Piecewise-Linear Modeling of Analog Circuits Based on Model Extraction from Trained Neural Networks

Gaurav Gothoskar and Alex Doboli
VLSI Systems Design Laboratory
Electrical and Computer Engineering Department
State University of New York at Stony Brook, Stony Brook, NY, 11794-2350
Email: {gaurav, adoboli}@ece.sunysb.edu

Simona Doboli
Department of Computer Science
Hofstra University, Hempstead, NY, 11549
Email: cscszd@hofstra.edu

Abstract

This paper presents a new technique for automatically creating analog circuit models. The method extracts piecewise linear models from trained neural networks. A model is a set of linear dependencies between circuit performances and design parameters. The paper illustrates the technique for an OTA circuit for which models for gain and bandwidth are generated. As experiments show, the obtained models have a simple form that accurately fits the sampled points. These models are useful for fast simulation of systems with non-linear behavior and performances.

1 Introduction

The need for mixed analog-digital designs is predicted to dramatically increase over the next years [1] [9]. The digital part of mixed-signal systems can be efficiently designed with a low effort using modern high-level, logic-level and physical-level design automation tools. In contrast, there is a lack of systematic design methods and efficient general-purpose synthesis environments for analog circuits [1] [9]. As a result, analog designs continue to seize a considerable portion of the total design time for mixed-signal systems [8] [9]. There is a persistent need for developing improved methods and tools to level the design productivity and quality of analog circuits. This paper presents an original analog circuit modeling method that can be efficiently employed for both circuit design and synthesis.

Analog circuit models (macromodels) express mathematical relationships between *significant* electrical and geometrical parameters of a circuit (like device sizes, layout parasitics, signal frequencies, noise etc) and *specific* performance attributes (such as circuit gain, bandwidth, power consumption, slew rate, harmonic distortion etc) [9]. For example, it is customary to formulate mathematical equations for an op amp gain and bandwidth as functions of transistor sizes and the value of the compensating capacitor [13]. Models are key components for both manual design and automated synthesis of analog circuits. A designer uses circuit models to capture the relevant dependencies in a design, and then find the design parameters (i.e. device sizes) according to the performance requirements that need to be satisfied [13]. Circuit synthesis tools use circuit models to improve the effectiveness of the exploration process and speed-up their convergence towards optimal solutions [14] [20]. In both cases, models must *accurately capture the behavior of circuits* without *increasing the complexity of their mathematical expression* [14].

Circuit models are very important for speeding up the convergence of simulation-based circuit synthesis tools. It has been reported that one of the important challenges is the large number of optimization variables that must be simultaneously tackled [11] [14]. This poses challenges to traditional exploration-based synthesis methods, which need a very long time to complete their search, or might even not converge towards a good solution [11]. A solution to this problem is to use models to speed up synthesis by guiding the search towards attractive solution space regions [6]. Most of the time, models are used to quickly find the performance attributes of the explored designs. Periodically, exhaustive circuit simulations are performed to correct the inaccuracies introduced by the models. Even for very sensitive designs such as RF mixer circuits, it has been shown that performance estimation through a combined circuit model evaluation and circuit simulation offers good accuracy levels while significantly reducing synthesis time [20].

Another important application of circuit models is for top-down design and synthesis of analog systems [5]. Top-down system synthesis proceeds in two steps [5]: the first step (called architecture generation) explores alternative architectures for a system expressed at an abstract level. The second step (named constraint generation) allocates performance constraints to each block in an architecture so that the overall system performances are optimized. Note that an abstract perspective on the analog circuits is maintained during both synthesis steps. As a result the constraints allocated to a block might be very difficult to be obtained with real analog circuits (for example large gains for large circuit bandwidths, high slew rates etc). Also, important aspects such as noise and layout parasitics are neglected during system design. Circuit models help eliminating these limitations by providing knowledge on (1) circuit performance trade-offs, (2) feasibility ranges for circuit performances, and (3) impact of physical-level elements such as noise and layout parasitics on circuit performances.

Circuit modeling techniques fall into two categories: (1) *physical modeling methods* and (2) *mathematical modeling techniques* [14]. Physical modeling methods simplify a circuit to a reduced sub-circuit that includes only the dominant devices

*Supported by Defense Advanced Research Projects Agency (DARPA) and managed by the Sensors Directorate of the Air Force Research Laboratory, USAF, Wright-Patterson AFB, OH 45433-6543

in the circuit. Such models are useful in offering a qualitative insight into the circuit but are limited in offering also a quantitative perspective. Models can be successfully used for circuit analysis but not for device sizing, circuit optimization and synthesis. Mathematical models capture quantitative relationships between the parameters and performances of a circuit. However, these models might not have any connection to the physical structure of the circuit. Non-linear regression methods are traditionally used to produce mathematical models [10] [3] [4]. The main limitation is that for a large number of data points, it is very difficult to find a single mathematical formula that accurately fits all points [14].

This paper presents a new technique for extracting piecewise linear models from trained neural networks. A model is a set of linear dependencies between circuit performances and design parameters. Dependencies are valid over a range of the parameters. Section 5 presents for an OTA circuit [13] the extracted models for gain and bandwidth as functions of frequency and layout parasitics. As experiments show, the produced piecewise linear models have a simple form that accurately fits the sampled points. Moreover, piecewise linear models are a promising method for approximating nonlinear behavior and performances with a small error [12]. There are powerful simulation methods that use piecewise linear models to quickly calculate system performances [12]. Our work addresses the need for a method to systematically create piecewise linear models used for simulation [12].

The model generation techniques starts with the step of training a neural network. A backpropagation algorithm is used for training until the desired accuracy is obtained at the output of the network. Next, a pruning method is applied to eliminate the neurons with insignificant contributions to the model. The size of the network is thus reduced without significantly affecting the modeling accuracy. Then, the sigmoidal activation function of each neuron is approximated with a piecewise linear function that includes three linear segments. Finally, the piecewise linear functions for input, hidden and output neurons are composed together to generate the final model of a circuit. The function composition algorithm is based on automatically expressing linear equations and inequalities for the neurons. Equations are then solved to find the feasibility (input) domain for each linear segment in the model.

The paper includes six sections. Section 2 presents related work on modeling with neural networks, and highlights the main contributions of this paper. Section 3 offers a theoretical description of the modeling problem. Section 4 presents the algorithm for extracting piecewise-linear models from trained neural networks. Section 5 illustrates the models generated for an OTA circuit. Finally, we put forth our conclusions.

2 Related Work on Modeling with Neural Networks

Neural networks have been successfully used in various types of problems, including classification and function approximation. They are able to learn any type of nonlinear mapping based on their well known property of universal approximators. The main problem of neural networks consists in their opaque representation of the knowledge embedded in the parameters of the model. Due to the nature of processing that takes place in a neural network - parallel distributed processing among connected neurons - it is very difficult to interpret what a neural network does.

Extracting symbolic knowledge out of a neural network would make the interpretation of the solution much easier. Several methods have been proposed for extracting rules from trained neural networks [19, 7, 2, 18] (for a review see [19]). Most techniques were developed for classification problems, and very few have been proposed for regression or function approximation problems [18, 16]. Previous extraction methods for classification problems attempt to translate a neural network model into a set of if-then rules. Rule extraction methods differ in the type of neural networks on which they are designed to work, in the type of rules they extract, in the complexity of the extraction algorithm and in the easiness of the rule interpretation.

Recently a new technique has been proposed to extract linear models for regression problems [16]. The method is similar to the one we use here. A neural network is first trained and pruned. Then, the activation function of each hidden neuron is substituted with a piecewise linear function with three or five regions. For each nonempty combination of hidden neuron regions a linear model is generated. The coefficients of the model depend on the weights of the network. The number of linear models that are generated is equal to the number of linear regions for the activation function of a hidden neuron to the power of the number of hidden neurons. The output neuron has a linear activation function. The limits of the constraints which define such a region are fixed and correspond to the values delimiting the piecewise linear regions of the activation function.

The main differences between our method and [16] consists in the way linear models are generated and in the way the set of constraints - which define the region where a linear model is valid - are extracted. In our approach a linear model is generated only when a region in the input space is valid, meaning that is has a nonempty solution set. Moreover, the constraint limits are adjusted until they represent the smallest possible region. Also, the constraint set is reduced by eliminating redundant information. In this way, we minimize the number of generated linear models and the number of constraints that define a linear model's valid region. The latter aspect is necessary to improve the understandability of the generated rules.

3 Problem definition

The task is to approximate the nonlinear function represented by a trained feedforward neural network with a piecewise linear mapping. The neural network considered here has three layers: an input layer $\mathcal{I}$, a hidden layer $\mathcal{H}$ and an output layer $\mathcal{O}$. The goal of the extraction method is to find a set of L linear models each of the following form: $\mathcal{L} = \{a_1^l x_1 + a_2^l x_2 + \ldots + a_I^l x_I,\ l = 1 \ldots L,\ a_{(.)}^l \in \Re\}$, where x_i is the output of a neuron in the $\mathcal{I}$ layer. The region in the input space where model l is valid is defined by a set of constraints of the following form: $\mathcal{C}^l = \{c_1^m x1 + c_2^m x_2 + \ldots + c_I^m x_I\ \{<, \leq, >, \geq\}\ d^m,\ m = 1 \ldots M^l,\ c_i^m,\ d^m \in \Re\}$, where M^l is the number of constraints for model l.

The model l is active if all the constraints in $\mathcal{C}^l$ are satisfied for a set of input values: $\{x_1, \ldots, x_I\}$ and inactive if at least one of the constraints is violated. The region in the input space where a constraint set $\mathcal{C}^l$ is satisfied is called the valid region of model l. All constraint sets $\mathcal{C}^l$ must satisfy the following requirements:

1. The valid regions of any pair of linear models must not intersect in any point in the input space: $\mathcal{C}^p \cap \mathcal{C}^r = \emptyset$, for $p \neq r$.
2. The set of constraints in $\mathcal{C}^l$ is minimal. By removing any constraint, the valid region for model l changes.

4 Model Extraction Method

The neural networks considered here are three layer feed-forward networks. There are N input neurons in the $\mathcal{I}$ layer, H hidden neurons in the $\mathcal{H}$ layer and O output neurons in the $\mathcal{O}$ layer. The weight matrix between the input and the hidden layer is $W^{IH} = \{w_{ji}, j = 1 \dots H,\ i = 1 \dots N+1\}$, where w_{ji} is the weight of the connection between the input neuron i and the hidden neuron j. The input layer and the hidden layer are augmented with a bias neuron. The weight matrix between the hidden and the output layer is $W^{HO} = \{w_{kj}, k = 1 \dots O,\ j = 1 \dots H+1\}$ with w_{kj} the strength of the connection between output neuron k and hidden neuron j.

The activation function of the hidden and output neurons is the sigmoidal function $\phi(x) = \frac{1}{1+exp(-\lambda x)}$, with $0 < \lambda \leq 1$. The weighted sum at the input of a hidden neuron and at the input of an output neuron are respectively:

$$h_j = \sum_{i=1}^{N} w_{ji} x_i, \quad h_k = \sum_{k=1}^{H} w_{kj} x_j. \tag{1}$$

where x_i is the output of the input neuron i. The output of the hidden neuron j is: $x_j = \phi(h_j)$, and the output of the x_k neuron is: $x_k = \phi(h_k)$.

First, the network is trained with the backpropagation algorithm [15] until the desired mean square error on the training and validation data sets is reached. Second, a pruning technique is applied to eliminate the insignificant weights.

The pruning repeatedly removes the most insignificant weight from the remaining weights until a stopping criteria is satisfied. The significance of a weight is proportional with the reduction in accuracy on both training and validation data. The accuracy reduction is measured iteratively as: $\Delta MSE(r) = MSE(r-1) - MSE(r)$, where $MSE(r)$ is the mean square error on the traininig and validation data of the neural network at step r of the pruning process. The weight that produces the least reduction in accuracy at each step is eliminated. The stopping criteria is the total loss of accuracy, which has to be smaller than a maximum limit. If a weight between a hidden and an output neuron is eliminated then the hidden neuron and all the weights between the input layer and the hidden neuron are also eliminated. The pruning is necessary in order to reduce the number of liner models.

The extraction algorithm starts with the pruned network. The idea is to approximate the nonlinear sigmoidal activation function with a piecewise linear function. One way to do this is by dividing the input space into three regions: for small x values $\phi(x)$ is approximately 0 – *the constant region 0*, for large values $\phi(x)$ is close to 1 – *the constant region 1* and for values in between $\phi(x)$ can be approximated with a linear function – *the variable region*. The corresponding linear mapping $g(x)$ is shown in Figure 4 and is defined as follows:

$$g(x) = \begin{cases} 0 & x < -\gamma \\ \frac{x+\gamma}{2\gamma} & -\gamma \leq x \leq \gamma \\ 1 & x > \gamma \end{cases} \tag{2}$$

where γ is a real value that separates the constant regimes from the variable one. The linear function of the variable region is a first order approximation of the Taylor series decomposition of $\phi(x)$ for $x = 0$. The value of γ is chosen such that it minimizes the error between the sigmoidal function $\phi(x)$ and the linear mapping $g(x)$. The linearization of the sigmoidal function can be done in other ways as well. One possibility to improve the accuracy of the linearization is by increasing the number of linear regions. In the case of $g(x)$ the number of linear regions is $R = 3$.

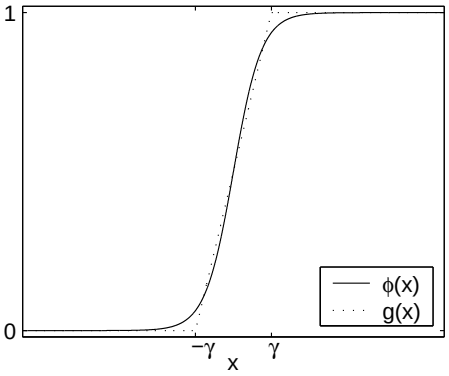

Figure 1: The graph of the sigmoidal function ($\phi(x)$) and of the approximated piecewise linear mapping ($g(x)$)

.

The next step consists in finding the the linear models and the regions in the input space where they are valid. For each region, a linear function in the input variables is generated by substituting the sigmoidal activation function $\phi(x)$ with the linear one $g(x)$ in $x_k = \phi(h_k)$. The coefficients of the linear functions depend on the weight values:

$$x_k = \phi(\sum_{h=1}^{H} w_{kj} \phi(\sum_{i=1}^{N} w_{ji} x_i)) \approx g(\sum_{h=1}^{H} w_{kj} g(\sum_{i=1}^{N} w_{ij} x_i)) \tag{3}$$

The main steps of the linear model extraction method are as follows:

1. *Linearization of the hidden neurons*. Extract and refine the sets of constraints $\mathcal{C}^{jr}$, $h = 1 \dots H$, $r = 1 \dots R$, with R the number of linear regions of $g(x)$. A set $\mathcal{C}^{jr}$ represents the region where the output of the hidden neuron j is given by $x_j = g_r(h_j)$, with $g_r(.)$ the r branch of the linear function $g(x)$. The refining process of a set of constraints C^{jr} is described below.

2. *Evaluatation of the weighted sum input to the output neurons (h_k).* Find all valid combinations of linear regions in the hidden neurons of the following form: $\mathcal{O}^p = \{r_1^p\ r_2^p\ \dots r_H^p\}$, $r_h^p \in \{1,\ 2,\ \dots,\ R\}$ and $p = 1 \dots R^H$. For each such combination, the output of all hidden neurons is completely specified, and therefore h_k can also be evaluated. The region in the input space where combination $\mathcal{O}^p$ is valid is defined by the constraint set: $\mathcal{C}^p = \mathcal{C}^{1r_1^p} \cap \mathcal{C}^{2r_2^p} \cap \dots \cap \mathcal{C}^{Hr_H^p}$. The process of intersecting sets of constraints is detailed below. A combination $\mathcal{O}^p$ is valid if its constraint set $\mathcal{C}^p$ delimits a nonempty region in the input space. The valid constraint sets $\mathcal{C}^p$ are refined as in step 1.

3. *Linearization of the output neuron.* For each valid combination region defined by $\mathcal{C}^p$, express the output of the network as a linear combination of input variables. Depending on the range of values for h_k^p - the weighted sum input to output neuron k for combination p - the activation function of the output neuron can be split into one or more linear regions ($g_r(h_k^p)$). The maximum number of linear models that can be generated is $R^{(H+1)}$. The set of constraints that define the validity region of a linear

model is obtained as follows: $\mathcal{C}^{pr} = \mathcal{C}^p \cap \mathcal{C}^r$, where $\mathcal{C}^r$ is the inequality constraint for the r branch of the linear function $g_r(x)$ (relation (2)). The set of constraints for each linear model $\mathcal{C}^{pr}$ is refined using the same methodology as in step 1. The linear models are expressed using relation (3), with $g(.)$ substituted with the right branch $g_r(.)$.

The process of refining a set of linear inequality constraints is now detailed and ilustrated for one of the hidden neurons. For example, the initial set of constraints for which hidden neuron j is in the constant region 0 ($x_j = g_1(h_j) = 0$) consists of the following ($\mathcal{C}^{j1}$):

$$C^{j1} = \begin{cases} \sum_{i=1}^{N} w_{ji} x_i & < & -\gamma \\ x_1 & \leq & M_1 \\ x_1 & \geq & m_1 \\ \ldots\ldots\ldots\ldots\ldots & & \\ x_I & \leq & M_I \\ x_I & \geq & m_I \end{cases} \quad (4)$$

where M_i and m_i are the maximum and respectively minimum values of the input neuron i. The set of constraints ($\mathcal{C}^{j2}$) corresponding to the variable region is composed of the same constraints, except for the first one, which is substituted with two: $\sum_{i=1}^{N} w_{ji} x_i \geq -\gamma$, and $\sum_{i=1}^{N} w_{ji} x_i \leq \gamma$. The set $\mathcal{C}^{j3}$ corresponding to the constant region 1 is defined similarly. The total number of C^{jr} sets is equal to $H\,R$.

The first step of the refining process consists in determining whether a set of constraints is valid, meaning that it has a nonempty solution set. For example, to verify the validity of C^{jr}, the set of constraints is passed to a linear programming solver with the first constraint as objective function, the optimization type - minimization (for $>$ or $\geq$) or maximization (for $<$ or $\leq$), and the rest of the inequalities as constraints. Any other constraint could have been chosen for verification of the existence of a solution. The initial set of variables has to be modified such that each variable is positive definite [17]. If the linear optimizer returns a solution which inside the limits of the first constraint, then the system of linear constraints C^{jr} is valid, otherwise C^{jr} is invalid.

The second step of the refining process adjusts the limits of all the constraints in a set $\mathcal{C}^{(.)}$. The goal is to reduce the region defined by the set of constraints. To exemplify this step we take a valid set of constraints C^{jr}. The limits of each constraint in the set $\mathcal{C}^{jr}$ are adjusted iteratively using the linear optmizer: at each step, a constraint becomes the objective function and a minimization is done if the inequality type of that constraint is $>$ or $\geq$ or a maximization, if the inequality type is $<$ or $\leq$. The right hand side of the constraint chosen as objective function is adjusted if the solution of the optimization is inside the limit (i.e. a minimum value greater than the right side coefficient for inequalities of type $>$ or $\geq$ or a maximum value smaller than the right side coefficient for inequalities of type $<$ or $\leq$). The procedure for adjusting the limits stops when none of the constraint limits undergoes any changes.

Next, we detail the process of intersecting sets of constraints. The purpose here is to eliminate redundant information between the sets of constraints that get combined. To show how this works, we take a combined set of constraints $\mathcal{C}^p = \mathcal{C}^{1r_1^p} \cap \mathcal{C}^{2r_2^p} \cap \ldots \cap \mathcal{C}^{Hr_H^p}$. The constraints are placed in the set $\mathcal{C}^p$ in an iterative process as follows: first, the set of constraints from the first hidden neuron ($\mathcal{C}^{1r_1^p}$) is added to $\mathcal{C}^p$, then each constraint from the subsequent sets $\mathcal{C}^j$, $j = 2 \ldots H$ is checked for similarity against all constraints already in $\mathcal{C}^p$. If a new constraint is similar to one already in $\mathcal{C}^p$ then the intersection between them is placed in the final set $\mathcal{C}^p$. If a new constraint is not smilar to any other in $\mathcal{C}^p$ then is simply added to the set $\mathcal{C}^p$. Two inequality constraints are similar if they have equal coefficients in the same input variables and similar inequality type. For example: $x_1 < 3.0$ and $x_1 \leq 2.0$ are similar and the intersection between them is $x_1 \leq 2.0$. In this way the number of constraints in the combined set $\mathcal{C}^p$ is kept at minimum. Once a set of constraints are intersected, the resulting set $\mathcal{C}^p$ is checked for validity and the limits refined in the same way as shown above.

The first requirement stated in the previous section - that the solution from any pair of constraints corresponding to two distinct linear models must be the empty set - is always true. The reason is that the set of constraints of each linear model ($\mathcal{C}^p$) is obtained by intersecting $\mathcal{C}^{jr}$ constraints for each hidden neuron. Each $\mathcal{C}^{jr}$ defines a linear region for a hidden neuron. All $\mathcal{C}^{jr}$ for the same hidden neuron j, but for distinct linear regions do not intersect: $\mathcal{C}^{jr_1} \cap \mathcal{C}^{jr_2} = \emptyset$, with $r_1,\ r_2 \in 1, \ldots R$ and $r_1 \neq r_2$ because the R linear regions defined by the function $g(x)$ do not overlap as well. The combined sets of constraints ($\mathcal{C}^p$) are obtained by intersecting the $\mathcal{C}^{jr}$ of all hidden neurons for a combination of the linear regions given in ($\mathcal{O}^p$). Two distinct sets of constraints $\mathcal{C}^{p_1}$ and $\mathcal{C}^{p_2}$ do not intersect because at least one hidden neuron must be in a different linear region.

5 Results

The method for linear model extraction is applied to model the frequency response of an analog circuit for different parasitic levels. The data is obtained using SPICE simulations of the analog circuit for a number of frequency (F) and parasitic (P) values: $F = 7$ and $P = 9$. Correspondingly, there are $F\,P$ gain values.

The two inputs - frequency (f) and parasitics (c)- are first scaled: $x_1 = \frac{log(f) - \mu(log(f))}{std(log(f))}$, with $\mu(.)$ the average and $std(.)$ the standard deviation. Similarly the parasitics input is also scaled (x_2). The output - the gain (g) - is translated (s_g) to fall inside the range of the sigmoidal activation function.

With these transformations, the data is split into a training (85%) and a test set (15%). A three layer neural network is trained and the best performance on the training set is obtained for a neural network with $H = 7$ hidden neurons. There are $I = 2$ input neurons one for freqency (x_1) and one for parasitics (x_2) and one output neuron - the gain. Figure 5 shows the simulation gain compared to the unscaled output of the trained neural network for four values of the parasitics. The network output approximates well the simulation data.

The trained network is pruned. From the initial set of weights between the input and hidden neurons ($(I+1)H = 21$) eight weights are eliminated. Because one of the hidden neurons gets disconnected completely from the input layer - all its weights were deleted - one hidden neuron is removed. The loss in accuracy for the pruned network is 4.19% of the original network mean square error.

The pruned network is linearized according to the procedure described in Methods section. First, the sets of constraints - $\mathcal{C}^{jr}$ - for all hidden neurons are found. Not all sets are valid, meaning that not all hidden neurons have output in all possible linear regions of $g(x)$. Second, the constraint sets $\mathcal{C}^p$ are obtained by intersecting the C^{jr} sets corresponding to combination of linear regions $\mathcal{O}^p$. From the initial number of combinations $R^H = 729$ only 18 are possible - all hidden neurons have solutions in the linear regions specified in each combination. Then, the 18 combinations are checked for validity and

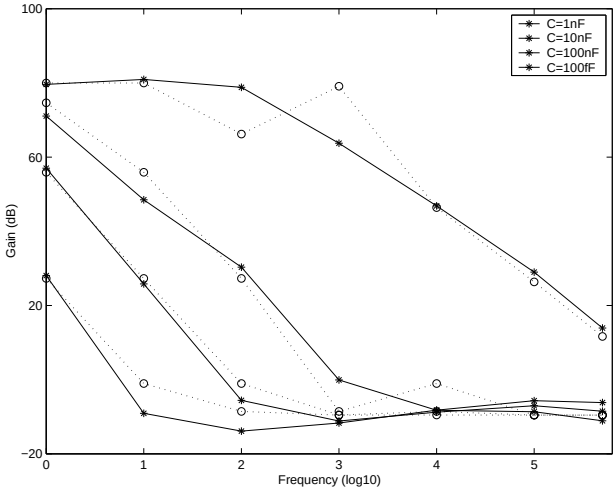

Figure 2: The gain frequency response of an analog circuit. With stars (*) are represented the network outputs and with circles (o) the SPICE values. The values of the parasitics for the four plots are: $C = 1nF$, $C = 10nF$, $C = 100nF$, $C = 100fF$. The values of the gain and are in the initial domain (unscaled).

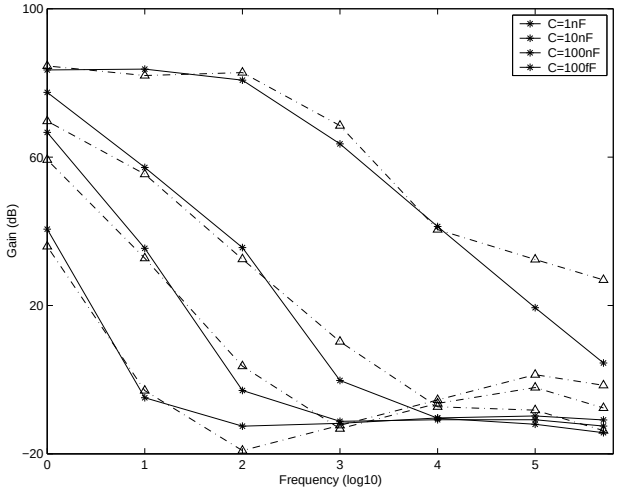

Figure 4: The gain frequency response of an analog circuit. With stars (*) are represented the outputs of the original neural network and with triangles (Δ) the outputs of the linear models. The values of the parasitics for the four plots are: $C = 1nF$, $C = 10nF$, $C = 100nF$, $C = 100fF$. The values of the gain and are in the initial domain (unscaled).

out of them 10 have a nonempty solution set. From the 10 valid combinations 6 generate a single linear model, corresponding to the variable region ($g_2(x)$) of the output neuron. The rest produce two linear models, one in the variable region and the other in one of the two remaining constant regions of the linear mapping $g(x)$.

For example, one of the linear models is: $x_o = 0.0201723x_1 + 0.127458x_2 + 1.17566$. The region where this output function is valid is defined by the set of 14 constraints with refined limits shown in Figure 3. In total, there are 14 linear models, 10 of them with a variable linear dependency in the input variables and 4 of them with a constant output.

Figure 4 shows the comparative results obtained with the initial neural network and with the linear models. The error between the nonlinear neural network model and the piecewise linear one is small in most of the points, but there are some areas where the error is bigger. These areas correspond to bigger linearization errors between the sigmoidal function and the linearized one.

6 Conclusions

A method was developed to extract piecewise linear models to approximate the nonlinear frequency response of analog circuits for different parasitics values. First, a neural network is trained to approximate the nonlinear mapping of the simulation points. A method of extracting piecewise linear models from the neural network is proposed, where the activation function of the neurons is approximated with a piecewise linear mapping. The number of generated linear models is reduced by checking the validity of each possible solution. The number of constraints that defines a region in the input space corresponding to a linear model is also reduced by eliminating redundant constraints. The extraction method was used to approximate the gain frequency plots of a simulated analog circuit for different parasitic values. The extracted piecewise linear model looses a little in accuracy compared to the initial neural network, but it gains in interpretability. One way to improve the accuracy of the linear models is to use instead of constant regions 0 and 1, variable regions where the output varies linearly with the input. Another possibility is to use the upper and lower limits of the weighted sum input to a hidden neuron to define a customized linear mapping for each neuron. It might be that the weighted sum input to a hidden neuron has a very restricted range, in which case the error for most input values will be large if we use a fix linearization function. For example, if most of the time the input to a hidden neuron falls around $\pm\gamma$ then the linearization error will be big. In this case it is better to find a more appropriate linearization mapping that reduces the error for that neuron.

References

[1] L. Carley *et al*, "Synthesis tools for mixed-signal ICs: Progress on Frontend and Backend strategies", *Proc. of DAC*, 1996, pp. 298-303.

[2] M.W. Craven and J.W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Proc. of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1994.

[3] W. Daems, G. Gielen, W. Sansen, "Simulation-Based Automatic Generation of Signomial and Posynomial Performance Models for Analog Integrated Circuits", *Proc. of the International Conference on Computer-Aided Design*, 2001.

[4] W. Daems, G. Gielen, W. Sansen, "An Efficient Optimization-Based Technique to Generate Posynomial Performance Models for Analog Integrated Circuits", *Proc. of the Design Automation Conference*, 2002, pp. 431-436.

$$\left\{\begin{array}{ccccccccc} 6.05483x_1+5.05147x_2 & < & 0.437622 & , & 6.05483x_1+5.05147x_2 & >= & -2.39329 \\ x_1 & >= & -0.587616 & , & x_1 & <= & 0.395531 \\ x_2 & >= & -0.387462 & , & x_2 & <= & 0.373985 \\ -1.22925x_1+2.98139x_2 & < & 1.40969 & , & -1.22925x_1+2.98139x_2 & >= & -1.64138 \\ -2.59169x_1-4.12291x_2 & < & 0.572377 & , & -2.59169x_1-4.12291x_2 & >= & -0.920589 \\ 0.547341x_1+1.21511x_2 & >= & -0.254318 & , & 0.547341x_1+1.21511x_2 & <= & 0.323216 \\ 0.108931x_1+0.688272x_2 & <= & 0.231289 & , & 0.108931x_1+0.688272x_2 & >= & -0.223593 \end{array}\right. \quad (5)$$

Figure 3: Constraint set defining the validity region for a linear model

[5] A. Doboli, "Specification and Design-space Exploration for High-level Synthesis of Analog and Mixed-signal Systems", *Ph.D. Thesis, University of Cincinnati*, 2000.

[6] N. R. Dhanwada, A. Nunez, R. Vemuri, "Hierarchical Constraint Transformation using Directed Interval Search for Analog System Synthesis", *Proc. of DATE*, 1999, pp. 328-335.

[7] L.M. Fu. Rule generation from neural networks. *IEEE Trans. Syst., Man., Cybern.*, 28:1114–1124, 1994.

[8] G.Gielen *et al*, "Analog circuit design optimization based on symbolic simulator and simulated annealing", *IEEE JSSC*, Vol.25, 1990, pp.703-711.

[9] G. Gielen, R. Rutenbar, "Computer Aided Design of Analog and Mixed-signal Integrated Circuits", *Proc. of IEEE*, vol 88, No 12, Dec 2000, pp. 1825-1852.

[10] R. Harjani, J. Shao, "Feasibility and Performance Region Modeling of Analog and Digital Circuits", *Analog Integrated Circuits and Signal Procesing*, Kluwer, 1996.

[11] M. Krasnicki, R. Phelps, R. Rutenbar, R. Carley, "MAELSTROM: Efficient Simulation-Based Synthesis for Custom Analog Cells", *Proc. of the 36th ACM/IEEE Design Automation Conference*, 1999, pp.945-950.

[12] D. Leenaerts, W. van Bokhoven, "Piecewise Linear Modeling and Analysis", *Kluwer*, 1998.

[13] K. Laker, W. Sansen, "Design of Analog Integrated Circuits and Systems", *McGraw Hill*, 1994.

[14] H. Liu, A. Singhee, R. Rutenbar, R. Carley, "Remembrance of Circuit Past: Macromodeling by Data Mining in Large Analog Design Spaces", *Proc. of the 39-th Design Automation Conference*, 2002, pp. 437-442.

[15] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.l. McClelland, editors, *Parallel Distributed Processing*, volume I+II. MIT Press, 1986.

[16] R. Setiono, W.K. Leow, and J.M. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Trans. Neural Networks*, 13(3):564–577, 2002.

[17] P.R. Thie. *An introduction to linear programming and game theory*. John Wiley & Sons, 1988.

[18] S. Thrun. Extracting rules from artificial neural networks with distributed representations. In D. Touretzky G. Tesauro and T. Leen, editors, *Advances in Neural Proceesing Systems*, volume 7. MIT Press, 1995.

[19] A.B. Tickle, R. Andrews, M. Golea, and J. Diederich. The truth will come to light: directions and challenges in extracting knowledge embedded within trained artificial neural networks. *IEEE Trans. Neural Networks*, 9(6):1057–1068, 1998.

[20] P. Vancorenland *et al*, "A Layout-Aware Synthesis Methodology for RF Circuits", *Proc. of ICCAD*, 2001, pp. 358-362.