XBM: A Crossbar Column-wise Binary Mask Learning Method for Efficient Multiple Task Adaption

Fan Zhang fzhang95@asu.edu
Li Yang lyang166@asu.edu
Jian Meng jmeng15@asu.edu
Yu (Kevin) Cao ycao17@asu.edu
Jae-sun Seo jseo28@asu.edu
Deliang Fan dfan12@asu.edu
Ira A. Fulton Schools of Engineering, Arizona State University, Tempe Arizona 85281

2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC) | 978-1-6654-2135-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ASP-DAC52403.2022.9712508

Abstract— Recently, utilizing ReRAM crossbar array to accelerate DNN inference on single task has been widely studied. However, using the crossbar array for multiple task adaption has not been well explored. In this paper, for the first time, we propose XBM, a novel crossbar column-wise binary mask learning method for multiple task adaption in ReRAM crossbar DNN accelerator. XBM leverages the mask-based learning algorithm's benefit to avoid catastrophic forgetting to learn a task-specific mask for each new task. With our hardware-aware design innovation, the required masking operation to adapt for a new task could be easily implemented in existing crossbar based convolution engine with minimal hardware/ memory overhead and, more importantly, no need of power hungry cell re-programming, unlike prior works. The extensive experimental results show that compared with state-of-the-art multiple task adaption methods, XBM keeps the similar accuracy on new tasks while only requires 1.4% mask memory size compared with popular piggyback. Moreover, the elimination of cell re-programming or tuning saves up to 40% energy during new task adaption.

I. Introduction

Nowadays, one practical limitation of deep neural network (DNN) is its high degree of specialization to a single task. It motivates researchers to develop algorithms that can adapt the DNN model to multiple tasks sequentially, meanwhile still performing well on the past tasks. This process of gradually adapting the DNN model to learn from different tasks over time is known as multi-task adaption. Fine-tuning is a natural way to adapt the current model (i.e., backbone model) to a new task. However, updating the parameters of the backbone model could result in the forgetting of old knowledge upon earlier tasks, thus degrading the performance. Such phenomenon is known as catastrophic forgetting, which widely exists in multi-task adaption. To alleviate the catastrophic forgetting, several mask-based methods have been proposed i.e., Piggyback and KSM [1, 2], which only learn a task-specific mask w.r.t all weights for each new task, while keeping the backbone model fixed.

From the DNN hardware accelerator design domain, DNN involves a huge amount of multiply–and-accumulate (MAC) operations and data movement. In traditional von Neumann architecture (e.g., CPU, GPU), the data movement consumes ∼100× higher energy than a floating-point operation which is also known as "memory wall" [3]. Recently, the in-memory computing (IMC) has attracted an increasing interest due to its ability to execute computing tasks directly within the memory array. Such ability significantly alleviates the "memory wall" issue [4, 5, 6, 7]. Among different volatile/non-volatile IMC designs, ReRAM crossbar based design is a promising candidate for the next generation DNN accelerator, due to its simple structure, high on/off ratio, high density, multi-bit per cell storage, and fabrication compatibility with CMOS [3, 4, 5, 8].

Motivation: Almost all existing works utilize ReRAM crossbar as an area and energy efficient hardware for deployment of DNN inference on a single specialized task or domain, but there is little consideration to support multiple task adaption based on ReRAM crossbar. In this context, to adapt the current model deployed in ReRAM crossbar for a new task, the most intuitive and straightforward way is to fine-tune the weight parameters (i.e., cell conductance) based on the new knowledge. However, this scheme will require to update the conductance of almost all cells to reflect the new set of fine-tuned weight parameters, which is inefficient and impractical in real-world multi-task learning due to limitation in both the ReRAM device (e.g., high re-programming power, limited endurance, etc.) and algorithm (e.g., catastrophic forgetting for large scale multi-task learning). As discussed earlier, the mask-based multi-task learning is one of the most popular methodologies nowadays to address the catastrophic forgetting issue. To apply the representative Piggyback [1] mask learning method to ReRAM crossbar hardware, it will require to learn a binary element-wise mask ($\{0, 1\}$) w.r.t all the weights for the new task, while keeping the backbone model fixed. Thus, to implement the learned mask in ReRAM crossbar hardware, it needs to either develop complex control circuits to individually turn on/off each cell in convolution computation, or reprogramming the cell conductance to reflect the mask value - '0' (meaning this cell should not be involved in the new task computing path). It could be easily seen that both possible designs will need large hardware overhead in either much more complex extra peripheral circuits or still

978-1-6654-2135-5/22/$31.00 ©2022 IEEE

re-programming partial ReRAM cell values. Also, since it is a element-wise mask, it requires a much large memory overhead for the learned new mask. For example, for an 8-bit DNN model, the learned new element-wise mask in Piggyback will cause memory overhead of 1/8 of total model size only for one new task.

Objective: These limitations motivate us to explore a new ReRAM crossbar friendly mask-based learning method that could leverage the mask based learning algorithm's benefit to avoid catastrophic forgetting in multi-task learning, and also could be easily implemented in existing crossbar based DNN accelerator hardware with minimal peripheral circuits and mask memory overhead, and more importantly, no need to re-program ReRAM cell values.

Contributions: This work is the first to propose a new crossbar friendly multi-task learning method, called *XBM* (Crossbar Binary Mask), which learns a crossbar column-wise binary mask for multi-task adaption, while keeping the backbone model fixed. Note that, in popular crossbar-based DNN accelerator weight mapping, each column corresponds to a group of kernels, e.g., a group of 8×3×3 kernels could be mapped to one column of one 72×72 crossbar array to implement parallel convolution computation. Therefore, in our XBM method, each column-wise binary mask value (1/0) controls the on/off of entire column, rather than each cell element in Piggyback. Through such, the above discussed objective could be achieved, with minor hardware peripheral circuit modification and no need to re-program any ReRAM cell value to implement masking operation. Our method is distinguished from prior works in the following aspects:

1. **Hardware friendly crossbar column-wise mask.** To reduce the peripheral circuit overhead for implementing the masking function in hardware and avoid power hungry re-programming of ReRAM cell in multi-task adaption, we are the first to design a new crossbar column-wise binary mask (XBM) based multi-task learning method, where each learned mask value (1/0) controls the on/off of entire crossbar column for the new task inference, instead of each element in prior works.
2. **Mask size reduction.** Another benefit of our XBM is the great reduction of mask size (thus, memory overhead) depending on the crossbar size. For instance, assuming 72×72 crossbar size, only a single mask value is needed in our XBM to control one column, i.e., a group of 8×3×3 kernels, instead of 72 separate mask values, with 72 times smaller mask size than prior element-wise mask.
3. **Gumbel-Sigmoid trick.** Different from the conventional hard thresholding method [1] to learn the binary mask, in our XBM learning, we propose to leverage the Gumbel-Sigmoid trick to better estimate the gradient of the mask during back-propagation.

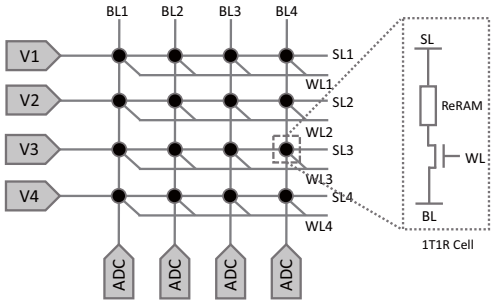

Fig. 1. ReRAM 1T1R crossbar array.

II. Background

A. In-Memory Computing and NN Accelerator

Fig. 1 shows the basic structure of the 1T1R crossbar array which can efficiently perform the vector-matrix multiplication (VMM) operation. In the 1T1R array, the weight matrix is stored at the cross-point ReRAM cells as the conductance G while the input vector is fed through the horizontal SL as analog voltage V_{in} [8, 9]. According to the Kirchhoff's Current Law (KCL), $I_{BL} = G \cdot V_{in}$, the current on bit-line is the VMM result. A $m \times n$ sized crossbar array can perform the VMM operation in one step which reduces the time complexity from O(mn) to O(1). Since the 2D convolution can be transferred to VMM either by Toeplitz matrix or unrolling the convolution kernel. Recently a lot of ReRAM crossbar array based neural network accelerator designs have been proposed to leverage the IMC's energy efficiency and high throughput [3, 4, 5, 6].

Existing ReRAM crossbar designs focus on improving energy efficiency for fixed off-line trained model. Reprogramming is necessary if the dataset or task changes. Although Fouda et al. [10] proposed a mask based method for crossbar array, this mask is used only during off-line training. Moreover, that method is employed to alleviate sneak path problem, not for multi-task adaption.

B. Multi-Task Adaption

Multi-task adaption [11, 12] aims to build a model, which can adapt a task into multiple visual tasks/domains without forgetting previous knowledge, and meanwhile using as fewer parameters as possible. [12] proposes to recombine the weights of the backbone model via controller modules in channel-wise. [13] proposes domain-specific attention modules for the backbone model. One of the most related method is Piggyback [1], which solves the issue by learning task-specific binary masks for each task. They achieve this by generating the real-value masks which own the same size with weights, passing through a binarization function to obtain binary masks, which are then applied to existing weights. We denote the real-value mask and binary mask as m^r and m^b respectively. Then,

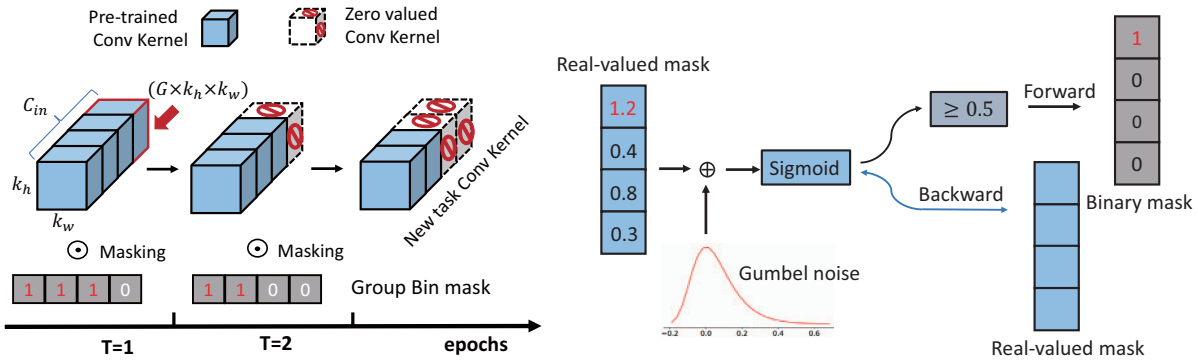

Fig. 2. **Left:** Overall working flow of our proposed XBM. **Right:** The binary masks are trained using a combination of Gumbel-Sigmoid function and hard thresholding.

the binarization function is given by:

$$\text{Forward}: \quad \boldsymbol{m}^{\mathrm{b}} = \begin{cases} 1 & \text{if } \boldsymbol{m}^{\mathrm{r}} \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$\text{Backward}: \quad \nabla \boldsymbol{m}^{\mathrm{b}} = \nabla \boldsymbol{m}^{\mathrm{r}} \tag{2}$$

Where τ is a constant threshold value. However, the gradient of binarization is non-differential during back-propagation. They use the straight-through estimator (STE) [14] to solve this problem, which estimates the gradient of real-value mask by the gradient of binary mask. Furthermore, [2, 15] combine the binary mask with additional floating-point scaling values to improve the adaption capacity, but suffer from even more computation and memory cost during the training procedure.

III. Methodology

In this work, we propose XBM, a crossbar-based column-wise binary mask learning method for fast and efficient multiple task adaption. Following the multiple task adaption setting in [2, 11], 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, we aim to learn a task-specific mask for each arriving task without changing the parameters of the backbone model. Specifically, given a convolution layer, we denote the weights $w^{(l)} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}} \times kh \times kw}$, where c_{in}, c_{out}, kh, kw refer the weight dimension of l-th layer, including #output channel, #input channel, kernel height and width, respectively. We also denote the dataset of the t-th task $(\mathcal{T}_t)$ as $D_t = \{x_t, y_t\}$, where x_t and y_t are vectorized input data and label pair. To adapt the pre-trained backbone model with the parameter $\{w_1\}$ from the initial task $\mathcal{T}_1$ to a new task $\mathcal{T}_t$ with crossbar deployment efficiency, we intend to learn a task-specific mask in column-wise $m \in \mathbb{R}^{G \times kh \times kw}$ that is applied to the fixed parameter w_1. By doing so, each mask element is shared by a column-wise $G \times kh \times kw$ kernels as shown in Fig. 2 (left). Based on this idea, to learn the task $\mathcal{T}_t$ by masking the fixed parameter w_1, the optimization objective can be mathematically formalized as:

$$\min_{\boldsymbol{m}_t} \mathcal{L}\Big(f(\boldsymbol{x}_t; \{\boldsymbol{m}_t \times \boldsymbol{w}_1\}), \boldsymbol{y}_t\Big) \tag{3}$$

In the following, we will present the detailed setting, learning method and hardware mapping of the column-wise mask respectively.

A. Column-wise mask

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 is difficult to realize. However, benefiting from the row/col wise parallelism, controlling the SL to turn on/off the entire row/column is an easy job for the existing crossbar design. 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. With the develop of deep learning in recent layers, the deep neural networks grow into more complex and larger structures, the size of one filter $C_{in} \times kh \times kw$ usually is too large to fit into a single crossbar column. A general solution is to further partition and then map one filter into multiple columns.

Therefore, we define the mask size as $G \times kh \times kw$ to make it consistent with the size of a crossbar column, namely column-wise mask, 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. In our design, the size of crossbar column is set as 72×1. Equivalently, we define the group size of the kernel-wise mask as $8 \times 3 \times 3$ with the group $G = 8$ in the algorithm.

B. Learning the binary mask

The conventional way [1] of generating the binary trainable mask is to train a learnable real-valued mask $(\boldsymbol{m}^r)$ followed by a hard threshold function (i.e., sign function) to binarize it as shown in Eq. (1). However, such a hard threshold function is not differential, and the general solution is to approximate the gradients by skipping the threshold function during back-propagation and update

the real-value masks directly. Different from that, in this work, we propose a method to better estimate the gradient by using Gumbel-Sigmoid trick as shown in Fig. 2 (right).

First, we relax the hard threshold function to a continuous logistic function:

$$\sigma(\boldsymbol{m}^r) = \frac{1}{1+\exp(-\beta \boldsymbol{m}^r)}, \tag{4}$$

where β is a constant scaling factor. Note that the logistic function becomes closer to the hard thresholding function for higher β values.

Then, to learn the binary mask, we leverage the Gumbel-Sigmoid trick, inspired by Gumbel-Softmax [16] 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:

$$p(\boldsymbol{m}^r) = \frac{\exp((\log\pi_0 + g_0)/T)}{\exp((\log\pi_0 + g_0)/T) + \exp((g_1)/T)}, \tag{5}$$

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. (5) as:

$$p(\boldsymbol{m}^r) = \frac{1}{1+\exp(-(\log\pi_0 + g_0 - g_1)/T)} \tag{6}$$

Benefiting from the differential property of Eq. (4) and Eq. (6), the real-value mask $\boldsymbol{m}^r$ can be embedded with existing gradient based 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 as mentioned in Eq. (1)) could have more accurate decision, resulting in better accuracy.

C. Hardware structure and weight mapping

Fig. 3 shows the ReRAM crossbar based NN accelerator design to support the proposed column-wise binary mask. Unrolled backbone model's convolutional kernel mapped to the ReRAM crossbar sub-array. To be consistent with the aforementioned group size, we use the 72×72 crossbar array. Each crossbar array can map a 72×8×3×3 convolution kernel. Any convolution kernel large than 72×8×3×3 will be partitioned into multiple arrays. In that case, each array will generate a partial sum instead of the activation. The global adder tree takes those partial sum and carry out the corresponding activation. Then send it to the global ReLU unit. In the crossbar array, each ReRAM cell stores 2-bit. We use two adjacent columns to represent the 4-bit weight. Shift-adder (SA) combines the two 2-bit results on BLs to generate the 4-bit partial sum activation.

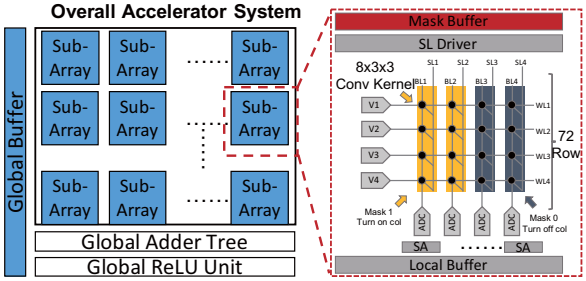

Fig. 3. ReRAM crossabr based NN accelerator architecture and weight mapping.

For the fine-tuning or Piggyback methods, we choose the re-programming method to update the fine-tuned weight or masked weight. During the inference, SL driver turns on the whole array's 1T1R cells to perform the 72×8×3×3 convolution simultaneously.

In order to support the column-wise binary mask, we add mask buffer to store the binary mask next to corresponding crossbar array, as highlighted in Fig. 3 associated with SL driver. To easily control the column on/off based on mask value, we connect the SL vertically instead of horizontally. The SL connects with cell transistor gates of the whole column. This way, the column-wise binary mask value could be sent to the SL driver's input to turn on/off the entire column, with no modification to other existing peripheral circuits. For one 72×72 crossbar array, the memory buffer overhead is 72 bits. For example, for the whole ResNet-50 model, with 8×3×3 group size, the total memory overhead is $23M/(8*3*3)/8 \approx 40KB$. Comparing with the 4-bit weight ResNet-50 model with $23M/2 = 11.5MB$, such 40KB mask buffer is only 0.35% overhead.

IV. Experiment

In this section we evaluate our proposed XBM from two aspects: algorithm and hardware. Similar to prior works, five image classification datasets are used: CUBS [17], Stanford Cars [18], Flowers [19], Wikiart [20], and Sketch [21]. We use the ResNet-50 as the backbone model which is pre-trained on ImageNet dataset [22].

A. Algorithm Evaluation

Table I shows the inference accuracy on different dataset. Here we use ResNet-50 as backbone model which is trained on ImageNet dataset with 4-bit weight and 4-bit activation quantization. The quantization method is adopted from PROFIT [23]. We choose the group size $G = 8$ in the experiment.

Fine-tuning the backbone model achieves the best accuracy in most datasets, since fine-tuning has the highest

TABLE I
MULTI-TASK ADAPTION ACCURACY

		Continual Learning (4-bit Quantization)	
	Finetune	Piggyback	XBM (This work)
CUBS	73.02%	74.47%	**75.53%**
Stanford_cars	85.92%	**86.85%**	85.96%
flowers	**95.34%**	91.09%	90.81%
Wikiart	**74.96%**	68.97%	67.6%
Sketches	**80.92%**	78.88%	76.95%

flexibility to change any weight to any quantized level. Although Piggybask can adapt any weight, the binary mask makes it lose some representation ability. Thus Piggyback shows slightly worse accuracy than fine-tuning. Our proposed XBM not only has the same binary limitation but also has the group concept. Those limitations suppose to further contaminate the accuracy. However, owing to the softmax trick that can better estimate the gradient, there is not much accuracy drop compared to the element-wise Piggyback, even though we share one mask value among 72 weights. Due to the group mask sharing, XBM's mask size is only $\frac{1}{72}$ of Piggyback. For the ResNet-50 backbone model, Piggyback's element-wise binary mask requires $23M/8 = 2.88MB$, while the XBM only consumes around 40KB.

Fig. 4 shows the mask sparsity among Piggyback and our XBM. Since the Finetune's sparsity values are too low, we do not plot them in this figure. Our proposed method always achieves more than 30% mask sparsity. Due to the group mask, it can be easily applied on crossbar array. Such high sparsity leads to more than 30% energy reduction, which will be explained in the next sub-section.

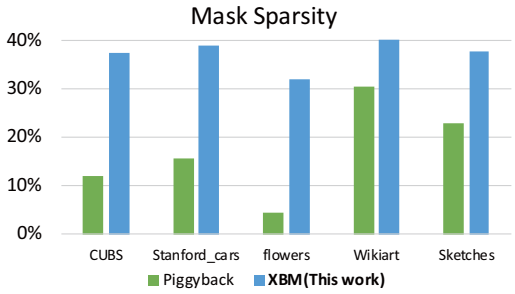

Fig. 4. Binary mask sparsity comparison

B. Hardware Evaluation

For a fair comparison, we implement all multi-task adaption methods on the same evaluation hardware platform as shown in Fig. 3.

We use the circuit level simulator NeuroSim [24] to evaluate the hardware performance of different learning schemes. The 4-bit quantized targeted DNNs are implemented based on 2-bit per cell HfO_2 1T1R ReRAM devices, characterized from [25] and projected to 32nm CMOS node. Table II and Figure 5 summarizes the detailed ReRAM array characteristics and total area consumption. Each ReRAM column is connected to a 5-bit

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 ($64 \times 112 \times 112 \times 4$)	8,490,034	0.003/bit/access
ReLU (128 units)	939.5	0.9

TABLE III
INFERENCE ENERGY PERIMAGE

	4-bit ResNet-50		
Method / Dataset	Finetune	Piggyback	**XBM(Binary Group Mask)**
CUBS	30.25 μJ	30.25 μJ	21.20 μJ
Stanford_cars	30.25 μJ	30.25 μJ	20.82 μJ
flowers	30.25 μJ	30.25 μJ	22.53 μJ
Wikiart	30.25 μJ	30.25 μJ	20.63 μJ
Sketches	30.25 μJ	30.25 μJ	21.12 μJ

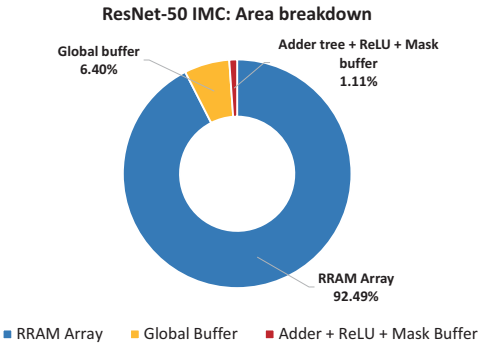

Fig. 5. Area breakdown of 4-bit ResNet-50 backbone model hardware deployment

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.

Table III summarizes the total energy consumption per input image (224×224×3). The element-wise masks generated by Piggyback [1] partially program the weights to zero, but it cannot effectively reduce the overall energy consumption since the rest of the cells along each column remains active. Therefore, the inference energy consumption is identical after fine-tuning or Piggyback [1] learning. The proposed XBM algorithm exploits the mask sparsity in a column-wise fashion. As a result, the entire column can be removed from the hardware inference process, and the overall energy consumption will be reduced. Compared to the normal fine-tuning and Piggyback [1] learn-

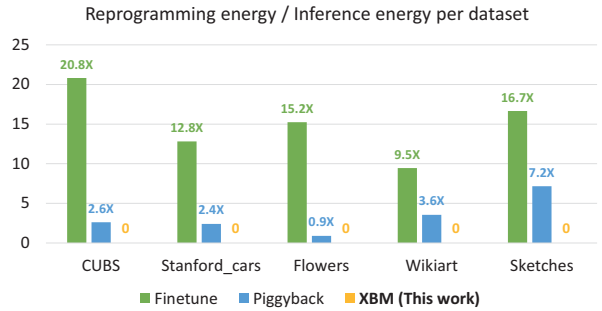

Fig. 6. Energy consumption of the reprogramming with different learning methods

ing schemes, the XBM can reduce the energy consumption by ∼1.5× with negligible hardware overhead.

Fine-tuning the model entirely or learning the element-wise masks requires reprogramming or even second-time deployment, which consumes enormous amounts of energy. The energy consumption caused by the weight increase/decrease during the programming can be computed based on the writing voltage, writing pulses, and conductance level changes [25, 26]. Fig. 6 demonstrated the energy consumption overhead of the reprogramming. Compared to the inference energy consumption of the entire test set, reprogramming the entire model causes a massive amount of energy overhead (over 20×). Such significant energy overhead of the previous method promoted our proposed method as the best solution. Learning the new features by turning off the ReRAM columns enables us to skip the reprogramming and second-time deployment.

V. Conclusion

In summary, we proposed XBM, a binary crossbar mask to efficiently deploy the multi-task adaption to crossbar-based neural network accelerator design with the consideration of hardware cost. Comparing with state-of-the-art methods, XBM do not need to change the neural network structure or re-program any ReRAM cell. Comparing with other mask-based method, XBM saves up to 40% inference energy and reduces the mask size to only 1.4% while maintaining the similar accuracy.

Acknowledgements

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

References

[1] 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.

[2] L. Yang *et al.*, "Ksm: Fast multiple task adaption via kernel-wise soft mask learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 845–13 853.

[3] S. Mittal, "A survey of reram-based architectures for processing-in-memory and neural networks," *Machine Learning and Knowledge Extraction*, vol. 1, no. 1, pp. 75–114, 2019.

[4] 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.

[5] X. Sun *et al.*, "Xnor-rram: A scalable and parallel resistive synaptic architecture for binary neural networks," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 1423–1428.

[6] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, ser. ISCA '18. IEEE Press, 2018, p. 383–396. [Online]. Available: https://doi.org/10.1109/ISCA.2018.00040

[7] D. Fan *et al.*, "Energy efficient in-memory binary deep neural network accelerator with dual-mode sot-mram," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 609–612.

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

[9] F. Zhang *et al.*, "Cccs: Customized spice-level crossbar-array circuit simulator for in-memory computing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020.

[10] M. E. Fouda *et al.*, "Mask technique for fast and efficient training of binary resistive crossbar arrays," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 704–716, 2019.

[11] S.-A. Rebuffi *et al.*, "Learning multiple visual domains with residual adapters," in *Advances in Neural Information Processing Systems*, 2017, pp. 506–516.

[12] A. Rosenfeld *et al.*, "Incremental learning through deep adaptation," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[13] S. Liu *et al.*, "End-to-end multi-task learning with attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1871–1880.

[14] I. Hubara *et al.*, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[15] M. Mancini, E. Ricci, B. Caputo, and S. Rota Bulo, "Adding new tasks to a single network with weight transformations using binary masks," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[16] E. Jang *et al.*, "Categorical reparameterization with gumbel-softmax," 2017.

[17] C. Wah *et al.*, "The Caltech-UCSD Birds-200-2011 Dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

[18] J. Krause *et al.*, "3d object representations for fine-grained categorization," in *2013 IEEE International Conference on Computer Vision Workshops*, 2013, pp. 554–561.

[19] M.-E. Nilsback *et al.*, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, 2008, pp. 722–729.

[20] B. Saleh *et al.*, "Large-scale classification of fine-art paintings: Learning the right metric on the right feature," *CoRR*, vol. abs/1505.00855, 2015. [Online]. Available: http://arxiv.org/abs/1505.00855

[21] M. Eitz *et al.*, "How do humans sketch objects?" *ACM Trans. Graph. (Proc. SIGGRAPH)*, vol. 31, no. 4, pp. 44:1–44:10, 2012.

[22] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," 2015.

[23] E. Park *et al.*, "Profit: A novel training method for sub-4-bit mobilenet models," 2020.

[24] 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, pp. 32.5.1–32.5.4.

[25] 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.

[26] P.-Y. Chen, X. Peng, and S. Yu, "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.