UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

INVERSE PROBLEMS WITH SVM-BASED RUN-TIME RECONFIGURABLE SYSTEMS

Andrea Boni, Fernando Pianegiani, Dario Petri

May 2004

Technical Report # DIT-03-059

Inverse Problems with SVMs–based Run-Time Reconfigurable Systems

A. Boni, F. Pianegiani, D. Petri

Department of Information and Communication Technology

University of Trento,

Via Sommarive, 14 – 38050 Trento, Italy

Phone: +39 0461 883902, Fax: +39 0461 882093, E–mail: petri@dit.unitn.it.

Index Terms

Inverse modeling, Support Vector Machines (SVMs), System on Chip (SoC), dynamic reconfiguration.

I. Introduction

In the last few years, the evolution of microelectronic technologies has promoted the development of measurement systems that extract the information of interest using learning by examples methodologies instead of a priori defined algorithms [1]. Application of learning by examples techniques, such as Artificial Neural Networks (ANNs), are appealing because allow to model a system without knowing its analytic structure, and using only a set of input/output samples, also called *training set*. Recently, important developments in Statistical Learning Theory (SLT) [2] have introduced new paradigms that overcome several drawbacks of ANNs such as the structure of the learning algorithm and the absence of a solid theoretical background. Among other methodologies based on theorems from SLT, the so–called Support Vector Machines (SVMs) seem to be the most appealing [1]. Their main characteristic is the structure of the learning algorithm, which consists in the solution of a simple constrained quadratic optimization problem. In practice, SVMs find a set of parameters during a *learning phase*, which are used in a *forward phase* to estimate the desired outcome. Hardware platforms suitable for the execution of such tasks are systems able to change at run–time their configuration in order to carry out different processing algorithms. Over the last few years, the development of Field Programmable Systems on Chip heralded the emerging technology of the hardware that can be dynamically reconfigured. The use of such architectures adds a new dimension to the design of adaptive measurement systems.

In this paper we face a general inverse–modeling problem [3] and describe the design and the implementation of a complete adaptive system based on SVMs and reconfigurable Field Programmable Gate Array (FPGA) devices. In section II the problem is formulated from a theoretical point of view. In section III simulation results on a typical equalization problem are given. Finally, in the last section, a complete description of the hardware–platform design of the considered case of study and the performances achieved with the hardware implementation are reported.

II. Problem formulation

Adaptive systems are applied in many fields, such as classification of input patterns, system identification, prediction and noise cancellation [3]. They are characterized by the coexistence of many interdisciplinary areas, such as ANNs, SLT and signal processing. Here, we focus our attention on inverse–modeling problems, where a special–purpose adaptive architecture can be fruitfully used in order to estimate a discrete signal $u(n)$, input of a nonlinear discrete system, on the basis of the signal $x(n)$ observed at the output of the system. Usually, the classical theory tackles this problem by finding an

optimal estimator, for example the Bayesian Maximum Likelihood (ML) detector, that provides an assessment $\hat{u}(n-D)$ of $u(n-D)$ through the observation of an r–dimensional *feature* vector $\boldsymbol{x}_n^{(r)} = [x(n), x(n-1), \ldots, x(n-r+1)]^T$, where D represents the intrinsic delay of the estimator and r the minimum number of channels useful to obtain a reliable estimation. Conversely, SVMs select an estimator of the input signal from a given class of functions on the basis of a set of m previous observations of the input and output signals:

$$\boldsymbol{z}^{(m)} = \left\{ \left(\boldsymbol{x}_i^{(r)}, u_i \right) \right\}_{i=0}^{m-1} \tag{1}$$

Notice that in the following of this paper the signals are identified by the indexes i and n during the *learning* and the *forward phases* respectively.

In [4] several advantages of SVMs with respect to the state of the art of equalization methods are reported. The authors also suggest two open issues, such as the need for both an efficient implementation and an adaptive processing, not resolved at that time. In this work we propose a solution for such requirements. Here, we provide some brief details on SVMs for classification, where a classifier has to be defined in order to separate two different set of observations [1], [2]. In accordance to the maximum generalization criteria formalized by the Vapnik and Chervonenkis' theory [2], in order to identify the best classifier for a given set of linearly separable observations $\boldsymbol{z}^{(m)}$, SVMs try to find the maximum–margin separating hyperplane, where such margin is defined as the maximum distance between the closest samples belonging to two different classes. However, in real–world problems the available set $\boldsymbol{z}^{(m)}$ is often not linearly separable. It is necessary to use a nonlinear function $\varphi : \Re^r \to \Re^R, R >> r$ that maps each element $\boldsymbol{x}_i$ of $\boldsymbol{z}^{(m)}$ in a new high–dimensional feature space, where the maximum–margin hyperplane can be found [1], [2]. From a mathematical point of view, the most important characteristic of SVMs consists in the fact that they do not require the explicit knowledge of the function φ. In effect, the nonlinear mapping is implicitly computed by a kernel function $K(\boldsymbol{x}, \boldsymbol{y}) = \varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{y})$ that typically can be linear ($K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y}$), Gaussian ($K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{y}\|^2 / 2\sigma^2\right)$) or polynomial ($K(\boldsymbol{x}, \boldsymbol{y}) = (1 + \boldsymbol{x} \cdot \boldsymbol{y})^p$). As a consequence, the structure of the estimator, used in the *forward phase*, is defined as $\hat{u}(n-D) = \sum_{i \in SV} \alpha_i u_i^* K\left(\boldsymbol{x}_i^{(r)}, \boldsymbol{x}_n^{(r)}\right) + b$, where $u_i^* = u_i$ for classification problems and $SV = \{i : \alpha_i \neq 0\}$ is the index set of the support vectors $\boldsymbol{x}_i$, with $i \in SV$. The parameters α_i and b are computed during the *learning phase* by solving an optimization problem in which the parameters α_i are constrained to lie in the box $[0, C], \forall i \in SV$. In the classification case, C is the parameter that controls the tradeoff between the generalization ability of the classifier and the number of misclassified input patterns, when $\boldsymbol{z}^{(m)}$ is not linearly separable in the feature space. In this work we use a Gaussian kernel, because of its capability of providing robust solutions in classification problems [1]. Finally, the parameters σ^2 and C are set after a model selection criteria [5].

III. Simulation results

In this section, we consider the equalization of a nonlinear channel as a typical case of study of inverse modeling problems. In such case, a symbol $u(n) \in \{+1, -1\}$, generated by a given source, has to be estimated by the receiver, after passing through a noisy channel having intersymbol interference of length N. The unpredictable nonlinear effects caused from the involved components (i.e., transmitter, channel, receiver) are usually modeled as FIR filters, plus a Gaussian distributed noise e with zero mean and variance σ_e^2. The following expressions describe such a kind of model:

TABLE I

PARAMETERS VALUES OF DIFFERENT MODELS

Model	h_0	h_1	h_2	c_1	c_2	c_3
1	1	0.5	-	1	0	-0.9
2	0.5	1	-	1	0.1	0.05
3	0.3482	0.9704	0.3482	1	0.2	-

TABLE II

BAYESIAN MAXIMUM LIKELIHOOD CLASSIFIER VS. SVM FOR *Model 1* AND *Model 2*

Model	D	ML(%)	SVM(%)	C–σ^2
1	0	14.4	15.6	1.6–0.4
1	1	5.2	5.4	1.6–0.4
1	2	3.7	3.5	3.2–1.6
2	0	13.7	12.1	1.6–1.0
2	1	4.3	4.6	1.6–1.0
2	2	0.7	0.7	1.6–1.0

$$\begin{aligned} \tilde{x}(n) &= \sum_{k=0}^{N-1} h_k u(n-k) \\ \hat{x}(n) &= \sum_{p=1}^{P} c_p \tilde{x}^p(n) \\ x(n) &= \hat{x}(n) + e(n) \end{aligned} \tag{2}$$

where N represents the duration of the filter time–response and P is the order of the nonlinearity. In the following we apply the ML [6] and SVM based approaches to the considered case and compare their performance, in terms of bit error rate. Note that the former method requires the knowledge of the channel (input constellation and noise statistics), whereas the latter works only by using a set of samples. In order to test the SVM classifier, several data have been collected according to equation (2) and by using three different nonlinear models of the channel (see table I): $N = 2$, $P = 3$ and Gaussian white noise (*Model 1*); $N = 2$, $P = 3$ and Gaussian colored noise (*Model 2*); $N = 3$, $P = 2$ and Gaussian colored noise (*Model 3*). In the *Model 2* and *Model 3*, the noise was generated by using the following FIR filtering:

$$e(n) = \frac{\sigma_e}{\sqrt{1+\xi^2}} w(n) + \frac{\sigma_e \xi}{\sqrt{1+\xi^2}} w(n-1) \tag{3}$$

where w is an uncorrelated noise with zero mean and variance $\sigma_w^2 = 1$, while $\xi = 0.75$. As a first experiment, we fixed $\sigma_e^2 = 0.2$ and considered three different delay values ($D = 0, 1, 2$) in order to maximize the performance of the equalizer. Moreover, a Gaussian kernel function has been selected. Table II reports the results for the *Model 1* and *Model 2*, obtained by considering 500 training samples and 3000 test samples. In the same table C and σ^2 represent the SVM hyperparameters found after a model selection process [5]. As a second experiment, the effect of the number of training samples (m) and the noise variance on the SVM performance have been tested by using *Model 3* as a function. Notice that the number of observations m is important because it determines the complexity of the required hardware platform and the delay of the system to change in the input signal. Table III shows the results, for $r = 3$ and $D = 2$. Tables II and III confirm the validity of the SVM–based approach, as the designed estimators often overcome the classical

TABLE III
SIMULATIONS WITH DIFFERENT TRAINING SAMPLES FOR *Model 3*

m	σ_e^2	ML(%)	SVM(%)	C–σ^2
500	0.1	1.8	1.7	4–0.1
500	0.2	5.2	4.7	2–1.6
500	0.3	8.9	7.5	4–6.4
500	0.4	11.9	10.5	4–12.8
128	0.1	1.8	1.7	16–1.6
128	0.2	5.2	6.1	32–0.8
128	0.3	8.9	8.3	8–3.2
128	0.4	11.9	10.7	32–12.8
64	0.1	1.8	1.8	16–1.6
64	0.2	5.2	7.5	16–0.8
64	0.3	8.9	9.0	8–6.4
64	0.4	11.9	11.9	32–12.8
32	0.1	1.8	2.0	8–0.8
32	0.2	5.2	8.1	16–0.8
32	0.3	8.9	10.8	8–3.2
32	0.4	11.9	13	8–12.8

maximum likelihood classifier. This behavior is due to the Gaussian distribution of the noise that, together with the use of Gaussian kernels, allows to design a classifier that is very close to the best one, if a sufficient number of training samples are considered [2]. As a final remark note that the performances reported on table III are just a test on a single realization composed of about 3000 samples obtained using the same seed. In the final paper we will report a detailed bootstrap–based statistical validation.

IV. System architecture and performances

This section focus on the design, implementation and performance analysis of an efficient hardware platform for the proposed equalization problem. The design of SVM classifiers is not new to the scientific community [7]. Here, instead of using a mixed–signal VLSI processor like in [7], an FPGA-based processor has been employed as target device. In such a way, a completely reconfigurable system on chip that adapt the characteristics of the estimator to the behavior of nonlinear transmission channels can be implemented. Current generation of FPGA platforms are powerful systems equipped with high density programmable logic and embedded Block RAMs (BRAMs), multipliers and hardware and software CPU cores. As shown in Fig. 1(a), such characteristics together with advanced techniques of dynamic reconfiguration have been used to design an architecture mainly composed by [8], [9]: a general purpose processor, which collects input data and acts as a system supervisor; a module that can be dynamically reconfigured to alternately implement the FIBS or the KTRON cores, which carry out the *learning* and *forward phases*, respectively. In particular, the FIBS core finds and stores in a memory the parameters α_i, b and the set of support vectors $\boldsymbol{x}_i$. The KTRON, so called for its similarities with the recently proposed Kernel Perceptron algorithm [10], receives the set of parameters found by the FIBS and estimates the input signal of the nonlinear system. The FIBS–KTRON module is dynamically reconfigured as soon as a learning process is required, when a new set of observation is available. In this first version, this occurs after a synchronization procedure between the transmitter and the receiver, executed at fixed intervals. In practice, the transmitter sends to the receiver an a priori known sequence, used for training.

In [6] a full description of the FIBS architecture is reported. In brief, it makes use

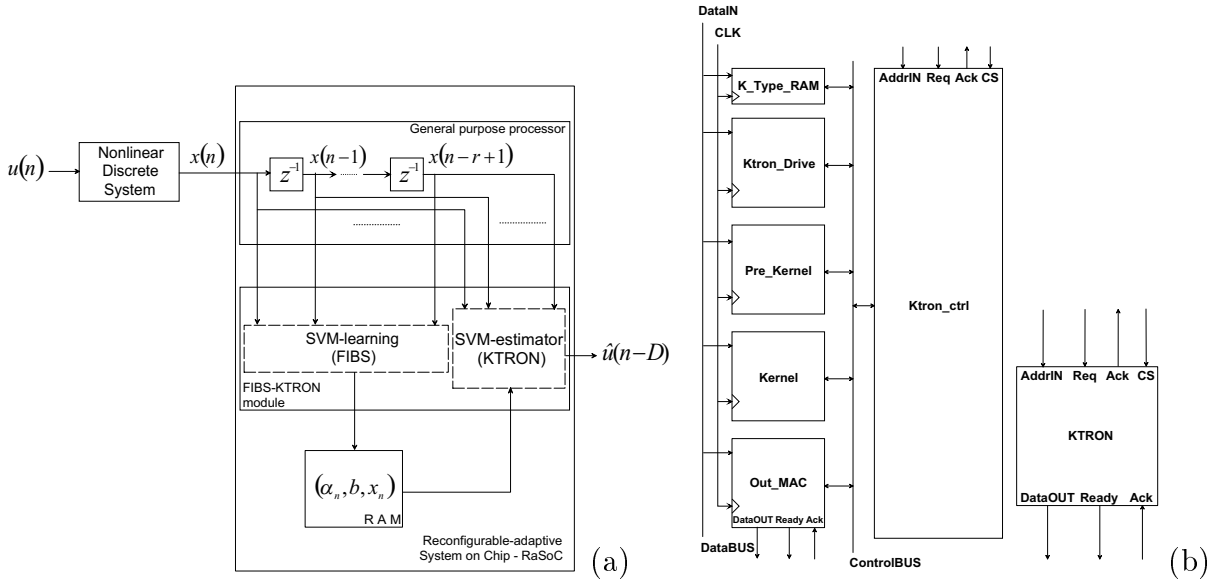

Fig. 1. Basic block diagram of an inverse–modeling estimator (a) and the KTRON architecture (b).

of a new algorithm for SVM learning, which is less sensitive to quantization errors with respect to the solution appeared so far in the literature. The core is composed of two parts: the first one exploits a recurrent network for finding the parameters of the SVM classifier; the second one uses a bisection process for computing the threshold b. Since the SVM classification function is very similar to the one realized by a perceptron, the proposed KTRON architecture, reported in Fig. 1(b), takes its inspiration from the TOTEM processor, which was recently implemented on a programmable logic device [11]. Starting from the VHDL high–level description of the TOTEM processor, an hardware implementation of the KTRON co–processor has been designed. The obtained architecture is shown in figure 1(b). *K_Type_RAM* is a simple flip–flop containing a flag indicating the type of computations carried out by the *Pre_Kernel* unit. *Ktron_Drive* contains the RAMs to storage both the set of support vectors and the input $\boldsymbol{x}$ to be processed. *Pre_Kernel*, the first processing unit, computes an inner product or a squared norm according to the value in the *K_Type_RAM* module. *Kernel*, a second processing unit, computes in practice the kernel function, by using a look-up table (LUT) in which kernel values are stored. Note that in this first version of the prototype, the division for $2\sigma^2$ has been implemented by approximating $2\sigma^2$ to a power of 2 and using a shift register. *Out_MAC*, the last computation unit, multiplies and accumulates the results provided by the *Kernel* unit. *Ktron_ctrl* is the main control unit of the core. In practice, only two embedded multipliers are used, one in the *Pre–Kernel* module, and another in the *Out_MAC* module.

In order to design the prototype of the KTRON core, 32 samples for training with $r = 2$ and a Gaussian kernel with $2\sigma^2 = 1$ have been used. Data were internally represented by 16–bit in 2's complement coding, with 3 bits and 13 bits for the integer and fractional parts, respectively. Such values were obtained after a model selection criterion, in order to reach a classification error of 4.3% on *Model 2* [6]. The whole architecture was implemented on a Xilinx Virtex II (XC2V1000) by using the Xilinx ISE 5.2i and XST as development and synthesis tools, respectively. Our core maps 280 Virtex–II Slices (5.6%) and works at a clock frequency of 100 MHz. Four embedded 2 KByte Block RAM (BRAM) of the Virtex II, used to store the support vectors, the weights, the $\boldsymbol{x}$ vector to be processed and the kernel LUT have been instantiated. In practice, the four BRAMs allow to store up to 100 support vectors of $r = 10$ features each. The number of clock cycles needed to obtain the result is around 430, for $m = 32$ and $r = 2$, and 7300, for $m = 100$ and $r = 10$. As a final remark, note that the percentage of logic used by the FIBS core is around 58% (2950 slices) and by the KTRON is about 6% (289 slices), while the remaining part of

the FPGA is used to implement the general purpose processor and all I/O interfaces.

References

[1] B. Scholkopf, A. Smola, *Learning with kernels*, The MIT Press, 2002.

[2] V. Vapnik, *Statistical Learning Theory*, Wiley, 1998.

[3] B. Widrow, *Adaptive Signal Processing*, Prentice Hall, 1985

[4] D.J. Sebald, J.A. Bucklew, *Support Vector Machine Techniques for Nonlinear Equalization*, IEEE Trans. on Signal Processing, Vol. 48, No. 11, 2000, pp. 3217–3226.

[5] D. Anguita, S. Ridella, F. Rivieccio, R. Zunino, *Hyperparameter Design Criteria for Support Vector Machines*, Neurocomputing, Volume 55, Issues 1–2, Pages 109–134, September 2003.

[6] D. Anguita, A. Boni, S. Ridella, *A Digital Architecture for Support Vector Machines: theory, algorithm and FPGA implementation*, IEEE Trans. on Neural Networks – Special Issue on Hardware Implementations, in press., 2003.

[7] R. Genov, G. Cauwenberghs: *Kerneltron: Support Vector Machine in Silicon*, IEEE Trans. on Neural Networks, in press., 2003.

[8] D. Mesquita, F. Moraes, J. Palma, L. Moller, N. Calazans, *Remote and partial reconfiguration of FPGAs: tools and trends*, Proc. Parallel and Distributed Processing Symposium, pp. 177–184, France, Apr. 22–26, 2003.

[9] Xilinx Inc., *Two Flows for Partial Reconfiguration: Core Based or Small Bit Manipulations*, Application Note No. 290, May 2002.

[10] D. Anguita, A. Boni, S. Ridella, *Digital Kernel Perceptron*, Electronics letters, Vol. 38, n. 10, pp. 445–446, 2002.

[11] S. McBader, L. Clementel, A. Sartori, A. Boni, P. Lee: *SoftTotem: an FPGA Implementation of the Totem Parallel Processor*, Proc. FPL2002, France, Sept. 2–4, 2002.