Machine Learning in Simulation-Based Analysis *

Li-C. Wang
University of California, Santa Barbara

Malgorzata Marek-Sadowska
University of California, Santa Barbara

ABSTRACT

This paper describes two separate learning flows for improving the efficiency of simulation-based design analysis. Machine learning concepts and methods are explained in the context of realizing the two learning flows. Experimental results are presented to demonstrate their feasibility. Generality of the proposed learning flows is illustrated using the kernel-based learning concept.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids

General Terms

Simulation

Keywords

Computer-Aided Design; Data Mining; Circuit Simulation

1. INTRODUCTION

Circuit simulation is indispensable for verifying the analog behavior of a design. For assessing the uncertainty of design behavior over process variations, Monte Carlo circuit simulation is one of the most popular approaches. However, circuit simulation can be time consuming. Hence, for a large and complex design, Monte Carlo circuit simulation can become prohibitively expensive.

Figure 1 depicts a functional view of the Monte Carlo style design analysis considered in this work. In this view, the design under analysis is seen as a mapping function $f()$. Inputs to the function comprise two sets of random variables. First, there are random variables modeling the input variations in the input space **X**. Furthermore, there are random variables

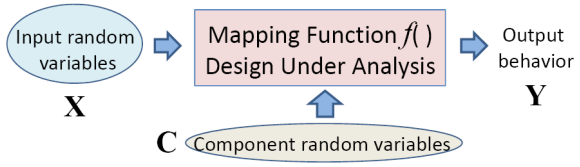

Figure 1: Functional view of a design analysis

modeling parametric variations associated with the components of the design. This component space is denoted as **C**. The function $f()$ is a mapping from (**X**, **C**) to the output space **Y**, i.e. $f : (\mathbf{X}, \mathbf{C}) \rightarrow \mathbf{Y}$. The function $f()$ can be computed through simulation. The objective of a Monte Carlo style design analysis is to evaluate the output variations with respect to the input variations from both the **X** space and the **C** space. For example, the analysis might be used to assess the performance yield of a design based a given process variation model.

As a specific application example, consider Monte Carlo circuit simulation of an analog design represented as a netlist of N transistors. The input space **X** comprises a set of possible input waveforms over a time period t, i.e. each sample $x_i \in \mathbf{X}$ is a waveform represented as a vector of voltage values over t time steps. Each component random variable models the size variation of a transistor, i.e. each sample $c_i \in \mathbf{C}$ is a vector of N transistor sizes. The objective of the analysis is to assess the behavior changes of the output waveforms due to the transistor size variations.

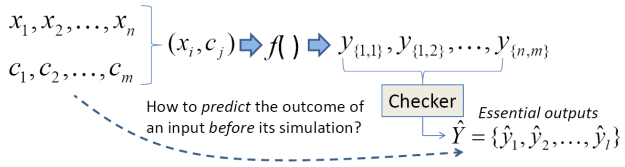

Figure 2: Formal view of the fundamental problem

Figure 2 provides a formal view of the fundamental problem considered in this research. In a Monte Carlo analysis, suppose that m samples $c_1, \ldots, c_m$ are drawn from the **C** space. Moreover, n samples $x_1, \ldots, x_n$ are derived from the **X** space. For example, these n samples can each be a waveform over the same time period $[0, t]$. As another example, circuit simulation is over a period $[0, T]$ and the period is divided into n successive time frames $[0, t_1]$, $[t_1, t_2]$,...,$[t_{n-1}, t_n]$ where $x_1, \ldots, x_n$ are waveforms from the n time frames. In the setting of Figure 2, every *input sample*, represented as a 2-tuple (x_i, c_j), produces a corresponding output $y_{\{i,j\}}$.

*This work is supported in part by Semiconductor Research Corporation, projects 2012-TJ-2268 and 2013-TJ-2466 and by National Science Foundation Grant No. 1255818.

ISPD'15, March 29–April 1, 2015, Monterey, CA, USA.

http://dx.doi.org/10.1145/2717764.2717786.

Suppose that with respect to the design analysis task at hand, a subset of the l outputs $\hat{y}_1, \ldots, \hat{y}_l$ would be sufficient to represent the relevant output space. Let them be called the *essential outputs*. In Figure 2, a *checker* is applied on the $n \times m$ outputs to identify the l essential outputs. Assume $l \ll n \times m$. This assumption of having a small subset of essential outputs is usually practical. As a simple example, in a Monte Carlo style analysis of a circuit performance parameter (e.g. timing), the analysis goal could be to verify the range of the parameter. Then, among all the outputs, only two are essential: the min and the max.

Let the input samples required to produce the essential outputs be called the *important input samples*. Ideally, one desires to simulate only the important input samples to save simulation cost. This idea is only feasible if one has a way to predict the importance of an input sample or a way to predict its output. This leads to the fundamental problem:

> "How to predict the outcome (i.e. either the importance or the actual output) of an input sample *before* the sample is simulated?"

1.1 Predicting the outcome of an input sample

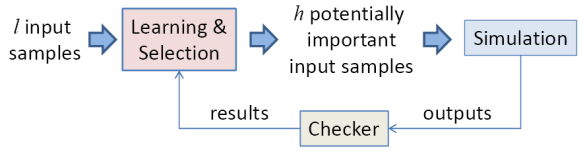

Figure 3: Learning to predict important inputs

Figure 3 and Figure 4 illustrate two approaches to the fundamental problem. Each approach is based on an iterative learning flow. In Figure 3, learning is based on the simulation results known in the current iteration, including both the known important and unimportant input samples. Once a learning model is constructed, it is used to predict the importance of input samples in the next iteration.

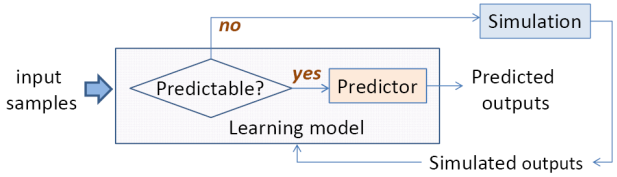

Figure 4: Learning to predict the outputs

In Figure 4, learning is for predicting the output of an input sample. The learning model comprises two parts: (1) a method to decide if an input sample is predictable, and (2) a predictor that produces the predicted output. Learning the predictor is based on the simulation results obtained so far, including the input samples simulated and their outputs. Note that while Figure 4 can be used to predict the importance of an input (by predicting its output), the learning flow can also be used to bypass simulation for those predictable inputs. This represents another way to save simulation cost. For example, given a circuit comprising multiple blocks, Figure 4 can be applied to one of the blocks and bypass simulation of some inputs to the block.

The rest of the paper is organized as follows. Section 2 introduces the related works. Section 3 states the concept of kernel-based learning. Section 4 and Section 5 describe the details and experimental results of the two learning concepts illustrated in Figure 3 and Figure 4, respectively. Section 6 concludes this paper.

2. RELATED WORKS

Existing research for improving the efficiency of simulation based design analysis can be divided into two categories. In the first category, the analysis focuses on component variations from the $\mathbf{C}$ space without explicitly considering the variations in the $\mathbf{X}$ space. In the second category, the analysis focuses on the input variations from the $\mathbf{X}$ space and ignores the variations in the $\mathbf{C}$ space.

For example, one notable area of research is Static Statistical Timing Analysis (SSTA) [2][3]. In SSTA each delay element (equivalent to a component in Figure 1) is modeled as a random variable according to process variations. Delay elements in SSTA are usually assumed to be pin-to-pin delays of a cell [3]. Circuit timing is a function of a set of delay random variables under the worst-case assumption on the input pattern space $\mathbf{X}$. The analysis is static because variation in the input pattern space is not considered. In SSTA, $\mathbf{Y} = f(-, \mathbf{C})$ is computed by propagating the delay distributions directly through the circuit. It does not involve random sampling of the $\mathbf{C}$ space and hence, avoids the high cost of Monte Carlo simulation of random samples.

The same idea of propagating probability distributions can be applied to low-level circuit analysis where the random variables are based on basic circuit elements such as resistors and capacitors. In low-level circuit analysis, the operators involved are no longer restricted to addition and maximization as those used in SSTA. Hence, the problem becomes more complex. For example, the work in [4] applies Polynomial Chaos Theory (PCT) [5] to low-level circuit analysis. In a PCT framework, distributions are modeled with orthogonal polynomials to facilitate their propagation through the circuit equations [4].

The work in [6] retains the idea of random sampling with Monte Carlo simulation. To improve efficiency, supervised learning techniques are applied to predict the irrelevant random samples from the component space $\mathbf{C}$. These irrelevant samples are discarded from simulation. In more recent works [7][8], advanced learning techniques are applied to develop an efficient framework for statistical analysis of circuit performance parameters. The framework is intended and optimized for applications where the underlying sources of variations are mainly from the component space $\mathbf{C}$.

In Figure 1, if one takes an extreme view that the mapping function is a processor or SoC, then the input space $\mathbf{X}$ becomes extremely large. In this case, considering component variations is no longer practical. Typically, for verifying an SoC, RTL simulation is used. Simulation cost is high due to the large input space $\mathbf{X}$ which needs to be covered. In this context, an input x becomes a functional test, e.g. a sequence of vectors.

The work in [9] assumes that a functional test is a sequence of 0/1 vectors of a fixed length. Then, the input space $\mathbf{X}$ essentially can be viewed as an N-dimensional space comprising all combinations of 0/1 vectors. With such a view, the work proposes a framework for reducing simulation time by identifying and simulating only the potentially important functional tests. Unsupervised learning technique, the one-class Support Vector Machine (SVM) [10], was applied to

learn and model the unimportant input subspaces to facilitate the selection of the potentially important inputs. The works in [11][12][13] extended the idea to analyze functional tests that are assembly programs.

An example to consider variations in both the **X** space and the **C** space is statistical delay testing, where the mapping function $f()$ is a gate-level circuit with w inputs. The 2^w possible input patterns constitute the **X** space. Then, component variations can be based on two sources: (1) variations of the delay elements due to process variations, and (2) variations in the delay defect sizes and locations.

Because it is not feasible to apply all 2^w input patterns, one crucial aspect of the statistical analysis is to identify the important input patterns that excite and observe delay defects under the statistical timing model. The early work in [14] is an example of an approach to this delay testing problem. However, the work relied on Monte Carlo simulation of random samples from the **C** space and no simulation time reduction was intended with respect to this space. Additionally, no statistical learning was applied.

3. KERNEL-BASED LEARNING

Figure 5 illustrates a typical dataset seen by a statistical learning algorithm (For more discussion, see e.g. [1]). When $\vec{y}$ is present and there is a label for every sample, it is called *supervised* learning [15]. In supervised learning, if each y_i is a categorized value, it is a *classification* problem. If each y_i is modeled as a continuous value, it is a *regression* problem. When $\vec{y}$ is not present and only **X** is present, it is *unsupervised* learning [15]. When some (usually much fewer) samples have labels and others have no label, the learning is called *semi-supervised* learning [16].

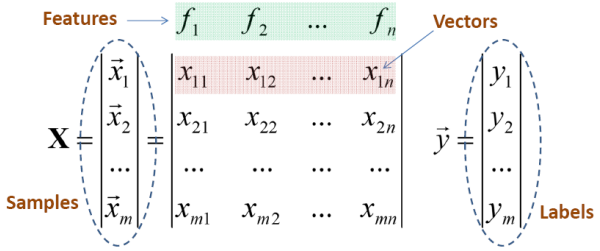

Figure 5: Typical dataset for a learning algorithm

In Figure 5, each sample from **X** is assumed to be encoded with n features $f_1, \ldots, f_n$. Hence, the characteristics of each sample are described as a vector $\vec{x_i}$ of n values. Figure 5 illustrates two fundamental challenges for applying a statistical learning algorithm in the learning flows depicted in Figure 3 and Figure 4:
(1) In our application, an input sample is a 2-tuple (x_i, c_j) which might not be given as a vector as shown in Figure 5.
(2) More importantly, the output $y_{i,j}$ might not be a scalar value, e.g., outputs can be waveforms. Note that this second challenge is only relevant to Figure 4. Figure 3 can be seen as an unsupervised learning flow because outputs are not what is to be predicted. Figure 4 is a supervised learning flow because outputs are to be predicted.

3.1 The importance of similarity measure

Many modern statistical learning algorithms follow the paradigm of *kernel-based learning* [17][18]. Figure 6 illustrates the basic concept of kernel-based learning.

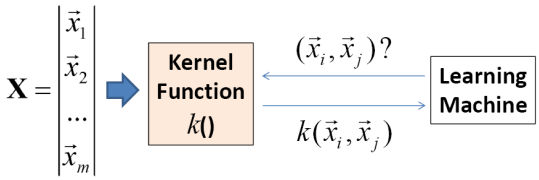

Figure 6: Kernel function vs. learning machine

In kernel-based learning, the learning machine, i.e. the learning algorithm such as a Support Vector Machine (SVM) algorithm [17], is not required to access the samples $\vec{x}_1, \ldots, \vec{x}_m$ directly. Instead, all the information required for the learning is coming from a *kernel function* $k()$ as shown in Figure 6. The kernel function measures the *similarity* between two samples based on some definition of similarity.

Kernel-based learning provides great flexibility to apply learning techniques in EDA and test applications [1], especially when the samples to be analyzed are not provided in vector formats like that shown in Figure 5. This is because from the learning perspective the representation of a sample is no longer important. What is important is how similarity between samples should be measured. This similarity measure, i.e. the kernel function, defines the space in which the learning is performed. With kernel-based learning, the challenge of having a proper input sample representation is alleviated. However, one still has the challenge of searching for a proper kernel function.

The kernel-based learning concept depicted in Figure 6 allows the separation of learning theories and methods from their specific implementation for a particular application context. To see this, consider Figure 3 and Figure 4 again. In Figure 3, we propose to develop the theories and methods to build a learning model $M_{sel}()$ to predict the importance of an input sample s by $M_{sel}(s)$. In Figure 4, we propose to develop the theories and methods to build another type of learning model $M_{pre}()$ associated with an evaluation scheme $E_{sel}()$. For a given sample s, $E_{sel}(s)$ indicates if s is predictable and $M_{pre}(s)$ provides the predicted output.

With the concept of kernel-based learning, we see that as long as a kernel function $k()$ is provided to measure the similarity between a pair of input samples s_1, s_2 as $k(s_1, s_2)$, a learning machine can operate on the similarity measures to build learning models. The actual representation of the input samples is irrelevant to the learning machine. Hence, theories and methods for learning the models $M_{sel}()$, $M_{pre}()$, $E_{sel}()$ can be based on the assumption of having the corresponding kernel functions. Then, the theories and methods are not specific to a particular type of input samples.

When the learning flows in Figure 3 and Figure 4 are applied to a particular simulation context, implementation of the specific kernel functions takes place. For example, in the context that each input x is a waveform over a time period and each component sample c is a vector of transistor sizes, one needs to implement a kernel function $k()$ such that for any pair of input samples $s_1 = (x_{i_1}, c_{j_1})$, $s_2 = (x_{i_2}, c_{j_2})$, $k(s_1, s_2)$ provides a similarity measure between them.

Without loss of generality, in the rest of discussion we assume that $0 \leq k() \leq 1$ where $k() = 1$ indicates the two samples are identical and $k() = 0$ tells that the two samples are most different. Note that one may also implement a separate kernel function $k_x()$ applied to the x samples and another function $k_c()$ applied to the c samples.

The implementation of kernel functions $k()$, $k_x()$ and $k_c()$ is application specific. The learning theories and methods to construct the learning flows in Figure 3 and Figure 4 are not. This separation allows the theories and methods developed from the proposed research to be applicable in a wide variety of different contexts.

4. PREDICTING IMPORTANT SAMPLES

Figure 2 shows that simulation of all input samples results in l *essential* output samples, $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_l\}$. Without loss of generality, we assume that these l output samples are those most dissimilar to each other based on a similarity measure $k_y()$. In other words, one can view these l outputs as the *representative* outputs of the space covered by all the $n \times m$ outputs based on the definition of $k_y()$. Note that this assumption is without loss of generality because we do not make any assumption on the kernel $k_y()$ itself. Hence, different definitions of $k_y()$ can be used to model different scenarios as how the essential outputs should be selected.

As a simple example, the l essential outputs can be selected with an iterative greedy method. The first essential output is selected randomly. Then, suppose the current set of essential outputs is $\hat{L} = \{\hat{y}_1, \ldots, \hat{y}_j\}$. For an output y, if $\forall \hat{y}_i \in \hat{L}$, $k_y(y, \hat{y}_i) < \delta$, then y is included as an essential output to the $\hat{L}$ set. For example, we can set $\delta = 0.8$ to mean that each essential output is different from all other essential outputs by at least 20% as measured by $k_y()$.

4.1 The essence of the learning problem

The learning flow in Figure 3 operates iteratively. In each iteration, h input samples are selected and simulated to produce h outputs. With the assumption that essential outputs are those most dissimilar to each other based on the similarity measure $k_y()$, it is intuitive to see that a feasible objective in each iteration is to select the h input samples that can produce outputs that are as much dissimilar as possible. The challenge, again, is that *before* simulation we do not know the outputs of those input samples to be selected and consequently, do not know how dissimilar their outputs would be.

For the input samples, we have another kernel function $k()$ that measures the similarity for a given pair of input samples. Suppose for every pair of input samples s_1, s_2 and their outputs y_1, y_2, we have $k(s_1, s_2) = k_y(y_1, y_2)$, i.e. input similarity = output similarity. Then, observe that in this case selecting important input samples becomes a trivial problem. This is because by selecting input samples that are most dissimilar to each other based on the kernel $k()$, we guarantee that the resulting outputs would also be equally most dissimilar to each other based on the kernel $k_y()$.

Usually, we have $k(s_1, s_2) \neq k_y(y_1, y_2)$. For example, two very different inputs can produce two very similar outputs, and vice versa. Then, the essence of the learning problem in Figure 3 can be thought of as the following: Starting from a given kernel $k()$, how to iteratively learn a kernel $k'()$ such that the similarity measures by $k'()$ are close to the similarity measures by $k_y()$?

This observation inspires the idea of *adaptive similarity measure* described below.

4.2 Iterative process with clustering

Figure 7 depicts an example of what might happen from one iteration to the next in the iterative learning flow shown in Figure 4. In this example, there are in total 14 samples. In each iteration, 3 samples are selected.

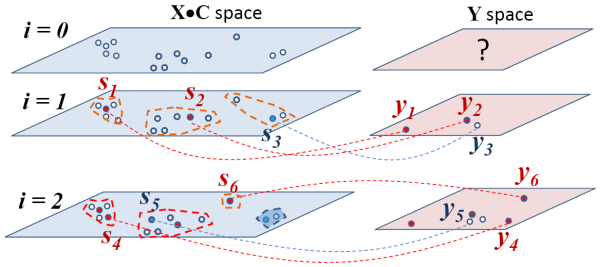

Figure 7: Iterative search for important inputs

For iteration $i = 0$, no sample is applied yet. Hence, there is no information on the $\mathbf{Y}$ space. Suppose that a kernel function $k()$ is given for measuring similarities between input samples in the input space $\mathbf{X} \bullet \mathbf{C}$. For iteration $i = 1$, three samples are selected. Because there is no information on the outputs in the $\mathbf{Y}$ space yet, in this iteration we simply cluster input samples into three groups of similar samples. Then, a representative sample is selected from each group. This is to achieve the effect that the selected three input samples are most dissimilar based on the given similarity measure function $k()$.

Suppose that the three selected inputs are s_1, s_2 and s_3, and after they are simulated, we discover that the corresponding y_1 and y_2 are essential, e.g. they are dissimilar by at least 80% based on the output kernel measure $k_y()$. But y_3 is not essential because it is similar to y_2 as shown in the figure. Then, for iteration $i = 2$, the question is, what would be a good strategy to take advantage of this new information observed in the $\mathbf{Y}$ space to help select the additional three input samples?

There are two pieces of information from simulating s_1, s_2, and s_3. First, it tells that the subspaces covered by the two clusters containing s_1 and s_2 respectively are important. Without further information, it makes sense to select additional input samples from those subspaces. Second, it tells that the subspace near s_3 is unimportant. Hence, it makes sense to exclude the subspace in the next selection.

Following these two ideas, in iteration $i = 2$, the subspace close to s_3 is blacked out. Clustering is then applied to the rest of the samples. Three samples s_4, s_5, and s_6 are selected. Samples s_4 and s_5 are selected from the same clusters as s_1 and s_2 before. Notice that s_6 is at a distance from s_3 and hence, it is not deemed unimportant because of s_3. In iteration $i = 2$, s_6 by itself forms a cluster and is selected.

4.3 Adaptive similarity measure

Figure 8 illustrates how the ideas discussed above can be accomplished without changing the clustering algorithm (for clustering algorithms, see, e.g. [22]). Given a kernel function $k()$, it implicitly defines a space where the similarity between a pair of samples is measured. In this space, clusters are formed. This is shown in the left plot of Figure 8. Suppose two clusters are formed and two samples s_1, s_2 are selected and simulated. Suppose the outcome is that s_2 produces an essential output while s_1 does not. Hence, s_2 is a known important sample and s_1 is a known unimportant sample.

The trick is that in the next iteration, a new space is created based on s_1 and s_2 and the clustering algorithm is

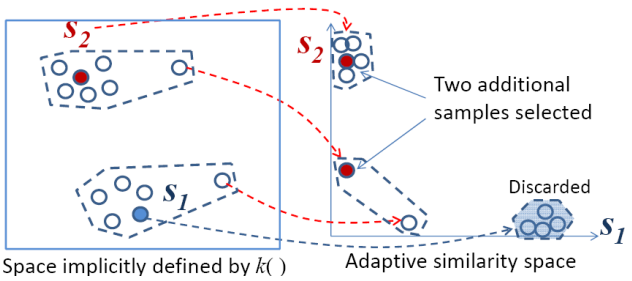

Figure 8: Adaptive similarity measure

applied to form clusters in this new space. In the new space, each sample s is placed at the coordinate $(k(s, s_1), k(s, s_2))$ in the 2-dimensional space defined by the two samples s_1 and s_2 and the given kernel function $k()$. Then, the similarity between two samples s_i and s_j in the new space is measured by $e^{-dist(s_i,s_j)^2}$ where $dist(s_i, s_j)$ calculates the distance between the two samples in the new space.

In the new space (the right plot), observe that the samples originally close to s_1 (and far from s_2) are now all close to the point $(1, 0)$ and form a cluster. This cluster is blacked out because s_1 is an unimportant sample. The samples originally close to s_2 (and far from s_1) are now close to the point $(0, 1)$ and form another cluster. One representative sample is selected because s_2 is an important sample. Furthermore, the remaining two samples form a cluster of their own and another representative sample is selected.

This simple example illustrates that without changing the clustering algorithm, by projecting the input samples into a new similarity measure space, the ideas discussed in Figure 7 can be realized. The *adaptive similarity measure* method is summarized as the following. Without loss of generality, assume that p samples s_1, ..., s_p have been simulated. Samples $s_1, \ldots, s_i$ are deemed important samples while s_{i+1}, ..., s_p are deemed unimportant. Then, consider all input samples not yet simulated:

1. For an input sample s, for each j, $i + 1 \leq j \leq p$, if $k(s, s_j)$ is greater than $\max\{k(s, s_q)|\forall q, 1 \leq q \leq i\}$, then s is removed from consideration in the current iteration, i.e., s is more similar (or "closer") to an unimportant sample than to *any* of the important samples.

2. For all the input samples not removed, project them into the space defined by the important samples s_1, ..., s_i and the given kernel function $k_x()$ by placing each sample s at the coordinate $(k(s, s_1)$, ..., $k(s, s_i))$ in the i-dimensional space. Measure similarities using the kernel $e^{-dist()^2}$ discussed above. Find clusters in this new space to select additional representative samples.

4.4 Experimental result

To assess the feasibility of the proposed adaptive similarity measure method, we designed an experiment in the context of Monte Carlo circuit simulation for an analog design. Figure 9 shows the circuit example, an Ultra-Wideband Phase Lock Loop (UWB-PLL) design used to tune the frequency of an impulse radio ultra wideband transmitter in [19]. The UWB-PLL comprises 949 transistors. Specific design details of the PLL can be found in [19]. For our experimental purpose, the design details are not that important.

To create a dataset as shown in Figure 2, we performed Monte Carlo simulation of 100 component samples c_1,..., c_{100} sampled from a process variation model for the transistors. Given a fixed frequency clock as input to the PLL, 3000 time steps were simulated for each sample. In each simulation, the first 500 time steps were ignored. From the remaining 2500 time steps, 50 waveforms were extracted, each with a time period of length 50 time steps. In total, there were 5000 waveforms collected at each net.

Then, a dataset was created for each of the four selected pairs of input nets and output nets, whose locations are marked in Figure 9 as $(I_1, O_1), \ldots, (I_4, O_4)$, respectively. In each case, we applied the proposed iterative learning flow to identify important input samples (waveforms).

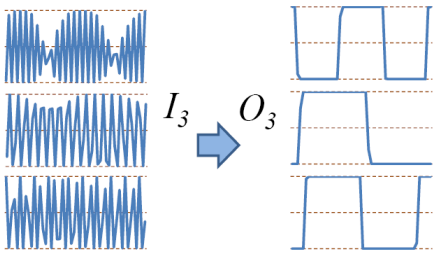

Figure 10: Input/output waveform examples

For essential output waveforms, we followed the rule discussed before that they each were at least 20% dissimilar to each other, based on a given kernel function $k_y()$. Figure 10 shows three examples of input waveforms that produce output waveforms dissimilar enough to be essential.

		Apply learning			Random	
In	Out	# Iters	# IS's	# EO's	# IS's	# EO's
I_1	O_1	4	128	31	700	31
I_2	O_2	4	222	64	2200	64
I_3	O_3	5	270	56	750	56
I_4	O_4	30	1979	50	2800	50

Table 1: Experimental results

Table 1 summarizes the results. The "# of Iters" column shows the number of iterations performed by the iterative learning flow. The "# of IS's" shows the total number of input waveforms selected as potentially important input samples. The "# of EO's" shows the total number of essential outputs covered by the selected input samples. For comparison, we also ran a process using random selection of input samples. Those results are shown in the last two columns.

As we can observe, random selection required selecting many more inputs to cover the same number of essential outputs, especially for the first three cases. This clearly demonstrates that the iterative learning flow had a positive effect on predicting the important input samples. With the fourth case, learning was not as effective as the previous three cases, because of the sequential circuit block (labeled "Counter") involved. This resulted in a sequential dependency of an input waveform at time t on the input waveforms before t, which was not modeled in the learning. This is an issue that requires further research.

4.5 Measuring similarity between two samples

In the experiments above, in each case a waveform is represented as a time-based vector of, say 50 values. In

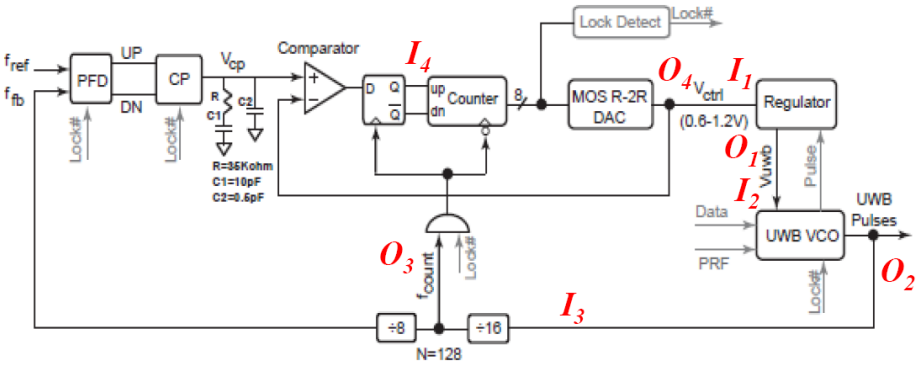

Figure 9: A UWB-PLL example circuit used in the experiments

other words, a waveform is represented as a vector $\vec{a} = [a_1, a_2, \ldots, a_{50}]$ where a_i is the amplitude value at time step i. Given two waveforms $\vec{a}$ and $\vec{b}$, one can apply popular kernel functions described in [18] to measure their similarity. For example, one can use the angle based kernel $cos(\vec{a}, \vec{b}) = (\vec{a} \cdot \vec{b})/(\| \vec{a} \| \| \vec{b} \|)$, or the distance based kernel function $e^{-dist(\vec{a},\vec{b})}$ discussed before. For example, in the experiments we used the distance based kernel function as the output kernel function $k_y()$.

Note that in the experiments, each input waveform x_i is associated with a component sample c_j. Hence, an input sample $s_{i,j}$ should be seen as a 2-tuple (x_i, c_j). And, for two samples (x_{i_1}, c_{j_1}) and (x_{i_2}, c_{j_2}), we need a kernel function $k()$ to measure their similarity $k((x_{i_1}, c_{j_1}), (x_{i_2}, c_{j_2}))$. The way we approached this problem was by defining two separate kernels $k_x()$ and $k_c()$ and let $k() = (k_x() + k_c())/2$.

We defined $k_x()$ as the kernel function to compare two waveforms. This is similar to $k_y()$ discussed above. We then defined $k_c(c_1, c_2)$ for two component samples c_1, c_2 to be based on the output samples obtained by simulating a small number of h input waveforms based on c_1 and c_2. Given c_1 and c_2, let their corresponding output waveforms based on the same input waveforms $x_1, \ldots, x_h$ be f as $y_{1,1}, \ldots, y_{1,h}$ and $y_{2,1}, \ldots, y_{2,h}$, respectively. We then calculated $k_c(c_1, c_2) = (\sum_{i=1}^{h} k_y(y_{1,i}, y_{2,i}))/h$. In other words, $k_c(c_1, c_2)$ is calculated as the average similarity between the output waveforms observed based on the two component samples with the same input waveforms. We then calculate $k()$ as $(k_x() + k_c())/2$.

5. PREDICTING THE OUTPUTS

Next, we will discuss the learning flow proposed in Figure 4. Realization of the learning flow demands two methods. The first method evaluates the predictability of an input sample. The second method learns a predictor from simulation results.

5.1 Intuition behind predictability of SVM

The predictability of modern learning algorithms such as Support Vector Machine (SVM) [17] is explained through the statistical learning theory framework [20]. In this section we will first illustrate the intuition of the predictability notion from the SVM perspective. Then, based on the intuition, we will develop new concepts to capture the notion of predictability in the context of Figure 4.

In a learning problem setting, a set of sample points (x_1, y_1), ..., (x_n, y_n) are given to be learned from (note: each x_i can be a vector, and each y_i is a scalar value). An SVM model is of the form [17]:

$$f(x) = (\sum_{x_i \in SV} \alpha_i k(x_i, x)) + b \quad (1)$$

SV is a subset of the input samples. Without loss of generality, let SV $= \{x_1, \ldots, x_l\}$. Each $x_i \in SV$ is called a *support vector*. $k()$ is the kernel function. Each $k(x_i, x)$ measures the similarity between a support vector x_i and the input vector x to be predicted. The coefficient α_i is the weight associated with $k(x_i, x)$. The learning objective is usually defined for a given prediction error function. For example the error function calculates the sum of square errors: $Err(f) = \sum_{i=1}^{n} (y_i - f(x_i))^2$.

With a kernel $k()$ and an error function given, learning by an SVM algorithm means to determine the following three things: (a) the set SV, (b) the α coefficients, and (c) the constant b in equation (1) above. Because the non-support vectors $x_{l+1}, \ldots, x_n$ are not used in the calculation of the SVM model $f(x)$, they can be seen as the samples used to *validate* the model.

Conceptually, the quantity $\frac{n-l}{n}$ can be thought of as a measure for predictability of the model $f(x)$ [20]. This is because a smaller l (size of the set SV) leads to a larger number of *validation* samples. In other words, if the model can predict more validation samples (based on a given error requirement), then the dataset is more predictable.

Therefore, we make two observations: (1) The essential purpose of learning is to decide how many samples can be predicted by other samples. In a sense, it can be seen as a compression process. (2) When more samples can be predicted by other samples, the predictability for the given dataset is higher.

5.2 SVM models with two support vectors

Suppose a SVM model comprises only two support vectors x_1, x_2. The model becomes $f(x) = \alpha_1 k(x_1, x) + \alpha_2 k(x_2, x)$ (assuming $b = 0$). Assume that the kernel function $k(x, z)$ is a Gaussian kernel $k(x, z) = e^{-\|x-z\|^2}$. Figure 11 shows the behavior of three models for $x_1 = 1$, $x_2 = 2$, and $x = 1.1, 1.2, \ldots, 1.9$.

In the first case $f_1(x)$, we have $\alpha_1 = 4$ and $\alpha_2 = 4$. The predicted y values are shown. Observe that the predicted y value of a x_i is closer to $f(x_1)$ if x_i is closer to x_1. Similar

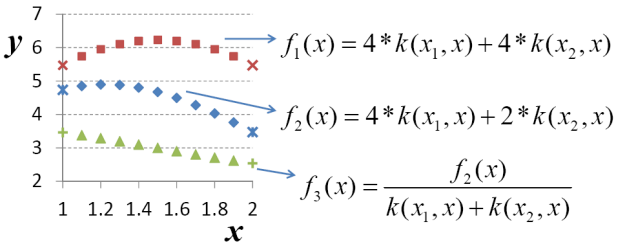

Figure 11: SVM models based on two SVs x_1, x_2

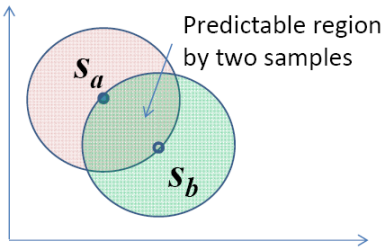

Figure 12: A predictable region

$$\begin{array}{c} s_a \qquad\qquad s_b \\ \begin{vmatrix} k(s_a,s_1)y_a & k(s_b,s_1)y_b \\ \vdots & \vdots \\ k(s_a,s_i)y_a & k(s_b,s_i)y_b \end{vmatrix} \rightarrow \begin{matrix} y_1 \\ \vdots \\ y_i \end{matrix} \end{array}$$

Figure 13: Local dataset to verify a pred. region

observation can be made to x_2. In the second case $f_2(x)$, we have $\alpha_1 = 4$ and $\alpha_2 = 2$. Notice that in this case the largest y value no longer occurs at $x = 1.5$. This is due to the two non-equal weights α_1, α_2.

Observe that by changing α_1, α_2, the model is capable to capture a variety of convex functions. In the third case $f_3(x)$, we take $f_2(x)$ and normalize it with the similarity sum $k(x_1, x) + k(x_2, x)$. This results in a linear function. Hence, if we desire to model a linear behavior between two samples, we can use the normalization method. Figure 11 illustrates that a simple two-SV learning model can implement a variety of interpolation functions between two samples x_1, x_2. We will use this observation to develop a notion of predictability in the context of Figure 4.

5.3 Constructing local predictors

Given a circuit, we partition the circuit into individual circuit elements (CEs). For example, partitioning can be based on design blocks or subjected to user choice. Partitioning allows the learning flow to be applied on individual CEs rather than on the entire circuit.

For learning a predictor for a CE, suppose a set of samples $(s_1, y_1), \ldots, (s_n, y_n)$ has been simulated and is available for the learning. For example, these samples are simulated inputs and outputs obtained during the circuit simulation from the previous iterations, in the iterative learning process of Figure 4.

With a set of the simulated samples, we can try to learn a single model $f(s) \rightarrow y$ for all samples, but this could be difficult if $f(s)$ is restricted to a two-sample model as shown in Figure 11. Hence, instead of learning a single model, we can try to learn a set of *local predictors*, each based on two input samples.

To construct a local predictor, we will first select two input samples s_a, s_b. Assume using the kernel $k() = k_x() + k_c()$ as discussed in Section 4.5 before. We first define a (potentially) predictable region as: $\forall s : k(s, s_a) \leq k(s_a, s_b) \wedge k(s, s_b) \leq k(s_a, s_b)$. Figure 12 illustrates how this region looks like in a 2-dimensional plane - it is simply the intersection of two circles (for $k()$ that is distance-based).

The next step is to extract a local dataset for the learning. Figure 13 illustrates the local dataset consisting of all the input samples $s_1, \ldots, s_i$ that fall inside the predictable region.

Then, the learning is to construct a two-sample model based on s_a and s_b to predict the outputs of $s_1, \ldots, s_i$ within a given accuracy, e.g. by finding the two α's coefficient values in the model. If this can be done, then a local predictor is found. In actual application, a new input sample s is first checked to see if it is inside a predictable region. If it is inside the region of a local predictor, then the output of s is predicted by the local predictor. For a given CE, one can build many local predictors based on many input pairs.

In the proposed approach to apply the learning flow in Figure 4, predictability of an input sample is decided at two levels: (1) The first level is based on information and complexity measures. At this level a circuit element is decided if it is suitable to apply learning. If it is not, then no simulation cost saving is attempted on the circuit element. (2) The second level is based on the predictable region defined by two input samples associated with a local predictor. This region defines when the local predictor can be applied.

5.4 Experimental Result

The experiments were based on the setup discussed in Section 4.4. For the UWB-PLL, 16 circuit elements (CEs) were selected, each with one input and one output. The sizes of the CEs were from 10 to 20 transistors. As discussed in Section 4.4, 5000 waveforms were collected at each input point with corresponding 5000 waveforms collected at the output point. Hence, for each CE we had 5000 samples.

We took the first 1000 sample points as the *training dataset* and used the remaining 4000 sample points as the *validation dataset*. With a training dataset, we selected up to 6K pairs of input samples. For each pair we tried to learn a local predictor. If this succeeded, we counted it as a success. The success rates are reported in the row "Success %." For each CE, the collection of the local predictors were applied to the 4000 samples in the validation dataset to check how many of their outputs were predictable by at least one of the predictors. The percentages of the predictable outputs are reported in the row "Predictable %."

Results are shown in Table 2. The CEs are divided into two groups. The first group comprises CEs of various types. The second group comprises CEs of divider type. In the first group, CEs 6, 7 and 8 are less predictable, reflected in their low success % and low predictable %. Note that a low success % does not imply a low predictable % and vice versa. In the second group, the predictable % numbers are generally high. This shows that divider type of CEs are quite predictable. Note that in the table, the predictable % number could be thought of as a potential saving of the

	Group (1)								Group (2)							
CE index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Success %	54.7	11.7	12.2	55.1	25.8	8.51	9.49	10.9	27.3	32.1	39.5	28.5	24.6	39.1	19.3	17.6
Predictable %	96.3	77.2	75.7	74.2	62.3	26.7	33.5	43.3	95.5	94.5	94.3	93.6	91.9	90	79.4	82.2

Table 2: Summary of results for 16 circuit elements from UWB-PLL

simulation cost on the particular CE. This is because those outputs are predicted rather than simulated.

6. CONCLUSION

In this work, we present two learning flows to be applied in the context of simulation-based analysis. Both learning flows are designed to improve the efficiency of simulation analysis. In the first flow, the importance of input samples is predicted, and efficiency is improved by discarding unimportant input samples in the analysis. In the second flow, the outputs of selected inputs are predicted and simulation cost is saved by not simulating the selected inputs. We explain the machine learning concepts and methods needed to implement the two flows and present experimental results to demonstrate their feasibility. The proposed learning flows are generic and can be applied in different application contexts. Changing from one context to another requires implementation of specific kernel functions suitable for the respective application. The applicability of the proposed learning flows is not yet fully explored and is subjected to future research.

Acknowledgment: The authors thank Samantha Alt, Kou-Kai Hsieh, Sebastian Siatkowski, and Chia-Ling Chang from UCSB for their invaluable help on preparing the paper.

7. REFERENCES

[1] Li-C. Wang, Magdy Abadir. Data Mining In EDA - Basic Principles, Promises, and Constraints. in *ACM/IEEE Design Automation Conference*, 2014.

[2] C. Visweswariah, k. Ravindran, K. Kalafala, S.G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, J. G. Hemmett. First-order incremental block-based statistical timing analysis. in *IEEE Trans. CAD*, v25, 10, 2006, pp. 2170 Ű- 2180.

[3] Xin Li, Jiayong Le, Lawrence T. Pileggi. Statistical Performance Modeling and Optimization. Now Publishers, 2007.

[4] A. Monti, F. Ponci, Member, T. Lovett. A polynomial chaos theory approach to uncertainty in electrical engineering. in International Conference on Intelligent Systems Application to Power Systems, 2005.

[5] F. Augustin, A. Gilg, M. Paffrath, P. Rentrop and U. Wever. Polynomial chaos for the approximation of uncertainties: Chances and limits. in *European Journal of Applied Mathematics*,v 19, 02, 2008, pp. 149-190.

[6] Amith Singhee, Rob A. Rutenbar. Statistical Blockade: A Novel Method for Very Fast Monte Carlo Simulation of Rare Circuit Events, and its Application. in *DATE* 2007, pp. 1-6.

[7] Xin Li, Wangyang Zhang and Fa Wang. Large-scale statistical performance modeling of analog and mixed-signal circuits. *IEEE Custom Integrated Circuits Conference* (CICC), 2012, pp. 1-8.

[8] Xin Li, Fa Wang, Shupeng Sun and Chenjie Gu. Bayesian model fusion: a statistical framework for efficient pre-silicon validation and post-silicon tuning of complex analog and mixed-signal circuits. *IEEE/ACM International Conference on Computer-Aided Design* (ICCAD), 2013, pp. 795-802.

[9] Onur Guzey, Li-C. Wang, Jeremy Levitt and Harry Foster. Functional Test Selection Based on Unsupervised Support Vector Analysis. in *Design Automation Conference*, 2008, pp. 262-267.

[10] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the Support of a High-Dimensional Distribution. in *Journal Neural Computation*, v 13, 7, 2001, pp. 1443-1471.

[11] Po-Hsien Chang, Li-C. Wang, Jayanta Bhadra. A Kernel-Based Approach for Functional Test Program Generation. In *International Test Conference*, 2010, pp. 1-10.

[12] Wen Chen, Nik Sumikawa, Li-C Wang, Jayanta Bhadra, Shaun Feng, Magdy S. Abadir. Novel Test Detection to Improve Simulation Efficiency — A Commercial Experiment. *ACM/IEEE International Conference on Computer-Aided Design*, 2012, pp. 101-108.

[13] Li-C. Wang. Data Mining in Functional Test Content Optimization. in *Asian and South Pacific Design Automation Conference*, to appear, Jan 2015.

[14] Mango C.T. Chao, Li-C. Wang, and Kwang-Ting Cheng. Pattern selection for testing of deep sub-micron timing defects. in *Design Automation and Test in Europe*, 2004, pp. 1060-1065.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning - Date Mining, Inference, and Prediction. *Springer Series in Statistics*, 2001.

[16] Olivier Chapelle, Bernhard Schölkopf and Alexander Zien. *Semi-Supervised Learning*. The MIT press, 2010.

[17] Bernhard Schölkopf, and Alexander J. Smola. Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press, 2001.

[18] J. Shawe-Taylor, and N. Cristianini. Kernel Methods for Pattern Analysis. *Cambridge University Press* 2004.

[19] M. Elzeftawi, *Compact Low-Power Low-Noise Neural Recording Wireless Channel for High Density Neural Implants (HDNIs)*, Dissertation, University of California, Santa Barbara, December 2012.

[20] Vladimir Vapnik. The nature of Statistical Learning Theory. 2nd ed., *Springer*, 1999.

[21] Sayan Mukherjee and Vladimir Vapnik. Support Vector Method for Multivariate Density Estimation. A.I. Memo No. 1653, C.B.C.L. Paper No. 170, MIT 1999.

[22] http://scikit-learn.org/stable/user_guide.html#user-guide