Exploiting Application Locality to Design Fast, Low Power, Low Complexity Neural Classifiers

Cesare Alippi
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Milano, Italy
alippi@elet.polimi.it

Fabio Scotti
Department of Information Technologies
Università degli Studi di Milano, Crema, Italy
fscotti@dti.unimi.it

Abstract— **The paper provides a design methodology for embedded classifiers particularly effective in those applications characterised by a temporal locality of the inputs. By exploiting application locality we reduce computational complexity and cache misses (hence speeding up the execution) as well as power consumption. A gated-parallel neural classifier has been found to be a particularly suitable structure since only one sub-classifier is active at time, the others being switched off. Results from industrial applications show that the suggested design methodology provide an accuracy comparable with more traditional classifiers yet yielding a significant complexity and execution time reduction.**

I. INTRODUCTION

In general, classifiers are designed by keeping in mind accuracy performance without taking care of constraints. This has pushed the related research to suggest techniques solely aiming at performance maximisation e.g., see, [1] without taking care of implementation-related aspects in the early phases of the classifier design cycle. Up to the best of our knowledge the unique implementation-related aspects considered at high abstraction level are related to the robustness/sensitivity issue for their impact on finite precision representation [2] and complexity reduction [3].

In this paper we address the classifier design problem by considering a variant of the gated-parallel model which allows the designer for integrating high level application properties in the classification module more effectively than standard monolithic, parallel, cascade, linear or hierarchical families [4]. In particular, instead of considering a single classifier (e.g., monolithic) dealing with the whole classification problem, we envisage a number of sub-classifiers activated by a master enabling module. The resulting "multiplexed classifier" is such that the master module enables only a sub-classifier at time, the others being switched-off. The grouping output module envisaged in the gated-parallel literature disappears for computationally complexity reasons and we allow sub-classifiers to be different in model nature and topology. A multiplexed classifier design is particularly effective in those applications possessing the temporal locality property (a sub-classifier enabled at time t has high probability to be enabled also at time $t+\tau$) and the partitioning property (the enabling module + sub-classifier complexity is smaller than that of the monolithic one).

In particular, the locality property, as in cache design, holds when patterns to be classified are time dependent, as it happens in classifiers working in several controlled environments. While the locality property grants that the enabled sub-classifier stays in the cache with high probability (cache misses arise only when another sub-classifier is loaded), the partitioning property implies that its computational complexity is smaller than that of the monolithic one (and hence, in addition to an independent execution gain, it is likely the sub-classifier to fully reside in the cache). In addition to the relevant power saving associated with reduction of cache misses in a SW implementation, all sub-classifiers but the active one can be switched off with a further gain in power consumption reduction. On the algorithm execution front both properties grant a reduced execution time as it will be explained in subsequent sections.

The novelties of the paper do not reside in the classifier structure but in the suggested design methodology exploiting application properties and integrating, at application level, accuracy and complexity/low power consumption constraints. The structure of the paper is as follows. The design issues for multiplexed classifiers are given in section II. Section III, by receiving the classifier delineated in section II, explores the design space with genetic algorithms to identify the most adequate classification families. Experimental results are given in section IV.

II. A MULTIPLEXED CLASSIFIER DESIGN

The design of a multiplexed classifier requires solution to the following subproblems:

1 identification of a suitable number k of sub-classifiers and clustering of the classification input space in k sub-domains, one for each sub-classifier;
2 selection of the features relevant to each sub-classifier;
3 design and configuration of the master enabling module;
4 selection and configuration of the k sub-classifiers.

Each step contributes to design a performing classifier as well as influencing the complexity (power consumption and execution time) aspect. To ease and make effective the methodology, we separate the design in two phases aiming at decoupling the accuracy from the complexity issue. In the first phase (steps 1-3) the focus is primarily on complexity: complexity reduction is obtained by acting on the number of

0-7803-8834-8/05/$20.00 ©2005 IEEE.

classifiers and their topology yet leaving enough degrees of freedom for a subsequent integration of the accuracy requirement. The second phase (step 4), primarily focuses on accuracy maximization by considering and configuring adequate models for sub-classifiers meanwhile penalizing complex solutions. We now detail the operations needed to accomplish steps 1-3.

A. Domain partitioning

The problem associated with the determination of the optimal sub-classifiers domain coincides with a data clustering problem w.r.t. the input space. Clustering can be carried out by considering the designer's favourite algorithm; here we considered a supervised Fuzzy C-Means clustering technique [5] for its effectiveness and computational simplicity. The number of clusters may either be given by the user or can be identified by the clustering algorithm as suggested in section 2.4 or, again, by considering the following heuristic: for each subdomain *k* must be increased by one whenever its partitioning into two halves provides two classifiers each of which having an input complexity below the starting one. The rationale is based on the fact that child sub-classifiers requiring a reduced number of input features are less complex than the father one requiring more inputs. On the other hand, by increasing *k* we increase the complexity of the master module: as such, a balance between the two must be identified. We empirically discovered that a reasonable *k* for industrial applications is below 7.

B. Feature selection

The feature selection step is fundamental in reducing the complexity of the multiplexed classifier since it addresses identification of the minimal number of input features to be presented to each sub-classifier; since sub-classifiers operate in a sub-domain it is likely that only few features will be sufficient to solve the local classification tasks. The designer can consider his favourite feature selection method: here, we considered the method based on a feature relevance analysis to solve the feature selection problem suggested in [6] for its proven effectiveness in industrial applications.

C. Master module construction

The master module can be easily built with a KNN classifier trained over the *k* centers of class identified in step II.A. During the operational phase of the multiplexed classifier the master module receives the pattern to be classified and activates a sub-classifier. A KNN master module provides a good accuracy/complexity compromise when the number of sub-classifiers is not very high, we should opt for a neural network of regression type otherwise. We suggest the designer to select the least complex solution between the two when *k* is above 7.

D. Complexity and application-level properties

In the following, we denote by Φ_{MC} the multiplexed classifier, by Φ_M the monolithic classifier and by C the complexity function. Without loss of generality we consider the monolithic classifier as the reference one for its easy configuration and wide use; of course we could use any other optimal classifier designed over the whole data set. Computational complexity must be associated with the application code profiling which, for a classification problem and a multiplexed classifier design, can be related to each sub-classifier activation. In order to evaluate the complexity of a computational flow we consider a classic approach which assigns a complexity cost (weight) to each high level instruction (e.g., see [7]). Indicative values for weights are given in table 1 where we reasonably assumed one clock cycle execution for register to register instructions. Weights must be intended as non-dimensional values relative to the integer sum (whose value is 1). The complexity (i.e., C or C_I) of a compiled code can then be estimated as the sum of the frequencies of each instruction amplified by the corresponding weight. We are assuming that non basic instructions, such as squared roots or activation functions for neurons, are either stored in a LUT table or expanded in series by using basic instructions. At this abstraction level we cannot differentiate cache accesses from memory ones; in the following we consider the average weight between the two every time we access memory.

Once the clustering algorithm has partitioned the application space, it is natural to associate an activation probability $P_I = \int_{\Omega_I} p(x)dx$ to the generic I-th sub-classifier. In other words P_I denotes the probability that an input pattern to be classified belongs to the I-th sub-classifier domain of complexity C_I (C_I accounts both for the complexity of the sub-classifier and that of the master module.). The complexity can then be expressed as

$$C(\Phi_{MC}) = \sum_{I=1}^{k} P_I C_I \qquad (2.1)$$

where the unknown probability can be estimated by the activation frequencies coming from the application profiler.

TABLE I
Weights associated with instructions of the classifiers code

Instruction group	*Weight*	*Instruction group*	*Weight*
Register Transfer Operation	$\alpha = 1$	Activation Function Evaluation	$\alpha = 52$
Floating Point Sum and Product	$\alpha = 1$	Read/Write Cache	$\alpha = 10$
Integer Sum and Product	$\alpha = 1$	Read/Write Memory	$\alpha = 100$

Several applications are characterized by time-dependent inputs so that two consecutive input patters are related in time. If we assume that the transformation from the input space to the feature space is related in time then also the associated input features will be time related. It is easy to extend temporal locality properties of caching mechanisms to multiplexed classifiers if we assume that the trajectory described by inputs in the feature space is confined in a limited region as it happens in automatic controlled applications. To test whether temporal locality is interesting to an application or not we can generate the $T_I(\tau)$ curve

which provides, for different values of τ, the probability that classifier I active at time t is *continuously* active up to time t+τ. Similarly, we can consider sequential locality.

Locality properties have an immediate positive effect in the caching mechanism with an obvious impact in power consumption reduction. This positive effect is based on the fact that only a sub-classifier is active at a time and that properly configured sub-classifiers are most of time significantly less complex than the monolithic classifier. An automatic procedure for selecting the number of classifiers *k* can be derived by exploiting the $\mathrm{T}_I(\tau)$ curves: since it is our interest to maximize temporal locality we can select the optimal *k* as the one maximizing the ensemble locality $\sum_{I=1}^{k}\sum_{\tau=1}^{N}\mathrm{T}_I(\tau)$, where *N* is the number of samples.

III. Exploring the design space

Once the topological structure of the classifier has been fixed the next design step requires configuration of each sub-classifier. This operation requires, for each sub-classifier:

1) identification of the most suitable classification structure (KNN, RBF, FF, SVM,…);
2) determination of the classification model family/kernel within each structure;
3) configuration/training of each sub-classifier model.

Obviously, the goal of the design activity is to configure a multiplexed classifier Φ_{MC} solving the accuracy/complexity tradeoff. To this end we considered the figure of merit (a different approach would involve generation of Pareto's optimal solutions):

$$J_{M,MC} = \min_{\Phi_{MC}\in\Theta}\left(\gamma\frac{C(\Phi_{MC})}{C(\Phi_M)} + \frac{A(\Phi_{MC})}{A(\Phi_M)}\right) \qquad (3.1)$$

where $A(\cdot)$ is the accuracy performance function (e.g., estimated with cross-validation, k-fold Cross-validation or Leave-one-Out). In the following, we consider cross-validation techniques for $A(\cdot)$ and the mean squared error loss function. $\gamma \geq 0$ is a weighting term for the accuracy and the complexity contributions (in our applications we identified a good accuracy/complexity performance with γ=0.1). To optimize (3.1) we envisaged a genetic algorithm optimization [8] by adopting standard selection, crossover and mutation genetic operators. The chromosome structure reflects the structure of the multiplexed classifier Φ_{MC}. In particular, information related to model hierarchy and model family is coded in the chromosome for each of the *k* sub-classifiers composing the classifier. As a consequence, the chromosome is composed of 2k genes. For each sub-classifier the first gene codes the model hierarchy, the second the model family. KNN, FF, RBF and linear classification hierarchies have been considered. Of course, we can decide to enlarge the genes allele by adding other classification families, i.e., SVM, Bayesian classifiers, etc. The second gene codes the value K, the norm in KNN and the number of hidden units and activation functions in FF and RBF.

IV. Experimental results

In this section we apply the methodology to three industrial applications. Application -I1- refers to a quality analysis process for the stainless steel laser cutting industry: the goal is to judge during the operational phase the local quality of the cut. Physical features are the cutting speed, the pressure of the shielding gas and four features associated with the sparks produced during the cutting: the spark jet presence, the angle of sparks core w.r.t the normal, its wideness angle and the angle containing the whole sparks jet. Industrial application –I2- is related to a 3d-laser scanner for railroad tracks profile analysis [9]. A processing systems extracts the track profile from the retrieved images as enlightened by a laser beam. A classification core is needed to identify, within each image column, the presence of a laser reflection.

The features extracted are minimum and maximum intensity of the column, their difference, the standard deviation and the maximum value of the convolution of the column intensity with a reference gaussian pattern and the energy of the derivate of the column intensity. Application –I3- is related to a quality analysis for the laser spot welding in the electronics manufacturing industry [10]. We identified that the relevant features to the classification problem (good/no-good weld) are the laser pulse energy, time of the first significant minimum of the back-reflected laser power signal, the time of turnpoint for the temperature sensor and the plume delay. After having applied the first three steps of the methodology we identify the topological structure and the P_Is of the sub-classifiers which, for the considered benchmarks, are given in table 2. We observe that, depending on the particular nature of the application, the number of sub-classifiers, their activation probability PI, the number of features f and the model nature can be significantly different. The table also contains, for sake of completeness, the complexity c of each sub-classifier and the sub-classifier type, information derived from step 4 of the methodology. No linear sub-classifiers have been selected here.

TABLE II

the Φ_{MC} s; f = feature number, c = complexity, type = classifier type (FF=feedforward, DEG=degenerated), PI = activation probability for sub-classifier I

	k	$\Phi_{MC,1}$		$\Phi_{MC,2}$		$\Phi_{MC,3}$	
I1	3	f: 0 c: 1	Type: DEG P1: 0.25	f: 5 c: 1158	Type: FF P2: 0.29	f:4 c: 1496	Type: FF P3: 0.46
I2	2	f: 7 c:752	Type: FF P1: 0.14	f: 5 c: 616	Type:FF P2: 0.86	-	-
I3	3	f: 3 c:1496	Type: FF P1: 0.37	f: 4 c: 1259	Type:FF P2: 0.17	f:4 c: 1022	Type: FF P3: 0.44

The $T_I(\tau)$ curves are given in figure 1 where each sub-figure contains the curves associated with all sub-classifiers composing a multiplexed classifier. We represented in a continuous line the reference curve $k^{-\tau}$ defined as the one whose inputs are independent and identically uniformly distributed. Under such hypotheses there is no time-dependency and temporal locality is absent. The more a $T_I(\tau)$ curve is above the reference curve the more the sub-classifier possesses local temporal property.

The considered industrial applications show good local temporality. For instance, module $\Phi_{MC,2}$ of application I2 shows a high activation probability from table 2 which means that the sub-classifier (P2=0.86) is more active than its companion (P1=0.14): in a Least Recently Used cache block substitution policy the module will stay in the cache with high probability (we also have to remind that sub-classifiers are in general smaller than the monolithic one). In addition, $\Phi_{MC,2}$ possesses an extremely good temporal locality profile from figure 1: the probability that $\Phi_{MC,2}$ will be active for all next 20 input patterns is above 0.8 with an obvious impact on cache miss reduction.

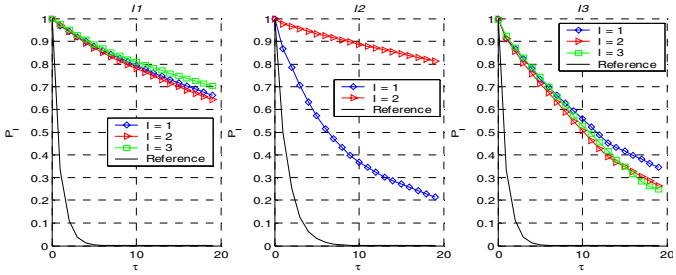

Figure 1. The $T_I(\tau)$ curves for the envisaged applications

TABLE III
Comparison of results in complexity and processing time

COMPLEXITY TABLE			Φ_{MC}		Φ_C	
Experi-ment	Complexity Gain $\frac{C(\Phi_C)}{C(\Phi_{MC})}$		Accuracy (% classif. error)	Complexity	Accuracy (% classif. error)	Complexity (type)
I1	357,04		5	1629	2	581624 (KNN)
I2	1,99		1	1573	1	3125 (FF)
I3	3,69		18	1690	20	6236 (FF)
PROCESSING TIME TABLE			Φ_{MC}		Φ_C	
Experi-ment	Execution Gain $\frac{T(\Phi_C)}{T(\Phi_{MC})}$		AMD athlon xp 1.7GHz 500MB ram	Intel P4 2 GHz 750BB ram	AMD athlonXp, 1.7GHz 500MB ram	Intel P4 2 GHz 750BB ram
	AMD	Intel	T µs	T µs	T µs	T µs
I1	348,0	356,8	255	353	88742	125952
I2	1,8	1,7	241	335	425	585
I3	3,5	3,5	273	373	956	1319

We can finally consider step 4 of the methodology, which requires the creation of the Φ_{MC} classifiers. The reference monolithic classifier Φ_C required in (3.1) has been selected among a set of monolithic classifiers with the same procedure delineated to create the Φ_{MC} ones. The initial population required by GAs was composed of 20 randomly initialized Φ_{MC}; the genetic algorithms procedure evolved for 100 generations. Results are shown in table 3.

Our experiments show that the methodology can produce multiplexed classifiers whose accuracy is comparable to the monolithic one but with a significant reduction in complexity. In some cases the complexity gain, which expresses how many times the monolithic classifier is more complex than the multiplexed one, is very high. Finally, we estimated the execution time per pattern presentation of both monolithic and multiplexed classifiers on two processors; the presence of not removable operating system processes was averaged over 100 presentations of the whole data set for each benchmark. We experienced that the gain in execution time is comparable, within fluctuations associated with uncontrollable OS processes (Windows 2000pro), with the estimated gain in complexity given in table 3 hence showing the effectiveness of the estimates (on which the genetic optimisation phase is based).

CONCLUSIONS

The paper presents an application level methodology for designing classification cores characterised by a reduced complexity and improved execution time by partitioning the classification space in sub-domains ruled by dedicated sub-classifiers. The multiplexed nature of the classifier design allows us to activate only a sub-classifier at a time while others are switched off with an immediate reduction in power consumption in applications experiencing temporal locality. The presence of the partitioning property, in addition to locality ones, reduces cache misses as well as computational complexity with also a gain in execution time.

REFERENCES

[1] C. M. Bishop, Neural Networks for Pattern Recognition. New York: Oxford, 1995.

[2] C.Alippi, L.Briozzo: Accuracy vs. Precision in Digital VLSI Architectures for Signal Processing, IEEE-TC, Vol 47. No.4 ,1998

[3] P.J. Edwards, A.F.Murray, Towards Optimally Distributed Computation, Neural Computation, Vol.10, 1998

[4] A.K. Jain, R.P.W. Duin, and Jianchang Mao, "Statistical pattern recognition: A review", IEEE-TPAMI, vol. 22, no. 1, 2000.

[5] A.K. Jain, R.C. Dubes, "Algorithms for Clustering Data". N. J. Englewood Cliff, Prentice Hall,1988.

[6] C. Alippi, P. Braione, V. Piuri, F. Scotti, "A methodological approach to multisensor classification for innovative laser material processing units" Proc. IEEE-IMTC, Budapest, Hungary, 2001

[7] D. Brooks, and al., "Wattch: A framework for architectural-level power analysis and optimizations", 27th ISCA, 2000.

[8] L.Davis, Handbook of GAs, Van Nostrand Reinhold, 1991

[9] C. Alippi, E. Casagrande, V. Piuri, F. Scotti, "Composite Real-Time Image Processing for Railways Track Profile Measurement", IEEE-TIM, vol. 49, 2000

[10] C.Alippi, T.Blom: Neural Networks for Measurement and Instrumentation in Laser Processing, NATO-Sciences Series: "Neural Networks for Instrumentation, Measurement and Related Industrial Applications", IOS Press, 2002