XST: A Crossbar Column-wise Sparse Training for Efficient Continual Learning

Fan Zhang Li Yang Jian Meng Jae-sun Seo Yu (Kevin) Cao Deliang Fan
fzhang95@asu.edu lyang166@asu.edu jmeng15@asu.edu jseo28@asu.edu ycao17@asu.edu dfan12@asu.edu
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe Arizona 85281

***Abstract*—Leveraging the ReRAM crossbar-based In-Memory-Computing (IMC) to accelerate single task DNN inference has been widely studied. However, using the ReRAM crossbar for continual learning has not been explored yet. In this work, we propose XST, a novel crossbar column-wise sparse training framework for continual learning. XST significantly reduces the training cost and saves inference energy. More importantly, it is friendly to existing crossbar-based convolution engine with almost no hardware overhead. Compared with the state-of-the-art CPG method, the experiments show that XST's accuracy achieves 4.95% higher accuracy. Furthermore, XST demonstrates ∼5.59× training speedup and 1.5× inference energy-saving.**

***Index Terms*—Continual Learning, In-Memory-Computing, Sparse Learning**

I. Introduction

In continual learning, given a pre-trained DNN model, conventional model fine-tuning for new tasks could easily forget old knowledge, thus degrading the learning performance on earlier tasks. Such a phenomenon is known as *catastrophic forgetting*. Recently, structure-based learning methods [1], [2] show the capability of alleviating the forgetting problem, which learn task-specific weights while freezing the weights of previous tasks as *preserved model*. In addition, inspired by the mask-based learning methods [3], they also apply a *learnable binary mask* to the preserved model, so as to further improve accuracy by selecting important weights. However, these methods require a complex two-stage *grow-then-prune* training procedure. For example, to adapt a preserved model to a new task, CPG [1], as a representative work, first learns all the free weights in the first stage and then element-wisely prunes it in the second stage to obtain the sparse weight and release space for next tasks, which suffers from huge training cost.

On the hardware side, the growing DNN model demands explosive multiply-and-accumulate (MAC) operations and data movement. In Von Neumann's architecture, such massive data movement may consume ∼2 orders higher energy than data processing. This phenomenon is known as the "Memory Wall". Recently, In-Memory-Computing (IMC) attracts much attention as a promising solution to the "Memory Wall" issue, due to directly processing the data within memory and eliminating data movement. Many different IMC designs have been proposed based on either volatile or non-volatile memories [4]–[7]. Among those designs, the ReRAM crossbar-based design is given transcendent expectations due to its simple structure, high on/off ratio, multi-bit per cell, non-volatility, and great compatibility with existing CMOS fabrication. Although many ReRAM crossbar-based designs have been proposed as area & energy-efficient computing core to support DNN inference, there is little exploration for continual learning, a practical and essential application in the real world. To apply the state-of-the-art CPG [1] mask-based continual learning to ReRAM crossbar hardware, it will require applying a binary element-wise mask for the fixed preserved model. However, implementing such element-wise masking scheme in ReRAM will inevitably bring significant hardware overhead in either much more complex peripheral circuits or consuming high power to reprogram ReRAM cell of the preserved model. Moreover, since the element-wise mask has the same dimension of weight parameters, it requires a large memory overhead for the learned new mask for each task.

These limitations motivate us to explore a new ReRAM crossbar friendly mask-based continual learning method that could leverage the mask based learning algorithm's benefit to avoid catastrophic forgetting in multi-task learning, as well as friendly with existing crossbar based DNN accelerator hardware with minimal peripheral circuits modification and mask memory overhead. Moreover, it should avoid power-hungry re-programming the ReRAM cells(i.e., preserved model).

In this work, we propose, XST, a new crossbar friendly sparse training framework for continual learning, which learns crossbar column-wise mask and sparse weight, taking advantages on both hardware inference implementation and software training. The key techniques are summarized as following:

1) **Hardware-friendly crossbar column-wise mask and sparse weight pattern.** To reduce the peripheral circuit overhead and avoid power hungry re-programming of ReRAM cells, motivated by our prior work [8], we also adapt a crossbar column-wise binary mask based continual learning method, where each learned mask value controls the on/off of entire crossbar column, instead of each element. Such column-wise design greatly shrinks the mask size, leading to memory overhead reduction.
2) **Sparse continual learning method.** We leverage sparse training method for continual learning, which adapts a novel *drop-and-grow* mechanism to learn a small portion of new weights for each task. We apply a learnable crossbar column-wise mask to the preserved model, so as to further improve the accuracy by selecting important weights(i.e., crossbar columns) for current task. Different from the conventional hard thresholding method [3] to learn the binary mask in CPG [1], we leverage the Gumbel-Sigmoid trick to better estimate the gradient of the mask during back-propagation.

II. Background

A. In-Memory Computing and NN Accelerator

Fig. 1 shows the basic structure of the ReRAM 1T1R array. Resistive memristor and access transistor sandwiched by horizontal SL and Vertical BL. It can not only work as a normal RAM to store/read data, more importantly, intrinsically support the vector-matrix-multiplication(VMM) operation. For the VMM operation, the matrix is stored at the crossbar intersections as conductance **G** where the input vector feed into

978-3-9819263-6-1/DATE22/©2022 EDAA

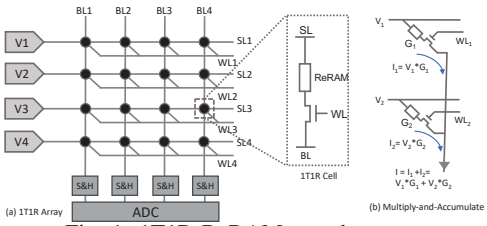

Fig. 1: 1T1R ReRAM crossbar array.

the crossbar through horizontal SL as input voltage. For the i, j_{th} 1T1R ReRAM cell, the current through that device will be $I_{i,j} = V_i \times G_{i,j}$. It is doing the multiplication in the analog domain between voltage and conductance. Since the vertical BL connects all 1T1R cells on the same column, the current throughout the BL is the summation of all the branches' current. Therefore, by sensing the accumulated current on BL, the MAC operation is achieved, i.e., $I_j = \mathbf{V} \times \mathbf{G}_{:,j}$, as shown in Fig. 1(b). Based on the ReRAM 1T1R structure, many NN accelerators have been proposed, e.g., ISAAC [4], PRIME [5], DPE [7], pipelayer [6], etc.

B. Single-task Sparse Training

Sparse training aims to train sparse neural networks with a fixed parameter count and a fixed computational cost throughout the whole training procedure. Rigl [9] develops the drop-and-grow mechanism that uses the magnitude-based method to prune and dense gradients to regrow connection in the same training iteration. Specifically, given a random initialized network with pre-defined sparsity, it first drops/prunes α fraction of weights, which have the smallest magnitude. Then it regrows the same number of new weights with the largest gradient magnitude, to keep the fixed parameter and computation count during the training. In addition, Rigl shows that gradually decaying the ratio of drop and grow α by following a cosine annealing could achieve better accuracy. In this work, for the first time, we adapt such mechanism as a backbone technique to develop continual learning algorithm.

III. Methodology

The overview of the proposed XST is illustrated in Fig. 2, which includes the overflow of both off-line training and mapping to ReRAM crossbar based IMC for inference. Following the general continual learning setting [1], new tasks ($\{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_N\}$) arrive sequentially and past tasks cannot be used for training future tasks. Based on this, XST adapts the sparse training method (Fig. 2(d)) to learn a small fraction of task-specific weights for each task. Meanwhile, to adapt the preserved model for previous tasks to current task, XST also leverages the selective masking method to learn a binary mask on the preserved model, so as to select the important weights for current task, as shown in Fig. 2(a). Then after the off-line training of each task, we update the crossbar array to support the new learnt weights, without re-programming the preserved model, for online inference, as shown in Fig. 2(b). More importantly, XST designs the learnable parameters (i.e., column-wise mask and column-wise sparsity of weight) both in crossbar column-wise (Fig. 2(c)), where each value controls the operation of the entire crossbar column, enabling hardware friendly crossbar mapping.

A. Column-wise parameter pattern

According to the 1T1R crossbar's structure, the transistor's gates are connected by SL either horizontally or vertically. Then, individually controlling each transistor to apply a binary element-wise mask will inevitably cause large overhead. However, benefiting from the row/col wise parallelism, controlling the SL to turn on/off the entire row/column is friendly for the existing crossbar design with minor overhead. In the conventional convolution kernel mapping method, the kernel has been divided by output feature map dimension. For example, a $C_{out} \times C_{in} \times kh \times hw$ kernel will be reshaped to a $(C_{in} \times kh \times kw, C_{out})$ sized 2D matrix, where C_{in}, C_{out}, kh, kw refer the weight dimension of a convolutional layer, including #output, #input channel, kernel height and width, respectively. With the development of deep learning in recent years, DNNs grow into more complex and larger structures, the size of one filter $C_{in} \times kh \times kw$ usually is too large to be mapped into a single crossbar column. A general solution is to further partition and then map one filter into multiple columns.

In this work, the proposed method mainly includes two types of learnable parameters: the new learned weights for current task, and the binary mask for the preserved weights of previous tasks motivated by our prior work [8]. Based on the analysis above, we define both the binary mask and the sparsity pattern of new weights in column-wise. For the column-wise mask, we represent mask as $G \times kh \times kw$ to make it consistent with the size of a crossbar column, where the group $G \in \{1, C_{\text{in}}\}$. By doing so, a single mask value can control the entire column of a crossbar array, which improves the computation efficiency significantly compared to element-wise mask. Similarly, for the new learned weight, we define the weight sparsity size as $G \times kh \times kw$. By doing so, only part of weights, where are not all zeros values, need to perform the energy-hungry programming to the corresponding columns without affecting other columns.

B. Column-wise Sparse Continual learning

As shown in Fig. 2(a), the proposed sparse continual learning algorithm adapts the sparse training method (Fig. 2(d)) to learn a small fraction of task-specific weights for each task. Meanwhile, to adapt the preserved model for previous tasks to current task, XST also leverages the selective masking method to learn a binary mask on the preserved model. In the following, we present our method in the sequential-task manner:

Learning Task 1: Similar to sparse train a network model for a single dataset from scratch, given the first task, we first randomly initialize a certain sparsity s_1 of the weights, but in column-wise. Then, the *drop-and-grow* mechanism [9] is adapted as mentioned in Section II-B to learn a small fraction of the weights $1 - s$. After training, the current model will serve as the preserved model for next task. Note that, as the difficulty of the training from scratch, we leverage a smaller sparsity ratio to train the task 1 (i.e., 0.7 in our experiments).

Learning Tasks 2, ..., T: Given a network with a pre-define model size, to guarantee new weights can be learnt for

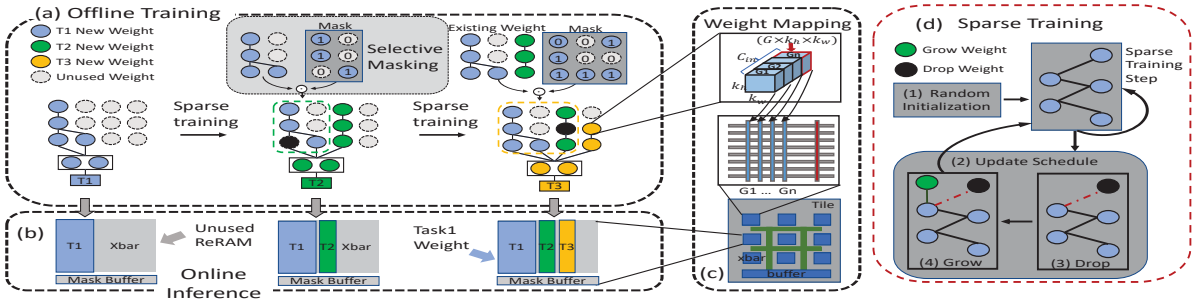

Fig. 2: Overview of the proposed XST. (a) The training workflow. (b) The inference workflow. (c) The column-wise ReRAM crossbar weight mapping. (d) Sparse training technique.

each task, we uniformly assign the same sparsity ratio $s_i = (1 - s_i)/(T - 1)$. Assume that in task t_i, the model that can handle task 1 to t_{i-1} has been built as a preserved model. We adapt two techniques to learn the model for task t_i:

1) **Sparse learning on new weights.** To learn the weights for current task t_i, we adapt the drop-and-grow strategy, meanwhile freezing the weights of previous tasks.
2) **Selective masking on preserved weights.** We apply a learnable column-wise mask to the preserved model, so as to select the important weights for current task. Note that, these two techniques are independent and hence can be jointly optimized in the same training process.

C. Column-wise Selective Masking

To learn the binary mask, we leverage the Gumbel-Sigmoid trick, inspired by Gumbel-Softmax [10] that performs a differential sampling to approximate a categorical random variable. Since sigmoid can be viewed as a special two-class case of softmax, we define $p(\cdot)$ using the Gumbel-Sigmoid trick as:

$$p(\boldsymbol{m}^r) = \frac{\exp((\log\pi_0 + g_0)/T)}{\exp((\log\pi_0 + g_0)/T) + \exp((g_1)/T)}, \quad (1)$$

where π_0 represents $\sigma(\boldsymbol{m}^r)$. g_0 and g_1 are samples from Gumbel distribution. The temperature T is a hyper-parameter to adjust the range of input values, where choosing a larger value could avoid gradient vanishing during back-propagation. Note that the output of $p(\boldsymbol{m}^r)$ becomes closer to a Bernoulli sample as T is closer to 0. We can further simplify Eq. (1) as:

$$p(\boldsymbol{m}^r) = \frac{1}{1 + \exp(-(\log\pi_0 + g_0 - g_1)/T)} \quad (2)$$

Benefiting from the differential property of Eq. (2), the real-value $\boldsymbol{m}^r$ can be embedded with gradient back-propagation training. To represent $p(\boldsymbol{m}^r)$ as binary format $\boldsymbol{m}^b$, we use a hard threshold (i.e., 0.5) during forward-propagation of training. Because most values in the distribution of $p(\boldsymbol{m}^r)$ will move towards either 0 or 1 during training, generating the binary mask by $p(\boldsymbol{m}^r)$ (instead of the real-value mask $\boldsymbol{m}^r$ directly) could have more accurate decision, resulting in better accuracy.

IV. Evaluation

Similar to the prior works, we adopt the CIFAT-100 dataset and divide the dataset into 20 superclasses(equal to 20 tasks), each superclass contains 5 sub-classes. Each sub-classes has 2500 training images, and 500 tesing images. We employed VGG16 with batch normalization layers and expanded the model to 1.5×. All the 20 tasks have been trained and tested sequentially. Each task has been trained 100 epochs.

A. Algorithm Evaluation

Table I shows the classification accuracy of CPG, XST (Element-wise), and XST (Column-wise) where the quantization method is adopted from [11]. We choose the group size $G = 8$ in the experiment. Although CPG and XST both are using the mask to select important exist weights and growing new weights. Benefit from the Gumbel-sigmoid trick and efficient sparse learning, XST shows higher accuracy than the CPG under the same setup. CPG first tries to explore the whole available weight, then prunes the weight while maintaining similar accuracy. XST not only has the weight drop but also has the weight grow mechanism to more accurately find the most critical weights. Therefore, XST has higher flexibility to tune the weights and better estimate gradient to generate binary masks, resulting in performance improvement. Even column-wise XST only has 0.51%(floating number) accuracy drop than CPG. Our proposed XST also shows strong robustness to quantization. With 4-bit quantization, CPG drops more than 6% accuracy while XST only has 1.56% and 0.72% accuracy degradation for element-wise and column-wise, respectively. Comparing with the floating number CPG, even the 4-bit quantized column-wise XST shows competitive performance.

Fig. 3 shows the training time comparison for each task. Although XST has more complex grow and drop steps, thanks to the efficient sparse learning that significantly reduces the training effort. Unlike CPG, XST does not require the most time-consuming part, gradually pruning. Therefore, on average, to achieve the accuracy in Table I, CPG takes ∼5.59× more time than XST. Due to the column mask sharing, XST's mask size is only $\frac{1}{72}$ of CPG, which leads to an easier and more efficient hardware design.

B. Hardware Evaluation

We implement our XST and other competitor methods on the same hardware platform. We use the circuit level simulator NeuroSim [12] for hardware performance evaluation with different schemes. The 4-bit quantized VGG-16-1.5× is implemented based on 2-bit per cell HfO_2 1T1R ReRAM devices, characterized from [13] and projected to 32nm CMOS node. Table II summarizes the detailed ReRAM array characteristics and total area consumption. Each ReRAM column is connected

TABLE I: Continual Learning Accuracy

	Method \ Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	AVG
4-bit Quantization	Ours(Element-wise)	67.6	77.6	76.8	80.4	85.6	86.6	82.8	85	86	90	89.2	84.4	89.8	83.4	54.6	76.4	73	73.6	87.8	94.4	**81.25**
	Ours(Column-wise)	66.6	78.6	78.2	79.8	86.2	83.2	80.8	85.2	85.2	89.6	88	79.6	84.4	81.4	52	72.4	67.4	73.2	88	93.6	**79.67**
	CPG(Element-wise)	59.4	74.4	73.8	75	81.8	80.2	80.4	82	77	82.4	78.2	79.8	80.8	71.6	44.4	64	63.4	68	85.2	92.6	**74.72**
Floating number	Our(Element-wise)	65.6	78.8	78.4	81.6	86.4	86	84	86.8	85.4	89.4	90	86.2	89.4	83.6	57.2	76	73.4	74.4	91	94.8	**81.92**
	Our(Column-wise)	66.2	77.6	78.4	82.4	86.8	85.6	81.4	82.8	83.8	89	89.2	82.8	86.2	81.6	55	73.2	68.8	74.2	89	93.8	**80.39**
	CPG(Element-wise)	65.2	76.6	79.8	81.4	86.6	84.8	83.4	85	87.2	89.2	90.8	82.4	85.6	85.2	53.2	74.4	70	73.4	88.8	94.8	**80.9**

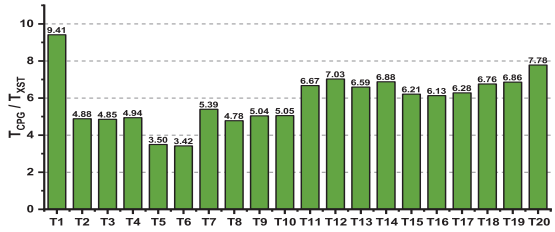

Fig. 3: Training time comparison

TABLE II: Hardware specification

RRAM Sub-Array		
Components	**Area** (μm^2)	**Energy** (pJ)
Memory Array (72×72)	84.93	
Switch Matrix (WL and SL)	457.3	1.1
SAR ADC (5-bit)	8,409.3	8.3
Shift-Add-Input	1,412.9	6.8
Shift-Add-Weight (2 col use 1)	825.8	1.0
Mask Buffer (72×1)	190.4	0.003/bit/access
Total	11,380.2	17.2
Peripheral Circuits		
1 stage AdderTree (128 units)	2,510.3	4.4
2 stage AdderTree (128 units)	7,740.1	13.7
3 stage AdderTree (128 units)	18,408.8	32.6
Global Buffer ($96 \times 32 \times 32 \times 4$)	1,039,596	0.003/bit/access
ReLU (128 units)	939.5	0.9

to a 5-bit successive approximation register (SAR) analog-to-digital converter (ADC). To avoid frequent off-chip memory access, we choose the global buffer as the same size of the largest feature map during the inference process.

Performing the inference with the CPG learning requires all the ReRAM columns to stay ON for all the sub-tasks. Therefore, the inference energy consumption per task is identical after fine-tuning or CPG [1] learning. On the other hand, the proposed algorithm updates the weight and mask in a column-wise fashion for different sub-tasks. Compared to CPG learning, the proposed XST learning will only use part of the ReRAM sub-array for each task. Therefore, the inference energy consumption will be reduced. XST can achieve $\sim$2.8$\times$ inference energy reduction for the early learning tasks.

Another major hardware disadvantage induced by CPG learning is the element-wise reprogramming and resetting. We quantified the energy consumption based on the writing voltage, writing pulses, and conductance level changes [13], [14]. As shown in Figure 4, the element-wise reprogramming and resetting of the CPG learning [1] lead to inconsistent and massive energy consumption during the entire continual learning phase (T2 to T20). The proposed XST resets the weights by turning off the corresponding ReRAM columns. As a result, the hardware resetting cost of the XST becomes zero. During the continual learning process, XST replenishes the weights in a group-wise manner, corresponding to the consistent reprogramming cost of the ReRAM columns. Compared to the CPG learning, the XST-trained model achieves 2.5$\times$ reprogramming energy reduction along sub-tasks.

V. Conclusion

In summary, we propose XST, a hardware-friendly crossbar column-wise sparse learning method to efficiently deploy

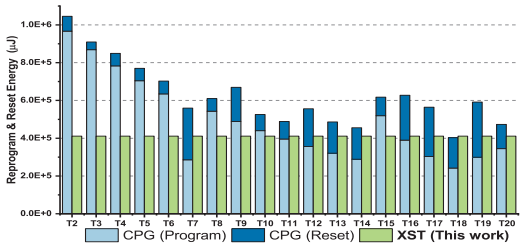

Fig. 4: Energy consumption of the reprogramming and resetting

continual learning to ReRAM crossbar based neural network accelerator with the consideration of hardware cost. Comparing with CPG, our XST shows 4.95% accuracy improvement, $\sim$5.59X training speedup, 1.5$\times$ inference energy saving, and 1.8X$\times$ re-programming energy saving on CIFAR-100 dataset with VGG16-BN(1.5$\times$) model.

Acknowledgment

This work is supported in part by the National Science Foundation under Grant No.2003749, No.1931871, No. 2144751

References

[1] S. C. Y. Hung *et al.*, "Compacting, picking and growing for unforgetting continual learning," 2019.

[2] J. Yoon *et al.*, "Lifelong learning with dynamically expandable networks," *arXiv preprint arXiv:1708.01547*, 2017.

[3] A. Mallya *et al.*, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 67–82.

[4] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 14–26.

[5] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[6] L. Song *et al.*, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 541–552.

[7] M. Hu *et al.*, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[8] F. Zhang *et al.*, "Xbm: A crossbar column-wise binary mask learning method for efficient multiple task adaption," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022.

[9] U. Evci *et al.*, "Rigging the lottery: Making all tickets winners," in *International Conference on Machine Learning*. PMLR, 2020.

[10] E. Jang *et al.*, "Categorical reparameterization with gumbel-softmax."

[11] Y. Li *et al.*, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," in *International Conference on Learning Representations*, 2020.

[12] X. Peng *et al.*, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *IEEE International Electron Devices Meeting (IEDM)*, 2019.

[13] W. Wu *et al.*, "A methodology to improve linearity of analog RRAM for neuromorphic computing," in *IEEE Symposium on VLSI Technology*, 2018, pp. 103–104.

[14] P.-Y. Chen *et al.*, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 12, pp. 3067–3080, 2018.