Fast FPGA System for Training Nonlinear Support Vector Machines *

Mudhar Bin Rabieah and Christos-Savvas Bouganis
Department of Electrical and Electronic Engineering
Imperial College London
mob10@imperial.ac.uk
christos-savvas.bouganis@imperial.ac.uk

Abstract

Support Vector Machines (SVMs) are powerful supervised learning methods in machine learning. However, their applicability to large problems has been limited due to the time consuming training stage whose computational cost scales quadratically with the number of examples. In this work, a complete FPGA-based system for nonlinear SVM training using ensemble learning is presented. The proposed framework builds on the FPGA architecture and utilizes a cascaded multi-precision training flow, exploits the heterogeneity within the training problem by tuning the number representation used, and supports ensemble training tuned to each internal memory structure so to address very large datasets. Its performance evaluation shows that the proposed system achieves more than an order of magnitude better results compared to state-of-the-art CPU and GPU-based implementations.

1 Introduction

Training Support Vector Machines on large datasets is a very challenging and time consuming process as the computational complexity for many solvers is $O(n^2)$, where n is the number of training points [1]. This hinders the applicability of such algorithms in situations where the characteristics of the data change over time. As a result, the need for constant retraining is desirable. Evidence of this can be seen in the google flu trend (GFT) where the need for model retraining was emphasized [2].

In this work, a complete FPGA-based system for accelerating nonlinear SVM training is presented. Ensemble learning is proposed to address large datasets and take advantage of available FPGA on-chip memory blocks. This approach allows each training subproblem to fully realize the parallelization potential. In addition to ensemble learning, cascaded multi-precision training flow is proposed by exploiting FPGA reconfigurability. This flow exploits the higher parallelization factor at the lower precision phase, which produces a reduced dataset for the higher precision phase to generate a more accurate model.

2 Background

Support Vector Machines (SVMs) are supervised learning methods that construct a hyperplane to classify data between two classes {-1,+1}. Given a set of examples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $y_i \in \{-1, +1\}$ and $x_i \in \mathbb{R}^d$. The hyperplane w is found by solving the following optimiza-

*This work was presented at the International Conference on Field-programmable Logic and Applications (FPL) 2015.

tion problem:

$$min\ \frac{1}{2}\|w\|^2 + C\sum_i \xi_i,\ \textbf{s.t.}\ y_i(\langle w, x_i\rangle - b) \geq 1 - \xi_i,\ \xi_i \geq 0 \tag{1}$$

where ξ are slack variables to account for misclassified data. C is a constant to control the trade-off between the error and the simplicity of the model w. The classification function is: $y = sign(\langle w, x\rangle - b)$.

Gilbert Algorithm: In [3], it was shown that Gilbert's algorithm [4] for solving the nearest point problem can be used to find the solution for SVM training (the slack variables ξ will be squared in optimization problem to easily transform the non-separable problem into a separable one). Gilbert's algorithm tries to find the point s^* on a given secant hull S which is the nearest to the origin. To describe the steps of Gilbert's algorithm, we need to define the following:

- The nearest point on the line segment connecting points a and b:

$$[a, b]^* = (1 - \lambda)a + \lambda b, \tag{2}$$

$$\lambda = \begin{cases} \frac{-\langle a, b-a\rangle}{\|b-a\|^2} & \text{if } 0 < -\langle a, b-a\rangle < \|b-a\|^2; \\ 0 & \text{if } -\langle a, b-a\rangle \leq 0 \\ 1 & \text{if } \|b-a\|^2 \leq -\langle a, b-a\rangle \end{cases} \tag{3}$$

- Contact function:

$$g_S^*(x) = s_{m0} \text{ where } \langle x, s_{m0}\rangle = max\langle x, s_m\rangle,\ s_m \in S \tag{4}$$

Now, at iteration k the solution is $w_k = [w_{k-1}, g_S^*(w_{k-1})]^* = (1-\lambda)\langle w_{k-1}, x\rangle + \lambda\langle g_S^*(w_{k-1}), x\rangle$. In [3], it was suggested that the angle should be used as a stopping criteria: $\frac{\langle w_k, w_{k-1}\rangle}{\|w_k\|\|w_{k-1}\|} \sim 1$.

In order to make Gilbert's algorithm applicable to SVM, the secant hull S is defined in terms of the two classes X and Y as $S = X - Y$. Now, $g_S^*(-w_{k-1})$ can be decomposed as:

$$g_S^*(-w_{k-1}) = g_X^*(-w_{k-1}) - g_Y^*(w_{k-1}) \tag{5}$$

Kernel functions can be used instead of dot product operations $\langle ., .\rangle$, which extends SVM to nonlinear cases.

Related Work: Parallel hardware structures (e.g. GPU, FPGA) have been targeted as a means for accelerating SVM training. In [5], the whole kernel matrix is calculated on the GPU and passed on to the CPU. In [6], the MapReduce framework was applied to the GPU. The advantage of using the MapReduce framework on a GPU instead of a cluster of computers is local synchronization between GPU processors. Another GPU implementation is GTSVM [7]. In this implementation, special attention was placed for sparse datasets (datasets with many zero features). Here, vectors with similar sparsity patterns, are grouped sequentially. This allows for a coalesced memory access.

Recently, FPGAs were also targeted as a means of acceleration. In [8, 9], kernel operations were done as fixed point operations. Both implementations demonstrated that for many datasets, the solutions accuracy was not affected. Both implementations are used as a co-processor. The FPGA handles kernel evaluations, whereas the rest of the learning algorithm is run on the CPU. In addition to using fixed point operations, the kernel function could be sped up by approximation. This approach was applied in [10]. Here, the exponential operation in the Gaussian kernel was replaced by the second-order Taylor series expansion. In [11], the kernel operation is divided between fixed point and floating point operations. The fixed point domain handles the dot product between the attributes of the data point. This is fed to the kernel processor to complete the evaluation of the function in the floating point domain.

In the previous implementations, the fixed structure of GPUs hinder their ability to exploit any heterogeneity within the training problem. As for the FPGA implementations mentioned above, some implementations as in [8, 9], only implement part of the training algorithm on the FPGA which reduces the parallelization potential. In [11], where a full implementation is proposed, no provision was made in the case when the dataset does not fit the block rams available. In addition, all the

previously mentioned FPGA implementations do not exploit the reconfigurability of such structures. These are the issues that our proposed framework addresses. It allows for custom precision per attribute. Also, for homogeneous datasets, a multi-stage multi-precision flow is proposed. Finally, ensemble learning is deployed to allow each training instance to fully fit the block rams.

3 Framework Overview

Fig. 1a shows an overall view of the framework architecture. Hardware module (FPGA) is responsible for running Gilbert algorithm to solve an SVM training instance. Software modules, which run on the CPU, are responsible for preprocessing the data (SVM Data), generating a customized hardware (SVM Configuration, Synth Tool), communication and setting up the training process (SVM Train) and running the classification process (SVM Classify). Problem description is a high level description of the training problem which includes kernel function used (RBF, polynomial ...) and number and types of dataset attributes (real-valued, categorical, boolean and integer).

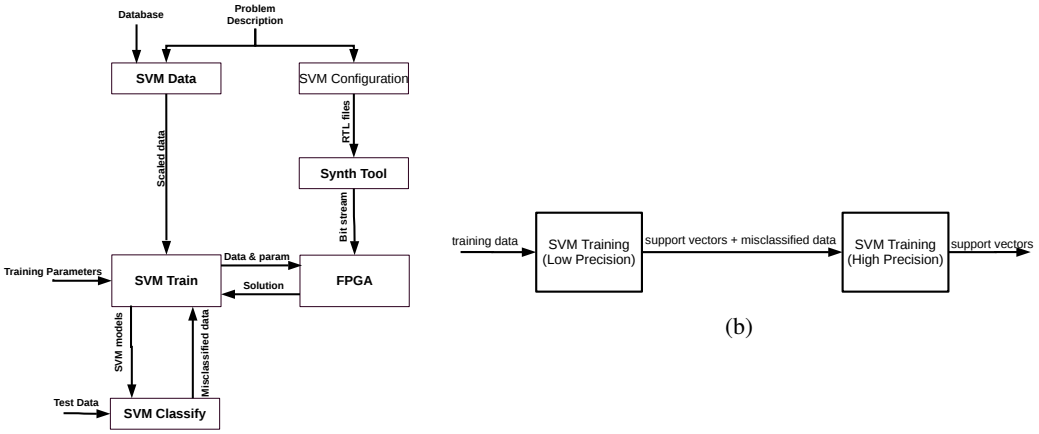


Figure 1: On the left (a): Framework Overview. (b): Low precision to high precision two stage training flow

SVM Ensemble: FPGAs provide low latency high throughput on-chip memory blocks which can be instantiated multiple times. To fully realize the parallelization potential of such memory blocks, an SVM ensemble is created by dividing the original data into several smaller groups, which fit the on-chip memory blocks. Each group is fed to the FPGA to be trained independently. Theses training models are aggregated (majority voting, weighted sum ...) at the classification phase.

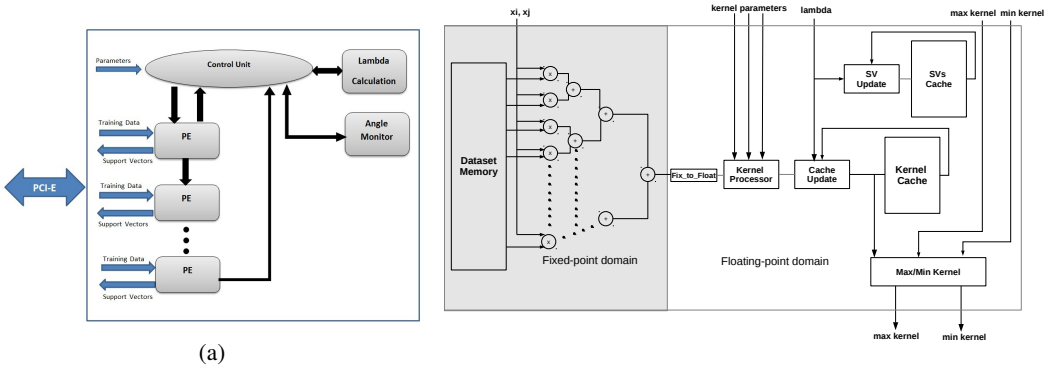
Cascaded Multi-precision Training A multi-precision training flow is proposed by exploiting the reconfigurability properties of FPGAs. At the lower precision phase, the training runs faster since more processing units can be instantiated. The support vectors generated together with misclassified data from this phase are passed to a higher precision phase. At the higher precision phase, less processing units are instantiated; however, this is mitigated by reducing the number of data points at this phase since not all the original training data is passed along. This scheme has the potential of speeding up the training process if the reduction of data points is significant. It also has the potential of reducing the final number of support vectors. This speeds up the classification since its execution time is a function of the number of support vectors.

4 Hardware Implementation

Top Level Architecture: The hardware module is responsible for executing Gilbert training algorithm. At the top level, several processing elements (PE), each with its own on-chip memory blocks, are stacked together. The processing elements evaluates the functions $g_X^*(-w_{k-1})$ and $g_Y^*(w_{k-1})$,

which are basically equivalent to finding $min\langle w_{k-1}, x_i\rangle$ and $max\langle w_{k-1}, x_j\rangle$, where $x_i \in X$ (class +1) and $x_j \in Y$ (class -1). Each processing element computes its local minimum and maximum and pass it to the next until the last processing element produces the global minimum and maximum.

In addition to the processing elements, the hardware module consists of "lambda calculation" and "angle monitor" blocks to manage the next iteration and monitor the stopping criteria. All this is managed through the main control unit. The (lambda calculation) block evaluates λ according to equation 3. The (angle monitor) block evaluates $\frac{\langle w_k, w_{k-1}\rangle}{\|w_k\|\|w_{k-1}\|}$ which is used as a stopping criterion. For both blocks, any dot product (kernel) operations are performed using the processing elements.



(a) (b)

Figure 2: On the left (a): Top level hardware architecture. (b): Processing element architecture

Processing Element architecture: Fig. 2b shows the architecture of the Processing Element. The kernel computations are divided between fixed and floating point domains. The dot product operations that appear in the kernel function are performed in fixed point. The rest of the kernel operations are done in floating point. To achieve maximum throughput, each attribute within the dataset is fed directly to a dedicated multiplier. Also, each attribute can have its own precision to fully utilize the FPGA resources. Each processing element updates the solution (SVs cache) corresponding to the training points it processes. Caching is used to speed up the overall algorithm (Kernel Cache). This is done by realizing that $\langle w_k, x\rangle = (1-\lambda)\langle w_{k-1}, x\rangle + \lambda\langle g_S^*(w_{k-1}), x\rangle = (1-\lambda)cache + \lambda\langle g_S^*(w_{k-1}), x\rangle$.

5 Evaluation Results

The FPGA system is implemented on Xilinx board ML605 (Virtex-6 XC6VLX240T) with an overall clock frequency 62.5Mhz (the core of the system can reach much higher frequencies but it was clocked down to to match the reference clock of the PCI port instantiated) The communication between the PC and FPGA board is carried through PCI port utilizing RIFFA framework [12].

The tests were performed on our system as well as SVM^{light} [13], GPUSVM [6] and GTSVM [7]. SVM^{light} was run on an Intel Core i7-3770 machine with 16GB RAM on board. Both GPU implementations were run on Nvidia Quadro K4000 GPU. Three datasets were tested, namely: adult (32K training points with 14 heterogeneous features), forest covertype (class 2 vs. all, 522K training points with 54 heterogeneous features) [14] and MNIST (odd vs. even, 60K training points with 784 homogeneous features) [15]. For the MNIST dataset, several additional tests were performed on the FPGA platform: full precision (8 bits per attribute), low precision (4 bits per attribute) and a hybrid two stage training scheme (Low precision + High precision). For all the datasets, the kernel used was the RBF kernel. For the MNIST and Covetype datasets, ensemble training is applied for the FPGA since the datasets do not fit the on-chip memory blocks.

Table 1, shows a summary of the training results (data preparation and setup times are not included for all implementations). The FPGA implementation shows significant speed ups compared to the other implementations across all datasets (especially when the number of points is large as the svm ensemble mode is applied here). The cascaded scheme achieves similar results to the full precision

Table 1: Summary of results

Data Set	Implementation	Test Error(%)	Training Time	speed up	SVs
Adult	SVM^{light}	14.8	80s	1x	18152
($C = 1, \gamma = 0.05$)	GPUSVM	17.2	5s	16x	18344
	GTSVM	15	4s	20x	19138
	FPGA	16.8	0.5s	160x	17845
Forest Covertype	SVM^{light}	13.9	43200s	1x	277102
($C = 10, \gamma = 0.125$)	GPUSVM	13.9	1850s	23.4x	277402
	GTSVM	29.9	1200s	36x	278564
	FPGA	14	15s	2880x	294490
MNIST	SVM^{light}	4.6	2062.75s	1x	43733
($C = 10, \gamma = 0.125$)	GPUSVM	5	425.7s	4.8x	43731
	GTSVM	4.7	70s	29.5x	43584
	FPGA (8 bits)	4.8	6s	343.8x	46671
	FPGA (4 bits)	5	3s	687.6x	47812
	FPGA (4 + 8 bits)	4.8	8s	257.8x	43882

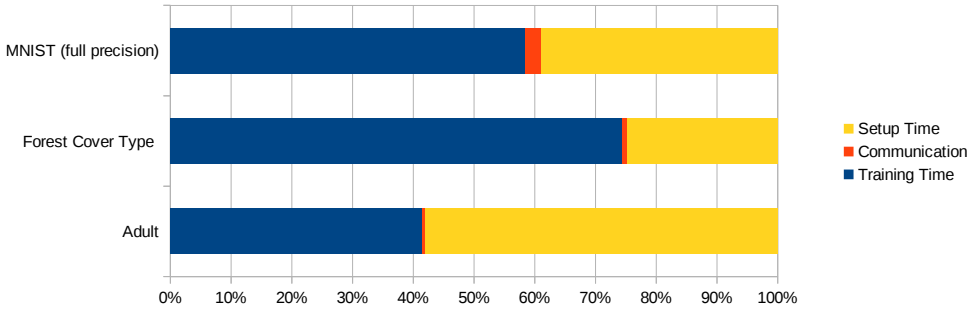


Figure 3: Setup and communication times overhead

case in terms of accuracy. However, it takes slightly more time in training. This is because the transition from the low precision to high precision phase did not result in a significant reduction in the number of points. However, the final tally of the number support vectors was reduced which is beneficial at the classification stage.

If setup time is included, the the overall end to end training time increases. This setup time includes the processing of data into fixed point values, configuring the training parameters (kernel parameters, regularization parameter ...), preparing data buffers and configuring communication channels between PC and the FPGA. However, the cost of setup time is only considerable at the start of the training process . Fig. 3 shows the data preparation time with respect to communication overhead between the FPGA and the PC (communication between SVM train module and FPGA) and the FPGA training time.It is clear that for small datasets (e.g Adult) the setup time becomes more considerable. As for the communication overhead, it is negligible except for the case of MNIST. This can be attributed to the fact that the dataset has many attributes which greatly reduce the number of points in each ensemble.

6 Conclusion

In this paper, a complete FPGA-based system for accelerating nonlinear SVM training has been presented. The proposed framework utilizes a cascaded multi-precision training flow, exploits the heterogeneity within the training problem, and supports ensemble learning. Performance evaluations shows that the proposed system outperforms other implementations across different datasets while still maintaining comparable accuracy.

References

[1] L. Bottou and C.-J. Lin, "Support vector machine solvers," *Large scale kernel machines*, pp. 301–320, 2007.

[2] D. M. Lazer, R. Kennedy, G. King, and A. Vespignani, "The parable of google flu: traps in big data analysis," 2014.

[3] S. Martin, "Training support vector machines using gilbert's algorithm," in *Data Mining, Fifth IEEE International Conference on*. IEEE, 2005, pp. 8–pp.

[4] E. G. Gilbert, "An iterative procedure for computing the minimum of a quadratic form on a convex set," *SIAM Journal on Control*, vol. 4, no. 1, pp. 61–80, 1966.

[5] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris, "Gpu acceleration for support vector machines," in *Procs. 12th Inter. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2011), Delft, Netherlands*, 2011.

[6] B. Catanzaro, N. Sundaram, and K. Keutzer, "A map reduce framework for programming graphics processors," in *Workshop on Software Tools for MultiCore Systems*, 2008.

[7] A. Cotter, N. Srebro, and J. Keshet, "A gpu-tailored approach for training kernelized svms," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 805–813.

[8] H. P. Graf, S. Cadambi, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and I. Dourdanovic, "A massively parallel digital learning processor," in *Advances in Neural Information Processing Systems*, 2008, pp. 529–536.

[9] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A massively parallel fpga-based coprocessor for support vector machines," in *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on*. IEEE, 2009, pp. 115–122.

[10] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, "Accelerating machine-learning algorithms on fpgas using pattern-based decomposition," *Journal of Signal Processing Systems*, vol. 62, no. 1, pp. 43–63, 2011.

[11] M. Papadonikolakis and C.-S. Bouganis, "A scalable fpga architecture for non-linear svm training," in *ICECE Technology, 2008. FPT 2008. International Conference on*. IEEE, 2008, pp. 337–340.

[12] M. Jacobsen, Y. Freund, and R. Kastner, "Riffa: A reusable integration framework for fpga accelerators," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. IEEE, 2012, pp. 216–219.

[13] T. Joachims, "Making large scale svm learning practical," 1999.

[14] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[15] Y. Lecun and C. Cortes, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/