ENABLING HARDWARE RELAXATIONS THROUGH STATISTICAL LEARNING

Zhuo Wang *Naveen Verma*

Department of Electrical Engineering, Princeton University, Princeton, NJ, USA 08544

ABSTRACT

Machine-learning algorithms are playing an increasingly important role in embedded sensing applications, by enabling the analysis of signals derived from physically complex processes. Given the severe resource constraints faced in such applications (energy, functional capacity, reliability, etc.), there is the need to think about how the algorithms can be implemented with very high efficiency. This paper examines the opportunities on three levels: (1) inherent resilience against computational errors, enabling some degree of fault tolerance; (2) top-down training of statistical models using data explicitly affected by errors, enabling substantial fault tolerance; and (3) bottom-up specification of inference kernels based on preferred hardware implementation, enabling reduced hardware complexity. Implementations employing the last two approaches are proposed and evaluated through hardware measurements and simulation.

Index Terms— Embedded Systems, Low-energy Design, Sensing Systems, Statistical Learning, Hardware Reliability

1. INTRODUCTION

Electronic systems have enabled high-value functions in a broad range of applications. The explosion in embedded sensing technologies leads to a natural progression, where such functions can now be applied to all forms of physical signals. This enables the application of electronics in ways and on scales much greater than before, but it also raises critical challenges. We identify two challenges that are particularly noteworthy: (1) generally, physical signals are derived from diverse and complex processes, such that specific information of interest is represented within complex correlations; and (2) embedded operation imposes severe constraints on system resources (energy, communication bandwidth, form factor, etc.).

The first challenge is algorithmic. Powerful algorithmic tools, particularly from the domain of machine learning, have emerged that are increasingly being applied in embedded sensing applications [1]. By enabling data-driven methods of constructing models for signal analysis, these allow high-quality inferences to be derived even when signals are too complex to model or understand analytically. With regards to the second challenge, however, an important problem arises, which is that machine-learning algorithms have generally not been optimized or even viewed from the perspective of system resource constraints. In fact, studies have shown that straightforward mapping of machine-learning algorithms to implementations can lead to energy- and hardware-intensive systems [1].

The purpose of this paper is approach the algorithmic capabilities enabled by machine learning with first-order consideration of resource-constrained implementation. In fact, we find that statistical learning enables the possibility to accommodate non-ideal behaviors in the implementation which have traditionally required substantial resources to resolve when left to the implementation level (namely via circuits/device optimizations). To couple principles with system realizations, this paper focuses on applications that perform classification on sensor data (pointing out that classification/recognition represents a particularly important class of functions in emerging sensing applications [2]). The remainder of this paper is organized as follows. The next section discusses statistical learning, abstractly, from the perspective of system resources, and proceeds to identify three major levels on which substantial relaxations can be achieved to system implementation. The next two sections explore two of these levels in detail, quantitatively showing the impact possible in systems.

2. STATISTICAL LEARNING IN THE CONTEXT OF SYSTEM RESOURCES

2.1. System Resources

System resources refer to elements that are finite and consumed or occupied when performing desired functions. Resources of primary concern in embedded systems include energy, communication bandwidth, functional capacity (amount of hardware and/or processing throughput of hardware), etc. The resources are generally highly interrelated and coupled through system tradeoffs. For instance, constraints on communication bandwidth can be alleviated by local processing (compression) of data, requiring energy be expended to reduce and derive the results. Recently, another constraint of high importance has emerged, namely reliability. Reliability again exhibits strong coupling with other system resources. In fact, reliability challenges have primarily come about due to aggressive scaling of CMOS technology, which is motivated by benefits that result in energy and functional capacity. Through continued CMOS scaling, reliability concerns have elevated to the point of being identified by the International Technology Roadmap for Semiconductors (ITRS) as one of the most important challenges facing the semiconductor industry [3]. Looking beyond CMOS, the adoption of emerging technologies (e.g., tunneling FETs, carbon nanotube FETs, etc.), which are being considered to continue the benefits to energy and functional capacity [4], are also primarily limited by reliability concerns in manufacturing [5].

Though all are highly interrelated, the primary system resources focused on in this paper are energy and reliability, due to their importance in embedded systems. We briefly consider the question of how energy and reliability can be addressed within a system. With regards to energy reduction, several approaches have been proposed, ranging from the circuit level (e.g., reduction of the supply voltage [6]) to the algorithmic level (e.g., transformations to reduce the computations required for implementing a given function [7]). Both approaches have proven highly effective but are ultimately limited, by

Support is provided by SRC, NSF (CCF-1253670), as well as Center for Future Architectures Research (CFAR) and Systems on Nanoscale Information fabriCs (SONIC), two of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

the need for finite transistor threshold voltages in the former case, and by the extent to which algorithmic transformations can be applied generically in the latter case. An alternate approach to energy reduction is to implement a given function by explicitly choosing kernels that are easier to implement with the underlying technology (e.g., MOSFETs). By exploiting the physics that the technology presents as a means of implementing complex kernels, particular kernel functions can potentially be implemented with very high efficiency [8]. The problem is that the particular kernels that are efficiently implemented with the technology may not address the function required. In this paper, we explore how statistical learning can in fact enable high-performance *application-level functions* (namely classification) while also enabling a high-level of flexibility in the precise *kernels* we are required to implement.

With regards to enhancing reliability, again many approaches have been explored from the circuit level (e.g., transistor up sizing [9]) to the algorithmic level (e.g., employing coarse estimators to detect occasional errors in hardware blocks [10]). In fact, algorithmic approaches have shown great promise, primarily because they target application-level specifications for reliability rather than expending resources on addressing low-level failures, whose manifestations can be made tolerable or inconsequential. Such approaches emphasize that the low-level data being processed by a system implementation does not map directly to the application-level information of interest. This leads to the question of how much failure on the implementation level (in the form of low-level hardware faults) can be incurred before the information of interest in an application is degraded. In this paper, we explore how statistical learning can be used to achieve system performance that corresponds with the *fundamental level of information* retained in the processing being performed by a system implementation.

2.2. Abstract Implementation of Machine-learning Algorithms

Generically, inference using a machine-learning algorithm consists of two functions: (1) construction of a model, via statistical learning over (previous) observations of data; and (2) the application of that model within a kernel function to derive inference over (real-time) observations of data. To bring the focus to systems that will be evaluated in this paper, Fig. 1 shows a generic system for classification based on a supervised machine-learning algorithm. Model construction is performed by the trainer and the class membership is derived by the detector. Both the trainer and the detector employ feature-extraction stages, which pre-process input data to enhance pattern recognition (e.g., by reducing the data dimensionality over which statistical learning must be performed, by computing a representation that is invariant to signal parameters not of critical importance, etc.). The trainer takes as input a training set, consisting of data instances (that adequately represent the statistics of the data over which inference is to be performed) as well as labels corresponding to the true class membership of the training data instances, and provides as output a model in some form that can be used by a kernel function to perform classification. The detector takes as input the model and testing data, and provides as output a class decision.

An important observation in typical sensing applications is that input data exhibits stationary statistics, or at least statistics that vary slowly compared to the rate of detection decisions required. This implies that training can be performed infrequently and slowly (i.e., with some level of reduced functional capacity). Thus, resources expended on training are of lesser concern since these can be amortized over the long term operation of the system. On the other hand, detection is performed continuously and possibly under real-time constraints. Thus, resources expended on detection are of primary concern. Of course, in practical systems, implementation of the trainer is a concern, and the resources required can not all be readily amortized (for instance, minimal hardware to enable processing of training data must be available and may not be easily repurposed after training). Accordingly, in this paper we will briefly consider implementation solutions and challenges related to the trainer; however, our primary focus will be on implementing the detector with minimal resources, by exploiting statistical learning in the trainer. Opportunities available for resource-efficient implementation of the detector are identified below on a high level, before going into details.

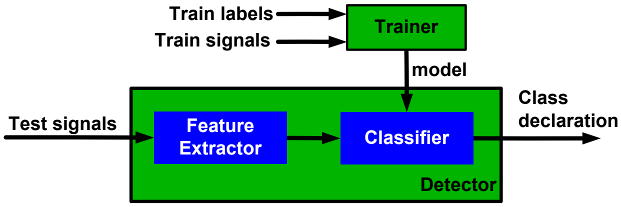


Fig. 1. Generic supervised machine-learning algorithm.

2.3. Resource-efficient Implementations

Three opportunities for resource-efficient implementation of sensing and inference systems are identified:

1. **Inherent Resilience.** Applications focusing on high-level inferences exhibit inherent resilience against some degree of low-level faults. In fact, machine-learning algorithms often afford an additional level of resilience due to their computational structures. Inherent resilience comes about because, generally, high-dimensionality input data is employed to derive low-dimensionality decisions. This raises the possibility of redundancy in the input data (e.g., in the form of correlation between input features) which can mitigate the impact of errors. Machine-learning algorithms may introduce additional resilience because they often employ many low-level computations towards derivation of a high-level result. For instance, in a support-vector machine (SVM), an inner product is computed between a feature vector and multiple support vectors, making occasional errors tolerable [11]. As we show in the following section, however, substantial potential beyond inherent resilience exists, for instance, through the next two opportunities.

2. **Data-driven hardware resilience (DDHR).** The process of statistical learning establishes a model that can be utilized in a particular kernel function to perform inference on input feature data. Two aspects are critical in determining the integrity of the inference: (1) the extent to which the training set used to derive the model is representative of the input feature data; and (2) the extent to which the input feature data retains the information required for computing the inference. This means that through statistical learning, complex perturbations in the feature-data statistics, which might come about due to low-level failures (faults) in the implementation, are acceptable provided these aspects are ensured. We refer to this approach as data-driven hardware resilience (DDHR) and show that it substantially enhances reliability in the face of hardware faults, beyond the level inherently achievable.

3. **Hardware-driven-kernel learning (HDKL).** While DDHR exploits statistical learning to train a model that can accommodate errors when computing a particular kernel function, another opportunity raised by statistical learning is in the choice of kernel function itself. In particular, model training pertains to a particular kernel function; appropriately adapting training can enable the use of alternate kernel functions that are preferred from the perspective of resource-efficient implementation. Thus, while DDHR primarily addresses reliability in a top-down manner, HDKL addresses energy by making the kernel function required an implementation-level choice in a bottom-up manner. As we will describe, the chosen kernel may limit the achievable strength of classification in terms of its ability to fit the data statistics during training. However, this can be addressed through the approach of boosting, whereby strong classification can be achieved from an ensemble of weak classifiers, under specific conditions that are typically easily met.

In the following sections the approaches of DDHR and HDKL are described, along with specific implementations. Experimentation, based on FPGA emulation and transistor-level simulation, is pursued by considering applications.

3. DATA-DRIVEN HARDWARE RESILIENCE (DDHR)

This section starts by providing an overview of DDHR, and then analyzes a system implementation that first address faults in the feature-extraction stage of a detector. This helps analyze the approach. Then, a practical system implementation is described that applies DDHR to both the feature-extraction stage and the classification stage, enhancing system-wide reliability. In addition to the detector, implementation of the trainer is also considered.

3.1. Approach

To illustrate the concept and a possible system implementation, Fig. 2 shows a generic DDHR system [12]. The system consists of feature-extraction processors, which are assumed to be fault affected, and classification kernels as well as a trainer (implemented on an embedded microcontroller), which for the moment are both assumed to be fault free. In practice, protection against faults can be achieved through an implementation that expends resources (for instance, higher energy by raising the supply voltage [9], or lower functional capacity by reducing lithographic density [13]). For reference, equivalent gate-count numbers are shown from an application implementation described below. As seen, the fault-protected blocks account for a small portion of the detector (∼7.0%), and later we show how this can be reduced much further (to ∼2.6%). As mentioned, training is performed with low duty cycle (just once for each classification kernel, in the implementations below), so its hardware is shared and its energy is amortized over the operation of the rest of the system.

The key to DDHR is constructing a model using training data that has explicitly been affected by errors manifesting due to non-idealities (such as variations and faults) affecting a particular instance of hardware [12]. This leads to *an error-aware model*. While, generally, low-level hardware non-idealities cause unpredictable errors, by training the model to error-affected data that comes about due to the particular occurring non-idealities, accurate inferences based on that model can be derived. A key requirement is that the data-statistics must be consistent during training and detection, and thus, generally, the non-idealities cannot be time varying. For illustration, measured data from an implemented application are shown in Fig. 3. As seen, errors (emulated using an FPGA) cause the feature-vector distributions to be altered with respect to the error-free distributions. However, provided that the class separation is maintained, high classification performance can be restored by training the error-aware model to the new distributions.

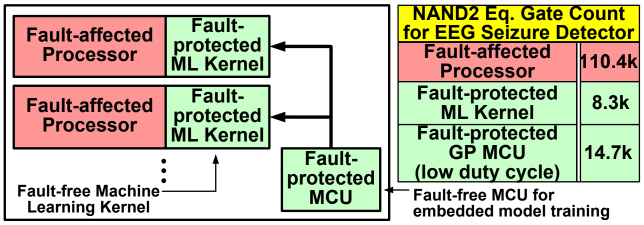

Fault-affected Processor
Fault-protected ML Kernel
Fault-affected Processor
Fault-protected ML Kernel
Fault-free Machine Learning Kernel
Fault-protected MCU
NAND2 Eq. Gate Count for EEG Seizure Detector
Fault-affected Processor
110.4k
Fault-protected ML Kernel
8.3k
Fault-protected GP MCU (low duty cycle)
14.7k
Fault-free MCU for embedded model training

Fig. 2. DDHR leverages minimal fault-protected blocks for system resilience.

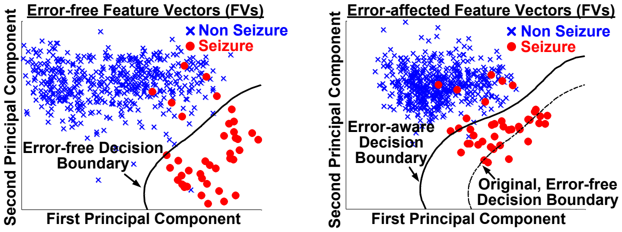


Fig. 3. Error-aware model trains to data affected by hardware non-idealities (data is measured from seizure-detection system).

3.2. Experimentation and Analysis

To demonstrate and analyze the DDHR approach, two system implementations are pursued: (1) an EEG-based seizure detector; and (2) and ECG-based cardiac arrhythmia detector. In the seizure detector, the feature-extraction stage corresponds to a bank of FIR filters and accumulators to compute the spectral-energy distribution of each EEG channel over a 1sec. epoch, and the classification stage corresponds to a support vector machine (SVM). In the arrhythmia detector, the feature-extraction stage corresponds to a cascade of FIR filters for computing a wavelet transform of the ECG signal, and the classification stage corresponds to an SVM [12].

3.2.1. Experimentation Approach

For each application, faults are introduced into the feature-extraction stage, by synthesizing a register-transfer-level (RTL) design description to a gate-level netlist, and then editing the netlist to insert error-control gates on randomly chosen nodes, as shown in Fig 4. The error-control gates emulate stuck-at-0/1 faults, which are widely-used fault models for a range of static defects at the hardware level. The netlists with error-control gates are then mapped to an FPGA. The signals $errorVal[i]$ and $errorCtrl[i]$ enable configuration both of the fault rate (i.e., number of nodes affected by stuck-at condition) and the precise fault-affected nodes. Detailed characterization is thus pursued by scaling the fault rate and testing multiple randomized fault configurations (i.e., affected nodes) at each rate.

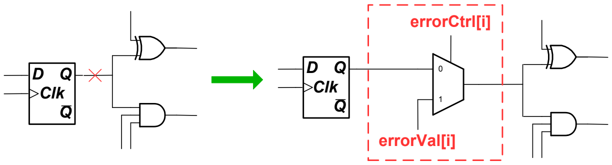


Fig. 4. Stuck-at faults achieved by introducing error-control gates (multiplexers) on randomly-selected outputs.

3.2.2. *Experimentation Results*

Following FPGA-based experimentation of the systems, analysis of the feature-vector errors is performed. First, the bit-level errors are considered. As shown in Fig. 5a and d, the bit-level errors for both systems are high, with bit-error rates (BERs) in the range of 10-50%. We also find that the errors are large in magnitude, affecting high-order bits with equal probability as ow-order bits. However, the overall performance of the systems is shown in Fig. 5b,c,e,f. We see that without an error-aware model, the performance rapidly degrades, suggesting that the inherent reliability of machine-learning algorithms is limited in the face of practical faults occurring on a substantial level. On the other hand, an error-aware model, constructed using the DDHR approach, substantially and consistently restores performance to very high fault rates.

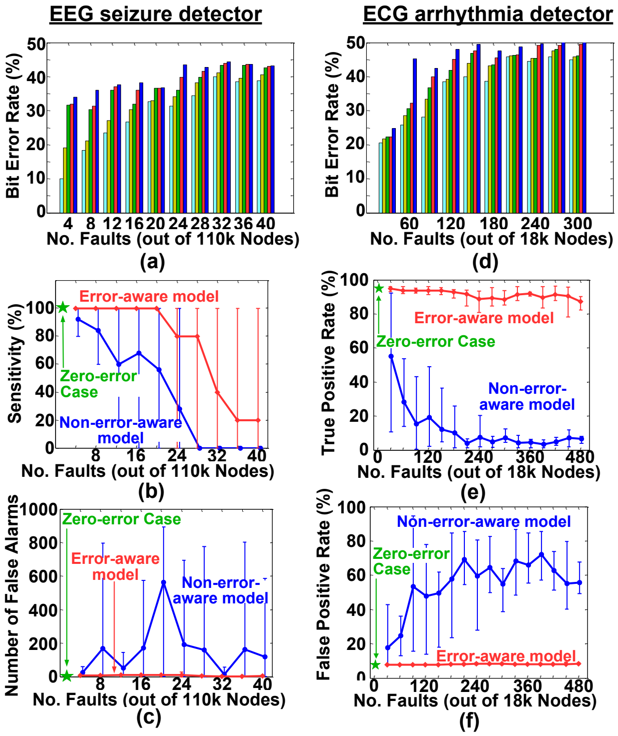


Fig. 5. Bit-error statistics of the computed feature vectors for different fault instantiations for (a) EEG seizure detector and (d) ECG arrhythmia detector; Performance of the systems with respect to the fault rates, for the cases with and without DDHR for (b)(c) EEG seizure detector and (e)(f) ECG arrhythmia detector.

3.2.3. *Analysis Based on Mutual Information*

Following from the system performance observed, we seek to understand what limits the performance of DDHR, if not bit-level errors. In fact, what we find is that, with information for classification embedded in the class distributions of the feature data, bit-level errors do not have direct correspondance with performance. Bit-level errors merely alter the distributions, changing the way that information is encoded. Thus, through statistical learning, the new encoding can be appropriately modeled. What ultimately limits performance is the level of information for classification that is retained in the new distributions. To quantify this, we compute the mutual information between the error-affected feature vectors and the class membership, over all of the fault configurations tested. Fig. 6 shows scatter plots of the system performance versus the mutual information for both systems (the systems are trained to have true negative rate of 98% and 95% respectively, so that comparison can be performed based on true positive rate). As seen, the system performance achieved corresponds strongly with mutual information. This suggests that, rather than being limited by bit-level error metrics, DDHR is able to achieve performance limited more fundamentally by an information-level metric.

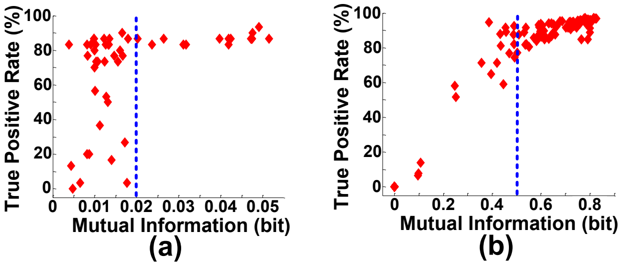


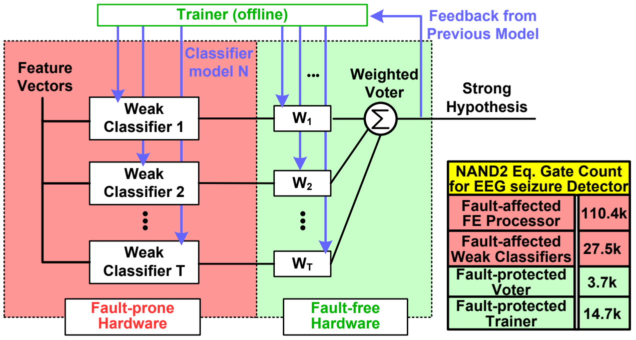
Fig. 6. Mutual information between error-affected feature data and feature-data class membership shows high correspondance with system performance for (a) seizure detector and (b) cardiac-arrhythmia detector.

3.3. Error Adaptive Classifier Boosting (EACB)

Though illustrative, the DDHR systems above are limited in that they require a fault-protected classification stage for applying the error-aware model. Generally, DDHR is able to address errors that occur before the stage at which the error-aware model is applied. Thus, it may appear that the classification kernel itself is required to be fault protected. However, this can be overcome by employing an algorithm that decomposes the classification kernel into stages wherein statistical learning is applied successively. The classification algorithm we focus on is Adaptive Boosting (AdaBoost), which leads to an approach within the classifier referred to as error-adaptive classifier boosting (EACB) [14].

3.3.1. *EACB Approach*

AdaBoost forms a strong classifier by combining an ensemble of weak classifiers [15]. In typical applications of AdaBoost, weak classifiers are limited by simple kernels that are restricted in their ability to fit to training data. In EACB, the weak classifiers are further limited by errors due to non-idealities in the hardware implementation. Fig. 7 show the algorithmic architecture for an EACB classifier, consisting of the following: (1) T fault-affected weak classifiers; (2) a fault-protected voter, implemented as a T-input signed adder (inputs and sign bits correspond to the classifier weights and outputs, respectively); and (3) a fault-protected trainer, which is required infrequently and whose implementation, based on a simple microcontroller, is described below. Within this architecture, the weak-classifiers are trained in successive stages, called iterations. During each iteration, the classification outputs from all previously-trained weak classifiers are used. This enables the current classification model to adapt to errors in the previous classifiers, thus enabling errors to be overcome. A key benefit from the theory of AdaBoost is that the weak classifiers are required to perform only marginally better than 50/50 guessing, raising the opportunity for substantial enhancement to reliability [15].



NAND2 Eq. Gate Count for EEG seizure Detector	
Fault-affected FE Processor	110.4k
Fault-affected Weak Classifiers	27.5k
Fault-protected Voter	3.7k
Fault-protected Trainer	14.7k

Fig. 7. Algorithmic architecture of error adaptive classifier boosting (EACB).

3.3.2. *EACB Demonstration and Characterization*

The EACB concept is demonstrated for the EEG-based seizure detector. The weak classifiers are implemented as decision trees, which are commonly used within AdaBoost [15]. Various decision-tree configurations have been analyzed (1-node trees, 4-node trees, 7-node trees) [14], and results provided here are for 1-node trees (stumps) for illustration. As before, the system is implemented by synthesizing an RTL design description to a gate-level netlist, inserting error-control gates, and mapping to an FPGA for emulation. The equivalent gate counts from the implementation are shown in Fig. 7, implying that the fault-affected hardware in the detector can now account for over 97% of the total hardware.

Measured system performance is shown in Fig. 8. As seen previously, without DDHR the performance degrades rapidly. However, with DDHR now applied in the classification stage (through EACB) the performance is restored to high fault rates.

3.3.3. *Low-complexity Trainer*

Since training occurs infrequently (one time, in the system demonstrations above), some of the resources it expends (e.g., energy) can be amortized over the real-time operation of the detector. However, some resources (e.g., special hardware required) must be included in the system, and, generally speaking, cannot be repurposed for use in the detector. This results in unrecoverable overhead. For instance, we consider trainer implementation via software executing on a simple microcontroller, typically available in a system for general-purpose functions. For the system implementation considered, the microcontroller is an OpenMSP, with roughly 14.7k equivalent gate count. A software implementation, which is both energy intensive and unable to run in real-time, is acceptable for the trainer since the energy can be amortized and real-time operation is not required. However, ensuring low generalization error during training requires using a training set of adequate size, leading to large embedded-memory requirements. To overcome this, [14] presents a training algorithm that exploits training over multiple iterations, by acquiring a new training set at each iteration in order to enhance training set diversity without increasing the instantaneous memory required. As a result, the training memory is reduced by >65× and training energy is reduced by >10×, compared to conventional AdaBoost. For the seizure-detection system considered, training is achieved via the OpenMSP with 6.4 kB of memory and 5 M clock cycles.

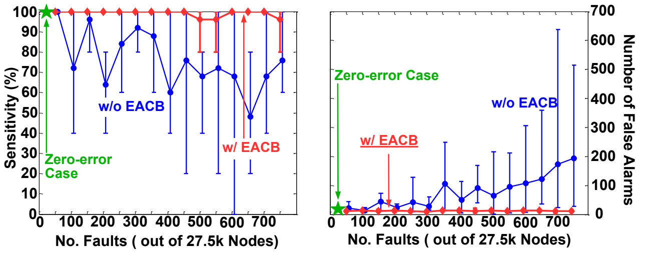


Fig. 8. Seizure-detector performance with and without EACB (error bars show worst/best performance over five fault-injected instantiations).

4. HARDWARE-DRIVEN-KERNEL LEARNING (HDKL)

Traditional system design follows a top-down discipline, where an algorithm is first defined and an implementation, falling within specifications, is then developed by employing standard topologies for computation. Though this leads to a generalized methodology for system implementation, it can be resource intensive because the standard topologies do not necessarily correspond to kernels that are most easily implemented in the technology. Hardware-driven-kernel learning (HDKL) is derived from the recognition that statistical learning raises the opportunity to choose the precise kernel function through which a model is applied. This enables us to select the kernel function for implementation in a bottom-up manner, specifically based on resource-efficient implementation in a technology.

4.1. Approach

To present HDKL in more concrete terms, we consider the implementation of a particular classification kernel. In the following discussion, bold-face font represents vectors, while regular font represents scalars. A linear classifier is a widely-used kernel. Given in Eq. 1, the kernel function involves a dot product between an input feature vector $\mathbf{x}$ and a decision vector $\mathbf{w}$, corresponding to linear coefficients derived from training.

$$h_{\mathbf{w}}(\mathbf{x}) = sign(\mathbf{w}\cdot\mathbf{x}+b) = sign(\sum_{j=1}^{m} w_j \cdot x_j + b) \qquad (1)$$

Though an implementation based on digital multipliers can be developed, Fig. 9 considers a simpler implementation formed from two series-connected transistors. This results in a function involving two input variables, voltages v_{xj} and v_{wj}, and an output variable, current i_j. The feature-vector elements x_j can be mapped to the analog voltage v_{xj} in any manner chosen (noting that this also has the system benefit of avoiding the need for analog-to-digital conversion of x_j, which will typically correspond to a sensor output); we consider a linear mapping within the allowed voltage range of the transistors. Thus, what remains is to find an analog voltage v_{wj} corresponding to the decision-vector elements w_j. Generally, w_j can take on positive or negative values from training. To accommodate this, a differential implementation is used, wherein each w_j is mapped to a pair of inputs v_{wj+} and v_{wj-}, with one of these set to $0V$ (giving nearly no output current) depending on the desired sign.

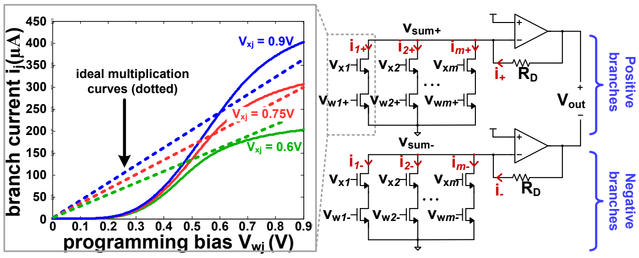


Fig. 9. Implementation of a classification kernel that exploits the transfer function of a circuit based on the series connection of two MOSFETs (transfer function shown is for implmentation in a 32nm CMOS technology).

4.1.1. *Inference based on the Implementation*

The function $i_j(v_{xj}, v_{wj})$ implemented by the circuit is shown from simulation of transistors in a 32nm CMOS technology. As seen, the implementation gives a poor approximation of a multiplier. Indeed training this kernel function as a linear classifier, will result in low performance. However, the training algorithm can be adapted to accommodate the kernel function. In particular, training of a linear classification kernel is often performed via an algorithm such as linear regression. This involves optimizing the objective function shown in Eq. 2, where $\mathbf{x}^{(k)}$ and $y^{(k)}$ represent the k^{th} training example and label, respectively. We note that this objective function, corresponding to a dot-product kernel, leads to convex optimization, which can be solved with low computational complexity.

$$min_{\mathbf{w},b} \sum_{k=1}^{n} (\mathbf{w} \cdot \mathbf{x}^{\mathbf{(k)}} + b - y^{(k)})^2 \tag{2}$$

However, for the implementation shown in Fig. 9, rather than classification based on a dot-product kernel, the classification computation has the form given in Eq. 3. This is based on the circuit transfer function obtained, and leads to a new objective function given in Eq. 4. Here, we have added the scaling parameter a to ensure that the v_{wj} values derived from training fall within the allowed range for the transistor gate voltage.

$$h_{\mathbf{v_w}}(\mathbf{v_x}) = sign(a \cdot i(\mathbf{v_w}, \mathbf{v_x}) + b) = sign(a \cdot \sum_{j=1}^{m} i_j(v_{wj}, v_{xj}) + b) \tag{3}$$

$$min_{\mathbf{v_w},w,a} \sum_{k=1}^{n} (a \cdot i(\mathbf{v_w}, \mathbf{v_x}^{(k)}) + b - y^{(k)})^2 \tag{4}$$

4.1.2. *Training the Kernel*

Having developed an appropriate objective function, the next question is how to solve this to arrive at the parameters $\mathbf{v_w}$ required within a particular detector. The solution is complicated by three factors: (1) the analytical form of $i(\mathbf{v_x}, \mathbf{v_w})$ is unknown, based on potentially complex circuit-level behaviors; (2) the objective function is no longer convex, since $i(\mathbf{v_x}, \mathbf{v_w})$ is not convex with respect to $\mathbf{v_w}$; (3) transistor-level variations render the output current i a probability distribution with respect to $\mathbf{v_w}$ and $\mathbf{v_x}$. Below, we discuss solutions to each of these challenges.

To accommodate the complex analytical form taken on by $i(\mathbf{v_x}, \mathbf{v_w})$, we pursue a numerical approach wherein $\mathbf{v_w}$ and $\mathbf{v_x}$ are discretized to retain corresponding values of i obtained either from measurement or simulation. A numerical approach of this manner has the benefit of addressing arbitrary kernels that may be chosen based on hardware implementation. Of course, sufficient granularity of discretization is required for accuracy. In practice, we find this is easily achieved, particularly since hardware variations can be substantial, and granularity beyond the level of variations is not required (though, as described below, variations must be appropriately handled). However, even with modest granularity, an exponentially increasing number of discretized values of i will need to be measured and retained as the $\mathbf{v_w}$ vector dimensionality increases. This necessitates important design considerations for the kernel-function implementation. In particular, in the implementation of Fig. 9, the two-transistor branches yield an output current, which in general depends on the series voltage $V_{sum+/-}$, thus leading to an interdependence between all branch currents $i_j(v_{wj}, v_{xj})$. However, employing an operational amplifier generates a virtual ground at $V_{sum+/-}$, making the series voltage invariable, breaking the interdependence. Thus only a single numerical representation of $i_j(v_{wj}, v_{xj})$ must be measured and retained, and the overall current i can be represented as a sum over the individual branch currents i_j.

Even though the above approach simplifies definition of the objective function, optimization must still be performed over a discretized space that increases exponentially with the $\mathbf{v_w}$ vector dimensionality. With the objective function now defined numerically by the current from the two-transistor implementation, we are faced with non-convex optimization. So, rather than efficient analytical approaches such as those possible for training a linear classifier (Eq. 4), training now takes on minimum-value search over a potentially large space in order to find the optimal $\mathbf{v_w}$. In particular, with l quantization levels and m dimensionality, the search space is l^m. In the demonstration below, with $l = 121$ and $m = 6$, this amounts to $3e12$ points, making optimization computationally expensive. Fortunately, however, for practical hardware implementations, such as that in Fig. 9, the kernel function, and thus the objective function, is likely to be smooth with respect to each v_{wj}. This means points in the space can be sampled, initially with coarse granularity, but with successively increasing granularity by 'zooming-in' on the sampled minimum [16]. Done in stages, the computational complexity thus reduces to $\sim r\, l^{\frac{m}{r}}$, where r corresponds to the number of successive stages of searching.

The remaining challenge is variations inherent in the implementation. For instance, transistor variations make the function shown in Fig. 9 stochastic. This in fact matches the challenge addressed by DDHR. In particular, using EACB with the kernel enables boosting in a manner that adapts to the errors that result due to the particular variations affecting the implementation. Transistor variations are considered in the demonstration below.

4.2. HDKL Demonstration

To demonstrate the HDKL approach, simulation is pursued of the kernel-function implementation in Fig. 9. For evaluation, comparison is performed of three system approaches:

1. **DDHR via EACB.** Given a particular intended kernel function, DDHR attempts to address deviations in the output that occur due to faults in the kernel-function implementation. An implementation resulting in a different kernel-function from that intended can be viewed as causing deviations of the output. As we show, however, such severe deviations from the intended kernel result in low performance.
2. **EACB with hardware-driven input variable mapping.** One might assume that different kernel functions can be handled simply by mapping the intended kernel-function inputs to those of the implemented kernel function through a non-linear function based on the implementation (such as that in Fig. 9). Then, training may be proceed assuming the intended kernel function. We show, however, that this is insufficient since training now presumes a relationship that does not hold when the kernel is presented with inputs $\mathbf{v_x}$ that are different from those in the training set.
3. **EACB with hardware-driven-kernel learning (HDKL).** HDKL enables training based on the implemented kernel function, and EACB enables deviations in the kernel-function outputs due to hardware variations. In addition to this, if the implemented kernel results in a weak classifier, unable to fit to the application data statistics, boosting via EACB enables realization of a strong classifier, thus generalizing the HDKL approach for applications.

4.2.1. Demonstration Approach

To demonstrate the approaches, corresponding trainers are developed in MATLAB. For the first two cases, the trainer employs an objective function based on an ideal linear-classification kernel function (Eq. 2), solvable analytically. For the last case, the trainer employs an objective function based on the series-transistor kernel function (Eq. 4), solvable numerically.

Simulation of the kernel-function implementation (shown in Fig. 9) is performed based on a 32nm CMOS technology. While training is performed assuming a nominal kernel function, practical implementations of the kernel function employed in a detector are susceptible to transistor variations. To account for variations, a statistical model of the kernel-function implementation is developed in MATLAB so that detector simulation with a large dataset can be performed (full statistical simulation at the circuit level is not viable with a large dataset). To develop a statistical model of the kernel-function implementation, Monte Carlo simulations of a transistor are first performed using a foundry-provided statistical transistor model, wherein a number of parameters are varied. The resulting output current distribution is then mapped to a gate-source voltage distribution through the nominal saturation-regime transistor-current characteristic. This leads to an approximately normal gate-source voltage distribution [9], which can thus be employed within the MATLAB model to simulate statistical variations in the kernel-function implementation. The result of a 1k-point Monte Carlo simulation of the transistor current characteristic is shown in Fig. 10a, where we see a roughly constant deviation of gate-source voltage for a given drain current, suggesting that referral of the variation to a gate-source voltage deviation is a reasonable approach over a large range of gate-source voltage inputs. Fig. 10b shows the resulting distribution of gate-source voltage deviation observed at a mid-range drain current of $I_{out} = 750\mu A$. As shown, fitting to a normal distribution can be achieved. The statistical MATLAB model of the kernel-function implementation is thus developed by employing the simulated current transfer function for the series connected transistors (Fig. 9), with deviations applied to the nominal v_{xj} and v_{wj}, drawn from the approximated normal distribution.

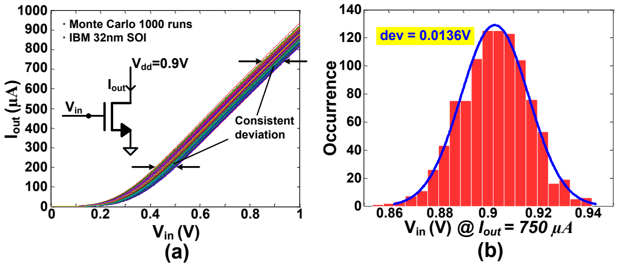


Fig. 10. Monte-Carlo simulation approach for generating a statistical model of the kernel-function implementation based on (a) characterizing transistor variations in IBM 32nm technology, and (b) referring these to gate-source voltage deviations modeled via a normal distribution.

4.2.2. Demonstration Application

The application used for demonstration is image detection based on a 6×6 array of pixels. The pixel data corresponds to the images show in Fig. 11. The dataset consists of 150 instances composed of five shapes (cross, el, ring, tetris, triangle) under 30 different contrast and brightness conditions. For training and testing, instances are randomized and 5-fold validation is employed. Five one-versus-all binary detectors are trained for the five shapes. The performance metrics are the true-positive/-negative rates. For feature extraction, the 36 sensor channels are ranked based on their correlation with the class membership, and the top six channels are selected.

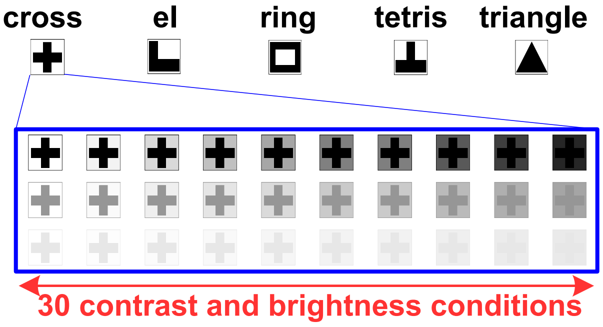


Fig. 11. Image dataset, consisting of various shapes and background/lighting conditions.

4.2.3. Demonstration Results

Fig. 12 shows the results obtained for the three approaches considered, with up to 6 iteration of EACB boosting. For reference, the performance is also shown for an ideally-implemented (double-precision, variation-free) detector based on an SVM classifier (with radial-basis-function kernel). We see that, in all cases, HDKL results in high performance (near that of an SVM) with very few iterations (usually one iteration). The performance achieved is substantially beyond that of the other two approaches.

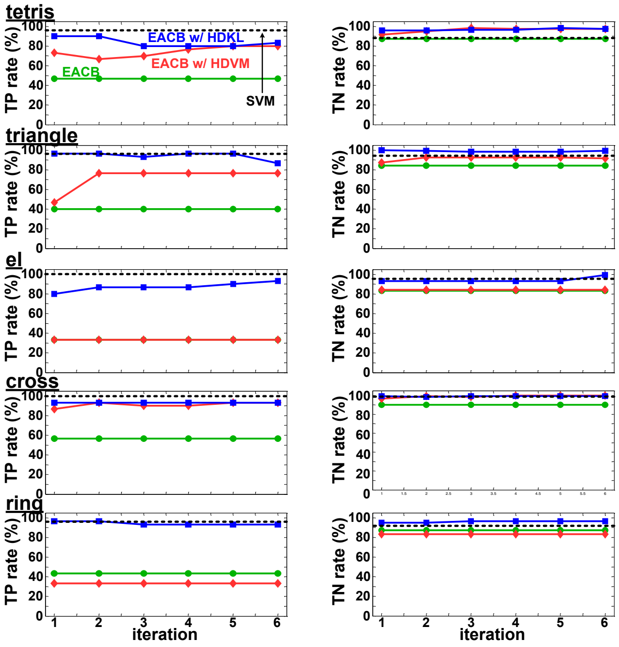


Fig. 12. One-versus-all classification results for the five shapes, respectively.

5. CONCLUSIONS

Given the increasing importance of machine-learning algorithms in resource-constrained sensing applications, this paper examines the possibilities raised by the algorithms for enabling efficient implementations. Two approaches are discussed in detail: data-driven hardware resilience (DDHR) and hardware-driven-kernel learning (HDKL). These take different approaches to addressing implementation challenges. DDHR takes a top-down approach, aiming to adapt an inference model to errors that manifest due to non-idealities in the implementation of a pre-specified kernel function. On the other hand, HDKL takes a bottom-up approach, aiming to adapt the inference model to enable specification of a new kernel function driven by implementation considerations. Generally, the function being implemented can strongly impact energy and complexity. Thus HDKL has potential to substantially address implementation resource constraints. Both DDHR and HDKL are demonstrated in systems, showing that the aims are effectively met in each case. However, several challenges and oportunities arise. With DDHR, the primary question is how to realize generalized architectures for training, particularly in the context of supervised-learning algorthms where training labels must be provided along with implementation-specific error-affected data. A possible architecture that addresses a class of applications is presented in [14], where estimated labels are derived on line. With HDKL, in general, the implemented kernels do not lead to convex or analytical objective functions for training the inference model. This causes increasing computational complexity with respect to the number of features and their granularity of numerical sampling. However, typically, the implemented kernels do lead to smooth objective functions, opening the possibility for computationally-efficient non-convex solvers. One such approach is desribed. Nonetheless, we view these and similar directions as offering great promise. Research activities in these areas are likely to have substantial impact in advancing embedded sensing systems and broadening the applicability of machine-learning algorithms.

6. REFERENCES

[1] K. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," *J. Solid-State Circuits*, vol. 48, no. 7, pp. 1625–1637, 2013.

[2] P. Dubey, "Recognition, mining and synthesis moves computers to the era of tera," *Tech. at Intel Magazine*, 2005.

[3] L. Wilson, "International technology roadmap for semiconductors (ITRS)," 2013.

[4] A. Balijepalli, S. Sinha, and Y. Cao, "Compact modeling of carbon nanotube transistor for early stage process-design exploration," in *Int. Symp. on Low Power Electronics and Design (ISLPED)*. IEEE, 2007, pp. 2–7.

[5] M. Bohr, "New era of scaling in an SoC world," .

[6] A. Wang and A. Chandrakasan, "A 180mV FFT processor using subthreshold circuit techniques," in *Int. Solid-State Circuits Conf.*, 2004, vol. 1, pp. 292–529.

[7] K. Lee, S. Kung, and N. Verma, "Improving kernel-energy trade-offs for machine learning in implantable and wearable biomedical applications," in *Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2011, pp. 1597–1600.

[8] R. Sharpeshkar, "Ultra low power bioelectronics: Fundamentals, biomedical applications, and bio-inspired system," 2010.

[9] J. Kwong and A. Chandrakasan, "Variation-driven device sizing for minimum energy sub-threshold circuits," in *Int. Symp. on Low Power Electronics and Design*. ACM, 2006, pp. 8–13.

[10] N. Shanbhag, "Reliable and energy-efficient digital signal processing," in *Design Automation Conference*, June 2002.

[11] V. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. Chakradhar, "Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 555–560.

[12] Z. Wang, K. H. Lee, and N Verma, "Overcoming computational errors in low-power sensing platforms through embedded machine-learning kernels," *to appear IEEE Tran. on VLSI*.

[13] P. Gupta, K. Jeong, A. Kahng, and C. Park, "Electrical assessment of lithographic gate line-end patterning," *Journal of Micro/Nanolithography, MEMS, and MOEMS*, vol. 9, no. 2, pp. 023014, 2010.

[14] Z. Wang, R. E. Schapire, and N. Verma, "Error-adaptive classifier boosting (EACB): Exploiting data-driven training for highly fault-tolerant hardware," in *Int. Conf. on Acoustics, Speech and Signal Processing*. IEEE, 2014, pp. 3884–3888.

[15] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*, The MIT Press, 2012.

[16] R. Strongin and Y. Sergeyev, *Global optimization with non-convex constraints: Sequential and parallel algorithms*, vol. 45, Springer, 2000.