Fast and Accurate Training of Ensemble Models with FPGA-based Switch

Jiuxi Meng, Ce Guo, Nadeen Gebara, Wayne Luk
Department of Computing, Imperial College London
Email: {jiuxi.meng16, c.guo, n.gebara17, w.luk}@imperial.ac.uk

***Abstract*—Random projection is gaining more attention in large scale machine learning. It has been proved to reduce the dimensionality of a set of data whilst approximately preserving the pairwise distance between points by multiplying the original dataset with a chosen matrix. However, projecting data to a lower dimension subspace typically reduces the training accuracy. In this paper, we propose a novel architecture that combines an FPGA-based switch with the ensemble learning method. This architecture enables reducing training time while maintaining high accuracy. Our initial result shows a speedup of 2.12-6.77 times using four different high dimensionality datasets.**

I. Introduction

Recent years have witnessed a significant surge in the size of data available for machine learning tasks. Though this is great for improving training accuracy, it is computationally challenging. Reducing the high dimensionality of the data becomes crucial for accelerating machine learning. Random Projection (RP) has been proved to be a powerful method for dimensionality reduction. However, increasing the sparsity of the RP matrix and thereby reducing the projected dimensionality inevitably results in loss of information within the dataset. This, in turn, impacts the training accuracy of machine learning applications.

An ensemble classification system comprises of multiple classifiers. These classifiers can be diverse in model type as well as predictive accuracy. The diversity of the classifiers allows them to correct and compensate each others' shortcomings, thereby achieving a collective accuracy that is higher than that of the individual classifier. Combining RP and ensemble systems allows classifiers to gain diversity from data in different sub-spaces, and the system to recover the accuracy loss with the ensemble.

Network switches as the pivot of data centre networks are possible places to introduce diversity. However, commodity network switches are fixed in functionality, so are not suitable for computational tasks. Previous work [1] shows the capability of FPGAs for implementing network switches. However, such design typically leaves computational resources such as Digital Signal Processing (DSP) blocks unused, which can be utilised to implement additional functions. We therefore propose an FPGA based network switch architecture to deploy RP ensemble method across multiple workers, with the switch section responsible for distributing training data and the DSPs for random projection.

The challenges of designing such systems are: **(C1)** Processing large dimensional data requires a large amount of FPGA resources, thereby limiting the scalability of the system. **(C2)** Performing fast computation requires storing the RP matrices on-chip, and this becomes challenging as the dimensionality of input data increases due to limited on-chip storage. Through model analysis, we tackle the aforementioned challenges, and provide a generalisable solution for various scenarios. The main contributions of this paper are:

1) A novel architecture for random projection ensemble method that utilises the spare resources of an FPGA-based switch.
2) Potential speedups in training whilst maintaining or even surpassing the single model accuracy.
3) A performance model that benefits future deployment of our system.

II. Background and related work

Fox et al. [2] propose the first FPGA based RP method to make training scalable on FPGAs. However, the limited resources left for RP in their design force them to use high sparsity RP matrices for hardware simplicity, which results in large error for the projected subspace. It also limits their design to low-dimensional datasets, whereas the benefits of RP are more significant for high-dimensional datasets. In contrast, our work devotes all the idle computational resources on an FPGA-based switch to performing RP, allowing us to explore RP using Li's method [3] without introducing additional error.

III. Proposed System Architecture

The system comprises two major parts, as shown in Fig. 1, the FPGA-based switch for data pre-processing and the worker nodes for training. The system starts with a network storage server, sending the training data to the FPGA through a high-speed interface. Upon entering the FPGA-based switch, the data within the packets are either sent directly to the output port (forwarding mode), or enter the DSPs where matrix multiplication is performed with the pre-stored RP matrices (processing mode) before being sent to the output port depending on the request field in the header. The worker nodes then use the received data to perform training. We make no assumptions on the type of worker nodes, and they could range from simple development boards to server clusters.

Our design can be deployed as an **extra layer** of switching between the ToR switch and server nodes, illustrated by the right half of Fig. 1. This approach allows a reduction of the number of workers connected to each FPGA and keeps the functionality of the traditional rack. As a result, more resources on the FPGA can be utilised to compute random projection.

2160-052X/20/$31.00 ©2020 IEEE
DOI 10.1109/ASAP49362.2020.00023

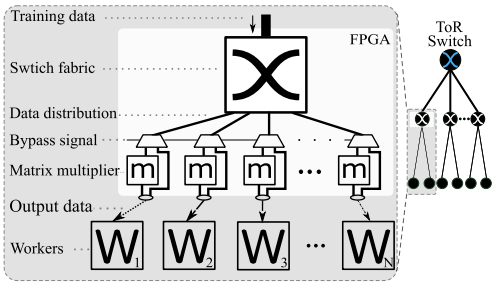

Fig. 1: System flow.

Replacing the ToR switch with FPGA-based switches, in contrast, would require the FPGA to scale with the number of servers connected to the ToR switch and this would limit the resources available for computation. Such a hierarchical approach allows us to address **C1** as the number of worker nodes increases.

On-chip storage avoids extra data transfer latency of the RP matrix from the off-chip memory and is necessary to perform fast data projection within the FPGA-based switch. However, the size of the projection matrix can become considerably larger than the capacity of the FPGA when the data dimension increases. Observing that the sparse matrix consists of only three types of values (positive, negative and zero) as shown in [3], we propose a compression method that represents the matrix value with 2 bits, and restores values only when used. This method enables storing the RP matrix on-chip with little logic resource overhead, thereby addressing challenge **C2**.

IV. Theoretical model

RP can be performed by the switch and/or by the worker nodes. Let α be the fraction RP computed by the switch, the rest $(1-\alpha)$ computed by the workers. The overall system time consists of (a) the RP time $T_{project}(\alpha)$ (b) release time of output ports $t_{release}(\alpha)$ and (c) the training time of the worker T_{train}. It is possible to implement the RP with systolic matrix multiplication to allow the overlapping of random projection and data release time. Therefore, we take the greater of $T_{project}(\alpha)$ and $T_{release}(\alpha)$. Ignoring the latency caused by the network and the pipeline, the total system time is as follows:

$$T(\alpha) = \max(T_{project}(\alpha), T_{release}(\alpha)) + T_{train} \tag{1}$$

$$T_{project}(\alpha) = \frac{\alpha X}{\theta^{switch}_{project}} + \frac{(1-\alpha)X}{\theta^{worker}_{project}} \tag{2}$$

where X is the number of data points for training a sub-model in the ensemble; $\theta^{D}_{project}$ is the throughput of random projection for device D measured by the number of data instances projected per second.

The release time to a worker is the total data traffic $F(\alpha)$ divided by the bandwidth b from the switch to the worker. The data traffic for each worker includes the original data and projected data sent to the worker, i.e.

$$F(\alpha) = \alpha X k + (1-\alpha) X m \tag{3}$$

where m and k are respectively the number of dimensions of the original data and the projected data. Assuming that all workers have the same bandwidth b measured by the number of data entries transferred per second, the release time is

$$T_{release}(\alpha) = \frac{F}{b} = \frac{\alpha X k + (1-\alpha) X m}{b} \tag{4}$$

We propose to handle all calculations for random projection in the FPGA-based switch with spare resources, which corresponds to $\alpha = 1$. In contrast, in a conventional system with an ordinary switch, the random projection works solely on the workers, which corresponds to $\alpha = 0$. The difference in performance is as follows:

$$T(1) - T(0) = X\big[\max(\frac{1}{\theta^{switch}_{project}}, \frac{k}{b}) - \max(\frac{1}{\theta^{worker}_{project}}, \frac{m}{b})\big] \tag{5}$$

The proposed system outperforms the conventional one if and only if $T(1) - T(0) < 0$. This inequality holds if and only if:

$$\max(\frac{1}{\theta^{switch}_{project}}, \frac{k}{b}) - \max(\frac{1}{\theta^{worker}_{project}}, \frac{m}{b}) < 0 \tag{6}$$

Note that inequality 6 is independent of the number of data points X or the time spent on sub-model training T_{train}.

Inequality 6 is very likely to hold under practical settings. In particular, there can only be four cases for the maximum operations.

1) $\max(\frac{1}{\theta^{switch}_{project}}, \frac{k}{b}) = \frac{k}{b}$ and $\max(\frac{1}{\theta^{worker}_{project}}, \frac{m}{b}) = \frac{m}{b}$. Since $k < m$ is the fundamental setting of random projection and $b > 0$, we have $\frac{k}{b} < \frac{m}{b}$. Therefore inequality 6 obviously holds.
2) $\max(\frac{1}{\theta^{switch}_{project}}, \frac{k}{b}) = \frac{k}{b}$ and $\max(\frac{1}{\theta^{worker}_{project}}, \frac{m}{b}) = \frac{1}{\theta^{worker}_{project}}$. In this case, $\frac{k}{b} < \frac{m}{b} \leq \theta^{worker}_{project}$. Therefore, similar to the previous case, inequality 6 still holds.
3) $\max(\frac{1}{\theta^{switch}_{project}}, \frac{k}{b}) = \frac{1}{\theta^{switch}_{project}}$ and $\max(\frac{1}{\theta^{worker}_{project}}, \frac{m}{b}) = \frac{1}{\theta^{worker}_{project}}$. In other words, the random projection time is the bottleneck of both systems. In this case, the inequity holds if $\theta^{switch}_{project} > \theta^{worker}_{project}$, which holds as long as DSPs on the FPGA have a speed advantage over the worker in terms of matrix multiplication.
4) $\max(\frac{1}{\theta^{switch}_{project}}, \frac{k}{b}) = \frac{1}{\theta^{switch}_{project}}$ and $\max(\frac{1}{\theta^{worker}_{project}}, \frac{m}{b}) = \frac{m}{b}$. In this case, whether inequality 6 holds is more data-dependent and hardware-dependent. However, we found that the case is very unlikely to appear practically.

V. Experiment setup and performance estimation

The experiments are primarily carried out in simulation with binary classification datasets from UCI repository [4]. A two-layer MLP classifier model from Keras [5] is used in our experiment. The simulation is open sourced[1]. Based on Eq. 7, training time is obtained from simulation and switch time is calculated based on our model. The simulation flow is illustrated in Fig. 2. To study the impact of dimension

[1]https://github.com/jiuxi-pub/Ensemble_learning

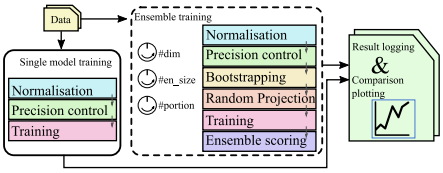

Fig. 2: Simulation flow.

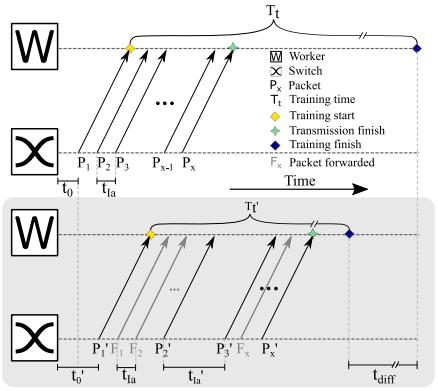

Fig. 3: Time graph. Top: Forward mode (without RP). Bottom: Processing mode (with RP).

reduction on the training time and accuracy, different number of workers ($\#en\_size$), size of dimensionalities ($\#dim$) and the portion size of bootstrapping ($\#portion$) are passed as input parameters to the simulator.

Fig. 3 illustrates the system time flow in more practically, with the top half showing the time for training without RP and the bottom half with RP. t_0 represents the time for the first element in the dataset to leave the switch. t_{Ia} is the time interval for each packet to arrive at the worker. T_t is the total training time taken by the worker. Without RP, training data are simply forwarded to the worker without processing, resulting in shorter t_0 and t_{Ia} in the top graph than the ones in the bottom. Packet forwarding during random projection is also shown in the bottom graph, indicated by the grey arrows in between the packets with RP. From the time graph, it is not hard to observe that a speedup from our design requires t_{diff} to be greater than zero.

To prove our point, four high dimensionality datasets are tested with their performance improvements shown in Tab. I. As a proof of concept, we assume a 4x4 10Gbps (256bit data width) switch is implemented on a Xilinx vu9p FPGA with 6840 DSPs, as presented in [1], [6]. A previous 4x4 network switch designed by the NetFPGA-SUME project [7] shows less than 15% of on-chip memory usage and less than 10% logic usage on a Virtex-7 chip. The same implementation on an Ultrascale+ chip will leave us with more flexibility for the RP function. We further assume a 16-bit fixed-point number for data representation as it only requires a single DSP block. Floating-point multiplication is avoided for minimal resource utilisation. The impact of fixed-point versus floating-point is modelled by a conversion precision loss in the simulation.

Overall system time is represented by the sum of time spent on the switch (T_{switch}) and the training time of workers (T_t)

TABLE I: Estimated result.

Name	Size used	Original dimension	Reduced dimension	Estimated Speedup
Ad [8]	3279	1558	200	3.28
epsilon [9]	55000	2000	1000	2.12
gisette [4]	6000	5000	1000	6.77
realsim [10]	5783	20958	3500	3.51

as shown in Eq. 7.

$$\begin{aligned} T_{single} &= T^F_{switch} + T^F_t, \\ T_{ensemble} &= T^P_{switch} + T^P_t \end{aligned} \tag{7}$$

where superscript F and P stand for forwarding mode and processing mode of the switch respectively. On the switch side, time is measured by the difference between the first entry and the last departure of packets. T^P_{switch} in terms of clock cycles is given by:

$$T^P_{switch} = \underbrace{t_{in} + t_{mult}}_{t_0} + (X_{sub} - 1) * t_{Ia} + t_{out} \tag{8}$$

where X_{sub} is the number of elements in the dataset for each sub-model; t_{in} is the time for the first element to enter the switch; t_{out} is the time for the last element to leave the switch, which is shorter than t_{in} due to the dimension reduction; t_{mult} is the time for multiplying a single element of the dataset with the RP matrix. t_{Ia} equals to t_{mult}, as shown in Fig. 5.

Packet forwarding time for single model training is shorter as no multiplication time is required. The switching latency is typically a few hundred nanoseconds and is a one time expense for continuous packet transfer, which becomes negligible compared with the total forwarding time. It can be modelled as: $T^F_{switch} = X_s * t_{out} = X_s * t_{in}$, where X_s denotes the number of elements in the single model training dataset.

The speedup S of our proposed system compared with a single model can then be calculated by:

$$S = \frac{T_{single}}{T_{ensemble}} = \frac{T^F_{switch} + T^F_t}{T^P_{switch} + T^P_t} \tag{9}$$

VI. Result and Discussion

In general, our software simulation result shows an increase in training accuracy and a decrease in training time with the ensemble method; the reduced dimension is typically less than half of the original. We choose to have 15 members for the ensemble method, as they achieve stable performance for most tested datasets. Fig. 4 shows the training time against accuracy for different binary classification datasets. The original data are represented with hollow markers, whereas experimental data are represented with solid filled markers. Take the Ad dataset for example; our method uses only 12.83% of the original dimension and 30.38% of the original training time to achieve the same accuracy of single model training. Increasing the dimensionality to 50% extends the training time but results in even higher accuracy than single model training. For the epsilon data set, the proposed method experiences small variance in terms of accuracy, but with only less than 0.5% lower in accuracy for some dimensions. This can be explained as the result of an insufficient ensemble member size. The gisette dataset gives the highest time reduction among all the

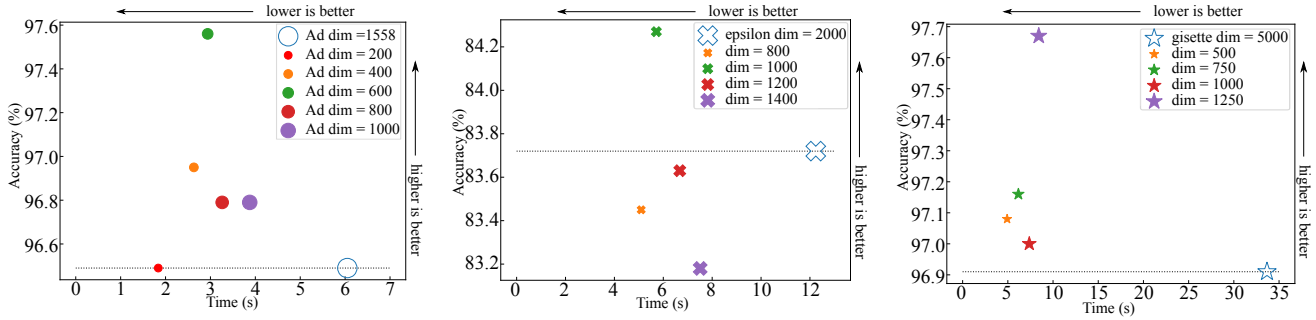

Fig. 4: Experimental results showing the impact of the proposed system on training time and accuracy.

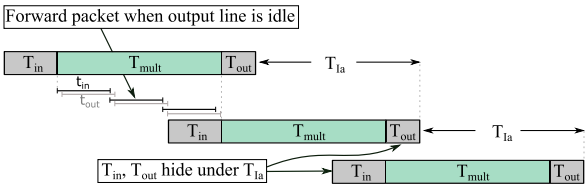

Fig. 5: Time variables in the system.

tests. Note that the training time of lower dimension data is given by the slowest member.

A case study will be used to illustrate Eq. 9. The Ad dataset under testing has a total of 3279 elements, in which 80% ($X_s = 2623$) is used for single model training. A portion of 80% from X_s is chosen as sub-training data ($X_{sub} = 2098$) for this case study. Assume the switch is running with a 200MHz clock rate, and let C denotes its period which is 5ns. We have $t_{in} = 1558 * 16/256 \approx 98C$. This latency only needs to be considered once as mentioned in Fig. 5. With limited DSP resources on the FPGA, matrix multiplication is broken down into column-wise vector multiplication. For a system with each FPGA-based switch connecting to two workers, to simplify the calculation, we allocate 1558*2 DSPs to each matrix multiplier module. We have multiplication time $t_{mult} = t_{Ia} = 1558kC/(1558 * 2) = 1/2kC$. The remaining elements take $t_{mult} \times (X_{sub} - 1) = 209700C$. After dimension reduction, a vector of 200 dimensions takes $T_{out} = 200 \times 16/256 \approx 13C$ cycles. In total, $T^P_{switch} = (98 + 209700 + 13)C = 1.049e-3s$. Finally, given that the training time for the reduced dimension dataset is 1.838s measured by the worker, we have $T_{ensemble} = 0.001049 + 1.838 = 1.8390s$

With our FPGA-based switch serving as a simple forwarding switch, the time spent on the switch is given by the total number of cycles that the dataset requires to be transferred. Therefore, $T^F_{switch} = X_s * t_{in} = 2623 * 98C = 0.00128s$. Together with training time of 6.049s obtained from the worker, $T_{single} = 0.00128 + 6.049 = 6.0502s$. Using Eq. 9, we obtain the speedup as $S \approx 3.288$. In addition, we determine the optimal value of α for the various models considered and summarise the results in Tab. II.

TABLE II: RP speedup result with selected α value.

	RP time (s)		α_{opt}
Name	$\alpha = 0$	$\alpha = 1$	
Ad	0.1145	0.0021	0.980
epsilon	0.1867	0.1760	0.515
gisette	0.3686	0.0320	0.920
realsim	0.7012	0.7756	0.475

VII. Conclusion

Compared with single model training, our system with ensemble learning achieves a higher training accuracy with reduced training time. It can be extended to cover multiple FPGAs for processing datasets with large dimensionality. Further work will explore how our design can benefit other computations, and how such computations can be implemented automatically and efficiently on the switch.

Acknowledgment. The support of of UK EPSRC (grant number EP/L016796/1, EP/I012036/1, EP/L00058X/1, EP/N031768/1 and EP/K034448/1), Microsoft and Xilinx is gratefully acknowledged.

References

[1] J. Meng, N. Gebara, H.-C. Ng, P. Costa, and W. Luk, "Investigating the feasibility of FPGA-based network switches," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 218–226.

[2] S. Fox, S. Tridgell, C. Jin, and P. H. Leong, "Random projections for scaling machine learning on FPGAs," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 85–92.

[3] P. Li, T. J. Hastie, and K. W. Church, "Very sparse random projections," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 287–296.

[4] D. Dua and C. Graff, "Uci machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[5] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[6] P. Papaphilippou, J. Meng, and W. Luk, "High-performance FPGA network switch architecture," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 76–85.

[7] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 gbps as research commodity," *IEEE micro*, vol. 34, no. 5, pp. 32–41, 2014.

[8] N. Kushmerick, "Learning to remove internet advertisements," in *Agents*, 1999, p. 175.

[9] "LIBSVM data: Classification (binary class)," https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#epsilon.

[10] "LIBSVM data: Classification (binary class)," https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#real-sim.