International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 17, No. 3 (2003) 385–404
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

SILICON SUPPORT VECTOR MACHINE WITH ON-LINE LEARNING

ROMAN GENOV

Department of Electrical and Computer Engineering,
University of Toronto, Toronto, ON M5S 3G4, Canada
roman@eecg.toronto.edu

SHANTANU CHAKRABARTTY* and GERT CAUWENBERGHS†

Department of Electrical and Computer Engineering,
Johns Hopkins University, Baltimore, MD, 21218, USA
**shantanu@jhu.edu*
†*gert@jhu.edu*

Training of support vector machines (SVMs) amounts to solving a quadratic programming problem over the training data. We present a simple on-line SVM training algorithm of complexity approximately linear in the number of training vectors, and linear in the number of support vectors. The algorithm implements an on-line variant of sequential minimum optimization (SMO) that avoids the need for adjusting select pairs of training coefficients by adjusting the bias term along with the coefficient of the currently presented training vector. The coefficient assignment is a function of the margin returned by the SVM classifier prior to assignment, subject to inequality constraints. The training scheme lends efficiently to dedicated SVM hardware for real-time pattern recognition, implemented using resources already provided for run-time operation. Performance gains are illustrated using the *Kerneltron*, a massively parallel mixed-signal VLSI processor for kernel-based real-time video recognition.

Keywords: Support vector machine (SVM); large margin classifiers; quadratic programming (QP); sequential minimum optimization (SMO); matrix-vector multiplication (MVM); pattern recognition; kernel machines; analog array processors.

1. Introduction

On-line learning is the key to computational intelligence in a real-time setting, sustained by adapting continuously to changes in the environment. Examples of applications requiring continuous learning and adaptation range from channel equalization in communications to learning concept drifts in surveillance applications. Incorporating on-line learning functions in hardware for real-time pattern recognition impose speed and memory constraints that can only be met by implementing simple learning architectures on silicon.[4]

Support Vector Machines (SVMs)[23] have emerged as a principled approach to machine learning for classification and regression, demonstrating state-of-art performance in many applications and offering an attractive alternative to artificial neural network and expert-based approaches.

From a hardware perspective SVMs are attractive because of their simple functional form that lends naturally to parallel implementation. Massive parallelism results in very large throughput and energetic efficiency as needed for real-time recognition and mobile operation. In essence, SVMs perform template matching using a kernel distance metric, and construct a linear combination of the kernel matching scores to arrive at the classification answer. The kernel evaluations at the core of SVM computation are implemented most efficiently in a scalable array processor architecture with fine-grain distributed memory.[7] The large-margin formulation of SVMs produces a sparse representation that reduces the required number of stored templates in the array to the number of support vectors, usually a small fraction of the training data.

The challenge of implementing SVMs in silicon is to incorporate on-line learning into the architecture, to arrive at the support vectors and their weighting coefficients while training data is presented. SVM training entails solving a linearly constrained quadratic programming (QP) problem with box inequality constraints.[3] In principle, the QP procedure can be formulated as a constrained Hopfield neural network, with a natural analog circuit implementation.[1] The problem with this approach is the area of the implementation scales with the square of the number of data points, which becomes impractical for very large data sets, particularly in an on-line setting.

Several approaches to efficient SVM training in software have been formulated, that decompose the QP task into a sequence of smaller scale optimization subproblems.[11,13,18,21] Yet, each of these decomposition methods operate on several (at least pairs of) training points and their coefficients simultaneously, by selection criteria that do not directly lend to on-line implementation. Most attractive for efficient implementation are on-line methods that assign support vectors and their coefficients sequentially, in the order in which training data is received. Stochastic approximation techniques based on stochastic gradient descent[6,14] are particularly simple to implement, but require multiple passes through the data for convergence or give approximate results. Incremental SVM learning[5] formulates the exact QP solution recursively, but assumes internal state variables to update previously assigned coefficient values, requiring additional resources in hardware implementation.

An on-line sequential SVM training scheme, tailored for efficient use with dedicated parallel kernel hardware, is presented in Sec. 2. The algorithm assigns the coefficient of the most recently presented training sample based on its label and the decision function, and adjusts the bias term accordingly. Section 3 presents the *Kerneltron*, a massively parallel VLSI processor for real-time kernel-based pattern recognition, as the basis for on-line training experiments given in Sec. 4.

2. Support Vector Machines and On-Line Learning

In this section we first briefly review basics of SVMs and kernel computation in light of efficient implementation. Then we present an on-line training scheme and corresponding implementation architecture. We will assume two-class SVM classification although it is straightforward to extend the results to the multiclass case and to SVM regression.

2.1. *SVM classification*

SVMs are rooted in principles of statistical learning theory which formulate bounds on the generalization performance of a learning model based on a structural measure of its complexity.[23] SVMs minimize *structural risk* by maximizing a measure of margin in the classification (or regression) of the training examples.

Margin maximization leads to a solution expressed uniquely in terms of *support vectors* — usually a small subset of training vectors located on the margin. An input pattern $\mathbf{X}$ is classified into class $y \in \{-1, +1\}$ according to

$$y = \text{sign}(f(\mathbf{X})) \tag{1}$$

where the decision function $f(\mathbf{X})$ takes the form of a linear combination of kernels $K(\mathbf{X}_m, \mathbf{X})$ between the presented vector $\mathbf{X}$ and each of the training vectors $\mathbf{X}_m$

$$f(\mathbf{X}) = \sum_m \alpha_m y_m K(\mathbf{X}_m, \mathbf{X}) + b \tag{2}$$

weighted by training labels y_m and coefficients α_m. The coefficients α_m are nonzero only for training data that are support vectors, so that the expansion (2) is sparse and the support vectors capture the relevant information present in the training data.[a] Values for the coefficients α_m and bias term b are determined by a QP training criterion that follows in Sec. 2.3.

$K(\cdot, \cdot)$ is a symmetric positive-definite kernel, that implies a transformation of the data to feature space in which the kernel represents an inner product.[3,9] In general, the feature map represents the data in higher dimensions, and for certain choices of kernels (such as Gaussian) the feature space becomes infinite dimensional. Therefore evaluating the inner product directly in feature space becomes impractical or impossible, and kernels offer a computationally tractable alternative. However, most practically used forms of kernels can themselves be represented as nonlinear transformations of inner products in *data space*, as shown next.

2.2. *Inner-product based kernels*

Most frequently used kernels $K(\cdot, \cdot)$, such as polynomial classifiers, multilayer perceptrons,[b] and radial basis functions,[22] can be generally classified as one of

[a] In what follows we will denote $\mathbf{X}_m$ interchangeably as training vectors with $\alpha_m \geq 0$, and as the subset of support vectors with $\alpha_m > 0$. M denotes the number of support vectors.

[b] With logistic sigmoidal activation function, for particular values of the threshold parameter only.

the following forms:

(1) Inner-product based kernels (e.g. polynomial; sigmoidal connectionist):

$$K(\mathbf{X}_m, \mathbf{X}) = k(\mathbf{X}_m \cdot \mathbf{X}) = k\left(\sum_{n=1}^{N} X_{mn}\, X_n\right). \tag{3}$$

(2) Radial basis functions (L_2 norm distance based):

$$K(\mathbf{X}_m, \mathbf{X}) = k(\|\mathbf{X}_m - \mathbf{X}\|) = k\left(\left(\sum_{n=1}^{N} |X_{mn} - X_n|^2\right)^{\frac{1}{2}}\right) \tag{4}$$

where $k(\cdot)$ is a monotonically nondecreasing scalar function subject to the Mercer condition on $K(\cdot,\cdot)$.[9,23]

For purposes of efficient realization in a mixed-signal architecture that will become evident in Sec. 3, we assume kernels of the inner-product type (3) or kernels reducing to inner product computation. The limitation to inner-product based kernels implies no loss of generality, as inner-products comprise the most intensive part of the computation in evaluating kernels of *both* types (3) and (4). Indeed, expansion of the L_2 norm in radial basis functions (4) reduces to inner-product form:

$$k(\|\mathbf{X}_m - \mathbf{X}\|) = k((-2\mathbf{X}_m \cdot \mathbf{X} + \|\mathbf{X}_m\|^2 + \|\mathbf{X}\|^2)^{\frac{1}{2}}) \tag{5}$$

where the last two terms depend only on either the input vector or the support vector. These common terms are of much lower complexity than the inner-products, and can be easily precomputed or stored in peripheral registers.

2.3. *Training formulation*

Soft-margin classification SVMs minimize a compound measure of structural and empirical risk, controlled by a single regularization parameter C. In the dual formulation of the training problem, the coefficients α_i are obtained by minimizing a convex quadratic objective function subject to box inequality constraints of the form:[23]

$$\min_{0 \le \alpha_i \le C} : \; H = \frac{1}{2} \sum_{i,j} \alpha_i Q_{ij} \alpha_j - \sum_i \alpha_i \tag{6}$$

with symmetric positive definite kernel matrix $Q_{ij} = y_i y_j K(\mathbf{X}_i, \mathbf{X}_j)$. The presence of a bias term b in (2) implies an additional linear equality constraint

$$\sum_i y_i \alpha_i = 0\,, \tag{7}$$

which can be absorbed by adding a Lagrange term to (6):[2]

$$\max_b \min_{0 \le \alpha_i \le C} : \; H = \frac{1}{2} \sum_{i,j} \alpha_i Q_{ij} \alpha_j - \sum_i \alpha_i + b \sum_i y_i \alpha_i\,. \tag{8}$$

The first-order conditions on H in (8) reduce to the *Karish–Kuhn–Tucker* (KKT) conditions:[2]

$$g_i = \frac{\partial H}{\partial \alpha_i} = \sum_j Q_{ij}\alpha_j + y_i b - 1 = y_i f(\mathbf{X}_i) - 1 \quad \begin{cases} \geq 0; & \alpha_i = 0 \\ = 0; & 0 < \alpha_i < C \\ \leq 0; & \alpha_i = C \end{cases} \tag{9}$$

$$h = \frac{\partial H}{\partial b} = \sum_j y_j \alpha_j = 0 \tag{10}$$

which divide the training data in three categories based on the sign of the margin variable g_i: *margin* support vectors ($g_i = 0$); *error* support vectors ($g_i < 0$) with coefficients at bound C; and the remaining discarded vectors ($g_i > 0$).

2.4. *Hardware complexity of SVM training*

The similarity between the constrained quadratic form (6) and the Lyapunov energy function of a Hopfield neural network suggests an analog circuit for SVM training in the form of a recurrent neural network in which neurons represent coefficients α_i, and connection strengths represent the kernel entries Q_{ij}.[1] Thus the size of the network scales with the square of the number of data points and practical hardware implementation is limited to small training problems in a batch-mode setting.

Incremental SVM learning[5] offers an exact on-line alternative with scalable implementation. The incremental approach updates the solution for addition of a single training sample $\mathbf{X}_i$ by incrementing the coefficient α_i and simultaneously adjusting previously assigned coefficients α_j $(j < i)$ to satisfy the KKT conditions (9) and (10) on the present and all previous training samples. Every margin-misclassified ($g_i \leq 0$) training vector $\mathbf{X}_i$ becomes a support vector and is stored in the array, and the corresponding (nonzero) coefficient α_i is computed using a recursive matrix operation of dimensions square in the number of margin (not error) support vectors. Since the number of margin vectors is usually small and does not grow with the number of training vectors, computation and storage requirements to implement incremental learning are modest, but imply an overhead in hardware implementation.

Simple on-line, approximate estimation of the α_i coefficients is obtained by performing purely sequential optimization and discarding updates in previous coefficients α_j $(j < i)$. This leads to efficient implementation as presented in the next section.

2.5. *On-line update rule*

On-line training allows to find a solution based on partial training data, and update it as more training data becomes available without the need to fully retrain the machine in batch mode. We formulate a simple SVM on-line learning algorithm

that assumes same core implementation resources as needed for computation of the decision function in (1), with minimal overhead in the implementation.

In the simplest on-line learning scenario each training sample $\{\mathbf{X}_i, y_i\}$ arrives sequentially, one sample every time instant i. For each new sample the coefficient α_i is adjusted to satisfy its own KKT margin conditions (9), while retaining the other coefficients α_j at fixed values.[c] This sequential assignment results in a simple update formula, in terms of the (kernel-normalized) decision function returned by the classifier prior to the new assignment:

$$\alpha_i = \begin{cases} 0; & g_i \geq 0 \\ -g_i/Q_{ii}; & -CQ_{ii} < g_i < 0 \\ C; & g_i \leq -CQ_{ii} \end{cases} \tag{11}$$

where $g_i = y_i f(\mathbf{X}_i) - 1 = \sum_{j<i} Q_{ij}\alpha_j + by_i - 1$. The update (11) performs sequential steepest descent of the objective function H in (8), and thus is guaranteed to converge asymptotically to the global minimum of H, with a global rate that depends on the conditioning of the kernel matrix Q_{ij} and value of regularization parameter C.

A particularly simple update rule is obtained for self-normalized kernels, which satisfy $Q_{ii} = K(\mathbf{X}_i, \mathbf{X}_i) \equiv 1$, avoiding the need for division and multiplication in implementation of (11). Self-normalized kernels include radial basis kernels of the form (4), and the savings during training justify the extra computational effort in constructing the L_2-norm using inner-product primitives in (5).

The value assignment for the bias term b is more involved, as the sequential coefficient assignment (11) violates the equality constraint (10) by construction. Standard SVM approaches to sequential minimum optimization[11,13,21] satisfy the equality constraint exactly through coupling updates in at least two coefficients simultaneously. The on-line sequential version presented here avoids the need for updating two or more coefficients simultaneously by adjusting the bias term along with the coefficient update (11) as to approximately satisfy the equality constraint. This could be accomplished through stochastic gradient ascent of the objective function H in (8), by specifying an incremental update

$$\Delta b = \mu_i \sum_{j \leq i} y_j \alpha_j \,. \tag{12}$$

The choice of learning rate sequence $\mu_i \geq 0$ deserves special attention.

Consider the change in margin variables g_j under influence of the change in bias term b. The objective of training is to satisfy all KKT conditions; however the sequential update (11) only satisfies the margin KKT condition (9) on the selected training sample. The bias update (12) regulates the sum $\sum_j y_j \alpha_j$ towards zero as long as $\mu_i \geq 0$. The amplitude of μ_i can be chosen as to minimize a measure of discrepancy in the remaining margin KKT conditions. A sensible (heuristic) choice

[c] For now we ignore the equality constraint (10).

for the update Δb is one that minimizes the L_ν norm ($\nu = 1, 2$) of the margin variables g_j for the *margin* support vectors: $\min_{\Delta b} : \sum_{j\in\mathcal{M}} |g_j + y_j \Delta b|^\nu$, where $\mathcal{M} = \{j | 0 < \alpha_j < C\}$. The L_2 norm ($\nu = 2$) results in an update $\Delta b^* = -\frac{1}{\#\mathcal{M}} \sum_{j\in\mathcal{M}} y_j g_j$ that centers the *mean* of the margin variables g_j over the margin vectors ($j \in \mathcal{M}$). Similarly, the L_1 norm ($\nu = 1$) leads to an update $\Delta b^* = -\text{median}_{j\in\mathcal{M}}\{y_j g_j\}$ that centers the *median* of g_j over $\mathcal{M}$. In either case, the polarity of the update could imply a negative value of μ_i in (12). A nonnegative value of μ_i is enforced by rectifying the update if necessary,

$$\Delta b = \begin{cases} \Delta b^* & \text{if } \Delta b^* \sum_i y_i \alpha_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

thus guaranteeing global asymptotic convergence towards satisfying the equality constraint (10).

Computation of Δb^* assumes updated g_j values accounting for cumulative changes in coefficients and bias terms, $g_j = \sum_{\ell\le i} Q_{j\ell}\alpha_\ell + b y_j - 1$. In the L_2-norm case (mean bias centering), the computation is carried through efficiently in recursive form:

$$\Delta b^* \leftarrow \Delta b^* - \Delta b - \Delta\alpha_i \frac{1}{\#\mathcal{M}} \sum_{j\in\mathcal{M}} Q_{ij} y_j \tag{14}$$

using state variable Δb^*, and with knowledge of the previous bias update Δb and coefficient update $\Delta\alpha_i$. The last term in (14) reduces to $-y_i \Delta\alpha_i \frac{1}{\#\mathcal{M}} \sum_{j\in\mathcal{M}} K(\mathbf{X}_i, \mathbf{X}_j)$ and amounts to a sum operation on select kernels computed in the evaluation of g_i in (11).

The accuracy of the sequential on-line training algorithm, evaluated on the UCI Diabetes dataset, is illustrated in Fig. 1. Components of the data are mean and variance normalized, and a Gaussian kernel is used with variance $\sigma^2 = 10$. For $C = 1$, batch (exact) training over the first 576 samples returns a test error of 21% on the remaining 192 samples. A single sequential on-line pass through the same training data (in random order) returns a test error of 23% for median bias centering, and 27% for mean bias centering. Clearly, the L_1 norm is more robust, but the computation of the L_2 norm using recursive update is simpler. Convergence and accuracy can be improved by repeated presentation[d] of the training data, as shown in Figure Figs. 1(c) and 1(d). For both median and mean bias centering, five sequential passes through the training data (in random order) suffice to deliver the baseline test error of 21%.

2.6. *On-line architecture*

A parallel architecture implementing on-line sequential SVM learning in conjunction with a dedicated kernel processor is depicted in Fig. 2. For simplicity the update

[d]Upon subsequent passes of previously presented data, the coefficient α_i needs to be reset to zero in the evaluation of the decision function $f(\mathbf{X}_i)$ and margin variable g_i prior to assignment.

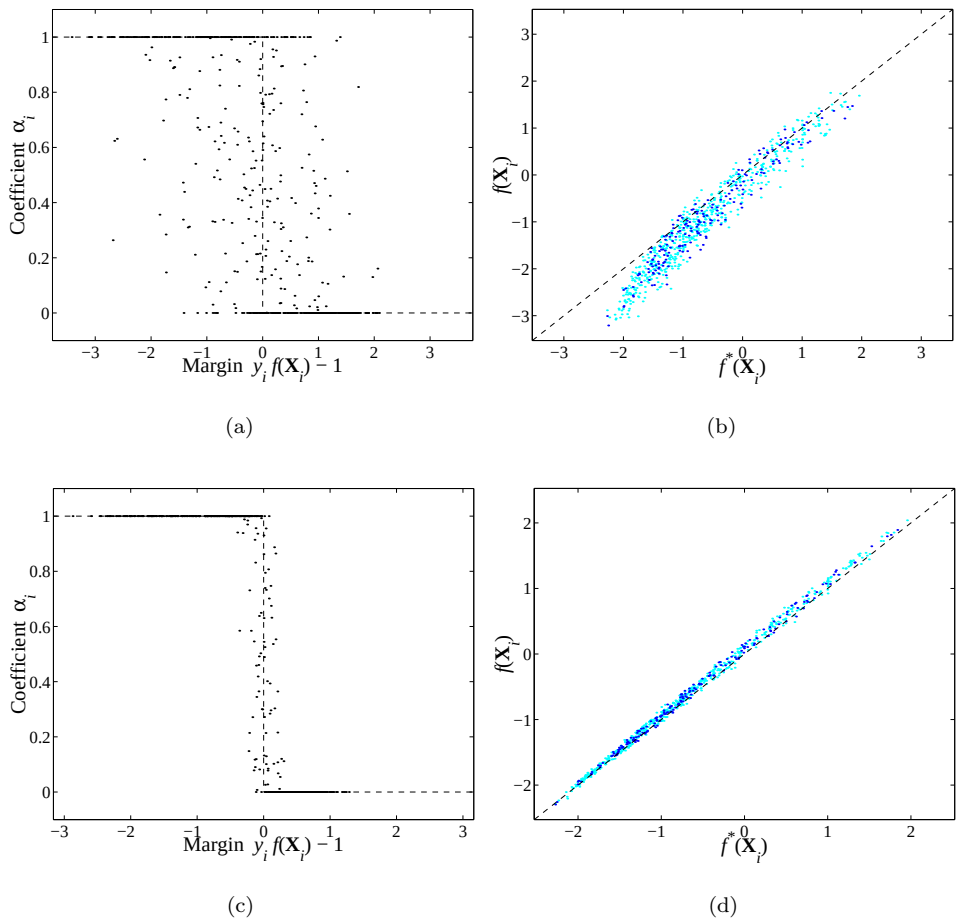
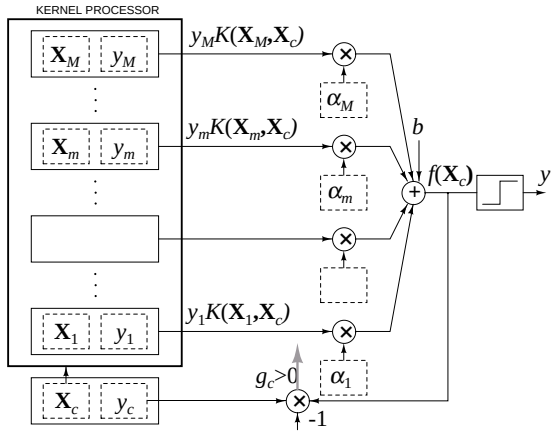


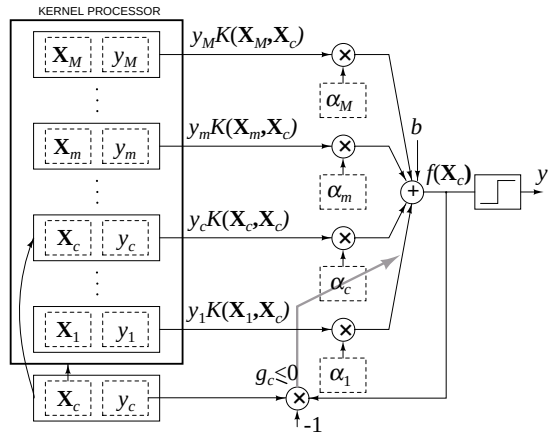
Fig. 1. Convergence and accuracy of on-line sequential SVM training, evaluated on the UCI Diabetes dataset for a Gaussian kernel. (a) Discrepancy in KKT margin conditions (9) after single pass (with mean bias centering) through the training set. (b) Discrepancy between on-line and batch (true) decision functions evaluated over training and test sets. (c and d) Same, after five on-line passes through the training set in random order.

in the bias term b is omitted from the diagram. The architecture for training shares the majority of units with those needed in run-time to arrive at the classification y from the decision function $f(\mathbf{X})$. Dedicated units for implementing training include the cell computing g_c from $f(\mathbf{X}_c)$, and local provisions for adaptation of α_c according to (11). The shared resources allow to efficiently interleave run-time and training modes of operation on the same integrated platform.

Physical memory constraints in the kernel processor hardware limit the number of support vectors that can be stored and processed in the array. Under ideal conditions, the number of support vectors is relatively small to enable integrated storage



(a)



(b)

Fig. 2. SVM learning parallel architecture implementing the on-line coefficient update rule (11). (a) A candidate training vector $\mathbf{X}_c$ and label y_c are presented. Kernels between $\mathbf{X}_c$ and the support vectors $\mathbf{X}_m$ evaluated by the array processor are linearly combined and weighted with stored coefficients α_m, to produce decision function $f(\mathbf{X}_c)$ and output y according to (2) and (1). If the margin variable $g_c = y_c f(\mathbf{X}_c) - 1$ is positive, the training sample has zero coefficient and is discarded. (b) If the margin variable g_c is negative, $\mathbf{X}_c$ becomes a support vector and is stored in a vacant memory location in the kernel array. The coefficient α_c is assigned a unique value satisfying KKT conditions (9) on the resulting value of g_c.

of all support vectors. For tasks with significant class overlap and noise, the number of error vectors can be large enough to exhaust available in-processor memory resources. Under such conditions a scheduling procedure is needed to maintain a fixed number of support vectors as limited by the memory. One possible schedule is a first-in-first-out procedure, which replaces the oldest support vector with the

new assignment, provided that the change results in lower global objective function H. Such a scheduling procedure enables the learning architecture to adapt its parameters under influence of environmental changes (drifting concepts).[14] Other possible schedules include selective removal of support vectors with smallest leave-one-out error.

2.7. *SVM computational resources requirements*

Sections 2.1 and 2.5 established that both run-time classification and on-line training can be accommodated in essentially the same architecture to compute the decision function (2). For general kernels described in Sec. 2.2, the computation of the inner products takes the form of matrix-vector multiplication (MVM), $\sum_{n=1}^{N} X_{mn}\, X_n$; $m = 1, \ldots M$, where M is the number of support vectors. For large scale problems as the ones of interest here, the dimensions of the matrix $M \times N$ are excessive for real-time implementation even on a high-end processor.

As a point of reference, consider the pedestrian and face detection task in Ref. 19, for which the feature vector length N is 1326 wavelets per instance, and the number of support vectors M is in excess of 4000. To cover the visual field over the entire scanned image at reasonable resolution (500 image window instances through a variable resolution search method) at video rate (30 frames per second), a computational throughput of 75×10^9 multiply-and-accumulate operations per second, is needed. The computational requirement can be relaxed through simplifying and further optimizing the SVM architecture for real-time operation, but at the expense of classification performance.[17,19]

3. Kerneltron: Massively Parallel VLSI Kernel Machine

The *Kerneltron* offers the computational power required for large-scale problems in high dimensions as just described. The general architecture is described next.

3.1. *Core recognition VLSI processor*

At the core of the system is a recognition engine, which very efficiently implements kernel-based algorithms, such as support vector machines, for general pattern detection and classification. The implementation focuses on inner-product computation in a parallel architecture for both run-time and training modes of operation.

As discussed in Sec. 2.5, both SVM classification and on-line learning are most efficiently implemented on the same chip, in a scalable VLSI architecture illustrated schematically in Fig. 3. The diagram is the floorplan of the *Kerneltron*, with the support vector matrix projected as a 2-D array of cells, and input and output vector components crossing in perpendicular directions alternating from one stage to the next. The architecture maintains low input/output data rate. Digital inputs are fed into the processor through a properly sized serial/parallel converter shift register.

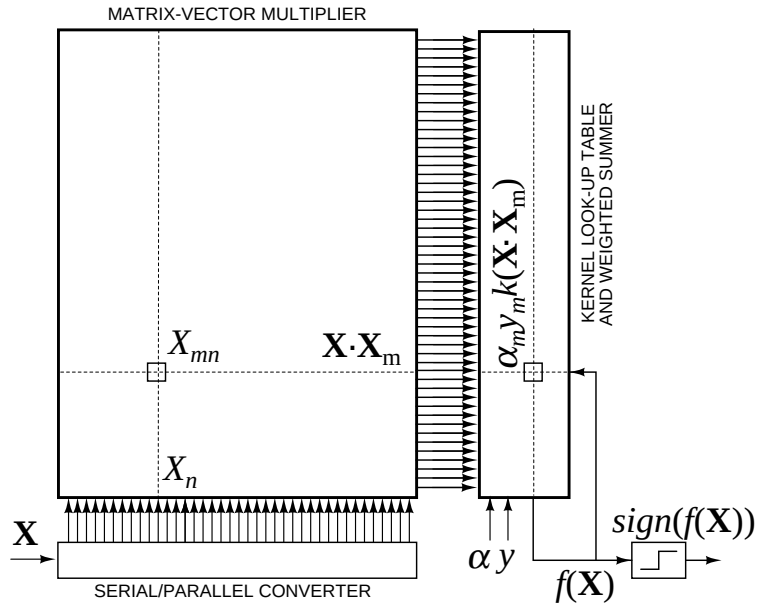


Fig. 3. The architecture of the core recognition processor for support vector machine classification and on-line training. Communication with outside modules is through a serial digital input/output interface for maximal flexibility and programmability, while the core internal computations are parallel and analog for optimal efficiency.

The classification decision is obtained in digital domain by thresholding the weighted sum of kernels. The kernels are obtained by mapping the inner-products $\mathbf{X} \cdot \mathbf{X}_m$ through the function $k(\cdot)$. During on-line learning, the α coefficient of a newly presented data point is adapted based on the value of the weighted sum of kernels. Adaptive assignment of the coefficient α_m is a task of low computational complexity that can be delegated off-chip, using the inner products $\mathbf{X}\cdot\mathbf{X}_m$ computed efficiently on the array.

3.2. *Mixed-signal computation*

Computing inner-products between an input vector $\mathbf{X}$ and template vectors $\mathbf{X}_m$ in parallel reduces to the task of matrix-vector multiplication (MVM). We will henceforth consider the general MVM setting

$$Y_m = \sum_{n=0}^{N-1} W_{mn} X_n \tag{15}$$

with N-dimensional input vector X_n, M-dimensional output vector Y_m, and $M \times N$ matrix of coefficients W_{mn}. Rows of matrix elements W_{mn} denote support vectors X_{mn}, and the outputs Y_m transform to kernels through activation of $k(\cdot)$ in (3) or (4).

3.2.1. *Internally analog, externally digital computation*

The approach combines the computational efficiency of analog array processing with the precision of digital processing and the convenience of a programmable and reconfigurable digital interface.

The digital representation is embedded in the analog array architecture, with matrix elements stored locally in bit-parallel form

$$W_{mn} = \sum_{i=0}^{I-1} 2^{-i-1} w_{mn}^{(i)} \tag{16}$$

and inputs presented in bit-serial fashion

$$X_n = \sum_{j=0}^{J-1} \gamma_j x_n^{(j)} \tag{17}$$

where the coefficients γ_j are assumed in radix two, depending on the form of input encoding used. The MVM task (15) then decomposes into

$$Y_m = \sum_{n=0}^{N-1} W_{mn} X_n = \sum_{i=0}^{I-1} 2^{-i-1} Y_m^{(i)} \tag{18}$$

with MVM partials

$$Y_m{}^{(i)} = \sum_{j=0}^{J-1} \gamma_j Y_m^{(i,j)} \tag{19}$$

and

$$Y_m{}^{(i,j)} = \sum_{n=0}^{N-1} w_{mn}^{(i)} x_n^{(j)} \ . \tag{20}$$

The binary–binary partial products (20) are conveniently computed and accumulated, with zero latency, using an analog MVM array as described next.

3.2.2. *Oversampling mixed-signal array processing*

The unit cell in the analog array combines a CID (charge injection device[10]) computational element[16,20] with a DRAM storage element.[7] The cell stores one bit of a matrix element $w_{mn}{}^{(i)}$, performs a one-quadrant binary–unary (or binary–binary) multiplication of $w_{mn}{}^{(i)}$ and $x_n{}^{(j)}$ in (20), and accumulates the result across cells with common m and i indices. An array of cells thus performs (unsigned) binary–unary multiplication (20) of matrix $w_{mn}{}^{(i)}$ and vector $x_n{}^{(j)}$ yielding $Y_m{}^{(i,j)}$, for values of i in parallel across the array, and values of j in sequence over time. A 256×128 array prototype using CID/DRAM cells is shown in Fig. 4.

The MVM partials (20) are quantized by a bank of oversampling analog-to-digital converters (ADCs), and the results accumulated in the digital domain according to (19) and (18). The precision of computation is limited by the resolution

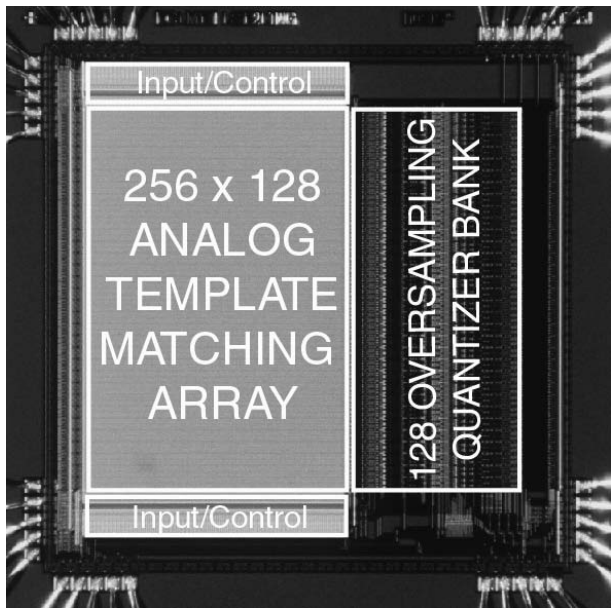


Fig. 4. Micrograph of the *Kerneltron* prototype, containing an array of 256×128 computational cells with embedded memory, and a row-parallel bank of 128 algorithmic $\Delta\Sigma$ ADCs. Die size is 3 mm $\times$ 3 mm in 0.5 μm CMOS technology.

of the ADCs digitizing the analog array outputs. The conventional delta–sigma ($\Delta\Sigma$) ADC design paradigm allows to reduce requirements on precision of analog circuits to attain high resolution of conversion, at the expense of bandwidth. In the presented architecture a high conversion rate is maintained by combining delta–sigma analog-to-digital conversion with oversampled encoding of the digital inputs, where the delta–sigma modulator integrates the partial multiply-and-accumulate outputs (20) from the analog array according to (19).

Figure 5 depicts one row of matrix elements W_{mn} in the $\Delta\Sigma$ oversampling architecture, encoded in $I = 4$ bit-parallel rows of CID/DRAM cells. One bit of a unary-coded input vector is presented each clock cycle, taking J clock cycles to complete a full computational cycle (15). The data flow is illustrated for a digital input series ${x_n}^{(j)}$ of $J = 16$ unary bits.

Over J clock cycles, the oversampling ADC integrates the partial products (20), producing a decimated output

$$Q_m^{(i)} \approx \sum_{j=0}^{J-1} \gamma_j Y_m^{(i,j)} \tag{21}$$

where $\gamma_j = 1$ for unary coding of inputs. Decimation for a first-order delta–sigma modulator is achieved using a binary counter. Higher precision is obtained in the same number of cycles J by using a residue resampling extended counting scheme.[15]

Additional gains in precision can be obtained by exploiting binomial statistics

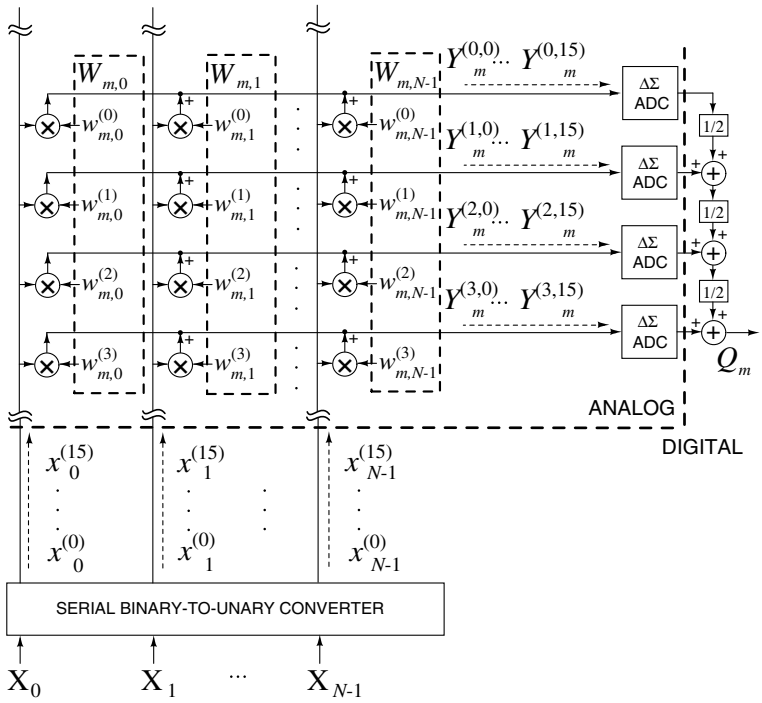


Fig. 5. Block diagram of one row of the matrix with binary encoded elements $w^{(i)}{}_{mn}$, for a single m and $I = 4$. Data flow of bit-serial unary encoded inputs $x^{(j)}{}_n$ and corresponding partial product outputs $Y^{(i,j)}{}_m$, with $J = 16$ bits. The full product for a single row $Y^{(i)}{}_m$ is accumulated and quantized by a delta–sigma ADC. The final product is constructed in the digital domain according to 18.

of binary terms in the analog summation (20).[8] In the present scheme, this would entail stochastic encoding of the digital inputs prior to unary oversampled encoding.

4. Results and Discussion

4.1. *Measured performance*

A prototype *Kerneltron* was integrated on a 3×3 mm^2 die and fabricated in 0.5 μm CMOS technology. The chip contains an array of 256×128 CID/DRAM cells, and a row-parallel bank of 128 algorithmic $\Delta\Sigma$ ADCs. Figure 4 depicts the micrograph and system floorplan of the chip.

The processor interfaces externally in digital format. Two separate shift registers load the templates (support vectors) along odd and even columns of the DRAM array. Integrated refresh circuitry periodically updates the charge stored in the array to compensate for leakage. Vertical bit lines extend across the array, with two rows of sense amplifiers at the top and bottom of the array. The refresh alternates between even and odd columns, with separate select lines. Stored charge corresponding to matrix element values can also be read and shifted out from the chip for test

Table 1. Measured performance.

Technology	0.5 μm CMOS
Area	3 mm × 3 mm
Power	5.9 mW
Supply Voltage	5 V
Dimensions	256 inputs × 128 templates
Throughput	6.5 GMACS
Output Resolution	8-bit

purposes. All of the supporting digital clocks and control signals are generated on-chip. The bank of $\Delta\Sigma$ ADCs dissipates 2.6 mW yielding a combined conversion rate of 12.8 Msamples/s. Table 1 summarizes the measured performance.

4.2. *On-line learning*

Experiments evaluating the performance of on-line sequential training on a simple visual object detection task are illustrated in Fig. 6. Inputs to the *Kerneltron* consists of blocks of 16 × 16 pixels of image data, quantized at 4-bit intensity resolution (in the range $\{-8, \ldots, +7\}$). The *Lena* sample image is divided in training and test sets as shown in Fig. 6(a), and a region around the left eye shown in the boxed inset defines the positive training labels. The matched filter response using the eye template over the image field is given in Fig. 6(b), responding strongly to the location of both the left (training) and right (test) eyes, but also to spurious other locations in the image. Offline (batch) SVM training with a Gaussian kernel at floating-point precision (variance $\sigma^2 = 1024$; $C = 1$) enhances the response to both eyes and suppresses other features in the training and test portions of the image, shown in Fig. 6(c). The trained classifier contains 80 support vectors including 55 margin vectors and 25 error vectors. The effect of on-line sequential training with mean bias centering at 8-bit fixed-point inner-product resolution on the *Kerneltron* is illustrated in Fig. 6(d). The resulting classifier is sparser with 77 support vectors, of which 55 margin vectors and 22 are error vectors. The peak number of support vectors in the course of on-line training is 122.

The implementation of the Gaussian kernel assumes the inner-product decomposition (5) where the input term $|\mathbf{X}|^2$ is obtained by storing the input vector in a dedicated template row location of the array prior to kernel evaluation. Exponentiation in the Gaussian kernel is circumvented using Euler's approximation

$$k(x) = \exp(-\frac{x^2}{2\sigma^2}) \approx \left(1 - \frac{1}{\nu}\frac{x^2}{2\sigma^2}\right)^{\nu} \tag{22}$$

for large ν, where the approximation holds for $x^2 \leq \nu 2\sigma^2$. A value $\nu = 3$ is used to reduce the number of multiplications required while suitably approximating the exponential.

Effects of quantization and imprecision at various levels are barely visible in Fig. 6(d).

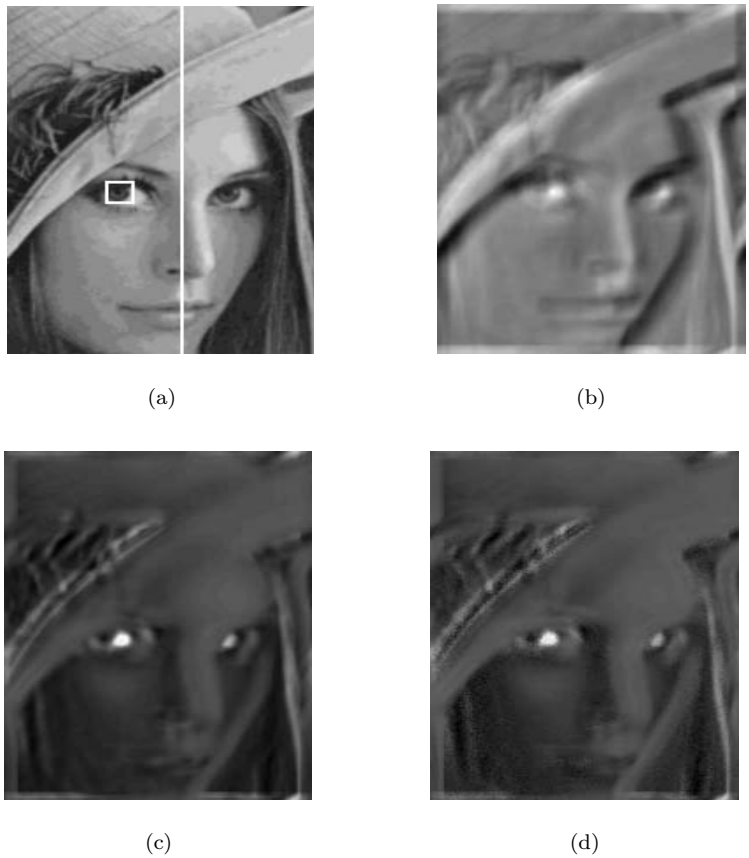


Fig. 6. Effects of on-line sequential training and finite precision on SVM adaptive visual object detection. (a) Lena image partitioned in training and test data. Inset box indicates positive training examples for eye detection task. (b) Matched filter response using left eye template. (c) Baseline batch-mode SVM training and test at floating-point kernel precision. (d) On-line sequential SVM training and test on *Kerneltron* at 8-bit inner-product precision.

4.3. *Applications*

The *Kerneltron* offers an efficient adaptive platform for real-time object detection and recognition, particularly in artificial vision and human–computer interfaces. Applications extend from SVMs to any pattern recognition architecture that relies on computing a kernel distance between an input and a large set of templates in large dimensions.

Besides throughput, power dissipation is a main concern in portable and mobile applications. Power efficiency can be traded for speed, and a reduced implementation of dimensions similar to the version of the pedestrian classifier running on a Pentium PC (27 input features)[17,19] could be integrated on a chip running at 100 μW of power, easily supported with a hearing aid type battery for a lifetime of several weeks.

One low-power application that could benefit a large group of users is a navigational aid for visually impaired people. OpenEyes, a system developed for this purpose[12] currently runs a classifier in software on a Pentium PC. The software solution offers great flexibility to the user and developer, but limits the mobility of the user. The *Kerneltron* offers the prospect of a low-weight, low-profile alternative. With integrated learning capabilities, the *Kerneltron* also powers the user with the ability to train the system on-line for automated recognition of custom-specified objects or people.

5. Conclusions

A massively parallel architecture implementing Support Vector Machine classification and training in very high dimensions has been presented. Inner-product based kernels are chosen for their generality and simplicity of implementation. The learning problem is formulated as a simple on-line sequential update rule that satisfies the margin KKT condition on the new point, while adjusting the bias term to approximately satisfy the equality constraint and the remaining margin KKT conditions. On-line training is implemented utilizing essentially the same computational resources as the classifier in run-time. The architecture lends itself to efficient mixed-signal implementation on the *Kerneltron*, a massively parallel support vector machine in silicon for large-scale kernel-based visual pattern recognition.

Acknowledgments

This research was supported by NSERC Discovery 261606-03, NSF IIS-0209289, ONR/DARPA N00014-00-C-0315, ONR N00014-99-1-0612, the Catalyst Foundation, and WatchVision Corporation. The chip was fabricated through the MOSIS service.

References

1. D. Anguita, S. Ridella and S. Rovetta, "Circuital implementation of support vector machines," *Electron. Lett.* **34**, 16 (1998) 1596–1597.
2. D. P. Bertsekas, *Nonlinear Programming*, Athena Scientific, Belmont MA, 1995.
3. B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifier," *Proc. Fifth Annual ACM Workshop on Computational Learning Theory*, 1992, pp. 144–152.
4. G. Cauwenberghs and M. Bayoumi, *Learning on Silicon, Analog VLSI Adaptive Systems*, Kluwer Academic, Norwell MA, 1999.
5. G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Adv. Neural Information Processing Systems*, *Proc. 2000 IEEE NIPS Conf.*, MIT Press, Cambridge, MA, 2001, pp. 409–415.
6. T.-T. Friessand, N. Cristianini and C. Campbell, "The kernel adatron algorithm: a fast and simple learning procedure for support vector machines," *Proc. 15th Int. Conf. Machine Learning* (*ICML98*), 1998, pp. 188–196.

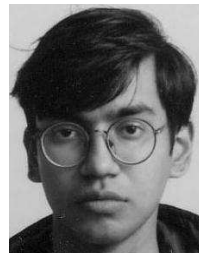
7. R. Genov and G. Cauwenberghs, "Charge-mode parallel architecture for matrix-vector multiplication," *IEEE Trans. Circuits Syst. II* **48**, 10 (2001) 930–936.
8. R. Genov and G. Cauwenberghs, "Stochastic mixed-signal VLSI architecture for high-dimensional kernel machines," *Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA, Vol. 14, 2002, pp. 1099–1105.
9. F. Girosi, M. Jones and T. Poggio, "Regularization theory and neural networks architectures," *Neural Comput.* **7** (1995) 219–269.
10. M. Howes and D. Morgan (eds.), *Charge-Coupled Devices and Systems*, John Wiley, 1979.
11. T. Joachims, "Making large-scale SVM learning practical," *Advances in Kernel Methods — Support Vector Learning*, eds. B. Schölkopf, C. Burges and A. Smola, MIT Press, 1999, pp. 169–184.
12. S. Kang and S.-W. Lee, "Handheld computer vision system for the visually impaired," *Proc. 3rd Int. Workshop on Human-Friendly Welfare Robotic Systems*, Daejeon, Korea, 2002, pp. 43–48.
13. S. Keerthi, C. Bhattacharyya and K. Murthy, "A fast iterative nearest point algorithm for support vector machine classifier design," Technical Report TR-ISL-99-03, Department of CSA, IISc, Bangalore, India, 1999.
14. J. Kivinen, A. J. Smola and R. C. Williamson, "On-line learning with kernels," *Adv. Neural Information Processing Systems* (*NIPS '2001*), Vol. 14, MIT Press, Cambridge, MA, 2002, pp. 785–792.
15. G. Mulliken, F. Adil, G. Cauwenberghs and R. Genov, "Delta–sigma algorithmic analog-to-digital conversion," *IEEE Int. Symp. Circuits and Systems* (*ISCAS'02*), Scottsdale, AZ, May 26–29, 2002, ISCAS'2002.
16. C. Neugebauer and A. Yariv, "A parallel analog CCD/CMOS neural network IC," *Proc. IEEE Int. Joint Conf. Neural Networks* (*IJCNN'91*), Seattle, WA, Vol. 1, 1991, pp. 447–451.
17. M. Oren, C. Papageorgiou, P. Sinha, E. Osuna and T. Poggio, "Pedestrian detection using wavelet templates," *Computer Vision and Pattern Recognition*, 1997, pp. 193–199.
18. E. Osuna, R. Freund and F. Girosi, "Training support vector machines: an application to face detection," *Computer Vision and Pattern Recognition*, 1997, pp. 130–136.
19. C. P. Papageorgiou, M. Oren and T. Poggio, "A general framework for object detection," *Proc. Int. Conf. Computer Vision*, 1998, pp. 555–562.
20. V. Pedroni, A. Agranat, C. Neugebauer and A. Yariv, "Pattern matching and parallel processing with CCD technology," *Proc. IEEE Int. Joint Conf. Neural Networks* (*IJCNN'92*), Vol. 3, 1992, pp. 620–623.
21. J. Platt, "Fast training of support vector machines using sequential minimal optimization," *Advances in Kernel Methods — Support Vector Learning*, eds. B. Schölkopf, C. Burges and A. Smola, MIT Press, Cambridge, MA, 1999, pp. 185–208.
22. B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio and V. Vapnik, "Comparing support vector machines with Gaussian kernels to radial basis functions classifiers," *IEEE Trans. Sign. Process.* **45**, 11 (1997) 2758–2765.
23. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer Verlag, 1995.

Roman Genov received the B.S. degree (highest honors) in electrical engineering from Rochester Institute of Technology, NY in 1996 and the M.S. and Ph.D. degrees in electrical and computer engineering from Johns Hopkins University, Baltimore, MD in 1998 and 2002 respectively. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Toronto.

Dr. Genov held engineering positions at Atmel Corporation, Columbia, MD in 1995 and Xerox Corporation, Rochester, NY in 1996. He was a visiting researcher in the Robot Learning Group at Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland in 1998 and in the Center for Biological and Computational Learning at Massachusetts Institute of Technology, Cambridge, MA in 1999.

Dr. Genov is a member of the IEEE Circuits and Systems Society and Solid-State Circuits Society. He received a Best Presentation Award at IEEE IJCNN'2000 and a Student Paper Contest Award at IEEE MWSCAS'2000.

His research interests include analog and digital VLSI circuits, systems and algorithms for parallel signal processing and adaptive computation with application to pattern recognition, focal-plane imaging, autonomous system design, and low-power instrumentation.



Shantanu Chakrabartty received the B.Tech. degree from the Indian Institute of Technology, Delhi and M.S. degree from Johns Hopkins University, Baltimore, MD, both in electrical engineering in 1996 and 2000, respectively. Since then he has been pursuing a Ph.D. degree at the Johns Hopkins University also in electrical engineering, with emphasis on designing adaptive classifiers on analog VLSI systems.

From 1996 to 1999 he was an engineer at Qualcomm Inc, San Diego, CA and was involved primarily in system and development aspects of India's first CDMA Wireless Local Loop subsystems.

His current research interests include hardware–software co-optimization for designing classifier systems for static and dynamic recognition problems. His interests also include nonlinear signal processing, large margin classification and low-power analog VLSI circuits and systems.

Gert Cauwenberghs received the Ph.D. in electrical engineering from Caltech in 1994, and is presently Professor of Electrical and Computer Engineering at Johns Hopkins University. He was Visiting Professor of Brain and Cognitive Science at Massachusetts Institute of Technology in 1999. He recently co-edited a book *Learning on Silicon* (Kluwer, 1999). He was Francqui Fellow of the Belgian American Educational Foundation in 1988, and received the National Science Foundation Career Award in 1997, the Office of Naval Research Young Investigator Award in 1999, and the Presidential Early Career Award for Scientists and Engineers in 2000. He is Associate Editor of the *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, and the *IEEE Sensors Journal*. He was Publication Chair at IEEE MWSCAS '2000, Analog Track Chair at IEEE ISCAS '2002, and Invited Sessions Co-Chair at ISCAS '2003. He chaired the Analog Signal Processing Technical Committee of the IEEE Circuits and Systems Society in 2001.

His research covers VLSI circuits, systems and algorithms for parallel signal processing, adaptive neural computation, and low-power coding and instrumentation.