EILE: Efficient Incremental Learning on the Edge

Xi Chen*, Chang Gao*, Tobi Delbruck, Shih-Chii Liu
Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland
{xi, chang, tobi, shih}@ini.uzh.ch

***Abstract*—This paper proposes a fully-connected network training architecture called EILE targeting incremental learning on edge. By using a novel reconfigurable processing element (PE) architecture, EILE avoids explicit transposition of weight matrices required for backpropagation to preserve the same efficient memory access pattern for both the forward (FP) and backward propagation (BP) phases. Experimental results on a Zynq XC7Z100 FPGA with 64 PEs show that EILE achieves 19.2 GOp/s peak throughput and maintains nearly 100% PE utilization efficiency for both FP and BP with batch sizes from 1 to 32. EILE's small on-chip memory footprint and scalability to match any available off-chip memory bandwidth makes it an attractive ASIC architecture for energy-constrained training.**

***Index Terms*—deep neural network, hardware accelerator, on-chip training, incremental learning, edge computing, FPGA**

I. Introduction

Pretrained neural networks suffer from catastrophic forgetting when trained for new classes or tasks [1]. Incremental learning (**IL**) algorithms allow a pretrained network to be trained online for new classes with slow forgetting of the old classes. **IL** is useful for edge devices, e.g mobile phones, in applications such as face adaptation and speech verification of new users. Because edge platforms have tight arithmetic and memory constraints, learning on large batch sizes is not possible. Additionally, [2] showed that using small batch sizes for training results in better generalization, more robust convergence, and needs less epochs.

Edge devices can only store a small number of collected data samples from the user, therefore, **IL** is best done with very small batch sizes (even down to 1) and reduced precision [3] to minimize memory footprint of the training process; however, small batch sizes reduce opportunities of reusing weights; thus, making it hard to achieve high utilization of arithmetic units. An edge IL accelerator must be carefully designed to ensure efficient off-chip memory access and high arithmetic unit utilization even with small batch sizes.

Full error backpropagation remains the most accurate training method. Neural network training with backpropagation requires a forward propagation (**FP**) phase to evaluate the loss function and a backward propagation (**BP**) phase to compute the gradient loss with respect to network parameters. The BP phase requires more arithmetic operations than the FP phase and has higher memory cost because of the need to store inter-layer activations and to fetch transposed weights from off-chip DRAM memory needed when training large networks. DRAM access is around 10X-100X more expensive than arithmetic operations [4] and fetching transposed matrices reduces DRAM energy efficiency and speed because it cannot exploit DRAM burst mode access.

Previous IL accelerators [5], [6] do not report the efficiency for training with small batches. [5] showed an ASIC IL design with 2 TOp/s/W power efficiency but the reported number is achieved by buffering all parameters of a small network on the on-chip SRAM (64 kB). [6] proposed a method to selectively update network parameters to reduce off-chip memory access; however, no throughput or power efficiency numbers were reported. Other implementations that support on-chip training on the edge include an FPGA design [7] that uses a special memory management unit to alleviate the impact of irregular memory accesses but the performance during BP is still below that of the FP. A recent study [8] exploits a recursive algorithm for training binary neural networks; however, the processing element (**PE**) utilization efficiency of FP and BP phases were not reported. Other implementations [9], [10], [11] use custom PE architectures to support on-chip training of different DNN architectures, but their performance either decreases during BP [9] or decreases with smaller batch sizes [10], [11].

This work proposes an IL hardware accelerator called **EILE**. It uses a novel reconfigurable PE array architecture that efficiently deals with the **transposed matrix problem** of asymmetric DRAM memory access in computations of the FP and BP phases without an explicit transpose of the weight matrix (Sec. III-B). We demonstrate an FPGA implementation of EILE that emulates its performance with a commonly used DRAM memory interface PC4-19200 (Sec. III-D). We show that it enables consistently highest throughput in both FP and BP and nearly full utilization of PEs for small batch sizes down to 1. The architecture is useful in practical edge IL applications, where a front-end network (e.g. CNN) computes pre-trained features and the last fully-connected layers are re-trained for incremental learning classification.

II. Background

Training using backpropagation involves three stages: **FP**, **BP**, and parameter update (**PU**). These stages are defined by the following equations:

FP: $$Z_l = W_l A_{l-1} + B_l \tag{1}$$

BP: $$\delta_{Z_l} = \delta_{A_{l-1}} \odot f_l'(Z_l) \tag{2}$$

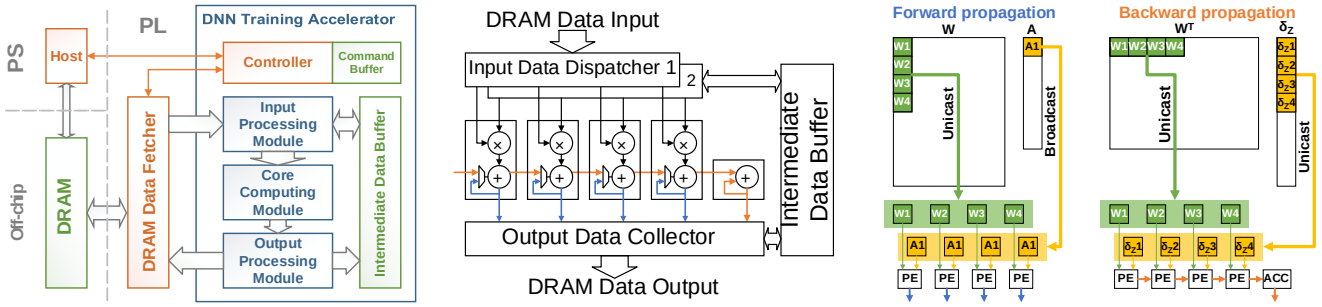
$$\delta_{A_{l-1}} = W_l^\intercal \delta_{Z_l} \tag{3}$$

$$\delta_{W_l} = \delta_{Z_l} A_{l-1}^\intercal \tag{4}$$

PU: $$W_l \leftarrow W_l - \eta \times \delta_{W_l} \tag{5}$$

This work was partially supported by the Swiss National Science Foundation, HEAR-EAR, 200021_172553 and the Samsung Institute of Advanced Technology.

*Xi Chen and Chang Gao are co-first authors.



(a) Top-level architecture. (b) PE architecture. (c) PE Parallel Mode (d) PE Cascade Mode

Fig. 1: EILE architecture and its PE modes.

Z_l, A_l, W_l, B_l are the pre-activation, activation, weight and bias of layer l respectively. The operator $\odot$ denotes the Hadamard product (element-wise product). f_l' is the derivative of the activation function at layer l. The symbol δ denotes the partial derivative of the loss $\mathcal{L}$ with respect to a variable, e.g., $\delta_{A_l} = \frac{\partial \mathcal{L}}{\partial A_l}$, and η is the learning rate.

It can be seen from these equations that DNN training mostly consists of matrix-matrix multiplication (MxM) or matrix-vector multiplication (MxV) operations when the batch size is 1. These operations are usually accelerated by distributing multiply-and-accumulate (**MAC**) operations across a number of PEs that compute in parallel. When the network parameters are stored in DRAM, the matrix transposition needed in Eqs. (3) and (4) creates the "transposed matrix problem". The physical architecture of DRAM means that memory fetches from DRAM are best carried out in bursts, along rows of the DRAM chip. Accessing DRAM data in a transposed manner requires a strided single data reading pattern, resulting in around 10-100X lower throughput and longer intervals between fetches. EILE proposes a novel array architecture which avoids this problem as described next.

III. Hardware Architecture

A. System Overview

The EILE architecture shown in Fig. 1a consists of three main modules: Input Processing Module (**IPM**), Core Computing Module (**CCM**), and Output Processing Module (**OPM**). A dedicated controller block interfaces with the host device and controls the overall training process. Part of the data is stored in the Intermediate Data Buffer (**IDB**, composed of on-chip BRAMs) to reduce DRAM fetches.

To solve the transposed matrix problem, EILE supports two modes of parallelism, *Parallel Mode* (**PM**) for FP and *Cascade Mode* (**CM**) for BP as shown in Fig. 1. We take a batch size of one as an example. In PM, a column section of weights is unicast (1→1) and an input activation element is broadcast (1→N) to PEs, so that the MxV results are obtained by accumulating weighted sum of weight columns sequentially; while in CM, a row section of transposed weights and a section of input errors are unicast to PEs, so that the MxV results are obtained by accumulating dot products sequentially. When executing CM in BP, the weight matrix is kept in place in DRAM without change of footprint and fetched in the same efficient order as in FP. The transposition is thus virtual, by changing the PE feed. Our method maintains nearly full utilization of PEs in both FP and BP stages.

B. PE Architecture

Fig. 1b shows the PE architecture that supports both PM and CM for computing different steps during training. Each PE has a multiplexer to select the source for one of the inputs of the adder in its MAC unit. During runtime, the PEs can operate in either PM (Fig. 1c) or CM (Fig. 1d).

1) PE Parallel Mode: The PE's multiplexer accepts an input from the PE's own buffer so that partial sums are accumulated within each PE in parallel.

2) PE Cascade Mode: Each PE's multiplexer accepts input from the preceding PE's adder, so that all PEs together form a pipelined adder chain along which the dot product is accumulated towards the final accumulator. This mode is exclusively used for calculating Eq. 3.

The ReLU activation and its derivative are implemented with multiplexers.

C. IPM and OPM

To support PM, CM and the different dataflow paths required by the training process, both IPM and OPM have a two-mode configuration as listed below:

1) *IPM Unicast Mode*: Different elements in a vector are sent to different PEs in parallel.
2) *IPM Broadcast Mode*: One element in a vector is sent to all PEs in parallel.
3) *OPM Parallel Mode*: Outputs from all PEs are stored in parallel to a set of consecutive addresses in the memory.
4) *OPM Reduction Mode*: The summation result from the final accumulator is stored to one address in the memory.

The combination of these two modules and PE's two modes enables the system to implement all the training equations presented in Sec. II.

D. Hardware Implementation

To validate the functionality of the EILE architecture, we implemented an accelerator in the programmable logic (**PL**) on a Zynq XC7Z100 system-on-chip (**SoC**). The ARM processing system (**PS**) core within the SoC is used both as the host device and for computing the training loss. The Xilinx IP

TABLE I: FPGA resource utilization

Resource		Utilization	Available	Percentage
LUT		40,492	277,400	15%
FF		46,690	554,800	8%
BRAM	Accelerator	167.5	755	22%
	Parameters*	512		68%
DSP		64	2020	3%

* Saved if parameters are stored onto DRAM.

'AXI Datamover' is used for data transfer between DRAM and the accelerator. For weight data fetches, to match a widely used DDR4 DRAM interface PC4-19200 having bandwidth B_{DRAM} of 19.2 GB/s, we use $P = 64$ PEs running at $f_{\mathrm{clk}} = 150$ MHz, according to $B_{\mathrm{DRAM}} = P \times W \times f_{clk}$ where $W = 2$ bytes stands for the word width. This choice also matches the maximum data width of 1024 bits (64 words) provided by AXI Datamover and AXI BRAM Controller IPs, so that each PE can be fed with data on each clock cycle. As a workaround to overcome the 4 GB/s DRAM bandwidth bottleneck of the SoC, we used 512 BRAM blocks (2 MB) to store network parameters. This way, EILE accesses the BRAM with the same pattern and data rate as it would for the target DDR4 DRAM.

Table I lists the FPGA resources. The accelerator itself uses 167.5 BRAM blocks ($\approx$0.65 MB, 22% of the PL). Using our 16-bit datatype and DSP unit's built-in multiplexer, each PE is mapped to one DSP unit in the PL. Each PE has a local accumulation buffer of 2 KB.

IV. Experimental Results

Many **IL** tasks require retraining of fully-connected classification layers [12], which usually have the most parameters in deep convolutional neural networks. We evaluate the performance of EILE by training a fully-connected network with 2 hidden layers (network size: 784-512-256-10, total 1 MB of parameters) on the full MNIST [13] handwritten digit dataset. Activations are quantized to fixed-point *Q(8, 8)* format while weights and gradients are quantized to *Q(2, 14)* format for batch size of 1, where *Q(m, n)* denotes the quantization using m bits for the integer part and n bits for the fraction.

A. Throughput and Power

Table II reports the training performance of EILE for different batch sizes. The peak numerical performance is 19.2 GOp/s for P=64 PEs and f_{clk}=150 MHz. From the 'Throughput' row, it is clear that EILE maintains nearly peak performance for FP and BP. The flexible PE architecture allows burst mode reads for both directions. Without it, the single stride reads needed for one pass would reduce the throughput of this pass by an estimated 8-50X if weights were stored on DRAM. The 'PE utilization' (i.e. fraction of clock cycles where PEs are active) is above 95% during both passes for all batch sizes.

For larger batch sizes, the parameters are updated less frequently, resulting in fewer memory fetches and thus less power consumption. The overall throughput is lower than either FP or BP throughput alone because the gradient calculation and parameter update are carried out in separate stages and thus the PU has around 50% efficiency. This problem can be resolved by combining both stages and consuming gradients on-the-fly.

TABLE II: Performance of EILE

Batch size		1	4	8	16	32
Power (W)	Baseboard*	7.30				
	Wall plug*	14.00	13.20	12.90	12.80	12.70
	Effective†	6.10	5.30	4.90	4.80	4.70
Throughput (GOp/s)	FP	18.90	19.09	19.14	19.15	19.16
	BP	18.39	19.00	19.09	19.14	19.16
	Total§	13.57	15.75	16.37	16.72	17.27
PE utilization‡	FP	**98.4%**	**99.4%**	**99.7%**	**99.8%**	**99.8%**
	BP	**95.8%**	**99.0%**	**99.4%**	**99.7%**	**99.8%**
	Total§	70.7%	82.1%	85.2%	87.1%	90.0%
Power efficiency (GOp/s/W)	FP	3.10	3.60	3.91	3.99	4.08
	BP	3.01	3.58	3.90	3.99	4.08
	Total§	2.22	2.97	3.34	3.48	3.67

* Power consumption is measured by a wall plug power meter.
† "Effective power" = Wall plug power - baseboard and fan power.
‡ "PE utilization" = measured average throughput / peak throughput.
§ "Total" term includes FP, BP, PU, the training loss calculation in the ARM processor and everything else including communication overhead.

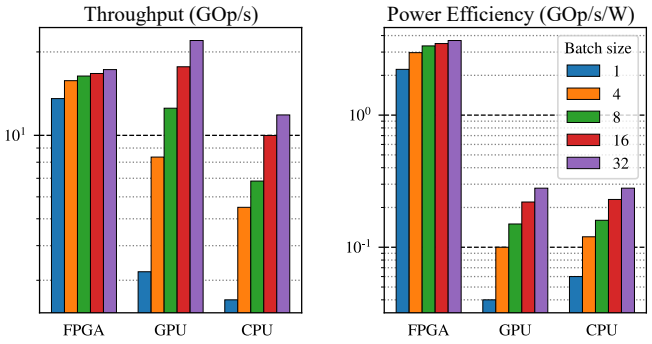


Fig. 2: Experimental results from EILE training.

The Xilinx Power Analyzer estimates the EILE SoC FPGA on-chip power consumption at 2.5 W for a signal switching rate of 50%. Static power is 0.27 W and dynamic power is 2.2 W. Most power (1.5 W) is consumed by the processing system (**PS**). The core power (training accelerator + BRAMs) is 410 mW, so the core power efficiency is about 35 GOp/s/W.

B. Performance Comparison

Fig. 2 compares EILE against training using PyTorch on a GPU (GeForce GTX 980 Ti) and a CPU (Intel Core i7-4770K CPU @ 3.50 GHz × 8). The GPU uses 16-bit floating-point (FP16), while the CPU uses FP32 since FP16 is not well supported for CPU on PyTorch. The runtime power of the GPU and CPU are 81 W and 44 W. Fig. 2 shows that EILE achieves more consistent throughput numbers through all batch sizes in comparison to the CPU and GPU where the highest throughput are achieved only at large batch sizes because of higher parallelism. In addition, EILE has 13-61X higher power efficiency due to its low power consumption.

V. Comparison with other work

Prior work (Table III) mainly reports peak throughput, which is usually not achievable because PE utilization is lower than 100%. Compared to prior work, EILE achieved the largest PE utilization in both FP and BP phases with batch sizes from

TABLE III: Comparison with other works

Work		DeepTrain [9]		[10]	[11]	GANPU [14]		Evolver [15]	**EILE**	
Network		CNN/RNN/FC		CNN/LSTM/FC	CNN/RNN/FC	GAN		CNN/FC	FC	
Process		ASIC 15nm		ASIC 14nm	ASIC 65nm	ASIC 65nm		ASIC 28nm	FPGA	ASIC 28nm‡
Area (mm²)		1.17		9	16	32.4		5.64	-	1.48
SRAM (KB)		976		2048	448	676		416	670	148
#MAC		480		500	1024	1344		256	64	256
Frequency (MHz)		2500		1500	200	200		268	150	500
Power (W)		2.64		-	0.196	0.647		0.036	0.41	0.49
Data type		FXP16	FXP32	FP16/32	FP16, FXP4/8/16	FP8	FP16	FXP2/4/8	FXP16	
Peak throughput* (GOp/s or GFLOp/s)	FP	4800	-	1500	204.8	1075	538	137	19.2	256
	BP	-	2400							
PE utilization	FP	88%-98%	-	92%-98%	-	-	-	89.7%	**98%-100%**	**99.6%**
	BP	-	50%-96%	80%-97%				85.1%	**96%-100%**	**97.5%**
Normalized throughput† (GOp/s or GFLOp/s)	FP	16.9-18.8	-	17.7-18.8	-	-	-	17.2	18.8-19.2	19.1
	BP	-	9.6-18.4	15.4-18.6				16.3	18.4-19.2	18.7

* Without exploiting sparsity.
† Given by $2 \times \#\mathrm{MAC} \times f_{\mathrm{clk}} \times \mathrm{PE\_Utilization}$, normalized to $\#\mathrm{MAC}$ = 64 and f_{clk}=150 MHz.
‡ Pre-layout synthesis and simulation results, batch-1 only.

1 to 32. Thus, once normalized to the same number of PEs and operating frequency, EILE achieves the largest normalized throughput in both FP and BP phase and the number is consistent across FP and BP phases. DeepTrain [9] was not designed for edge applications and has smaller throughput in the BP phase compared to the FP phase. To support different dataflows in FP and BP, [11] proposed a transposable PE array to exploit parallelism on multiple samples in a batch, thus PE utilization decreases with smaller batch sizes. [14] did not show PE utilization number and its normalized throughput cannot be calculated. [15] proposed a 2D PE array that exploits activation sparsity in FP and BP and reports lower FP/BP utilization but batch size is not mentioned. By contrast with prior work that used a 2D PE array, EILE proposes an optimized low-cost 1D PE array that achieves ∼100% PE utilization with batch sizes up to 32.

VI. Conclusion

This paper describes an edge **IL** accelerator implemented on FPGA. By introducing reconfigurable dataflow with uniform memory access patterns, EILE has ∼100% PE utilization and therefore consistent throughput in both FP and BP phases with batch sizes from 32 down to 1. Even in FPGA and not ASIC form, the power efficiency of EILE is higher than a desktop GPU and CPU by 1-2 orders of magnitude. The EILE architecture is scalable and particularly suitable for on-chip training on power- and memory-constrained edge devices. An ASIC implementation (where a floating point PE would be practical) of a 64-PE EILE would require less than 700 kB of SRAM and would be matched to the DRAM bandwidth $19.2\,\mathrm{GB/s}$ (Sec. III-D) provided by a standard PC4-19200 DDR4 interface. EILE's architecture allows for future optimizations such as weight pruning and zero-skipping technique for sparse data flow, which can boost performance and power efficiency by reducing data transfer and computation. EILE can be scaled up by simply grouping 1D arrays to form a 2D array, where it can also process larger batch sizes with 100% utilization. Future work includes extension to training of RNNs useful for speech recognition applications.

References

[1] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165.

[2] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *CoRR*, vol. abs/1804.07612, 2018. [Online]. Available: http://arxiv.org/abs/1804.07612

[3] Y. Hu, T. Delbruck, and S. Liu, "Incremental learning meets reduced precision networks," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[4] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[5] C. Chen, H. Ding, H. Peng, H. Zhu, R. Ma, P. Zhang, X. Yan, Y. Wang, M. Wang, H. Min, and R. C. . Shi, "OCEAN: An on-chip incremental-learning enhanced processor with gated recurrent neural network accelerators," in *ESSCIRC 2017*, 2017, pp. 259–262.

[6] J. Shin, S. Choi, Y. Choi, and L. S. Kim, "A pragmatic approach to on-device incremental learning system with selective weight updates," in *57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[7] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, "FPGA acceleration of recurrent neural network based language model," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 111–118.

[8] T. Guan, P. Liu, X. Zeng, M. Kim, and M. Seok, "Recursive binary neural network training model for efficient usage of on-chip memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2019.

[9] D. Kim, T. Na, S. Yalamanchili, and S. Mukhopadhyay, "DeepTrain: A programmable embedded platform for training deep neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2360–2370, Nov 2018.

[10] B. Fleischer, S. Shukla, M. Ziegler, J. Silberman, J. Oh, V. Srinivasan, J. Choi, S. Mueller, A. Agrawal, T. Babinsky *et al.*, "A scalable multi-TeraOPS deep learning processor core for AI training and inference," in *2018 IEEE Symposium on VLSI Circuits*, 2018, pp. 35–36.

[11] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H. Yoo, "A 2.1 TFlops/W mobile deep RL accelerator with transposable PE array and experience compression," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, Feb 2019, pp. 136–138.

[12] F. M. Castro, M. J. Marín-Jiménez, N. Guil, C. Schmid, and K. Alahari, "End-to-end incremental learning," in *ECCV 2018*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11216. Munich, Germany: Springer, Sep. 2018, pp. 241–257.

[13] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[14] S. Kang, D. Han, J. Lee, D. Im, S. Kim, S. Kim, and H. Yoo, "7.4 GANPU: A 135 TFLOPS/W multi-DNN training processor for GANs with speculative dual-sparsity exploitation," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 140–142.

[15] F. Tu, W. Wu, Y. Wang, H. Chen, F. Xiong, M. Shi, N. Li, J. Deng, T. Chen, L. Liu, S. Wei, Y. Xie, and S. Yin, "Evolver: A deep learning processor with on-device quantization–voltage–frequency tuning," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 2, pp. 658–673, 2021.