2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Embedded Electronics Systems for Training Support Vector Machines

Sergio Decherchi, Giovanni Parodi, Paolo Gastaldo, and Rodolfo Zunino, *Senior Member, IEEE*

***Abstract* — Training Support Vector Machines (SVMs) requires efficient architectures, endowed with agile memory handling and specific computational features. Such a process is often supported by embedded implementations on dedicated machinery, for example in applications requiring on-line training abilities. The paper presents a general approach to the efficient implementation of SVM training on Digital Signal Processor (DSP) devices. The methodology optimizes efficiency by a twofold approach: first, it suitably adjusts an established, effective training algorithm for large data sets; secondly, it reformulates the algorithm to best exploit the computational features of DSP devices and boost efficiency accordingly. Experimental results tackle the training problem of SVMs by using a high-end DSP architecture on real-world benchmarks, and confirm both the effectiveness and the general validity of the approach.**

I. Introduction

SUPPORT Vector Machines (SVMs) [1] represent one of nowadays' most effective methodologies for tackling classification and regression problems in complex, nonlinear data distributions [1-4]. Crucial advantages characterize the SVM model from both theoretical and practical perspectives.

First of all, the training problem is convex, which allows one to avoid local minima by using polynomial-complexity Quadratic Programming (QP) methods; secondly, the optimization problem involves inner products between patterns, hence the curse of dimensionality does not enter the complexity of the training process explicitly; finally, kernel-based representations allow SVMs to handle arbitrary distributions that may not be linearly separable in the data space. Of course, SVMs lie within the scope of sample-based models, and the ultimate result depends on the quality of the sampling process to build the training set.

An important feature of the overall model is the ability to handle huge masses of data, and a wide variety of efficient algorithms [5-10] have been developed for SVM training in such significant cases. These methods typically adopt an iterative selection strategy: first, limited subsets of (supposedly) critical patterns are identified and undergo partial optimization, then the local solution thus obtained is projected onto the whole data set to verify consistency and global optimality. Such a process iterates until convergence.

The consequent success of SVMs in real-world domains motivates continuing research toward embedded implementations on low-cost machinery. Programmable digital devices often represent the basic choice for the run-time support of trained SVMs [11], but FPGA technology may not prove efficient when dealing with systems that require training capabilities. In these cases, the basic features of the SVM model make the family of Digital Signal Processors (DSPs) possibly more appropriate.

Therefore, this paper describes a methodology for the embedded support of SVM training by means of DSP-based architectures. The research reconsiders the overall training problem and privileges the viewpoint of embedded implementations. This resulted in a hardware-oriented version of the well-known optimization algorithm by Keerthi *et al.* [8]. A reformulation of the basic local-optimization steps allows the algorithm to exploit the architectural features of the target processors at best, thus attaining highest efficiency. The method has been tested on a set of standard benchmarks, to verify the algorithm's effectiveness and the computational efficiency of the embedded system.

Thus, the method proposed in this paper leads to two main results: first, the enhanced version of the SVM-training algorithm well fits the limited hardware resources and the peculiar computational features of embedded devices; secondly, the speed-up values and the accuracies of the embedded SVM systems confirm the validity of the overall approach from an applicative perspective.

II. Theoretical Background

A. Support Vector Machines for classification

A binary classification problem involves using a set of patterns $Z = \{ (\mathbf{x}_l, y_l);\ l=1,..,np;\ y_l \in \{-1,+1\} \}$. The Support Vector Machine [1] proved a valuable algorithm for that task [1], and requires the solution of a Quadratic Programming problem:

$$\min_{\alpha} \left\{ \frac{1}{2} \sum_{l,m=1}^{np} \alpha_l \alpha_m y_l y_m K(\mathbf{x}_l, \mathbf{x}_m) - \sum_{l=1}^{np} \alpha_l \right\}$$

0-7803-9490-9/06/$20.00/©2006 IEEE

$$\text{subject to: } \begin{cases} 0 \le \alpha_l \le C, \forall l \\ \sum_{l=1}^{np} y_l \alpha_l = 0 \end{cases} \quad (1)$$

where α_l are the SVM parameters setting the class-separating surface, the scalar quantity C upper bounds the SVM parameters, and $K()$ is the kernel function, i.e., a basis for the SVM series expansion. If $\Phi(\mathbf{x}_1)$ and $\Phi(\mathbf{x}_2)$ are the points in the "feature" space that are associated with $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively, then their dot product can be written as $\langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle = K(\mathbf{x}_1, \mathbf{x}_2)$. An SVM supports a linear class separation in that feature space; the classification rule for a trained SVM is:

$$f(\mathbf{x}) = \sum_{l=1}^{np} \alpha_l y_l K(\mathbf{x}, \mathbf{x}_l) + b \quad (2)$$

where b is a bias. The set of $n_{SV} \le np$ patterns, for which non-null parameters α_l are found by solving (1), are called support vectors.

For an exhaustive presentation of the method, see [1-4]. The complexity of the hypersurface (2) is affected by both the value of C and the specific kernel adopted. When designing empirically trained classifiers, the crucial quantity clearly is the run-time classification error. Formal approaches to predicting generalization performance [12,13] often prove impractical, mainly due to the too wide bounds obtained. Conversely, empirical approaches such as cross-validation and *k*-fold [14,15] can yield useful generalization estimates, at the cost of reducing the number of training patterns.

B. *Efficient algorithms for SVM training*

The problem setting (1) has the crucial advantage of involving a quadratic-optimization problem with linear constraints. This ensures that the solution is unique and that it can be found in polynomial time. The problem formulation depends on the sample cardinality, *np*, hence an effective strategy for selecting the eventual support vectors (i.e., those $\mathbf{x}_l$ s.t. $\alpha_l \neq 0$) is crucial to a training method's effectiveness.

These aspects become critical in the presence of large training sets; in such cases, the widely accepted approach consists in focusing the optimization algorithm on subsets of training patterns in turns. The literature offers several algorithms for that purpose; "decomposition" methods [6] typically aim at identifying those vectors that seem more critical in satisfying the boundary constraints when minimizing (1).

Sequential Minimal Optimization (SMO) [7-10] involves the smallest possible subset of vectors, and considers pairs of patterns sequentially. Such a strategy has the crucial advantage that the solution of (1) when *np*=2 can be computed analytically and explicitly. Keerthi's selection method [8] uses a convergence process to the bias term in (2) to select the most relevant patterns for local optimization.

Likewise, Lin's Quadratic Programming library LibSVM [10] applies SMO on "working sets" of patterns that can be selected iteratively by a shrinking process. Lin's approach tackles the convergence problem by measuring the cost gradient, ∇f_l, at the l-th pattern $\mathbf{x}_l$. The selection strategy identifies the pair of patterns whose Lagrange coefficients $\{\alpha_i, \alpha_j\}$ most violate the Karush Kuhn-Tucker (KKT) conditions [8]; their indexes are computed as:

$$i = \arg\max_l \left\{ \left(-\nabla f_l ; \begin{cases} y_l = +1 \\ \alpha_l < C \end{cases} \right), \left(\nabla f_l ; \begin{cases} y_l = -1 \\ \alpha_l > 0 \end{cases} \right) \right\} \quad (3a)$$

$$j = \arg\min_l \left\{ \left(\nabla f_l ; \begin{cases} y_l = -1 \\ \alpha_l < C \end{cases} \right), \left(-\nabla f_l ; \begin{cases} y_l = +1 \\ \alpha_l > 0 \end{cases} \right) \right\} \quad (3b)$$

Two quantities are associated with these patterns, and rule the stopping criterion in LibSVM:

$$g_i \equiv \begin{cases} -\nabla f(\boldsymbol{\alpha})_i & \text{if } y_i = 1, \alpha_i < C \\ \nabla f(\boldsymbol{\alpha})_i & \text{if } y_i = -1, \alpha_i > 0 \end{cases} \quad (4a)$$

$$g_j \equiv \begin{cases} -\nabla f(\boldsymbol{\alpha})_j & \text{if } y_j = -1, \alpha_j < C \\ \nabla f(\boldsymbol{\alpha})_j & \text{if } y_j = 1, \alpha_j > 0 \end{cases} \quad (4b)$$

and the stopping condition that ensures convergence is written as:

$$g_i \le g_j \quad (5)$$

The above algorithm is highly efficient in terms of convergence speed. Empirical practice shows that it is of the most successful strategies to minimize the number of iterations in QP optimization.

C. *Embedded DSP-based Support for SVMs*

Field Programmable Gate Array devices (FPGAs) mostly represent the reference architecture for supporting the run-time operation of *trained* SVMs [11]. This is a direct result of the cost-efficient programming and limited power consumption featured by embedded electronic systems that host FPGA devices.

On the other hand, when application constraints require that the embedded system support on-line *training*, too, other families of computing devices with different features might possibly become more interesting.

In particular, the target hardware platforms should be endowed with: 1) agile memory handling for easy support of the pattern-selection strategies described in the previous section; 2) buffering and caching capabilities, for managing large sets of high-dimensional data effectively; 3) specialized arithmetic features to speed up computations and maximize efficiency; 4) limited cost, to ensure maximal market impact of the embedded technologies.

From an architectural viewpoint, the class of Digital Signal Processors (DSPs) might provide a suitable tradeoff

between computational power and cost efficiency. Indeed, DSP devices have been developed as a cost-effective solution to the implementation of embedded systems.

As compared with other software programmable devices (e.g. general purpose microprocessors), DSPs provide lower power dissipation and more flexibility. DSPs devices mainly feature specific hardware solutions to support math-intensive signal-processing applications in a computationally efficient manner.

In the specific context of SVMs, hardware-looping capabilities and SIMD architectures well fit the features of training algorithms, which are typically characterized by deeply nested loop cycles. Furthermore, modern high-end DSPs support configurable internal memory, which allow one to use on-chip memory either as a cache or as a mapped memory. Such a flexibility adds up to the conventional Harvard memory architecture; the dual-bus internal structure is virtually unique of DSP devices and boosts timing performance by allowing the DSP core to access program and data simultaneously.

DMA channels should then be used for data transfer from the external to the on-chip memory, so that the core can operate with limited latency. These features allow embedded implementations to exploit properly the limited amount of on-chip memory even when large data sets are involved. Incidentally, one notes that such a characteristic is hardly found on conventional microprocessors, which do not support directly addressable on-chip memory.

Finally, a pipelined architecture allows DSPs to process different instructions in parallel, thus obtaining a proper exploitation of the functional units on the device. On the other hand, when optimising the implementation of a computationally demanding algorithm, the ultimate effectiveness strongly depends on the inherent complexity of the algorithm itself. In particular, code sections with several conditional branches and jumps can dramatically degrade overall efficiency and performance by repeated *pipe flush* occurrences.

III. SVM TRAINING ON DSP-BASED ARCHITECTURES

The basic algorithm for SVM training described in LibSVM involves three main procedures (Fig, 1): 1) the selection of the working set, 2) the verification of stopping criteria, and 3) the update of the crucial quantities, namely, α_i, α_j, and $\nabla f(\boldsymbol{\alpha})$. The first two steps clearly play a crucial role to the ultimate effectiveness of the training algorithm.

Nevertheless, one should note that the *optimization* efficiency of decompositions algorithms is conventionally measured by the involved reduction in the number of iterations until QP convergence. When implemented on specific embedded devices, however, selection-based algorithms might exhibit peculiar features that ultimately tend to limit the *computational* efficiency of the overall system.

The crucial issue is that the implemented strategy should also take into account the internal architecture of the target electronic device. Toward that end, the research presented in this paper proposes a hardware-oriented reformulation of the optimization algorithm [8] for DSP-based embedded electronics.

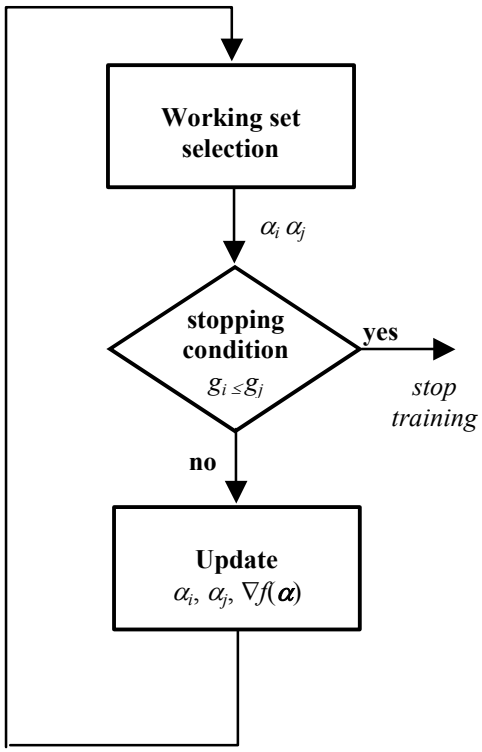

Fig. 1. The basic steps of the SVM training algorithm.

A. Training algorithm: reformulation and optimization

The algorithm by Keerthi *et al.* can be adapted by means of specific reformulations in the basic local-optimization steps, and the eventual optimization algorithm exploits a slightly modified heuristic. With respect to the original criterion formulated in the LibSVM library [10], the new heuristic proposed in this work replaces (4) with:

$$g_i \equiv \begin{cases} -\nabla f(\boldsymbol{\alpha})_i & \text{if } y_i = 1, \alpha_i < C \\ \nabla f(\boldsymbol{\alpha})_i & \text{if } y_i = -1, \alpha_i = C \end{cases} \tag{6a}$$

$$g_j \equiv \begin{cases} -\nabla f(\boldsymbol{\alpha})_j & \text{if } y_j = -1, \alpha_j < C \\ \nabla f(\boldsymbol{\alpha})_j & \text{if } y_j = 1, \alpha_j = C \end{cases} \tag{6b}$$

In summary, the new pattern-selection heuristic implied by (6) improves on computational efficiency by accepting the risk that a few patterns might violate one KKT condition.

From a hardware-oriented viewpoint, the rationale behind this approach is that fulfilling entirely the KKT conditions results in a computationally demanding task. On the other hand, one runs the risk that the method might reach a sub-optimal solution. Thus, the reformulation (6) specifically aims to balance solution accuracy and computational efficiency.

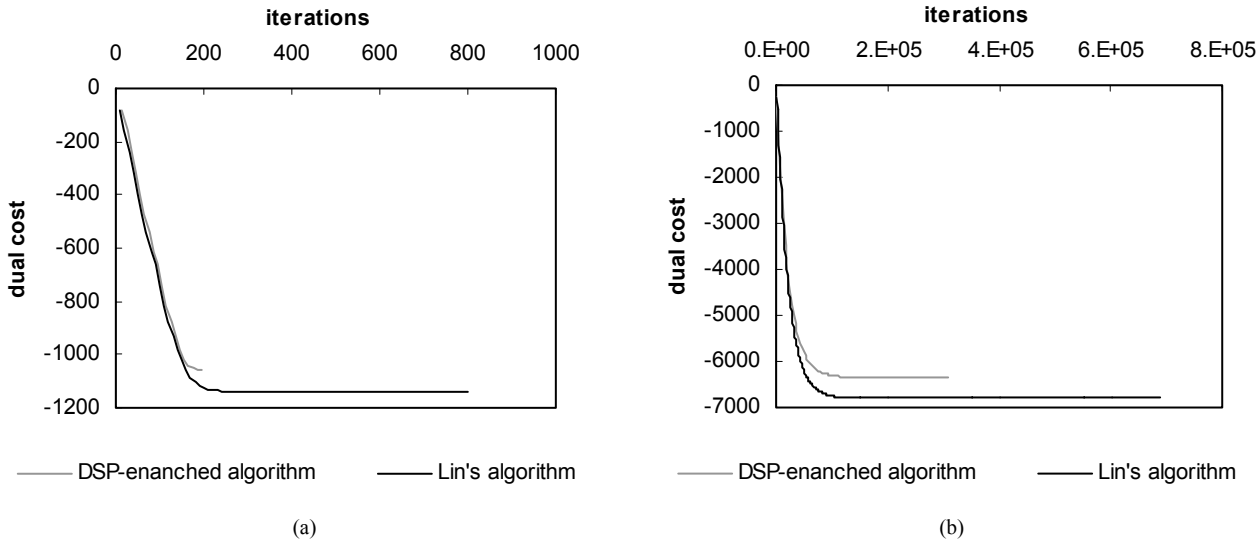

Fig. 2. Examples of training procedures performed by using the LibSVM criterion and the DSP-enhanced heuristic. The graphs plot the dual cost (1) versus the number of iterations. (a) Training results on the "Diabetes" testbed; (b) Training results for the "Sonar" testbed.

From a cognitive viewpoint, one might justify the above modified heuristic in terms of region representation. The Lagrange multipliers which determine the partial violation of KKT conditions (6) are associated with patterns that would anyway be identified as support vectors, and lie far from the margin area. Therefore, the eventual effect of the reformulated heuristic consists in a (possibly minor) slant of the margin area with respect to the optimal configuration.

The ultimate validation of this approach should clearly stem from experimental evidence. Toward that end, the graphs in Fig. 2 plot the dual cost (1) on the *y*-axis versus the number of iterations. The two graphs compare the results obtained by training an SVM with the original criterion (continuous line) with those resulting from the DSP-enhanced heuristic (6) (gray line). Fig. 2a presents the costs for the "Pima-Indians Diabetes" testbed [16], whereas Fig. 2b is associated with tests on the "Sonar" testbed [17].

The graphs show that the algorithm based on the DSP-enhanced selection strategy terminates in a number of iterations that is significantly smaller than that required by the original criterion.

```
for l = 0 to np

    if ( alpha[l] < C ) then
        if  (−grad[l] > g_i ) then
            if ( y[l] == 1 ) then
                g_i = −grad[l];   idxG1= l;
            endif
        endif
        if (−grad[l] > g_j ) then
            if ( y[l] == −1 ) then
                g_j  = −grad[l];   idxG2= l;
            endif
        endif
    endif

    if ( alpha[l] >0 ) then
        if ( grad[l] > g_j ) then
            if ( y[l] == 1 ) then
                g_j = grad[l];   idxG2= l;
            endif
        endif
        if ( grad[l] > g_i ) then
            if ( y[l] == −1 ) then
                g_i = grad[l];   idxG1= l;
            endif
        endif
    endif

endfor
```

Fig. 3. Pseudocode of LibSVM [10] strategy as per (4).

```
for l =0 to np
    sign= 1;
    if ( 0 <= alpha[l] < C ) then
        sign= −1;
    endif

    grad_new= grad[l]*sign;

    if ( grad_new > g_i ) then
        if  (y[l] == −sign) then
            g_i = grad_new; idxG1= k;
        endif
    endif

    if ( grad_new > g_j ) then
        if (y[l] == sign) then
            g_j = grad_new; idxG2= k;
        endif
    endif

endfor
```

Fig. 4. Pseudocode of the DSP-enhanced selection strategy as per (6).

On the other hand, the dual cost attained by the proposed algorithm only approximates the optimal solution; when considering that the resulting SVMs operate as classifiers, one should note that even if the dual costs differ from each other, the classification error is the actual cucial quantity. Further experiments on several other testbeds confirmed these conclusions.

In order to verify that the proposed heuristic contributes to limit the algorithm complexity, Fig. 3 and Fig. 4 give the pseudocodes for the original and the reformulated criterion, respectively. In the pseudocode, *grad*[*k*] indicates $\nabla f(\boldsymbol{\alpha})_k$, and *idxG1* and *idxG2* denote the indexes i, j associated with α_i, α_j, respectively. A comparison between the two pseudocodes shows that the modified heuristic sharply reduces the number of conditional operations (i.e., code segments of the kind "if...then...else") that lie in the inner section of the main loop. This in turn maximizes the efficiency of a DSP-based implementation, by limiting the occurrence of *pipe-flush* phenomena.

B. Training algorithm: basic porting

In fact, also step 3) involves some critical aspects for the DSP implementation since updating $\nabla f(\boldsymbol{\alpha})$ requires the access to the Hessian matrix Q. In general, the Hessian matrix needs an amount of memory that is available only in the memory external to the DSP. Thus, to implement the updating task efficiently, bandwidth becomes a critical aspect in transferring data from external RAM to the on-DSP memory.

The specific internal architectures of DSP devices allow one to attain optimal performance, as the Harvard schema provides separate, independent memory sections for program and data buses. Moreover, DMA channels allow independent memory loading from external to internal memory. Hence, in the proposed implementation, the DSP-internal RAM operates as a fast caching device, where matrix subsections are copied in turns. The eventual design strategy follows the architectural approach sketched in Fig. 5. The working set selection defines the columns of the Hessian matrix involved in the update procedure; then, DMA-channels support data loading from external to internal memory. As a result, data transfer from the cache proceeds in parallel with ongoing computations.

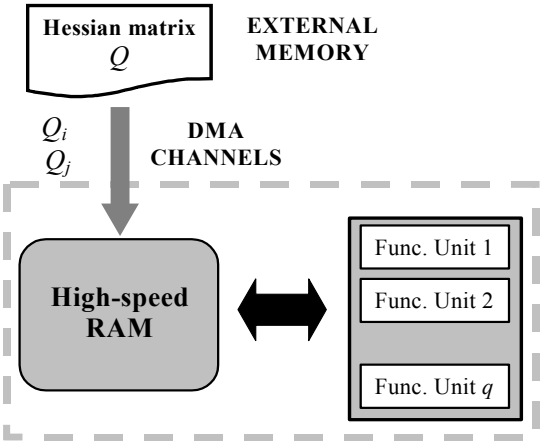

Fig. 5. The design strategy supporting data loading from the Hessian matrix.

To further improve the computational efficiency of the algorithm on the DSP architecture, specific reformulations involved the computation of α_j^{new}, which is defined by:

$$\alpha_j^{new} \equiv \begin{cases} \alpha_j + \dfrac{-g_i - g_j}{2(1+Q_{ij})} & \text{if } y_i \neq y_j \\ \alpha_j + \dfrac{g_i - g_j}{2(1-Q_{ij})} & \text{if } y_i = y_j \end{cases} \tag{7}$$

The division operation provided by (7) has been implemented by using two look-up tables, which provide the following values:

$$Lut1_{ij} = \frac{1}{2(1+Q_{ij})}; \quad Lut2_{ij} = \frac{1}{2(1-Q_{ij})}. \tag{8}$$

Thus, the division operator in (8) is replaced by a multiply operator, which better fits the architectural features of DSPs. The look-up tables are computed off-line and reside in the external memory. Hence, the proper value is loaded into the internal memory depending on the working set selection.

IV. Experimental Results

A. Application and architectural issues

This section describes the results obtained from the implementation of the proposed training algorithm on an "ADSP-BF533 Blackfin" Analog Devices® DSP [18] (denoted, in the following, as 'Blackfin' for brevity), running at a clock speed of 270 MHz. This device combine a 32-bit RISC instruction set with a dual 16-bit multiply accumulate (MAC) digital signal processing functionality. Moreover, the Blackfin DSP provides flexible Single Instruction, Multiple Data (SIMD) capabilities and support hardware looping and static branch prediction that allow for improvement while executing loop with conditional instruction inside them.

The memory architecture is hierarchical and provides a fast on chip memory (L1) and a slower off-chip memory. However, this development platform supports DMA data transfer between internal and external data memory, thereby fulfilling the bandwidth condition for optimal data-transfer rates. Besides, L1 memory can be configured as mapped memory or as cache in order to accommodate the configuration to specific algorithms.

Fixed-point representation brings about some loss in

TABLE I
PERFORMANCE COMPARISON BETWEEN THE ENHANCED ALGORITHM AND LIBSVM

(A) SPAM TESTBED

	Proposed heuristic (DSP)	LibSVM criterion (DSP)	LibSVM criterion (PC)
Iterations	86	172	55
Dual cost	-31.894	-31.978	-31.916
CE	1	1	1
SV	63	65	66
Clock cycles	598010	3194066	-

(B) BANANA TESTBED

	Proposed heuristic (DSP)	LibSVM criterion (DSP)	LibSVM criterion (PC)
Iterations	52	97	51
Dual cost	-96.772	-96.649	-96.863
CE	49	51	51
SV	99	98	98
Clock cycles	357755	1799341	-

(C) IONOSPHERE TESTBED

	Proposed heuristic (DSP)	LibSVM criterion (DSP)	LibSVM criterion (PC)
Iterations	54	69	47
Dual cost	-49.514	-49.707	-49.606
CE	12	12	12
SV	73	72	71
Clock cycles	378150	1301331	-

(D) PIMA-INDIANS DIABETES TESTBED

	Proposed heuristic (DSP)	LibSVM criterion (DSP)	LibSVM criterion (PC)
Iterations	49	49	49
Dual cost	-92.043	-92.043	-91.828
CE	51	51	50
SV	98	98	98
Clock cycles	334770	1254342	-

precision, as compared with the performance that could be attained by the higher resolutions provided by floating-point representations. Preliminary analysis pointed out that a 16-bit quantization level conveyed an acceptable degradation for the problem at hand. At the same time, the Q15 format representation allowed one to exploit the available 16-bit multipliers in parallel within the Blackfin DSP core.

B. Performance analysis (accuracy and optimisation)

The results presented in Table I compare the performances of the original training algorithm with the enhanced version implemented on the target DSP platform.

TABLE II
PERFORMANCE COMPARISON BETWEEN THE DSP-BASED IMPLEMENTATION OF SVM TRAINING AND THE PC-BASED IMPLEMENTATION OF THE ORIGINAL LIBSVM

	DSP	PC
Spam	1150 μs	573 μs
Banana	1300 μs	596 μs
Ionosphere	1224 μs	573 μs
Diabetes	1350 μs	550 μs

To allow a fair comparison, training sessions were always performed by using the following parameter settings: C=1, $2\sigma^2$=100.

Table I compares the performances of the two algorithms by giving the following quantities: the number of iterations required by the training procedure, the dual cost (1) attained at convergence, the classification performance of error percentage (CE), the number of support vectors (SV), and the clock cycles required. The last column gives the results obtained by completing the SVM training on a high-end PC supporting the original heuristic included in LibSVM.

Table I (A) presents the training results on the "Spam" testbed [19]; the dataset includes 100 patterns belonging to two possible classes, and involves a 57-dimensional distribution. Likewise, the sections Table I (B), I (C), and I (D) present the results obtained on the "Banana" [16], "Ionosphere" [19], and "Pima-Indians Diabetes" testbeds [16], respectively.

Empirical evidence confirms that the DSP-enhanced heuristic improves computational efficiency by attaining satisfactory performances in term of dual cost and digital cost. Experimental evidence shows that when implemented on a DSP LibSVM requires a computational effort that is five times larger than the effort required by the proposed heuristic. The proposed heuristic always outperforms the original heuristic in term of computational efficiency. Nonetheless, the dual cost and the classification error are very close to the reference values obtained by running LibSVM on a PC.

C. Performance analysis (timing and efficiency)

The results presented above prove that the DSP-enhanced version of the original LibSVM criterion well fits the limited hardware resources and the peculiar computational features of embedded devices. Nonetheless, it is worth to compare the computational performances of the DSP-based implementation of SVM training with the PC-based implementation of the original LibSVM.

To this purpose, Table II reports the timings required to complete the DSP-based SVM training and for the PC-based implementation of LibSVM, respectively. The experiment involved the blackfin DSP presented above (clock speed of 270 MHz) and a PC with a PentiumIII@550Mhz. The results refer to the four testbed presented above.

The figures show that the DSP training exploiting the proposed algorithm takes about two times the time required by the conventional training performed on a PC. To make a fair comparison, one should consider that the clock rate of the PC is more than twice the clock rate of the DSP. Hence, However, when normalizing clock speeds, these results provide evidence that the proposed DSP-enhanced training algorithm improves the computational efficiency of the embedded system.

V. Conclusions

The implementation of training algorithms for SVMs on embedded architectures differs significantly from the electronic support of trained SVM systems. This mostly depends on the complexity and the computational intricacies brought about by the optimization process, which implies a Quadratic-Programming problem and usually involves possibly large data sets.

When dealing with digital approaches to that problem, the specific requirements of agile memory handling and efficient support of massive arithmetic operations seem to indicate DSPs as a viable solution to the hardware-development problem.

At the same time, existing algorithms for efficient SVM training mostly consider the number of iterations as the key quantity to optimize to measure (and limit) complexity. In the context of embedded implementations, however, computational efficiency is often paramount because it determines the ultimate timing performance of the training device.

Therefore one expects that some modifications or reformulations of exiting methods may be introduced to best fit the specific architectural features of the supporting machinery. In the present work, a slight adjustment of the working-set selection and stopping criterion implemented in LibSVM allowed to reach remarkable improvements in efficiency when porting those algorithms on DSP platforms.

Speed-up measurements, comparing the resulting embedded systems with both the original criterion and a conventional PC-based implementation, always confirmed the validity and effectiveness of the proposed approach. Empirical evidence showed that the reformulated heuristic and the associate porting methodology well matched the optimal value attained by the original algorithm criterion, but at the same time greatly boosted the corresponding time and power-consumption performance.

The obtained results were obtained under pre-selected experimental conditions (i.e., hyperparameter settings), which were set arbitrarily and without loss of generality by exploiting previous knowledge. In a general context, choosing the hyperparameters requires one to perform a model-selection task, and in this case the availability of the efficient DSP-based platform may prove quite useful, by providing a cost-effective co-processing resource for the notable computational load involved by iterative model selection.

The main issues that remain to investigate in this research mostly concern a formal justification of the proposed heuristic, whose performance and cognitive interpretation open new vistas on the development of improved methods for effective SVM training. This is especially interesting when considering those classes of application domain, which require continuous, on-line training and involve high-dimensional data spaces.

References

[1] V. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[2] Christopher J.C. Burges, "A tutorial on Support Vector Machines for Pattern Recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121-67, 1998.

[3] N. Cristianini and J. Shawe-Taylor, *An introduction to Support Vector Machines*. Cambridge, U.K.: Cambridge University Press 2000.

[4] B. Schölkopf and A. Smola, *Learning With Kernels*. Cambridge, MA: MIT Press, 2002.

[5] C.-J. Lin, "On the convergence of the Decomposition Method for Support Vector Machines," *IEEE Trans. Neural Networks*, vol. 12, pp. 1288-98, 2001.

[6] T. Joachims, "Making large-Scale SVM Learning Practical," in *Advances in Kernel Methods - Support Vector Learning*, B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999.

[7] J. C. Platt, "Fast training of Support Vector Machines using Sequential Minimal Optimization," in *Advances in Kernel Methods - Support Vector Learning*, B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds. Cambridge, MA: MIT Press, 1999.

[8] S. Keerthi, S. Shevade, C. Bhattacharyya and K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, pp. 637-649, Mar. 2001.

[9] C.-J. Lin, "Asymptotic convergence of an SMO algorithm without any assumption," *IEEE Trans. Neural Networks*, vol. 13, pp. 248–250, Jan. 2002.

[10] C.-C. Chang, C.-J. Lin, *LIBSVM: a Library for Support Vector Machines*. Available at: http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[11] D. Anguita, A. Boni, and S. Ridella, "A digital architecture for support vector machines: theory, algorithm and FPGA implementation," *IEEE Trans. on Neural Networks*, vol. 14, no. 5, pp. 993-1009, Sept. 2003.

[12] V. Vapnik, E. Levin, Y. Le Cun "Measuring the VC-dimension of a learning machine", 1994, Neural Computation, 6:851—876.

[13] O. Chapelle, V. Vapnik, O. Bousquet, S. Mukherjee "Choosing multiple parameters for Support Vector Machines" *Machine Learning*, 2002, vol.46, No.1-2, pp.131-159

[14] K. Duan, S. Keerthi, A. Poo "Evaluation of simple performance measures for tuning svm hyperparameters", 2001, *Technical Report CD-01-11*, Dept. of Mech. Eng., Natl. Univ. Singapore, Singapore.

[15] D. Anguita, S. Ridella, F. Rivieccio, R. Zunino "Hyperparameter tuning criteria for Support Vector Classifiers" *Neurocomputing*, October 2003, No.55, pp.109-134

[16] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for AdaBoost," *Machine Learning*, vol. 42, no. 3, pp. 287-320, March 2001.

[17] J. M. Torres-Moreno and M. B. Gordon, "Characterization of the sonar signals benchmark," *Neural Processing Lett.*, vol. 7, pp. 1–4, 1998.

[18] Analog Devices, *ADSP-BF533 EZ-KIT Lite - Evaluation System Manual*. Available at: htttp://www.analog.com

[19] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, *UCI repository of machine learning databases*. Irvine, CA: University of California, Department of Information and Computer Science. Available at: http://www.ics.uci.edu/~mlearn/MLRepository.html