberg and L. Ljung, "Overtraining, regularization and searching for a minimun, with application to Neural Networks", *Int. Journal of Control*, no. 62, pp. 1391-1407 (1995)

[74] D.I. Soloway and J.T. Bialasiewicz, "Neural Network modeling of nonlinear systems based on Volterra series extension of a linear model", in Proc. IEEE Int. Symposium on Intelligent Control, pp. 7-12 (1992)

[75] V. Teppati, *Microwave device characterization techniques oriented to increase the measurements accuracy*, PhD Thesis, Politecnico di Torino (2002)

[76] T.R. Turlington. *Behavioral Modeling of nonlinear RF and Microwave devices*, Ed. Artech House, Boston, MA (2000)

[77] P. Van Esch and J. Verspecht, "The use of feedforward Artificial Neural Networks in the building of the non-linear behavioral models of microwave components in the frequency domain", in Proc. Int. Workshop on Advanced Black-Box Techniques for Nonlinear Modeling, pp. 222-227 (1998)

[78] F. Verbeyst and M. Vanden Bossche, "VIOMAP, the s-parameters equivalent for weakly nonlinear RF and microwave devices", *IEEE Trans. Microwave Theory Tech.*, vol. 42, no. 12, pp. 2531-2535 (1994)

[79] F. Verbeyst and M. Vanden Bossche, "The Volterra input-output map of high frequency amplifier as a practical alternative to load-pull measurements", *IEEE Trans. Instrumentation and Measurements*, vol. 44, no. 3, pp. 1-12 (1995)

[80] J. Verspecht, F. Verbeyst, M. Vanden Bossche and P. Van Esch, "System level simulation benefits from frequency domain behavioral models of mixers and amplifiers", in Proc. European Microwave Conference (1999)

[81] J. Verspecht, "Black-box modeling of power transistors in the frequency domain", in Pro. 4th Int. Workshop on Integrated Nonlinear Microwave and Millimeterwave Cicuits (INMMIC) (1996)

[82] V. Volterra, *Theory of Functionals and Integral and Integro-Differential Equations*, Ed. Dover, New York (1959).

[83] T. Wang and T.J. Brazil, "Volterra-mapping-based behavioral modeling of nonlinear circuits and systems for high frequencies", *IEEE Trans. Microwave Theory Tech.*, vol. 51, no. 5, pp. 1433-1440 (2003)

[84] T. Wang and T.J. Brazil, "The estimation of Volterra transfer functions with applications to RF power amplifier behavior evaluation for CDMA digital communication", in *IEEE MTT-S Int. Microwave Symp. Dig.*, pp. 425-428 (2000)

[85] D. Weiner and G.H. Naditch, "A scattering variable approach to the Volterra analysis of nonlinear systems", *IEEE Trans. Microwave Theory Tech.*, vol. 24, no. 7, pp. 422-433 (1976)

[86] J. Wood, D.E. Root and N.B. Tufillaro, "A Behavioral modeling approach to nonlinear model-order reduction for RF/Microwave ICs and Systems", *IEEE Trans. Microwave Theory Tech.*, vol. 52, pp. 2274-2284 (2004)

[87] J. Wood and D.E. Root, "The Behavioral modeling of RF/Microwave ICs using nonlinear time series analysis", in *IEEE MTT-S Int. Microwave Symp. Dig.*, pp. 791-794 (2003)

[88] W.P. Woods, A.H. Batchelor, S.R. Johnson and G.G.R. Green, "Comparison of the response of a time delay NN with an analytical model", in Proc. Int. Joint Conference on Neural Networks, vol. 2, pp. 1120-1125 (2002)

[89] J. Wray and G.G.R. Green, "Calculation of the Volterra kernels of nonlinear dynamic systems using an Artificial Neural Network", *Biological Cybernetics*, vol. 71, pp. 187-195 (1994)

[90] J. Wray and G.G.R. Green, "Neural Networks, approximation theory and finite precision computation", *Neural Networks*, vol. 8, no. 1, pp. 31-37 (1995)

[91] J. Xu, M.C.E. Yagoub, R. Ding and Q.J. Zhang, "Neural-based dynamic modeling of nonlinear microwave circuits", in *IEEE MTT-S Int. Microwave Symp. Dig.*, pp. 1101-1104 (2002)

[92] M. Yusof, "Modeling Volterra series based on input/output data", in Proc. 4th National conference on Telecommunication Technology, pp. 222-225 (2003)

[93] Q.J. Zhang and K.C. Gupta, *Neural Networks for RF and Microwave Design*, Ed. Artech House, Boston, MA (2000)

[94] Q.J. Zhang, K.C. Gupta and V.K. Devabhaktuni, "Artificial Neural Networks for RF and Microwave Design - From Theory to Practice", *IEEE Trans. Microwave Theory Tech.*, vol. 51, pp. 1339-1350 (2003)

[95] A. Zhu and T. Brazil, "Behavioral modeling of RF power amplifiers based on pruned Volterra series", *IEEE Wireless and Microwave components letters*, vol. 14, no. 12, pp. 563-565 (2004)