Piecwise-Linear Modeling of Analog Circuits using Trained Feed-Forward Neural Networks and Adaptive Clustering of Hidden Neurons

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

Gaurav Gothoskar and Alex Doboli
VLSI Systems Design Laboratory
Electrical and Computer Engineering Department
SUNY Stony Brook, Stony Brook, NY, 11794-2350
Email: {gaurav, adoboli}@ece.sunysb.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 - an amplifier circuit widely used in filters and A/D converters - for which models for gain and bandwidth were automatically generated. As experiments show, the obtained models have a simple form that accurately fits the sampled points and the behavior of the trained neural networks. 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 [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 [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 simulation 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 simulation and synthesis by guiding the search towards attractive solution space regions [6] [20]. 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].

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 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] [2] [3]. 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 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 training of 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 and weights with insignificant contributions. Then, the sigmoidal activation function of each hidden neuron is approximated with a piecewise linear function with a variable number of segments. The number of segments, its limits and the linear approximation on each segment are automatically determined by a clustering algorithm. Finally, the piece-
wise linear functions for the hidden neurons are composed together to generate the piecewise linear functions of the model output. The regions were each linear output model is active are found by iteratively solving a linear system of inequalities and adjusting its limits.

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 the 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 model would make the interpretation of the solution much easier. Several methods have been proposed for extracting rules from trained neural networks [19, 7, 1, 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.

Recently new techniques have been proposed to extract linear models for regression problems [16, 4]. The method proposed in this paper relates to these methods. The main steps of the extraction process are as follows: A 2-layer neural network is first trained and pruned. In [16] the activation function of each hidden neuron is approximated with a fixed set of piecewise linear functions - three or five. The input space is then split into a set of regions for each hidden neuron so that an input point in one of the regions activates one piecewise linear function. For each non-empty intersection of input regions - one for each hidden neuron - the output activation function can be expressed as a linear combination of the input variables. The coefficients of each linear model are a function of the network weights. The results obtained with this method are good, and show a decrease in the approximation error by increasing the number of piecewise linear regions per activation function.

The disadvantage of increasing the number of linear regions - either three or five - is fixed for all hidden neurons. But each hidden neuron can be active in a different region of the input space. The main steps of the linear model extraction method are as follows:

1. The valid regions of any pair of linear models must not intersect in any point in the input space: \( \mathcal{C}_p \cap \mathcal{C}_q = \emptyset \), for \( p \neq q \).

2. The set of constraints in \( \mathcal{C}_l \) is minimal. By removing any constraint, the valid region for model \( l \) changes.

4. Linear Model Extraction Method

The neural network considered here is a three layer feed-forward network. There are \( N \) input neurons in the input layer \( I \), \( H \) hidden neurons in the hidden layer \( H \) and \( O \) output neurons in the output layer \( O \). Without restricting the generality of the extraction method, the number of output neurons can be considered as equal to one. The weight matrix between the input and the hidden layer is \( W^{IH} = \{w_{ji}, j = 1 \ldots H, i = 1 \ldots N + 1\} \), with \( w_{ji} \) the weight of the connection between input neuron \( i \) and hidden neuron \( j \). The input layer and the hidden layer are both augmented with a bias neuron with a constant output of one. The weight matrix between the hidden and the output layer is \( W^{HO} = \{w_{kj}, k = 1 \ldots O, j = 1 \ldots H + 1\} \) with \( w_{kj} \) the strength of the connection between output neuron \( j \) and hidden neuron \( k \).

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

\[
\begin{align*}
h_j & = \sum_{i=1}^{N} w_{ji} x_i, \\
h_k & = \sum_{j=1}^{H} w_{kj} y_j, \\
\end{align*}
\]

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

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

1. Training and pruning the neural network. A neural network is first trained and then pruned. In this process, the number of hidden neurons is chosen such that small training and testing errors are obtained. This step is described in Section 4.1.
2. Linearization of the activation function of the hidden neurons. The values of the activation function of each hidden neuron are first clustered into a number of sets, each corresponding to a linear segment. The clustering algorithm determines: the number of clusters, its limits and the equation of the linear segment for each cluster - the slope and intercept of the linear segment that passes through the limits of a cluster. The clustering algorithm is presented in Section 4.2.

3. Extraction of the linear models. This step consists in finding the non-empty regions in the input space where combinations of linear regions in the hidden neurons are valid. For each such a region the linear model of the network output is expressed in terms of the network weights. This step is detailed in Section 4.3.

4.1 Training and Pruning the Neural Network

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

The pruning repeatedly attempts to remove the smallest weight from the remaining weights until a stopping criteria is satisfied. The criteria for removing a weight is the reduction in accuracy on both training and validation data points. 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 training and validation data of the pruned neural network at step \( r \). Pruning is done until the total accuracy reduction is kept under a fixed threshold (less than 10% from the error obtained with the original trained network). 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 a necessary step in order to obtain a small number of linear output models. After pruning, the network is retrained by fixing the pruned weights to 0, and allowing only the remaining weights to adapt.

4.2 Clustering Algorithm

The clustering algorithm starts with the retrained pruned network and its purpose is to approximate the nonlinear sigmoidal activation function of each hidden neuron with a group of piecewise linear functions. The number of linear regions as well as its limits are determined in the clustering process. The idea is to group input points - sampled from the input region of interest - that correspond to the same slope of the activation function. The main feature of the clustering algorithm is the stopping criteria which allows the algorithm to stop when an optimal number of clusters is found.

The first step consists in finding the activation values of each hidden neuron by using all the available input data points \((x_n)\) to evaluate the weighted sum \( h_j \) (relation (1)) and the output \( y_j \) given by the sigmoidal function \((g(h_j(x_n)))\).

Then, the output points \((y_j)\) are sorted in ascending order and only the distinct points are selected for clustering - \( N_c \). The clustering algorithm is a modified agglomerative clustering technique [5]. First, a linear segment passing through each pair of consecutive output values is defined by computing its slope and intercept. Then the distance between two such segments is defined as the cosine of the angle between them: \( d_{cosine}(c_{r1}, c_{r2}) = \frac{x_{y1} - y_{y2}}{x_{y1}^2 + y_{y2}^2} \), where \( c_{r(i)} \) are the indices of two segments or clusters, \( x_{y1}, y_{y2} \) the coordinates of the vector that passes through the only two points of the initial clusters.

The clustering starts with a number of clusters equal to the number of linear segments between consecutive output points. It then iteratively attempts at merging the closest pair of clusters together until a stopping criteria is reached. The criteria to stop merging is:

\[
J(t) = \frac{N_c(t)}{N_c - 1} + \frac{1}{N_c} \sum_{i=1}^{N_c} |y_i(x_n) - y_j(x_n)|
\]

where \( N_c(t) \) is the number of clusters at step \( t \), \( y_i(x_n) \) is the linear output for input point \( x_n \), \( y_j(x_n) \) is the original sigmoidal output. The first term of relation (2) penalizes a large number of clusters, while the second term penalizes a large linearization error. At the beginning of the clustering, the linearization error is zero and the penalty for the number of clusters is one: \( J(0) = 1 \). As merging of closest clusters continues, the first term goes down, while the second term goes up. Therefore, at the beginning, the values of the criterion function \( J(t) \) decrease while the penalty for a large number of clusters dominates compared to the linearization error. As merging progresses, the linearization error starts to become more important in the sum, and at one point the values of \( J(t) \) go up. At that moment the clustering stops. The resulting number of clusters determines the number of linear regions for hidden neuron \( j \).

The linear output \( y_j(x_n) \) is computed as follows. Each cluster has two limiting points, and a linear segment that passes through them. The slope \((a_j)\) and intercept \((b_j)\) of the linear segment which goes through the limits of the \( c_i\)th cluster is computed. Then the linear output of a point \( x_n \) - within the limits of the cluster \( c_i \) - is: \( y_j(x_n) = a_j x_n + b_j \).

The closest pair of clusters at each step in the algorithm is defined as follows. The only pairs of clusters eligible for merging are the ones that have a common limit point. Then the distance between any two adjacent clusters \( c_{r1}\) and \( c_{r2}\) is measured as:

\[
d(c_{r1}, c_{r2}) = \max_{k_1, k_2} \left\{ d_{cosine}(k_1, k_2) \right\} + \frac{1}{n_{r1} + n_{r2}} \sum_{k \in \cup r_{r1}, r_{r2}} |y_i(x_k) - y_j(x_k)|
\]

where \( k_i \) is the index of a segment defined at step \( 0 \) of the algorithm, which is now part of either \( c_{r1}\) or \( c_{r2}\) clusters at step \( t \). The first term - the maximum cosine distance between any pairs of segments in the two clusters - is a measure of how closely oriented are the segments in the two clusters - while the second term is the average of the absolute linearization error that would be introduced by merging clusters \( c_{r1}\) and \( c_{r2}\).

The pair of clusters with the minimum distance \( d(c_{r1}, c_{r2}) \) is merged.

After each merging the value of the criterion function \( J(t) \) is re-evaluated, and if it is bigger than the value at the previous step \( J(t-1) \) the algorithm stops. The results of the algorithm are: the number of clusters - \( N_c \) - for the activation function of the hidden neuron \( j \), the coordinates in the input space of the upper and lower bounds of each cluster and the slope and intercept of each cluster. The slope and intercept are obtained from the linear segment that goes through the limiting points of each cluster. The resulting linear segments cover all the activation values of the hidden neuron and any two adjacent segments overlap only in one point.

The end of the second step of the linear model extraction method consists in expressing the limits in the input space where each linear region of a hidden neurons is active. The limits are specified as a set of linear constraints. For example, for neuron \( j \), linear region \( r \), the set of constraints \( C^{j1} \) is:

\[
C^{j1} = \begin{cases}
\sum_{i=1}^{N} w_{ij}x_i \leq M_1, & x_1 \geq m_1 \\
\sum_{i=1}^{N} w_{ij}x_i \geq M_1, & x_1 \leq m_1 \\
x_N \leq M_N, & x_N \geq m_N
\end{cases}
\]
where \( m_r \) and \( M_r \) are the minimum and maximum values of the linear function in region \( r \). \( m_i \) and \( M_i \) are the limits of each input variable as they result from the clustering process.

### 4.3. Extraction of the linear models

Once the activation function of the hidden neurons is approximated by a piecewise linear mapping, the next step consists in finding the valid combinations of linear regions for the hidden neurons. Such a combination is given by a set of indices, where each index represents the active linear region of a hidden neuron: \( C^p = \{r^1, r^2, \ldots, r^p\} \in \{1, 2, \ldots, N, (j)\} \) and \( p = 1 \ldots n, (1), (2) \ldots N, (H) \), where \( N, (j) \) is the number of linear regions of hidden neuron \( j \). Each such combination is defined by a region in the input space given by the intersection of the sets of constraints: \( C^p = C^{o_1} \cap C^{o_2} \ldots \cap C^{o_n} \). The valid combinations are the ones for which the set of constraints in \( C^p \) defines a non-empty region in the input space.

The constraints are placed in the set \( C^p \) in an iterative process as follows: first, the set of constraints from the first hidden neuron \( c^{t_1} \) is added to \( C^p \), then each constraint from the subsequent sets \( c^{t_2}, j = 2 \ldots H \) is checked for similarity against all constraints already in \( C^p \). If a new constraint is similar to one already in \( C^p \) then the intersection between them is placed in \( C^p \), otherwise the new constraint is added to \( C^p \). Two inequality constraints are similar if they have equal coefficients in the same input variables and the same inequality type. For example: \( x_1 \leq 3.0 \) 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 \( C^p \) is minimal.

Next, the sets of constraints \( C^p \) are checked for validity and their limits refined. The validity of \( C^p \) is checked by a linear programming solver with the first constraint chosen as objective function, the optimization type - minimization (for \( \geq \)) or maximization (for \( \leq \)), and the rest of the inequalities as constraints. If the linear solver returns an acceptable solution then the input region defined by the \( C^p \) is non-empty, and therefore the combination is valid.

The goal of refining the limits of the constraints in each valid set \( C^p \) is to eliminate redundancy in the constraint limits. The limits of each constraint in the set \( C^p \) are adjusted iteratively using the linear optimizer: at each step, a constraint becomes the objective function and a minimization/maximization is done depending on the inequality type of the constraint, with the rest of the inequalities as constraints. The limit of the optimized constraint is adjusted if the solution returned by the solver is more restrictive. The procedure for adjusting the limits stops when none of the constraint limits undergoes any changes.

For each valid combination region defined by \( C^p \) the output of the network is expressed as a linear combination in the input variables: \( y^i = a^i_1 x_1 + a^i_2 x_2 + \ldots + a^i_n x_n + a^i_{n+1} \). The coefficients \( a^i_k \) are functions of the weights of the network and of the slopes and intercepts of the linear regions of the hidden neurons determined in the clustering algorithm. The set of linear models defined by coefficients \( a^i_k \), together with the set of constraints of the valid combinations \( C^p \) represent the result of the extraction method.

The first requirement stated in the Section 3 - that the intersection between any two sets of constraints \( C^{p_1} \) and \( C^{p_2} \) be the empty set - is always true. The reason is that the set of constraints of each linear model \( C^p \) is obtained by intersecting the \( C^{o_1} \) constraints for each hidden neuron. Each \( C^{o_1} \) corresponds to a linear region of a hidden neuron. Any two different \( C^{o_1} \)'s of the same hidden neuron do not intersect because the \( N, (j) \) linear regions defined by the clustering algorithm overlap only at the margins. Two distinct sets of constraints \( C^{p_1} \) and \( C^{p_2} \) 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 amplitude frequency response of an analog transconductance amplifier (OTA) [13] for different parasitic levels. The data is obtained using SPICE simulations of the analog circuit sampled in a large number of frequency and parasitic values. The two inputs - frequency \( (f) \) and parasitics \( (c) \) - and the output - the gain - are first normalized.

A three layer neural network with \( I = 2 \) inputs and \( H = 7 \) hidden neurons is trained such that the performance on both training and testing data are very good. A larger number of hidden neurons did not improve the approximation. The trained network is then 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 it is removed and the network retrained. Figure 1 shows the SPICE values of the gain compared to the unscaled output of the pruned neural network for seven parasitics values. The network output approximates very well the real values.

The clustering algorithm is then applied to each hidden neuron. Two of the hidden neurons have constant outputs given by the bias weight - the weights to the input variables were all pruned. The rest of the hidden neurons are clustered into 6, 9, 3 and 8 clusters respectively. Figure 2 shows the linear segments obtained by clustering for the hidden neuron with 6 clusters.

From a total of 1296 combinations of linear regions of the hidden neurons, only 136 have a non-empty solution set. For each valid combination, a linear model of the network output is computed. The result of the piecewise linear extraction method is presented in Figure 3. The dotted plot represents the piecewise linear model output, while the line represents the true values. It can be seen that the piecewise linear approximation is very accurate. Previous approaches to extract linear models from trained neural networks [16] and [4] using a fixed number of segments for each hidden neuron do not have the same accuracy. For example, Figure 4 shows the results of the linear extraction method presented in [4] on the same data set as in Figure 3. In this case the activation function of each

![Figure 1: The gain frequency response. The pruned neural network output is represented with dots and the SPICE values with lines. Each curve corresponds to a different value of the parasitics.](image-url)
hidden neuron is split into three linear regions, where the output is either zero, linear dependence or one. The accuracy of the extracted model is much worse than that obtained with the present method.

Similar results were obtained for modeling an operational amplifier circuit [13], and a high-frequency OTA as well.

We also developed piecewise-linear models for the current-voltage relationships at the terminals of popular analog building blocks such as differential stages, current mirrors, output stages etc. For example, the output stage was modeled as the dependence of the output current to the input voltages and the widths of the two transistors. Then, the simulation behavior of complex analog circuits were obtained using the extracted, piecewise linear models. The piecewise linear models were composed by using Kirchhoff’s and Ohm’s laws to obtain the current-voltage behavior at the terminals of the complex circuits. This behavior is useful for efficient AC and time-domain simulation. Due to its event-driven nature, this simulation is much more efficient than continuous time simulation. This is very important for analog circuit synthesis as it is well known that the large simulation time is the bottleneck of the synthesis cycle.

6. Conclusions

A method was developed to extract piecewise linear models to approximate the nonlinear mapping encoded in a trained neural network. The application domain is piecewise linear modeling of analog circuits. The model generation techniques includes (1) training of a neural network using the backpropagation method, (2) neuron pruning to eliminate neurons with insignificant contributions, (3) sigmoidal activation function approximation of hidden neurons as a piecewise linear function, and (4) hidden neuron composition to generate the piecewise linear models. The method automatically determines the best number of linear regions to approximate each hidden neuron activation function by using a modified clustering method. Experiments showed that the adaptive clustering significantly improves the accuracy of extracted models as compared to other extraction approaches. In general, the extraction method was shown to approximate very well the measured gain-frequency plots of an analog circuit simulated for different parasitic val-
ues.

References


