FPGA Based Nonlinear Support Vector Machine Training Using an Ensemble Learning

Mudhar Bin Rabieah
Department of Electrical and Electronic Engineering
Imperial College London
Email: mob10@imperial.ac.uk

Christos-Savvas Bouganis
Department of Electrical and Electronic Engineering
Imperial College London
Email: 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.**

I. Introduction

Support Vector Machines (SVMs) are powerful supervised learning methods in machine learning [1], and they have been deployed to tackle diverse problems. For example, they have been used successfully in many applications, such as: object recognition, e-mail filtering, DNA sequencing and speech recognition [2].

Training 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 [3]. 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 [4].

To tackle the problem of high computational costs of SVM training, several algorithmic techniques have been proposed. For example, decomposition techniques transform the training problem into a sequence of sub-problems known as working sets [3]. At each iteration, only the coefficients of the working set data are updated. SVM^{light} [5] deploy such techniques. Another approach is exploiting the geometric properties of SVM training [6].

Hardware improvements can contribute towards training acceleration. For example, GPUs are used to accelerate computationally intensive parts such as the dot product operations [7], [8]. Reconfigurable structures (e.g. FPGAs) have also been proposed to accelerate SVM training [9], [10], [11].

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.

II. Background

A. Support Vector Machine

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 optimization 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: $y = sign(\langle w, x\rangle - b)$.

The above optimization problem can be reformulated in the dual form as:

$$min\ \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle \tag{2}$$

$$\textbf{s.t.}\ w = \sum_i \alpha_i y_i x_i,\ \sum_i \alpha_i y_i = 0,\ 0 \leq \alpha_i \leq C$$

where α_i is the Lagrange multiplier. Kernel functions can be used instead of dot product operations, which extends SVM to nonlinear cases.

B. Gilbert Algorithm

In [6], it was shown that Gilbert's algorithm [12] for solving the nearest point problem can be used to find the solution for SVM training. Gilbert's algorithm tries to find the point s^* on

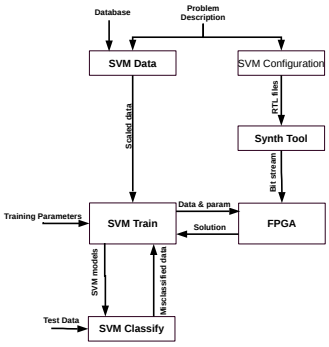

Fig. 1. Framework Overview

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:

$$[a,b]^* = (1-\lambda)a + \lambda b, \tag{3}$$

$$\lambda = \begin{cases} \frac{-\langle a, b-a\rangle}{\|b-a\|^2} & \text{if } 0 < -\langle a, b-a\rangle < \|b-a\|^2; \\ 0 & \text{if } -\langle a, b-a\rangle \leq 0 \\ 1 & \text{if } \|b-a\|^2 \leq -\langle a, b-a\rangle \end{cases} \tag{4}$$

- Contact function:

$$g_S^*(x) = s_{m0} \text{ where } \langle x, s_{m0}\rangle = max\langle x, s_m\rangle,\ s_m \in S \tag{5}$$

Now, Gilbert algorithm proceeds as follows:

1. choose random point $w_0 \in S$
2. $k = 0$
3. **Repeat**
   4. $k = k + 1$
   5. find $g_S^*(-w_{k-1})$ as described in equation 5
   6. $w_k = [w_{k-1}, g_S^*(w_{k-1})]^*$.
7. **Until** convergence

In [6], 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:

$$g_S^*(-w_{k-1}) = g_X^*(-w_{k-1}) - g_Y^*(w_{k-1}) \tag{6}$$

III. Framework Overview

The proposed framework consists of both software and hardware modules. Fig. 1 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).

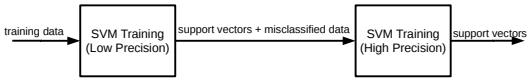

Fig. 2. Low precision to high precision two stage training flow

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). Such high level description masks the underlying hardware from the user which opens up the framework to non hardware designers and provides a consistent front end regardless of the FPGA platform used.

A. SVM Ensemble

FPGAs provide low latency high throughput on-chip memory blocks which can be instantiated multiple times. However, for many training problems, the memory requirements exceeds the on-chip memory available within the FPGA. A straightforward fix is to use external SDRAMs to store the data. However, the associated control logic as well as the access scheme limits the number of possible processing units instantiated which reduces the parallelization factor.

In the proposed framework, the solution is provided by creating an SVM ensemble of k problems each of the size N/k, where N is the total number of data points. This allows each training subproblem to fully realize the parallelization potential. Each group is fed to the FPGA and a model for each group is created independently. Theses models are aggregated (majority voting, weighted sum ...) at the classification phase.

B. Cascaded Multi-precision Training

A multi-precision training flow is proposed by exploiting the reconfigurability properties of FPGAs. In this scenario, a cascaded training flow is created as shown in fig. 2. 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 is guaranteed to produce a feasible solution since the solution produced at the lower precision phase satisfies the condition $\sum_i \alpha_i y_i = 0$ which is a feasible solution.

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.

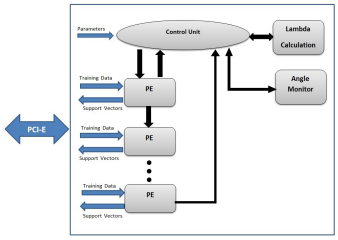

Fig. 3. Top Level SVM Architecture

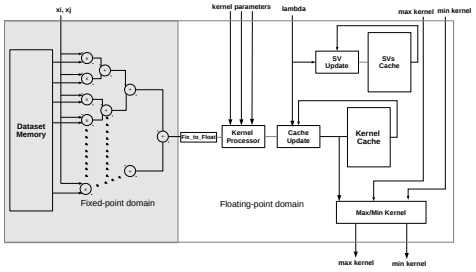

Fig. 4. Processing Element Architecture

IV. Hardware Implementation

A. Top Level Architecture

In the proposed framework, the hardware module is responsible for executing Gilbert training algorithm. The top level hardware architecture compromises of several processing elements (PE), each with its own on-chip memory blocks. Also, it 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 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). These operations translate to performing kernel operations between one single point and all the training data. The complexity of these functions is $O(n)$ where n is the number of data points. The instantiation of multiple processing elements allows the data points to be distributed between these elements. 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.

The (lambda calculation) block evaluates λ according to equation 4. Any dot product (kernel) operations are done utilizing the processing elements, which leaves the (lambda calculation) block with just some simple floating point scalar operations. The (angle monitor) block evaluates $\frac{\langle w_k, w_{k-1}\rangle}{\|w_k\|\|w_{k-1}\|}$ which is used as a stopping criteria. Again, any dot product (kernel) operations are performed using the processing elements.

B. Processing Element

Fig. 4 shows the architecture of the Processing Element. As mentioned earlier, processing elements performs kernel operations required for the evaluation of $min\langle w_{k-1}, x_i\rangle$ and $max\langle w_{k-1}, x_j\rangle$. The kernel computation is divided between fixed and floating point domains in a similar fashion to what was proposed in [9]. 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. This will allow for a heterogeneous dataset and fully utilize the FPGA resources. Unlike [9], each processing element updates the solution (SVs cache) corresponding to the training points it processes. This makes the solution update runs in parallel with evaluations of $min\langle w_{k-1}, x_i\rangle$ and $max\langle w_{k-1}, x_j\rangle$.

Also, caching is used to speed up the overall algorithm (Kernel Cache). At each iteration, the kernele function $k(w_k, x)$ is calculated to find the minimum and maximum. This is a very expensive computation since from equation (2): $w_k = \sum_i \alpha_i y_i x_i \Rightarrow k(w_k, x) = \sum_i \alpha_i y_i k(x_i, x)$. As a result, as the algorithm progresses, $k(w_k, x)$ becomes more challenging due to the increased number of support vectors. This can be solved by observing the recursive nature in which w_k is computed:

$$\begin{aligned} w_k &= [w_{k-1}, g_S^*(w_{k-1})]^* \\ &= (1-\lambda)w_{k-1} + \lambda g_S^*(w_{k-1}) \\ \Rightarrow k(w_k, x) &= (1-\lambda)k(w_{k-1}, x) + \lambda k(g_S^*(w_{k-1}), x) \\ &= (1-\lambda)cache + \lambda k(g_S^*(w_{k-1}), x) \end{aligned}$$

Now, only two kernel operations are required since $k(g_S^*(w_{k-1}), x) = k(g_X^*(-w_{k-1}), x) - k(g_Y^*(w_{k-1}), x)$

V. Evaluation Results

The FPGA system is implemented on Xilinx board ML605 (Virtex-6 XC6VLX240T). The maximum operating frequency of the core of the system is 150Mhz. However, the overall system is clocked down to 62.5Mhz 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 [13].

Three datasets were tested, namely: adult, forest covertype (class 2 vs. all) [14] and MNIST (odd vs. even) data sets [15]. The tests were performed on our system as well as SVM^{light} [5], GPUSVM [7] and GTSVM [8]. 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.

The adult dataset contains 32K training points with 14 heterogeneous features which fits completely in the FPGA. The forest covertype contains 522K training points with 54 heterogeneous features. The MNIST dataset contains 60K training points with 784 homogeneous features. Both covertype and MNIST do not fit the FPGA which require the use of SVM ensemble mode. The aggregation scheme for both is majority voting. For the MNIST dataset, several additional tests were performed on the FPGA platform: full precision

TABLE I
SUMMARY OF RESULTS

Data Set	Implementation	Test Error(%)	Training Time	speed up	SVs	Power
Adult	SVM^{light}	14.8	80s	1x	18152	77W
($C = 1, \gamma = 0.05$)	GPUSVM	17.2	5s	16x	18344	80W
	GTSVM	15	4s	20x	19138	80W
	FPGA	16.8	0.5s	160x	17845	6W
Forest Covertype	SVM^{light}	13.9	43200s	1x	277102	77W
($C = 10, \gamma = 0.125$)	GPUSVM	13.9	1850s	23.4x	277402	80W
	GTSVM	29.9	1200s	36x	278564	80W
	FPGA	14	15s	2880x	294490	6W
MNIST	SVM^{light}	4.6	2062.75s	1x	43733	77W
($C = 10, \gamma = 0.125$)	GPUSVM	5	425.7s	4.8x	43731	80W
	GTSVM	4.7	70s	29.5x	43584	80W
	FPGA (8 bits)	4.8	6s	343.8x	46671	5.7W
	FPGA (4 bits)	5	3s	687.6x	47812	6W
	FPGA (4 + 8 bits)	4.8	8s	257.8x	43882	5.85W

TABLE II
NUMBER OF PROCESSING ELEMENTS INSTANTIATED WITH RESPECT TO EACH DATASET

Data Set	Processing Elements Instantiated
Adult	20
Forest Covertype	20
MNIST (8 bits per attribute)	1
MNIST (4 bits per attribute)	3

(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.

Table I, shows a summary of the training results (data preparation and setup times are not included for all implementations). Test error is the percentage of misclassified data to the overall number of testing points. Power consumption of the FPGA implementation was estimated using XPower from Xilinx with a switching probability of 0.5. The power consumption of the other CPU as well as GPU implementations is reported from the data sheets of Intel Core i7-3770 and Nvidia Quadro K4000 GPU.

The FPGA implementation shows significant speed ups compared to the other implementations accross all datasets. As for the MNIST cases, it is obvious that the low precision case achieves the best training time due to the higher number of processing elements instantiated as shown in table II. The cascaded scheme achieves similar results to the full precision case in terms of accuracy. However, it takes slightly more time in training. This is due to the fact that 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.

VI. 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] V. N. Vapnik, "Statistical learning theory," 1998.

[2] L. Wang, *Support Vector Machines: theory and applications*. Springer Science & Business Media, 2005, vol. 177.

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

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

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

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

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

[8] 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.

[9] M. Papadonikolakis and C. Bouganis, "A heterogeneous fpga architecture for support vector machine training," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*. IEEE, 2010, pp. 211–214.

[10] 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.

[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] 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.

[13] 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.

[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/