Machine Learning Techniques for System Modelling

José António Vieira Veiga de Faria

Instituto Superior Técnico, Universidade de Lisboa.
jose.veiga.de.faria@ist.utl.pt

June 8, 2018

In electronics, anything that can be considered as a unit (simple devices, groups of devices, sub-circuits) can potentially be represented by a simple black-box macromodel. The purpose of this work is to research and analyze the use of learning methods as general modelling frameworks, whereby a model as simple and as accurate as possible can be generated. Support vector machines (single, multi-output) and neural networks were explored in depth and used to develop a robust modelling framework that can generate a black-box model for a given circuit's unit. A model evaluation engine where a SPICE component can be created from the model and used (as many times as necessary) was also produced. The framework was tested in multiple circuits beginning with simple linear circuits, progressing to small nonlinear circuits and ending with nonlinear circuits with a considerable number of components. Good generalization to data not used in training was achieved as well as improvement in the simulation time.

1. Introduction

Circuit simulation is intended to reproduce the behaviour of a given circuit. This process is nowadays an essential part of circuit design because it provides a means to predict and check circuit behavior before any costly fabrication is conducted. It is also useful for exploring alternatives and optimizing designs according to predefined objectives.

Simulation is not possible without a model of the circuit. This model is often given by a set of equations relating the variables of interest in the circuit or device under question. One of the major difficulties nowadays is that given the diminishing feature sizes of devices, it is often no longer possible to write simple behavioral equations based only on physical laws. Device models, such as transistors, are nowadays described by extremely complex models that involve tens or hundreds of parameters, many of them heuristic and with no physical interpretation. Values for such parameters are provided by circuit manufacturers that have painstakingly fitted what are often cumbersome parametric descriptions with the help of data obtained through measurements and observations. Often such models are not reusable when one moves from one technology to another or from one manufacturer to another, creating an additional problem for model generation.

The reasons above are one of the main motivations for this work, which is to develop modeling techniques that are somewhat independent from technological issues. Another motivation is the recent renewed interest in machine learning techniques which are now seeing widespread application. Such black-box models have the advantage that they can be computed for simple devices, groups of devices, sub-circuits or modules of a circuit. Anything that can be thought of as a unit that could be reused in multiple designs, simulated, fitted and a model generated. Like all black-box models, neither the form of the models nor its parameters allow much insight into the physical behavior of the subcircuit, but that can be acceptable if one thinks of the hierarchical fashion in which circuits are built. By encapsulating details of the circuit structure into the model its evaluation can be done more efficiently, which allows one to perform additional simulations while trying to optimize designs. In the case a certain circuit is not physically available this technique is also useful as its behaviour may later need to be recreated for some purpose.

The main contributions of this thesis are threefold: (1) the development of a framework (optimized, robust) to automatically create black-box models of various circuit's units and to study the effect variations in the regressors' parameters have in their performance; (2) the implementation of multioutput SV machines that might be advantageous in terms of model complexity or accuracy; (3) the creation of a model evaluation engine that allows circuit components to be created using the black-box models obtained and to be verified in the context of a circuit simulator.

It is important to mention that, in spite of being suitable for static circuits (circuits where for a given input the output is always the same), support vector regressors (SVR) and neural network regression techniques as implemented in this work are intrinsically limited in their ability to provide a full description of the behaviour of dynamic circuits. This is because the circuit's response may depend not only on changes on the input but potentially also on the internal state of the circuit or on the input wave frequency. Nevertheless,

the range of applicability of these types of approaches is sufficiently large to merit attention.

2. The simulation loop

The circuit simulator used in this work was Ngspice [1], an open source version of the SPICE program. SPICE can do DC, transient and AC analysis. At the core of its engine is the computation of the voltage at any circuit's node. To achieve this, Kirchhoff circuit equations need to be written, device models provided and the equations solved.

Let's assume each circuit's component has an equation that allows us to express its current (i) as a function of its voltage (V). This constitutive equation is represented by $i = g(V)$ which can be linear or non linear (in the general case it is non linear). Applying the Kirchhoff's current law (KCL) to a circuit node i (Figure 1 used as an example) gives a nonlinear equation.

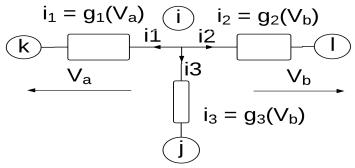

Figure 1: *Example node.*

$$\begin{aligned} &i_1 + i_2 + i_3 = 0 \\ &g_1(V_a) + g_2(V_b) + g_3(V_c) = 0 \\ &g_1(V_i - V_k) + g_2(V_i - V_l) + g_3(V_i - V_j) = 0 \end{aligned} \tag{1}$$

Putting together the KCL equations of all nodes results in a system of non linear equations which is represented by:

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{V}) = 0, \quad & \boldsymbol{F} : \mathbb{R}^N \to \mathbb{R}^N \\ & \boldsymbol{V} = [V_1 ... V_i ... V_j ... V_n]^T \\ & \textit{and} \quad \boldsymbol{F} = [F_1 ... F_i ... F_j ... F_n]^T \end{aligned} \tag{2}$$

where F_i is the KCL equation of node i.

If the initial assumption is not verified which means there are devices where the current cannot be written explicitly as a function of the voltage or its derivative (e.g. inductors or voltage-controlled voltage sources) additional equations must be added. This is accomplished by including such currents as new variables and adding the respective devices constitutive equations. These procedures are termed the Modified Nodal Analysis method (MNA).

In the case of a transient analysis, components such as inductors or capacitors introduce differential terms in the KCL equations. SPICE solves the ODE (ordinary differential equations) system by using temporal discretization. The differential equations are discretized in time leading to a system of approximate algebraic equations. One way to achieve this is, for instance, to use the backward Euler method in equation (3).

$$\begin{aligned} & x_{n+1} = x_n + h\frac{dx_{n+1}}{dt} \implies \frac{dx_{n+1}}{dt} = \frac{x_{n+1} - x_n}{h} \\ & \text{where} \quad h = \Delta t = t_{n+1} - t_n \end{aligned} \tag{3}$$

The solution of the MNA system provides the voltages in all nodes. In the general case this solution cannot be found using an analytical approach. Systems of non-linear equations are usually solved using iterative methods. SPICE uses an iterative method to get a good approximation of the solution, the Newton-Raphson method.

Approximating $F(\boldsymbol{V})$ with its Taylor series about a vector $\boldsymbol{V}^k$ we get

$$F(\boldsymbol{V}) \approx F(\boldsymbol{V}^k) + J_F(\boldsymbol{V}^k)(\boldsymbol{V} - \boldsymbol{V}^k) \tag{4}$$

where $\boldsymbol{J}_F$ is the jacobian matrix of $\boldsymbol{F}$ (the matrix of all first-order partial derivatives of the vector-valued function $\boldsymbol{F}$). The value of iteration $k+1$ according to the Newton method is the vector $\boldsymbol{V}^{k+1}$ that satisfies

$$\begin{aligned} F(\boldsymbol{V}^k) + J_F(\boldsymbol{V}^k)(\boldsymbol{V}^{k+1} - \boldsymbol{V}^k) = 0 \\ \implies J_F(\boldsymbol{V}^k)\Delta\boldsymbol{V}^{k+1} = -F(\boldsymbol{V}^k) \end{aligned} \tag{5}$$

where $\Delta\boldsymbol{V}^{k+1} = \boldsymbol{V}^{k+1} - \boldsymbol{V}^k$. This is a linear system that can be solved using appropriate methods, like direct methods such as Gauss' method, or iterative methods such as Gauss-Jacobi, Gauss-Seidel or conjugate-gradiente type algorithms.

The algorithm to solve the circuit can be summarized as:

$$\begin{aligned} & \boldsymbol{V}^0 = \text{Initial Guess}, k = 0 \\ & \text{Repeat}\{ \\ & \quad \text{Compute} \quad F(\boldsymbol{V}^k), J_F(\boldsymbol{V}^k) \\ & \quad \text{Solve} \quad J_F(\boldsymbol{V}^k)(\boldsymbol{V}^{k+1} - \boldsymbol{V}^k) = -F(\boldsymbol{V}^k) \quad \text{for} \quad \boldsymbol{V}^{k+1} \\ & \quad k = k + 1 \\ & \} \quad \text{Until} \quad \boldsymbol{V}^{k+1} - \boldsymbol{V}^k, f(\boldsymbol{V}^{k+1}) \quad \text{small enough} \end{aligned} \tag{6}$$

This algorithm is executed only once in a DC analysis and for each time point in a transient one.

2.1. Computational costs

The simulation of the circuit requires computing the solution that satisfies the set of equations at each time-point.

Since these equations are nonlinear this implies, at each time point, to iterate until convergence. At each iteration we need to compute the Jacobian matrix and the F vector and then solve the linear system. For that reason there are two costs: the cost of evaluating the device's equations in order to compute J and F and the cost of solving the system. To favor a small simulation time the devices' mathematical models should be simple (evaluation less costly as possible) and the system should have small dimensions.

The use of a black-box model to replace modules of circuits means the circuit will have less nodes and as a consequence matrix J and vector F will have less entries to be computed. Also since the mathematical model is supposed to be simple its evaluation will not take a long time.

3. Regression analysis

In order to produce a black-box model of a circuit' subsection the relationship between the subsection's inputs and outputs of interest must be estimated. In the general case this is what regression analysis does, it estimates the relationship between a dependent variables and one or more independent variables. The dependent variables represent the outputs whose variation is being studied (in our case subsection's node voltages or branch currents). The independent variables represent inputs, or in the experimental setting, the variables controlled by the experimenter (in our case the subsection's inputs).

There are many techniques for modelling this relationship and they all estimate a function of the independent variables named the regression function (see (7)). In equation (7) y represents the dependent variable, $\vec{x}$ represents the independent variables and $\vec{\beta}$ the model parameters.

$$y \approx f(\vec{x}, \vec{\beta}) \tag{7}$$

In parametric regression this function is defined in terms of a finite number of unknown parameters that are estimated from the data (ex: linear, polynomial regression). In non-parametric regression the function does not have a predetermined form but is constructed according to information derived from the data (ex: support vector machines, neural networks). Since we want to model subsections of circuits without having an idea of what the relationship between variables might look like, non-parametric regression is especially appealing because it uses data to determine the regression function form and we are not required to put forward a certain type of function. Support vector machines (with RBF or polynomial kernels) and multilayer perceptrons were the non-parametric regressors studied and implemented.

4. Multi-output SV machines

The standard SV regressor can only model output-input relationships for a single output at a time. Since circuits/subsections of circuits may have multiple inputs and outputs this approach gives as many black-boxes as the number of outputs. A research was conducted to see if there were multioutput approaches that allowed all the relationships to be modelled in a single model (which might have a cheaper evaluation). This is motivated by the fact that simulation time is linked to the complexity of the black-box models' equations which should be as simple as possible.

The possible less expensive evaluation relates to the fact that, although the problem of Multioutput Support Vector Regression is approachable as an aggregation of independent single output ε regressions $f_i : R^n -> Y$, it is likely that there will only be coincidental commonality between the SVs associated with the different outputs. The cost of this is incurred when making a prediction because the greater the number of non-common support vectors is, the greater the number of kernel computations needed when making a prediction. This cost may be negligible when kernel computations are inexpensive, however as they become more expensive (i.e. large number of input space dimensions) these costs may become significant.

The following three multioutput algorithms were implemented: Vector-Valued Support Vector Regression (VV-SVR) [2], Multi-output support vector regression (MSVR) [3] and Multi-output least-squares support vector regression machines (MLS) [4]. The last regressor mentioned does not enable sparsity (i.e dropping SVs which have meaningless contribution to the output) and so it cannot reduce the simulation time, but it is still interesting since it might generate more precise models than its counterparts.

5. Model generation engine

The model generation framework requires and uses data points obtained through simulation, using Ngspice, of electrical networks. The data representing the time or frequency evolution of the relevant output nodes is modeled against the data points from the relevant inputs.

The three multi-output support vector algorithms described in section 4 were implemented in R [5]: VV-SVR (with norm-1), MSVR and MLS. The svm function of the

R package e1071 was used as the standard support vector regressor.

5.1. Data generation

Data was generated in two different ways. In the first one the circuit's input voltage took the form of a chosen function (ex: sinusoidal, squared waveform), the time response of the system was obtained using a transient analysis and samples of the circuit's output were collected. In the second one a DC sweep between a minimum and maximum input voltage was done (with a chosen step size) and samples of the circuit's output collected. In both cases the samples of output voltages paired with the correspondent input voltages were used as data to train the regressor.

The regressor uses the training set to find values for the model parameters (to 'train' the model). To have an unbiased estimate of the model's performance in the universe, the performance is not assessed in the training set, but rather in a test set that is independent from the training set. The test set can be a percentage of training data (the data is randomly selected) or generated from a new ngspice file.

5.2. Input and output normalization

The radial basis function kernel (RBF) can be interpreted as a measure of similarity between two vectors. This function has image $]0, 1]$ and the closer its output value is to 1 the more similar two vectors are and vice-versa.

Suppose there is an example circuit with two inputs. The first input has range (0.1 to 0.8) and the second one has range (3000 to 50000). When computing the RBF kernel value of the input vectors, for instance, (0.1, 4000) and (0.8, 4000) with a third input vector the two values will hardly have any difference. For the support vector machine these vectors will be very similar and we just changed input 1 from its minimum to maximum value. The machine will be much more sensitive to input 2 than to input 1 which is not acceptable as it must be equally sensitive to each of the inputs.

In this work we decided to use the 'soft' normalization presented in [6]. For each input i the mean and standard deviation in the training set are computed. Then each component i of each training vector j is normalized according to equation (8).

$$\vec{x}_{j_{i_{norm}}} = \frac{\vec{x}_{j_i} - mean_i}{2 * sd_i} \tag{8}$$

Output normalization is also useful. In support vector machines the prediction formula for one output is:

$$Pred(\vec{x}) = \sum_{i=1}^{l} \alpha_i k(\vec{x_i}, \vec{x}) + b \tag{9}$$

The alpha coefficients in the standard SV regression have to satisfy the inequality $0 \leq \alpha_i \leq C$ and RBF kernel value is always less or equal to 1. If the output values are sufficiently large the sum used for prediction might never reach them. As an example let's say a model has 10 support vectors (10 non zero alpha values) and C = 16. This will give a prediction of maximum value 160 V. For that reason outputs were also scaled although for the majority of electronic circuits it may not be necessary because output voltage is usually not very high.

5.3. Parameters

The support vector regressors used have as parameters ε, C, σ, %T (explained next) and λ. The parameter λ is only used in the MLS which also does not require the parameter ε. To see how varying each parameter influences the SVRs' error and number of support vectors a standard value was chosen for each parameter. Then, a sweep between chosen values was done for each of the parameters while keeping the others fixed. Standard and sweep values decided are explained below.

Epsilon – ε

Ideally predicted output vectors should all fall inside the ε-insensitive region which means ε represents the maximum error we want to tolerate. One question that arises is how to choose this maximum error. Considering the circuit's inputs are sinusoidal the input standard deviation will be approximately equal to the sinusoide's root mean square (rms) value $\frac{Amplitude}{\sqrt{(2)}}$ (equal if number of training points are infinite). Dividing every input by 2 times the standard deviation, a point with rms value will have a normalized value of 0.5. If we chose ε = 0.001 we are saying that we want errors to be $0.5/0.001 = 500$ times lower than the rms of the sinusoide. If the sinusoide has amplitude of 5V the errors should not exceed $3.54/500 = 7mV$.

The standard value chosen for parameter was $1 * 10^{-3}$ and epson the sweep was done in values $5 * 10^{-2}$, $1 * 10^{-2}$, $5 * 10^{-3}$, $1 * 10^{-3}$ and $1 * 10^{-4}$.

Regularization C

Fitted models should have good accuracy in the training vectors and also generalize well to unseen vectors/data.

Unfortunately, excellent performance in these two areas (at the same time) is typically impossible to achieve. In order to improve generalization, one must sacrifice accuracy in the training set and vice-versa. The aim of regularization techniques is to improve generalization without substantially increasing the errors in the training set (avoid underfitting) by favoring the production of smooth functions.

In the particular case of support vector regressors regularization is implemented through the parameter 'C' which controls the bias-variance tradeoff. The standard value of C was chosen based on the C values in [3] and [2]. The C values were varied using powers of 10. As the standard a value of 100 was chosen and for sweep values $10^{-4:7}$.

***Sigma** – σ*

Sigma (σ) represents the standard deviation of the RBF kernel (Gaussian function) which is used as a similarity measure between two points. A large sigma value defines a Gaussian function with a large variance. Considering a fixed vector x_i the larger this value is the larger the region of vectors that are considered similar to x_i. On the other hand a small sigma value defines a Gaussian function with a small variance and two vectors are considered similar only if they are close to each other.

Below is a one-dimensional example of gaussians with sigma = 2 and sigma = 0.5. The x axis contains the value of the difference of vector $\vec{x}$ with fixed vector $\vec{x}_i$. If the sigma

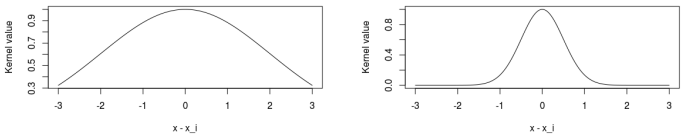

Figure 2: *One-dimensional example of RBF kernel (left: sigma = 2, right: sigma = 0.5).*

value is large the SV model will behave similarly when vectors close to each other are presented (rbf kernels in the prediction sum will have similar values). That means the model might be less complex having probably less support vectors but it might not be able to capture the complexity or 'shape' of the data. With a small sigma value every point will be treated differently by the machine. Models will be more complex but might overfit. The concept of overfitting refers to a model that has a good performance in the training set but is inefficient when predicting new data. Such a model is typically more complex than necessary (has more parameters than can be 'justified' by the data) which implies a costly evaluation, at least more than necessary.

The effect on the regressor's performance was evaluated by doing a sweep across the values $0.1, 0.2, 0.3, 0.4, 0.8, 1.5$. These values are justified because since training points are normalized most of their values will be between -0.5 and 0.5. However, in situations where there was a circuit's output for which really close input vectors gave substantially different values of it, the low sigma's sequence $seq(0.01, 0.02, 0.04, 0.06, 0.08)$ was used.

%T

The $\%T$ parameter represents the percentage of data points collected that are used for training (the sampling is done randomly). In order to see the effect of the number of data points used for training in the SVRs performance, models were trained using 40%, 60%, 80% and 100% of the data points collected.

***Lambda** – λ*

Lambda is the MLS parameter that controls the tradeoff between the common mode and the differential mode of the multioutput model. The more emphasis is put in reducing the common mode ($\lambda >> 1$) the more we are allowing/forcing the model's coefficients for each output to be different. On the other way, the more emphasis is put on the differential mode ($\lambda << 1$) the more the model's coefficients for each output will tend to be similar. The effect of changing the lambda values was evaluated by doing a sweep over the values $2^{seq(-10,10,2)}$ chosen according to the MLS paper [4].

5.4. Neural network regressors

A neural network regressor was implemented using the R nnet package suggested in [7]. Nnet implements feed-forward networks that are limited to a single hidden layer. However perceptrons with a single hidden layer can approximate essentially all useful functions. There might be cases in which to approximate the desired function the number of one layer hidden nodes necessary might be too high and we might want to explore a network with more layers in order to approximate the desired function with less nodes. However in the examples chosen there was no necessity to add more than one hidden layer.

5.4.1 Parameters

The nnet package uses the backpropagation algorithm. Besides choosing the training algorithm, implementation requires a choice of values for the neural network's parameters.

Input and output units

The number of input neurons and output neurons is equal to the number of inputs and outputs of our section/circuit.

Hidden layer units

The whole training set yields a set of nonlinear equations whose unknowns are weights. When there are less unknowns than equations, there will be no solution, unless the set of equations is redundant. Even the existence of an approximate solution implies that there must be some kind of redundancy, or regularity, in the training set. Regularities are what is expected to be maintained from the training set to the universe which means small networks are expected to generalize best if they can perform well on the training set.

We chose to adapt the rule of thumb suggested in [8] [1] and train neural networks where the ratio between the product of the number of training patterns by the number of outputs and the number of weights is $\frac{1}{10}, \frac{1}{9}, ..., \frac{1}{2}$. In the end the neural network with the best performance in the test set is picked. Note that for the same fraction value (ex: $\frac{1}{3}$) several neural networks are trained because of different decay values (explained next in this section).

Activation function of units

Nnet uses sigmoids as activation units. Linear units were used in the output layer because the outputs should to be able the whole range of possible target values which is wider than the range of values in the sigmoid.

Convergence criterion

The convergence criterions used are based in the fit criterion sum of square residuals.

$$SSE = \sum_{i=1}^{n} (y_i - y_{pred})^2 \tag{10}$$

- Stop when the SSE falls below the *abstol*. *Abstol* chosen was $1 * 10^{-4}$

$$SSE_{i+1} < abstol \tag{11}$$

- Stop when the training is unable to reduce the SSE by a factor of at least $1 - reltol$. *Reltol* chosen was $1 * 10^{-8}$.

$$SSE_{i+1} = SSE_i * (1 - reltol) \tag{12}$$

To ensure termination of the algorithm in case the convergence criterions are not satisfied, a maximum number of iterations is imposed (*maxit* = 10 000).

Regularization using Decay value

A method that consists of adding a regularization term (called weight decay) to the cost function being optimized was used. The use of weight decay seems both to help the optimization process and to avoid overfitting as mentioned in [9] where it is suggested $\lambda \approx 10^{-4}$ – 10^{-2} for least-squares fitting and variables of range one. The sequence $c(0, 10^{-4}, 5^{-4}, 10^{-3}, 5^{-3}, 10^{-2})$ was used and the model with best performance in the test set chosen.

6. Model evaluation engine

'SVR model' components were created using the Ngspice circuit extension XSPICE. The creation steps consist in: 1) create the code model directory and its associated model or data files, 2) inform Ngspice about the code model that has to be compiled and linked into Ngspice, 3) compile it into a shared lib and 4) load them into the ngspice simulator upon runtime.

After the circuit subsection is replaced, its internal nodes do not exist anymore. This means the KCL equations of the internal nodes won't appear in the vector **F** described in section 2. The vector (which represents the circuit's equations) will have a number of internal nodes less components (number of internal nodes less KCL equations) but also number of internal nodes less unknowns. It also means that all columns and lines of the partial derivatives matrix **J** (Newton method) corresponding to the internal nodes will disappear.

The new component can have as many outputs and inputs as desired and so associated with it there are $\mathbf{card}(inputs) + \mathbf{card}(outputs)$ currents/unknowns. These currents cannot be written as a function of the device's voltages and so modified node analysis has to be used. The currents are included as new variables and the device's constitutive equations added. As an example, for a device with one input and two outputs the equations to add would be:

$$\begin{aligned} i_{inp} &= i_{out1} + i_{out2} + i_{out3} \\ V_{out1} &= pf_1(V_{inp}) \\ V_{out2} &= pf_2(V_{inp}) \end{aligned} \tag{13}$$

where pf_i is the evaluation formula, of the chosen SVR algorithm, for output i. For the case of the standard SVR it is $Pred(\vec{x}) = \sum_{i=1}^{l} \alpha_i k(\vec{x_i}, \vec{x}) + b$ where $\vec{x}$ would be V_{inp} in this case.

The simulator also requires partial derivatives to allow it to solve the non-linear equations that describe the circuit behaviour. They are needed to compute the **J** matrix in the

[1] " The number of weights should be around or below one tenth of the product of the number of training patterns by the number of outputs (number of equations). In some situations, however, it may go up to about one half of that product."

Newton's method. The expressions for the derivatives of output subcircuit voltages V_{out} with respect to input subcircuit voltages V_{in} are introduced manually in the Model definition file. We must not forget that the SVR models work with normalized input and output voltages. This way the input voltages passed to the simulator have to be normalized and the output voltages computed by the model denormalized. The constitutive equation of the 'SVR model' component is therefore:

$$V_{out} = 2 * sd(output_{train}) * pf(nu(V_{in})) + mean(output_{train})$$
$$\text{where} \quad nu(V_{in}) = \frac{V_{in} - mean_(input_{train})}{2 * sd(input_{train})}$$
$$\text{and} \quad sd/mean(output_{train}) \quad \text{is the standard deviation/mean of output training wave} \tag{14}$$

Applying the chain rule the derivative of a V_{out} with respect to a V_{in} is:

$$\frac{\partial V_{out}}{\partial V_{in}} = 2 * sd(output_{train}) * \frac{\mathrm{d}pf}{\mathrm{d}nu}(nu(V_{in})) * \frac{\mathrm{d}nu}{\mathrm{d}V_{in}} \tag{15}$$

The C code function developed for the model definition file specifies the operations to be performed within the model on the data passed to it by the simulator. It is used in every iteration of the Newton method because everytime the model input voltage changes the output voltage has to be computed using the model expression (don't forget that for each time point in a transient analysis there can be many Newton iterations).

In the first invocation of the code model by the simulator the training vectors, model coefficients and normalization parameters are read from their respective files (written by the model generation engine) and stored in vectors. Since all the SV models are defined by the training vectors used, coefficients alpha, parameter b and sigma, all we have to do is provide different training set and SV model files if we want to model a new component. In each iteration of the Newton method Algorithm 1 is used, where $\vec{V_{in}}$ represents the sections inputs and $\vec{x_i}$ the inputs used for training.

7. Examples

The modelling framework was tested in 6 different circuits. Here a subset of the experiments done, that helps presenting the main findings, is described.

Algorithm 1 Code model

$partial_j \leftarrow 0$; $output_j \leftarrow 0$;
for $i = 1$ to Number train vectors **do**
  if $\vec{x_i}$ is a SV **then**
    Compute $k(\vec{V_{in}}, \vec{x_i})$
    for $j = 1$ to Number outputs **do**
      if $\alpha_{ij} \approx 0$ **then**
        continue
      else
        $output_j \leftarrow output_j + pf(k(\vec{V_{in}}, \vec{x_i}), \alpha_{ij})$
        $partial_j \leftarrow partial_j + \frac{\partial \vec{V_{out_j}}}{\partial \vec{V_{in}}}$ (compute the parcels correspondent to $\vec{x_i}$)
  else
    continue
$\vec{output_j} \leftarrow \vec{output_j} + \vec{b_j}$; Denormalize $\vec{output_j}$

7.1. Resistive circuit

All regressors were able to accurately model the input-output relationships of a resistive circuit in which all the internal nodes' voltages were considered as outputs (5 nodes). The multioutput regressor VV-SVR had less complexity (10 vs 36 SVs) and more accuracy than the standard SVR regressor. For that reason it was concluded that multioutput models might be beneficial when modelling multiple nodes/outputs of a circuit

7.2. Nmos inverter

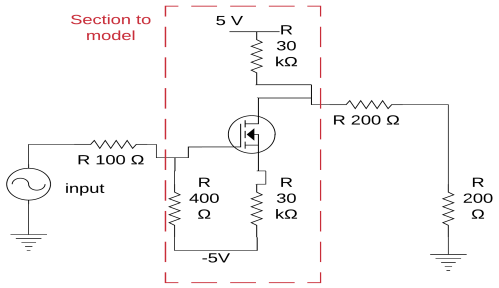

Figure 3: *Nmos Inverter.*

This example was conducted to see if training a subcircuit using data from the linear region of its output generalizes well to the non-linear region and vice-versa. In this example the highlighted section of the Nmos inverter circuit of Figure 3 was modelled. Results when training with distor-

tion/no distortion and testing with no distortion/distortion are shown in Figures 4/5 respectively. The parameter chosen and the complexity of the models are both identified in the legends. In the case of Figure 5 there was no parameter value that gave a mean relative error below 5% for all models so the parameter value chosen was the one associated with models that present less errors.
Although the linear region chosen incorporates a bit of the non-linear region it does not generalize well because the majority of the shape of the non-linear region is not known by the machine. The non-linear region generalizes much better because the linear region is contained in it. Also, the waveform used should not matter since the algorithms only care about input values which can be seen in the example of Figure 6 (good performance when the waveform used for testing is different than the one used for training). In conclusion one should try to include points in every region of the function we are trying to model and any waveform can be used for training.

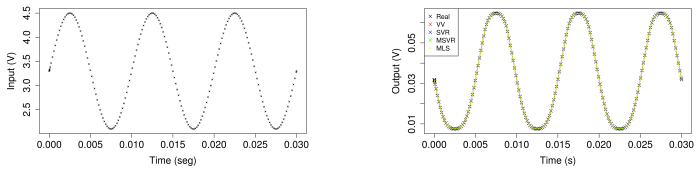

Figure 4: *Nmos train with distortion and test with no distortion. datapoints* $=$ 15% *and* VV : 16, SVR : 16, $MSVR$: 28, MLS : 28 *(left: test input waveform, right: test output waveform).*

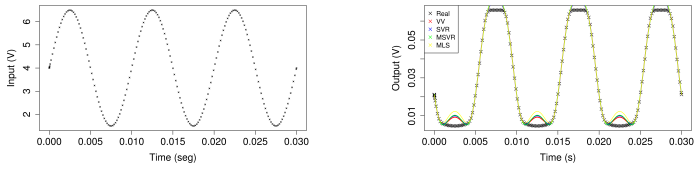

Figure 5: *Nmos train with no distortion and test with distortion.* $\sigma = 1.5$ *and* VV : 172, SVR : 154, $MSVR$: 175, MLS : 182 *(left: test input waveform, right: test output waveform).*

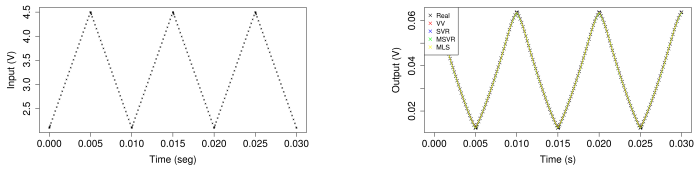

Figure 6: *Triangular wave: Train with distortion - test with no distortion (left: test input waveform, right: test output waveform).*

7.3. MOS 3a amplifier

The aim of this example is to see if the regressors can model the behaviour of a significantly more complex circuit than the previous experiments carried out in this work. This circuit has 3 outputs and is described in [10]. In order to model the circuit's behaviour it is necessary to know the output vs input functions for every output. With that in mind we provided the circuit with two input waves. The first one with an amplitude of 4.5V and the second one with an amplitude of 0.4V (in order to have enough points in the falling part of output1 represented in Figure 7), both with an offset of 1.65V. We collected 60% of the data points generated from the first wave and 40% of the data points generated from the second wave and that way we can have a clear idea of what the output vs input functions are. The functions obtained are in Figure 7. Training the regressors

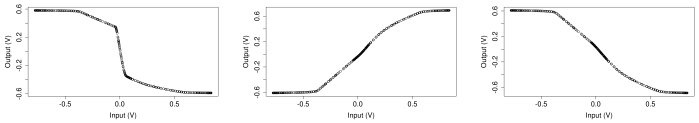

Figure 7: *3a circuit output vs input functions for every output (normalized) (from left to right, output1, output2 and output3).*

with these data points we want to see if there is a model that can accurately predict the circuit outputs for as much input waveforms as possible. In order to verify this we make two tests: 100mV amplitude and 5.5V amplitude input waveforms. The falling transition in the first output means there are really close inputs with substantially different output values. Therefore it might be beneficial to test lower sigmas so that the model does not treat close input points as equal. The low sigma sequence used was: 0.01, 0.02, 0.04, 0.06, 0.08.

Because the σ value is low, models don't know how to predict points that are not sufficiently close to any of the ones used for training. For that reason a good performance in the 5.5V test wave was only achieved after changing the amplitude of the training wave to 6V. All the algorithms were able to provide accurate models except the VV-SVR. A set parameters associated with a VV model that gives good results in both the 100mV test wave and the 5.5V test wave was not found (although there were sets for which the results in the 100mV test were good and sets for which the results in the 5.5V test were good). Therefore it can be concluded that the VV algorithm is not suitable for this circuit. As mentioned before it was also concluded that if a

low value of σ is used for training the model, prediction is only successful if there are training points very close to the one we want to predict.

7.4. Performance comparison between MLPs and SVRs

A comparison of the performance of a neural network (MLP) with the performance of the SVR machines was done, more specifically the number of operations and quality of the result obtained. The quality of the result was measured using the mean of the absolute errors squared (which from now on is referred to as *Abs_error*). Here the experiment done on the MOS 6a circuit [11] is presented. The *Abs_error* for each output and model complexity, for 100mV and 2V waveforms, are in Table 1. The best neural network had 50 hidden units and the best SVR model (MLS-SVR) 164 support vectors.

	Output1 (V^2)	Output2 (V^2)	Output3 (V^2)
Nnet	7.640e-07	8.440e-08	8.282e-07
SVR	7.534e-10	1.620e-09	2.453e-09
Nnet	7.660e-09	2.009e-06	2.127e-06
SVR	6.384e-09	1.092e-07	1.193e-07

Table 1: *MOS 6a circuit results for the 100mV wave (above) and 2V (below).*

Prediction *Abs_errors* using the neural network are slightly larger than those obtained when using the best SVR regressor. However the accuracy is still good (the mean relative error in each output is below 5%). In what concerns the number of operations to evaluate/predict a value (evaluation time):

- Number of multiplications: 164 * 3 + 164 = 656 (SVR) vs 200 (nnet)
- Number of kernel/sigmoid computations: 164 * 2 = 328 (SVR) vs 50 (nnet)

The cost of a kernel computation was considered to be similar to that of a sigmoid computation as they both require computing one exponential. As there are a lot more kernel computations and multiplications in the SVR model one should expect it to have a much higher evaluation time than the neural network model.

From this and other experiments it was concluded that SV regressors seem to be able to provide somewhat more accurate models but at the expense of a significant more costly evaluation time. However it was found that for an accuracy similar to that of the best neural network model a SVR model with an evaluation time closer to it, but still being significantly higher, can be obtained.

7.5. Analysis of the accuracy/complexity plots

By interpretation of error plots obtained for some experiments done some conclusions about the way changing the parameters values affects the accuracy and the complexity of the models generated were derived. Based on that we derived optimal values for the parameters sweep which are represented in Table 2.

Parameter	Sweep values
Data_points	(0.2, 0.4, 0.6, 0.8, 1) (%)
ε	(5e−2, 1e−2, 5e−3, 1e−3, 5e−4, 1e−4)
C	100
$low \quad \sigma$	(0.01, 0.02, 0.04, 0.06, 0.08)
$big \quad \sigma$	(0.1, 0.2, 0.3, 0.4, 0.8, 1.5)
λ	2e-10

Table 2: *Optimal values for the parameters sweep.*

7.6. Retraining

After providing a test set the user can choose the model type (VV, MSVR, SVR, MLS or all), the error type ($mean(abs\_errors^2)$, $mean(rel\_errors)$, $max(rel\_errors)$) and obtain the model with the lowest value of the error type chosen in the test set. For circuits with more than one output the best model is the one where the product of the output errors is smaller.

The script also identifies the test points that have a relative error above 10% in any of the outputs and asks the user if he wants to add them to the training set and retrain the model. This can be done for any desired model.

A simple diagram illustrating the script behaviour is presented in Figure 8.

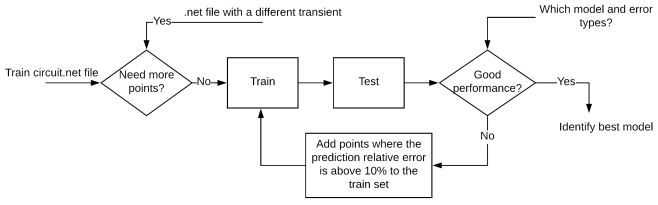

Figure 8: *Simple script diagram.*

7.7. Evaluating run-time complexity

A black-box model of the MOS 3a circuit was obtained using the model generation engine: first models were trained using a sinusoidal waveform (amplitude 4.5V, offset 1.65V, freq 1kHz) and then the one with the smaller value of mean relative error in a sinusoidal test waveform (amplitude 3.5V, offset 1.65V, freq 1.5kHz) was chosen. Since the RBF kernel is computationally expensive due to its exponential part, a polynomial kernel was also tested. The kernel chosen has the form $K(\vec{x}_1, \vec{x}_2) = (\vec{x}_1 \cdot \vec{x}_2 + 1)^d$ where d is the degree of the polynomial. In order to determine the value of d, degrees up to 4 were tested until accurate results in the training set were achieved. This only happened for $d = 4$ which was, for that reason, the degree chosen. After generating the polynomial models, a model with a good accuracy in the test set and a number of support vectors close to the ones of the RBF model was chosen.

After generating the black-box models the simulation times for a transient analysis with a step value of 14.3 μs, a final time of 1s and an initial time of 11 μs were obtained (results are in Table 3).

	Time (s)	Number of SV
Original circuit	9.018	-
Black-box RBF	2.934	67
Black-box poly	1.985	80

Table 3: *Simulation times and number of SV for the original circuit and for the black-box models.*

We can see that both the RBF and polynomial kernel models reduce the simulation time. In the case of more complex circuits, that have more complex components or a higher number of components, the reduction in the simulation time might be even bigger unless the complexity of the black-box models needed to model the input-output relationships increases substantially. Also as we can see from Table 3 the polynomial kernel computations are less expensive since a model with more support vectors has less the same evaluation time.

8. Conclusions

In this thesis the problem of modeling complete subcircuits or modules within circuits using a black-box approach was studied. A robust modeling framework was developed so that a black-box model for any given circuit can be generated. Along with this framework a model evaluation engine was also coded so that the black models can be easily converted in SPICE components.

The results showed that this black-box model approach is successful in simple circuits as well as in more complex circuits. It was also shown that black-box models are able to reduce the simulation time of circuits.

In the future it is necessary to study how to model dynamic systems and to test the frameworks' robustness on bigger/more complex circuits.

References

[1] Ngspice. `http://ngspice.sourceforge.net/`. Accessed: 2018-06-14.

[2] M. Brudnak. Vector-valued support vector regression. *IEEE*, 2006.

[3] A. Kowalczyk, J. Verrelst, L. Alonso, F. Pérez-Cruz, and G. Camps-Valls. Multioutput support vector regression for remote sensing biophysical parameter estimation. *IEEE Geoscience and Remote Sensing letters*, 2011.

[4] S. Xu, X. An, X. Qiao, L. Zhu, and L. Li. Multi-output least-squares support vector regression machines. *Pattern Recognition Letters*, 2013.

[5] The r project for statistical computing. `https://www.r-project.org/`. Accessed: 2018-06-14.

[6] N. Kumar. Using support vector machines effectively. `https://neerajkumar.org/writings/svm/`. Accessed: 2018-06-14.

[7] K. Shah. What are the best machine learning packages in r? `https://www.r-bloggers.com/what-are-the-best-machine-learning-packages-in-r/`. Accessed: 2018-06-14.

[8] L. B. de Almeida. Multilayer perceptrons. Adapted from section C1.2 of the Handbook of Neural Computation.

[9] W.N.Venables and B.D.Ripley. *Modern Applied Statistics with S*. Springer-Verlag New York, 2002.

[10] R. Póvoa, N. Lourenço, R. Martins, A. Canelas, N. Horta, and J. Goes. Single-stage amplifier biased by voltage-combiners with gain and energy-efficiency enhancement. *IEEE Transactions on Circuits and Systems II*, 2017.

[11] R. Póvoa, N. Lourenço, N. Horta, R. Santos-Tavares, and J. Goes. Cascode-free single-stage amplifier using a fully-differential folded voltage-combiner. *21st IEEE International Conference on Electronic Circuits and Systems (ICECS)*, 2014.