Wang F, Luo GJ, Sun GY *et al.* Area efficient pattern representation of binary neural networks on RRAM. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(5): 1155–1166 Sept. 2021. DOI 10.1007/s11390-021-0906-y

Area Efficient Pattern Representation of Binary Neural Networks on RRAM

Feng Wang[1], Guo-Jie Luo[1,*], *Member, CCF, ACM, IEEE*, Guang-Yu Sun[1], *Member, CCF, ACM, IEEE*
Yu-Hao Wang[2], Di-Min Niu[2], and Hong-Zhong Zheng[2]

[1] *Center for Energy-Efficient Computing and Applications, Peking University, Beijing 100871, China*
[2] *Pingtouge, Alibaba Group, Hangzhou 310052, China*

E-mail: {yzwangfeng, gluo, gsun}@pku.edu.cn; {yuhao.w, dimin.niu, hongzhong.zheng}@alibaba-inc.com

Received August 14, 2020; accepted August 26, 2021.

Abstract Resistive random access memory (RRAM) has been demonstrated to implement multiply-and-accumulate (MAC) operations using a highly parallel analog fashion, which dramatically accelerates the convolutional neural networks (CNNs). Since CNNs require considerable converters between analog crossbars and digital peripheral circuits, recent studies map the binary neural networks (BNNs) onto RRAM and binarize the weights to $\{+1, -1\}$. However, two mainstream representations for BNN weights introduce patterns of redundant 0s and 1s when dealing with negative weights. In this work, we reduce the area of redundant 0s and 1s by proposing a BNN weight representation framework based on the novel pattern representation and a corresponding architecture. First, we spilt the weight matrix into several small matrices by clustering adjacent columns together. Second, we extract 1s' patterns, i.e., the submatrices only containing 1s, from the small weight matrix, such that each final output can be represented by the sum of several patterns. Third, we map these patterns onto RRAM crossbars, including pattern computation crossbars (PCCs) and pattern accumulation crossbars (PACs). Finally, we compare the pattern representation with two mainstream representations and adopt the more area efficient one. The evaluation results demonstrate that our framework can save over 20% of crossbar area effectively, compared with two mainstream representations.

Keywords binary neural network (BNN), pattern, resistive random access memory (RRAM)

1 Introduction

Convolutional neural networks (CNNs)[1] have shown significant improvements in various applications, such as image classification and speech recognition. At the same time, in the convolutional layers and the fully-connected layers, CNNs contain a large number of memory accesses and multiply-and-accumulate (MAC) operations, which become the performance bottleneck. The emerging resistive random access memory (RRAM)[2] has been considered as a promising hardware to construct a CNN accelerator[3]. Previous studies[3–5] achieved about 1 000× throughput on RRAM compared with a CPU. On the one hand, it can perform MAC operations within the RRAM crossbars that store the weights, which saves most of the memory access overhead. On the other hand, the MAC operations are implemented in a highly parallel analog fashion. They set one vector as the input voltages and the other as conductance in the crossbar, thereby the output current represents their inner product according to Ohm's law. The degree of parallelism can easily reach the crossbar size.

However, the weights in traditional CNNs are typically represented by high-precision numbers, which pose a great challenge to the reliability of RRAM[4]. Also, the extra converters between the analog RRAM

Regular Paper

Special Section of 2020 CCF Integrated Circuit Design and Automation Conference

This work is partly supported by the National Key Research and Development Program of China under Grant No. 2020AAA0130400, Beijing Municipal Science and Technology Program of China under Grant No. Z201100004220007, the National Natural Science Foundation of China under Grant No. 62090021, Beijing Academy of Artificial Intelligence (BAAI), and Alibaba Innovative Research (AIR) Program.

*Corresponding Author

crossbars and the digital peripheral circuits take up most of the area and energy consumption[5]. Fortunately, recent studies on machine learning propose binary neural networks (BNNs)[6,7] and verify their effectiveness in various benchmarks. BNNs truncate the values of weights and activation into binary values $\{+1,-1\}$ during the inference stage. Thus, when mapping onto the RRAM crossbar[8–11], each RRAM cell only needs to store one bit. And low-precision converters, or even one-bit comparators, can replace high-precision converters. BNNs can be accelerated by RRAM crossbars with little negative effect mentioned above.

It is still challenging about how to represent BNN weights on RRAM efficiently in area. RRAM can only store non-negative data by conductance, thereby the negative weights cannot be represented on RRAM crossbars directly. To cope with this problem, previous studies propose two mainstream representations, the pos-neg representation[8,9] and the XNOR representation[10,11]. Although both of them successfully solve the negative weight problem, they store redundant 0s in 50% of the RRAM crossbars, regardless of the original weight distribution. Besides, we note that these two representations only contain 0s and 1s, thereby different columns may share some 1s' patterns, i.e., they all have 1s in the same few rows. It wastes area to store and compute multiple copies of a pattern.

In this work, we propose an area efficient weight representation framework based on the novel pattern representation for BNN weights to reduce the redundant 0s' and 1s' patterns in RRAM as many as possible. Our contributions are as follows.

- We design an architecture containing pattern computation crossbars (PCCs) and pattern accumulation crossbars for our novel pattern representation.
- We propose a weight representation framework for this architecture and formulate the pattern extraction and the pattern mapping problems.
- We prove that both problems are NP-hard and design heuristic algorithms to solve them.
- We experimentally evaluate our framework to show its area saving compared with mainstream representations.

2 Background and Motivation

2.1 BNN

In traditional CNNs, the convolutional layers and the fully-connected layers contribute a lot to the total run time and energy consumption due to the high-precision MAC operations, which becomes the performance bottleneck. To address this problem, BNNs[6,7] train and truncate the weights and activations to values $\{+1,-1\}$ without losing much accuracy. In this situation, the MAC operations over $\{+1,-1\}$ during the inference stage can also be converted to the XNOR operations over $\{1,0\}$ followed by the bit-counting operations.

2.2 RRAM-Based BNN Accelerator

Single-level-cell (SLC) RRAM[2] is a two-terminal device with two different states according to its resistance. The high resistance represents logical "0", and the low resistance represents logical "1". The RRAM crossbar can be configured for parallel MAC operations in an analog fashion. When we convert one input vector to the voltage signals applied to each row and store the other one as the conductance of an RRAM column, the output current of this column is the inner product between the voltage vector and the conductance vector. The parallelism can reach the crossbar width with negligible extra hardware. Fig.1 shows a $row_C \times col_C$ RRAM crossbar. The MAC operations in Fig.1 can be expressed as

$$I_k = \sum_{j=1}^{row_C} V_j G_{j,k}, \quad k = 1, 2, ..., col_C.$$

This feature can accelerate the MAC operations in the fully-connected and the convolutional layers of NNs. The weight matrix of the fully-connected layers can be directly mapped onto the RRAM crossbars. For the convolutional layers, one kernel is typically expanded to a one-dimensional vector and stored in an RRAM column. Different kernels of the same layer are aligned in the same crossbar and share the input feature maps. If the weight matrix (or the kernels of a layer) size exceeds the RRAM crossbar size, we usually split it into multiple crossbars and accumulate their partial sums outside.

2.3 Mainstream Representations of Negative Weights

RRAM can only store non-negative values. There are two mainstream representations for the negative weights in BNNs, as shown in Fig.2. The pos-neg representation[8] splits the positive and the negative weights into different crossbars and complements the vacant cells with 0s. It requires two crossbars as large

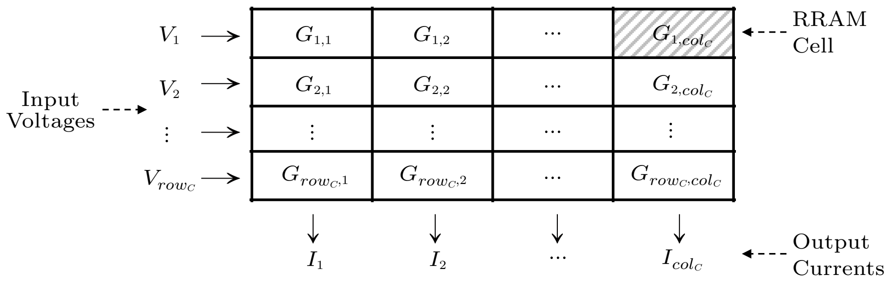


Fig. 1. Parallel MAC operations in an RRAM crossbar. In this paper, we ignore the hardware details of RRAM and represent an RRAM cell with a cell. We bold the borders of hardware cells to distinguish them from the weight matrices in the software level.

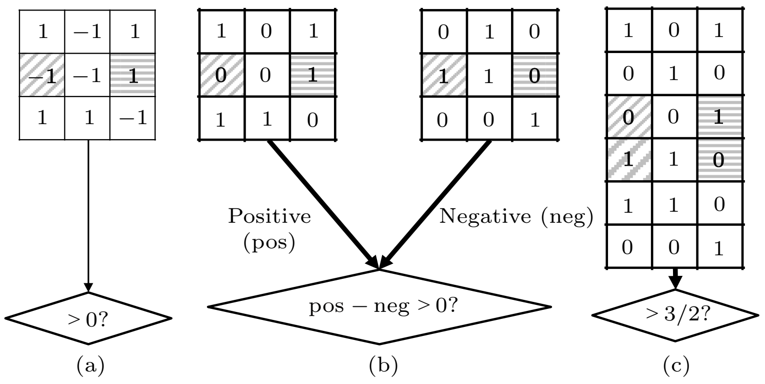


Fig. 2. Two representations of negative weights. The shaded −1 and 1 in the original weight matrix are mapped onto the shaded RRAM cells, respectively. (a) Weight. (b) Pos-neg representation. (c) XNOR representation.

as the weight matrix. This approach works for both traditional CNNs[3] and BNNs[9].

The XNOR representation[10] is proposed for BNNs only. It stores two bits $\langle 0, 1\rangle$ and $\langle 1, 0\rangle$ for -1 and 1 in the weights and inputs, respectively, to implement the XNOR operation. If the input equals the weight, their product is $(1, 0) \cdot (1, 0)^{\mathrm{T}} = (0, 1) \cdot (0, 1)^{\mathrm{T}} = 1$. Otherwise the product is 0, which is consistent with the XNOR operation. The judgement condition outside the crossbar has to be modified, and the detailed proof can be found in [11]. This representation requires a crossbar that is twice as large as the weight matrix.

2.4 Motivation

We observe that these two representations have a large number of redundant 0s and 1s, which wastes the RRAM crossbar area. First, both representations introduce 0s in exactly 50% of the crossbars so that they require twice the area of the original weight matrix. It is worth noting that the extra 0s come from the representations but not the 1s' distribution in the weight matrix or the network sparsity. Second, the RRAM crossbars only contain 1s and 0s, thereby different columns may have a high similarity. We can extract the same rows of these columns as patterns and store only one copy.

It is challenging to reduce the redundant 0s and 1s. SNrram[12] is an RRAM architecture that exploits the network sparsity by discarding the whole empty columns in the weight matrix, but it cannot deal with the BNN representation problem where 0s and 1s alternate. Although previous studies[13] propose some pattern extraction methods for other hardware targets, these methods cannot be applied to RRAM directly. For example, Chi and Jiang[13] synthesized BNNs on FPGAs by extracting same and opposite patterns, but RRAM crossbars do not support the subtraction operations required by opposite patterns. Moreover, it is difficult to keep the high parallelism if we only store 1s' patterns. Different patterns correspond to different inputs, but the parallelism in RRAM crossbars requires the alignment of different columns.

3 Architecture

We design an architecture for our novel pattern representation, which inherits most aspects of standard RRAM-based accelerator. At the top view, the architecture is hierarchical, which contains multiple processing elements (PEs) for matrix-vector product operations, as well as pooling units and sigmoid units for the other operations.

Fig.3 shows PE details of PEs. Each PE processes one BNN layer. Crossbars in a PE are divided into two parts, PCCs and PACs. For a given input feature map, we first send it to all PCCs according to their configurations. Once the patterns are computed, their values are converted to the voltages as inputs, and the output feature map is achieved in the analog fashion.

PCCs are used for computing partial sums of patterns. We do not store the original weight matrix in the RRAM crossbars. Instead, we extract patterns from it and only store patterns in the crossbars. A pattern appears in at least one RRAM crossbar according to its length and occupies one column in each crossbar. Our pattern mapping follows two principles. First, patterns in the same crossbar have to be aligned such that they can be computed in parallel. Second, patterns in different crossbars should have no sharing inputs such that we do not need to copy inputs. The number of PCCs depends on both the pattern number and the pattern length.

PACs are used for accumulating partial sums of PCCs. We record the indices (relationship) between the output feature maps and the patterns in the PACs. The index of one output occupies one column. The intersection of a pattern and an output is set to 1 if they are related. If a pattern appears in multiple PCCs, each part of the separated patterns needs to occupy a PAC row. We arrange the patterns in the PACs in the same order in PCCs to reduce the routing cost. If an output is related to more than n patterns, we accumulate these patterns with an adder tree. Assuming that $n = 128$, a two-level adder tree can accommodate 128^2 patterns, which meets the requirement of most BNNs. The number of PACs depends on both the pattern number and the relation between patterns and original outputs.

After mapping, the orders of inputs, outputs and patterns will change, thereby we carefully reorder the crossbar inputs to match the output order. PCCs except for the first layer are reordered according to PACs of the previous layer. PACs are reordered according to PCCs of the same layer. For example, the order of PCC-1-1 and PCC-1-2 outputs is $\{p_1, p_3, p_2\}$, thereby the order of PAC-1-1 inputs is also $\{p_1, p_3, p_2\}$. The order of PAC-1-1 outputs is $\{o_4, o_2, o_3, o_1\}$, thereby the order of PCC-2-1 inputs is also $\{i_4, i_2, i_3, i_1\}$. Therefore, we do not need extra hardware to store and fetch indices.

We can further improve the area efficiency of our PE design with two methods. First, our design can fully utilize one dimension in both PCCs and PACs. Therefore, we can easily map the patterns or the index matrices of independent weight matrices, especially matrices of different layers, into the same crossbar to fully utilize a crossbar. For example, PCC-1-2 and PCC-2-1 can be merged into a 4×4 crossbar. Second, a crossbar can be configured to act as either a PCC or a PAC with negligible extra hardware. Although different BNNs require different ratios of PCCs to PACs, we can always fully utilize all crossbars rather than waste some PCCs or PACs.

4 Framework and Problem Formulation

This section proposes our weight representation framework and formulates the pattern extraction and mapping problems.

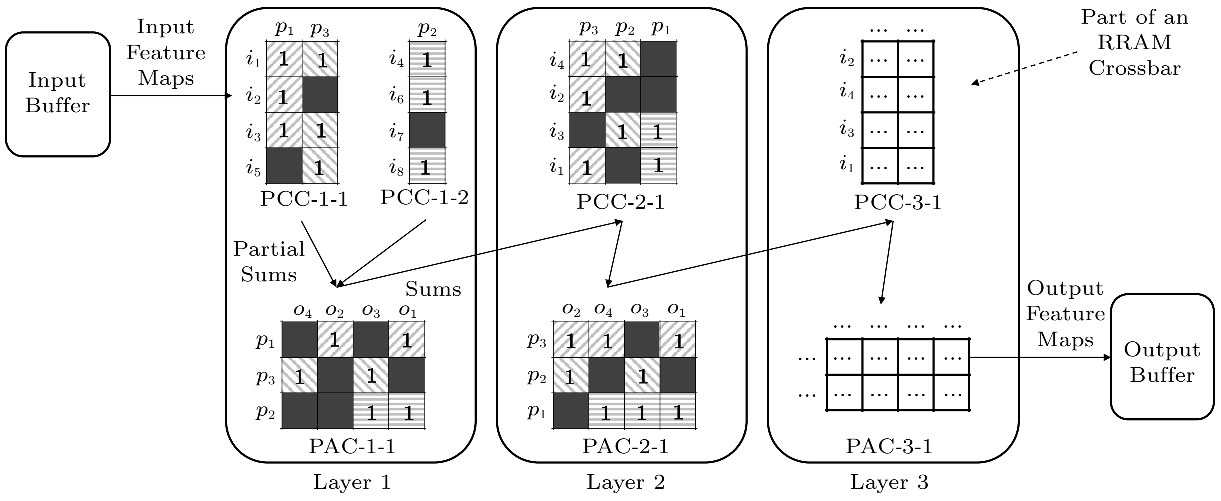


Fig.3. Processing elements (PEs) for pattern representation. The matrix-vector product operations are performed in PEs.

4.1 Framework Overview

Fig.4(a) depicts our four-step weight representation framework for the weight matrix of a layer that is already in the pos-neg or the XNOR representation.

Column Split. If the matrix is wider than the crossbar, we split it by column to several small matrices that are as wide as the crossbar. Two direct representations also require this step to ensure that the small matrix can fill in a crossbar.

Pattern Extraction. We extract patterns, i.e., submatrices, from each small weight matrix. 0s do not contribute to the results of MAC operations, thereby we extract patterns that only contain 1s. As shown in Fig.4(b), the weight matrix is exactly covered by three 3×2 patterns, and each final output can be represented by one or two patterns.

Pattern Mapping (PCC Part). The weight matrix rows are split into different PCCs according to the crossbar height. A few 0s are reintroduced due to the alignment requirement of parallelism. In Fig.4(b), three patterns extracted from the weight matrix are computed in two PCCs. Input rows $\{i_1, i_2, i_3, i_5\}$ and $\{i_4, i_6, i_7, i_8\}$ are separated into two PCCs such that each pattern only appears in one PCC and occupies a column.

Pattern Mapping (PAC Part). In Fig.4(b), PAC stores the relation matrices among four outputs and three patterns. For example, o_1 is represented by p_1 and p_2, thereby the intersections (p_1, o_1) and (p_2, o_1) are 1 in PAC. All of the three patterns occupy a row in PAC. Patterns are arranged in $\langle p_1, p_3, p_2 \rangle$ in both PCCs and PAC.

Area Comparison. Our pattern representation does not work in a few cases, e.g., the staircase 1s' distribution in Fig.5, which can be proved by enumerating all pattern extraction and mapping possibilities. As a result, we compare the area of pattern representation, i.e., the total area of PCCs ($area_{\text{PCC}}$) and PACs ($area_{\text{PAC}}$), with that of two direct representations ($area_{\text{direct}}$) to find a more area efficient representation:

$$area_{\text{pattern}} = \min\{area_{\text{PCC}} + area_{\text{PAC}}, area_{\text{direct}}\}.$$

4.2 Problem Formulation

The pattern extraction and mapping are two key steps in our framework. We give a formal definition to the pattern and formulate these two problems. Both problems try to minimize the total area of the pattern representation.

Pattern. For a binary weight matrix $\boldsymbol{W} = \langle row_W, col_W \rangle$, a pattern $\boldsymbol{p} = \langle row_p, col_p \rangle$ is a submatrix of $\boldsymbol{W}$ that only contains 1s. In other words, $row_p \subseteq row_W$, $col_p \subseteq col_W$, and $p_{xy} = 1$ always hold.

Pattern Extraction Problem. Given a binary weight matrix $\boldsymbol{W}$, we find n disjoint patterns $p_1, p_2, ..., p_n$ to exactly cover all 1s in $\boldsymbol{W}$ with the total area lower bound f_{ext} minimized.

$$\begin{aligned} \text{minimize} \quad f_{\text{ext}} &= \sum_{i=1}^{n} |row_{p_i}| + n \times |col_W| \\ &\leqslant area_{\text{PCC}} + area_{\text{PAC}}. \end{aligned}$$

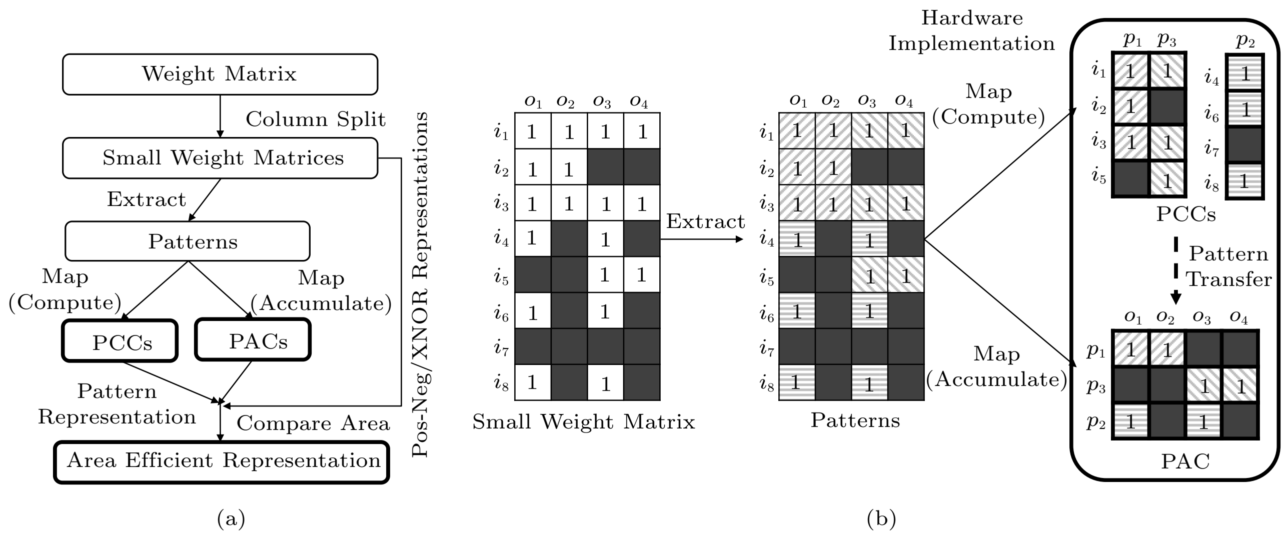


Fig.4. Framework based on the pattern representation. (a) Framework overview. (b) Example of the pattern extraction and mapping steps. Three patterns are labeled in different shadings, and the black cells are filled with 0s. We assume that the crossbar size is 4×4. The unused PCC columns and PAC rows are not shown. The area saving is $\frac{32-12-12}{32} = 25\%$.

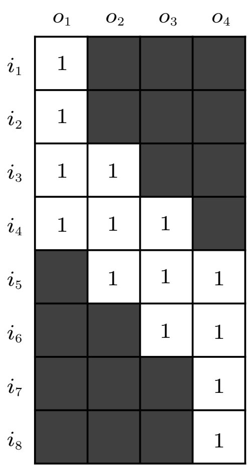


Fig.5. A staircase 1s' distribution where the direct representations are more area-efficient.

The exact values of $area_{\text{PCC}}$ and $area_{\text{PAC}}$ depend on the pattern mapping step, thereby we estimate them with lower bounds. $area_{\text{PCC}}$ is not less than the total pattern height $\sum_{i=1}^{n}|row_{p_i}|$, and the equal sign is obtained when no 0s are introduced later. The total pattern number in the PAC is not less than n, thereby $area_{\text{PAC}} \geqslant n \times |col_W|$. The equal sign is obtained when no patterns are split into multiple PCCs.

Pattern Mapping Problem. Given n patterns $p_1, p_2, ..., p_n$, we split the weight matrix rows into $\lceil\frac{|row_W|}{|row_C|}\rceil$ disjoint subsets $s_1, s_2, ..., s_{\lceil\frac{|row_W|}{|row_C|}\rceil}$ to exactly cover row_W, and the rows in a subset are mapped to the same crossbar. f_{map} is minimized. row_C is the number of rows in a crossbar.

$$\text{minimize} \quad f_{\text{map}} = \sum_{i=1}^{n} \sum_{j=1}^{\lceil\frac{|row_W|}{|row_C|}\rceil} (row_{p_i} \cap s_j \neq \emptyset) \propto area_{\text{PCC}} + area_{\text{PAC}}.$$

f_{map} is proportional to $area_{\text{PCC}}$ and $area_{\text{PAC}}$, and we care about how many parts of patterns are separated in our pattern representation by minimizing f_{map}. As explained before, if $row_{p_i} \cap s_j \neq \emptyset$, i.e., pattern p_i contains a row in s_j, part of this pattern will be mapped onto this PCC and occupy a column with $|row_C|$ cells. Thus, $area_{\text{PCC}} = |row_C| \times f_{\text{map}}$. Also, each part of the pattern will occupy a row in the PAC with $|col_C|$ cells, thereby $area_{\text{PAC}} \geqslant |col_C| \times f_{\text{map}}$. $area_{\text{PAC}}$ is a little larger when a PAC tree is needed.

According to our formulation, we can reduce both two optimization problems to the exact cover problem [14], a well-known NPC problem, and thus they are also NP-hard. We propose the heuristic algorithms to solve these problems next.

5 Algorithm Design

This section introduces our algorithms for the first three steps in the weight representation framework.

5.1 Column Split

To prevent our pattern representation from being worse than two direct representations, e.g., in the staircase distribution in Fig.5, we cluster "adjacent" weight columns in a small weight matrix by the K-means algorithm. We denote the set of 1s rows in a column col_i as row_i. We define the distance between two columns row_i and row_j ($|row_i| \leqslant |row_j|$) as f_{ext} of the weight matrix that consists of these two columns ($|col_W| = 2$). This two-column matrix can be covered by no more than three patterns, one pattern $\langle row_i \cap row_j, \{col_i, col_j\}\rangle$ for the common rows, and two patterns $\langle row_i - row_j, \{col_i\}\rangle$, $\langle row_j - row_i, \{col_j\}\rangle$ for the rest rows. Based on this observation, the distance between col_i and col_j can be computed as follows:

$$distance(col_i, col_j) = f_{\text{ext}} = \begin{cases} |row_i| + 2, & \text{if } row_i = row_j, \\ |row_j| + 4, & \text{if } row_i \subset row_j, \\ |row_i| + |row_j| + 4, & \text{if } row_i \cap row_j = \emptyset, \\ |row_i - row_j| + |row_j| + 6, & \text{otherwise.} \end{cases}$$

Since $distance(col_i, col_j)$ is always smaller under the first two conditions, the K-means algorithm prefers to cluster columns with equivalence or inclusive relationships into one small weight matrix.

5.2 Pattern Extraction

We construct the initial solution by iteratively matching the column with the least 1s with all of the other columns. For example, the optimal pattern extraction solution of Fig.4(b) can be obtained during initialization. At the beginning, o_2 has the least 1s. We match it with o_1 that also has 1s in the first three rows $\{i_1, i_2, i_3\}$, and we get p_1. Then, the rest of 1s in o_1 become the least, thereby we match o_1 with the other columns to generate p_2. Finally, the remaining 1s constitute p_3.

Each iteration during initialization can reduce at least one column in the weight matrix, and thus the initial solution contains no more than $|col_W|$ patterns. This algorithm performs well for unbalanced 1s. With the execution of the algorithm, columns with lots of 1s are gradually matched and become new columns with the least 1s. In the worst case, no columns have the inclusive relationship, and the algorithm generates a pattern for each column.

We design a simulated annealing (SA) algorithm to further optimize the patterns. In each iteration, we select two random patterns p_i and p_j ($|row_{p_i}| \leqslant |row_{p_j}|$ or $|col_{p_i}| \leqslant |col_{p_j}|$) that share rows or columns and generate a new pattern for their common rows or columns. Since p_i and p_j are disjoint, they cannot share rows and columns simultaneously. The rest parts of p_i and p_j are still reserved in the solution if existed. A better transition that can decrease f_{ext} is always accepted, while a worse one is accepted with a certain probability determined by the current temperature and Δf_{ext}.

There are six different situations in the transition function according to the relative positional relationship of p_i and p_j, as listed in Fig.6. The patterns other than p_i and p_j remain unchanged, thereby Δf_{ext} only depends on p_i, p_j, and new patterns. ① and ③ can improve f_{ext} by merging two smaller patterns to a larger one and reducing the pattern number. ② can improve f_{ext} by reducing the total pattern height. ④–⑥ may worsen f_{ext}, but they can reduce the size of p_i and p_j and increase the likelihood of subsequent pattern merging, which are effective for large patterns. Fig.7 gives an example where each column constitutes a pattern after initialization. The SA algorithm first splits patterns in o_1 and o_2 into three small patterns using transition ⑤ and then merges these small patterns with the other patterns using transitions ② and ①. The total pattern height drops by half after the SA algorithm.

5.3 Pattern Mapping

We design a greedy algorithm to minimize f_{map}, as shown in Algorithm 1. Each iteration from line 3 to line 8 selects a pattern with the least rows and adds its rows to S. Corresponding to the actual pattern mapping procedure, in each iteration, we map the pattern with the least rows that are not mapped onto PCC. The mapped rows of this pattern are placed in the corresponding PCCs, while the other rows are mapped in the PCCs that are not yet full. A new PCC is added when all of the existing PCCs are fully allocated. This algorithm prefers to map similar patterns sharing most of the rows in the same PCCs and try not to split them.

For example, $f_{\text{map}} = 3$ in Fig.4(b). p_1 is one of the shortest patterns, and we map it onto the left PCC first. Then, p_3 only has one unmapped row i_5. The fourth row in the left PCC is not allocated, thereby we map i_5 to this position. Finally, the left PCC is fully utilized, and we map p_2 to the right PCC. But if we map p_2 after p_1, i_1–i_4 will stay in a PCC. p_2 and p_3 will appear in two PCCs, and f_{map} becomes 5.

6 Experimental Evaluation

We experimentally evaluate our weight representation framework on two datasets MNIST and CIFAR-10. We adopt the BNN training algorithm in XNOR-Net[6] and slightly modify the network model to make it deeper as in [13]. The BNN model for MNIST contains seven convolutional or fully-connected layers, and its accuracy is 99.2%. The BNN model for CIFAR-10

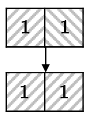
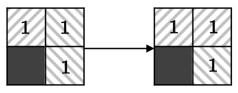
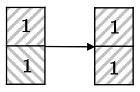
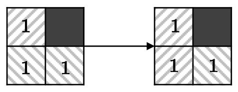
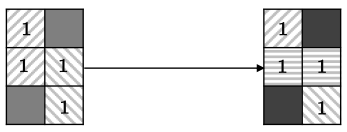
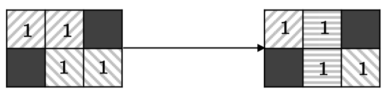
Position	① $row_{p_i} = row_{p_j}$	② $row_{p_i} \subset row_{p_j}$	③ $col_{p_i} = col_{p_j}$	④ $col_{p_i} \subset col_{p_j}$
Transition	$\langle row_{p_i}, col_{p_i} \cup col_{p_j} \rangle$	$\langle row_{p_i}, col_{p_i} \cup col_{p_j} \rangle$, $\langle row_{p_j} - row_{p_i}, col_{p_j} \rangle$	$\langle row_{p_i} \cup row_{p_j}, col_{p_i} \rangle$	$\langle row_{p_i} \cup row_{p_j}, col_{p_i} \rangle$, $\langle row_{p_j}, col_{p_j} - col_{p_i} \rangle$
Δf_{ext}	$-\lvert row_{p_i}\rvert - \lvert col_W\rvert < 0$	$-\lvert row_{p_i}\rvert < 0$	$-\lvert col_W\rvert < 0$	$\lvert row_{p_j}\rvert > 0$
Example				
Position	⑤ $row_{p_i} \cap row_{p_j} = row_{p_k} \neq \emptyset$		⑥ $col_{p_i} \cap col_{p_j} = col_{p_k} \neq \emptyset$	
Transition	$\langle row_{p_k}, col_{p_i} \cup col_{p_j} \rangle$, $\langle row_{p_i} - row_{p_k}, col_{p_i} \rangle$, $\langle row_{p_j} - row_{p_k}, col_{p_j} \rangle$		$\langle row_{p_i} \cup row_{p_j}, col_{p_k} \rangle$, $\langle row_{p_i}, col_{p_i} - col_{p_k} \rangle$, $\langle row_{p_j}, col_{p_j} - col_{p_k} \rangle$	
Δf_{ext}	$-\lvert row_{p_k}\rvert + \lvert col_W\rvert$? 0		$\lvert row_{p_i}\rvert + \lvert row_{p_j}\rvert + \lvert col_W\rvert > 0$	
Example				

Fig.6. State transition function of the simulated annealing algorithm, which generates a new pattern for the common rows or columns of p_i and p_j and reserves the rest parts of p_i and p_j. ①–③ can improve f_{ext}, while ④–⑥ may worsen f_{ext}.

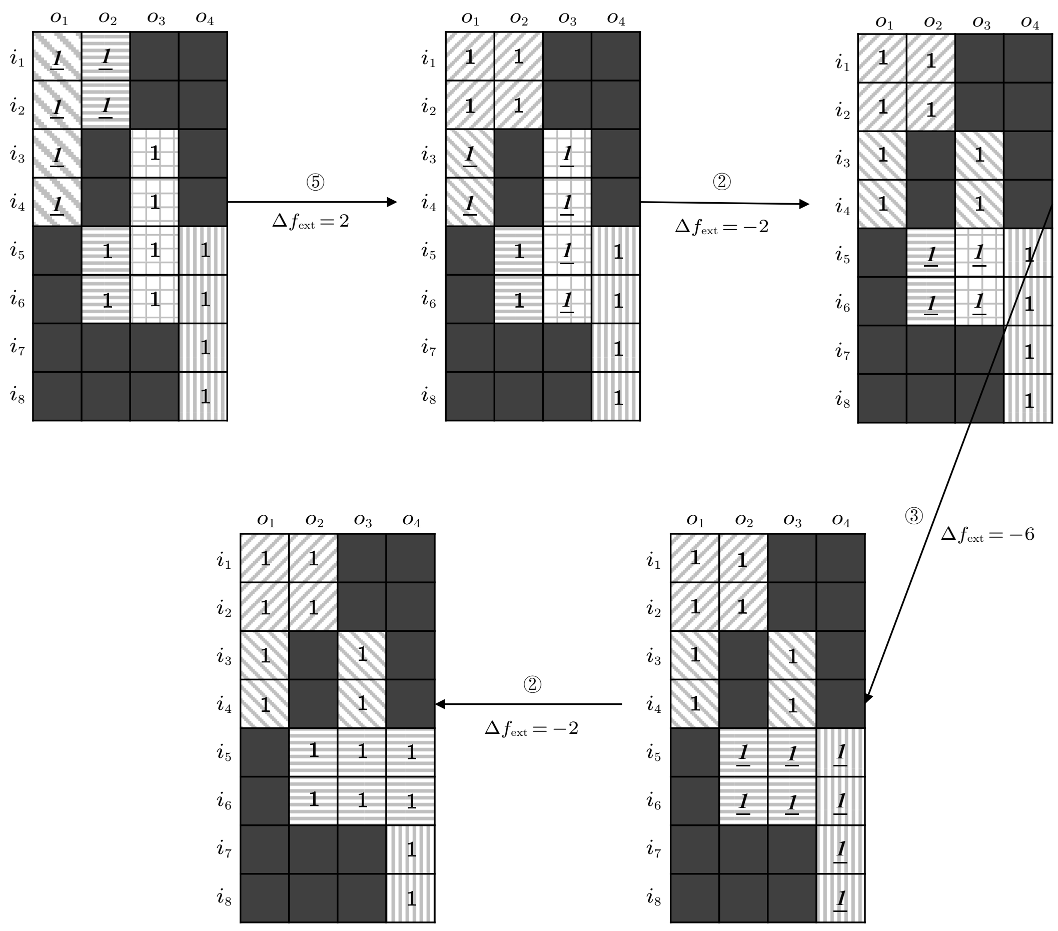


Fig.7. Example of the simulated annealing algorithm. The two selected patterns in the current step are italic.

contains 10 convolutional or fully-connected layers, and its accuracy is 86.3%.

Algorithm 1 . Greedy Pattern Mapping Algorithm

```
Input: $n$ patterns $p_1, p_2, ..., p_n$
Output: $\lceil \frac{|row_W|}{|row_C|} \rceil$ subsets $s_1, s_2, ..., s_{\lceil \frac{|row_W|}{|row_C|} \rceil}$
 1: $S = \emptyset$                    ▷ an ordered set storing $\langle s_1, s_2, ... \rangle$
 2: for $i = 1$ to $n$ do
 3:     $index = \arg\min_x |row_{p_x}|$
 4:     $S = \langle S, row_{p_{index}} \rangle$
 5:     Delete $p_{index}$
 6:     for $j = 1$ to $n$ do
 7:         $row_{p_j} = row_{p_j} - row_{p_{index}}$
 8:     end for
 9: end for
10: Split $S$ into $s_1, s_2, ..., s_{\lceil \frac{|row_W|}{|row_C|} \rceil}$
```

6.1 Area Evaluation

Table 1 lists the area saving of our framework based on two direct representations (X + pattern) respectively. $P+$ ($X+$) in the last column means pos-neg representation (XNOR representation) is better. We set the crossbar size to 128×128, the optimal design option validated in [10]. 0% area saving in a few cases indicates that we adopt the direct representations, which are more area-efficient than the pattern representation. The height and the width belong to the original weight matrix. The area of two direct representations is $\sum area_{\text{direct}} = 2 \times row_W \times (\sum col_W)$.

Overall, our representation framework saves more than 20% of area for both direct representations and both networks. The area saving of a layer varies from 0% to 30%. Our pattern representation outperforms direct representations in more than 70% of cases, and its area saving exceeds 20% for half of the cases, especially for larger weight matrices that are much wider and higher than the crossbar. Our framework based on two direct representations achieves a close area saving. Since power is proportional to the number of crossbars, we derive that our representation can also save more than 20% of power on RRAM crossbars.

Table 1. Area Saving of Our Weight Representation Framework

Dataset	Layer	row_W	$\sum col_W$	$\sum area_{\text{direct}}$	pos-neg + Pattern		XNOR + Pattern		Better
					$\sum area_{\text{pattern}}$	Saving (%)	$\sum area_{\text{pattern}}$	Saving (%)	
MNIST	1	500	50	50 000	38 100	23.80	50 000	0.00	P+
	2	800	256	409 600	275 072	32.84	294 400	28.13	P+
	3	256	256	131 072	113 152	13.67	99 840	23.83	X+
	4	256	256	131 072	111 616	14.84	102 400	21.88	X+
	5	256	256	131 072	120 832	7.81	103 680	20.90	X+
	6	256	256	131 072	105 472	19.53	99 328	24.22	X+
	7	256	10	5 120	5 120	0.00	5 120	0.00	Tie
	All	–	–	989 008	769 364	22.21	754 768	23.68	X+
CIFAR-10	1	180	96	34 560	30 848	10.74	30 208	12.59	X+
	2	864	48	82 844	59 136	28.70	82 944	0.00	P+
	3	432	256	221 184	161 920	26.79	168 960	23.61	P+
	4	2 304	384	1 769 472	1 285 120	27.37	1 356 800	23.32	P+
	5	3 456	1 024	3 538 944	5 263 360	25.64	5 299 200	25.13	P+
	6	1 024	64	131 072	102 912	21.48	131 072	0.00	P+
	7	64	128	16 384	16 384	0.00	16 384	0.00	Tie
	8	128	128	32 768	28 288	13.67	24 320	25.78	X+
	9	128	128	32 768	27 008	17.58	25 600	21.88	X+
	10	128	10	2 560	2 560	0.00	2 560	0.00	Tie
	All	–	–	9 401 600	6 977 536	25.78	7 138 048	24.08	P+

6.2 Effect of Feature Map Size

The pattern representation cannot improve much for two types of small matrices. First, if the matrix width is much smaller than the crossbar width, we can hardly extract large patterns with multiple columns to reduce the redundant 1s. Thus, the direct representations are usually better. This usually happens in the first convolutional layer, e.g., (MNIST, 1) of the XNOR representation, and the last fully-connected layer, e.g., (MNIST, 7) and (CIFAR-10, 10) of both representations. Second, if the matrix height is too small, our algorithm prefers to extract extremely short patterns. Since each pattern needs to occupy a whole column in a PCC regardless of its height, these extremely short patterns will introduce lots of redundant 0s in PCCs. If the matrix height is equal to or less than the crossbar height, e.g., (CIFAR-10, 7) of both representations, the direct representations are better. If the matrix is a little higher than the crossbar, e.g., (MNIST, 3) of the pos-neg representation, our framework achieves a low area saving (5%–20%).

6.3 Effect of Direction Representation Type

Despite of the same area, two direct representations have different shapes. The pos-neg one doubles the weight matrix width, while the XNOR one doubles its height, which leads to different area saving of our weight representation framework in several layers. Given the analysis above, pos-neg + pattern performs better for narrow matrices, e.g., (MNIST, 1), while XNOR + pattern performs better for short matrices, e.g., (CIFAR-10, 8).

6.4 Effect of Crossbar Size

Fig.8 shows the area saving of our pattern representation under different crossbar sizes. Here we take four representative layers (two large layers and two small layers) as an example. For two large layers (CIFAR-10, 4) and (CIFAR-10, 5), 128 is the best crossbar size. When the crossbar size reaches 256, we cannot extract such long patterns that too many 0s are reintroduced. When the crossbar size falls to 64, outputs are related to more patterns, and thus we need more PAC trees to record indices. For two small layers (CIFAR-10, 7) and (CIFAR-10, 8), our representation only works when the crossbar size is smaller than the layer size. In summary, 128 is the best crossbar size for two BNNs used in the experimental evaluation.

6.5 Throughput Evaluation

We evaluate the throughput of our representation, as shown in Fig.9. The hardware parameters are adopted from [8]. We can see that our representation achieves a little throughput improvement on two BNNs, which mainly comes from area saving. Traditional representations usually require multi-level adder trees to accumulate partial sums. On contrary, our representation only needs to accumulate partial sums of patterns. One-level PACs are usually enough.

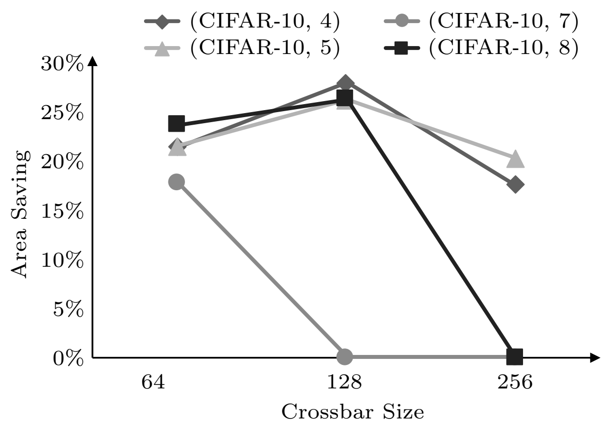


Fig.8. Area saving under different crossbar sizes.

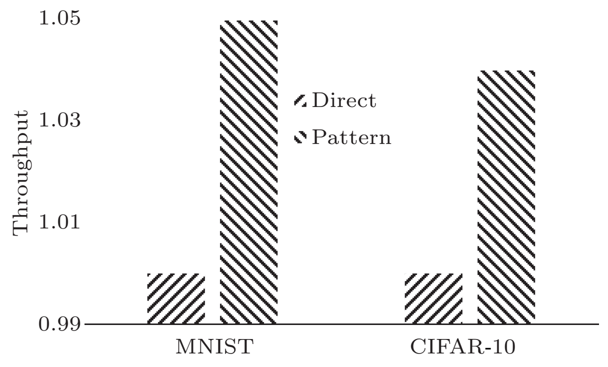


Fig.9. Throughput comparison. Throughput of the better direct representation is normalized to 1.

6.6 Comparison on Another BNN Type

In a normal BNN, the weights may also be represented by $\{+1, 0\}$. This can be directly mapped onto RRAM without differential crossbars. We also evaluate our weight representation in this situation, as shown in Table 2. We can see that our representation still performs well and saves about 30% area on average. The area saving rate is a little higher than that in Subsection 6.1 due to unbalanced 1s and 0s. Since 1s or 0s occupy most of the weight matrices, the similarity between different columns increases, and it is easier to mine patterns. In the extreme case of all 1s or 0s in the matrix, one pattern is enough. Recent studies[15] propose some other mapping methods, which are orthogonal to our work. We can further compress the crossbars with only 0s and 1s using our pattern representation.

Table 2. Area Saving of Our Weight Representation Framework on Another BNN Type

Dataset	Layer	Area Saving (%)
MNIST	1	28.32
	2	37.25
	3	31.35
	4	24.41
	5	21.48
	6	25.88
	7	0.00
	All	30.53
CIFAR-10	1	12.96
	2	29.01
	3	30.27
	4	34.64
	5	27.38
	6	23.44
	7	0.00
	8	29.30
	9	25.39
	10	0.00
	All	28.66

7 Related Work

Previous studies[3,16,17] accelerate the convolutional layers and the fully-connected layers of CNNs with analog MAC operations in RRAM crossbars. They also design corresponding data encoding schemes, peripheral circuits, software and hardware interfaces to improve the performance of the entire architecture.

Three improvements are made in subsequent studies. First, due to the limited accuracy of traditional multi-bit RRAM crossbars, [8,10] use single-bit RRAM crossbars to implement BNNs, and [18] further proposes quantization algorithms to implement configurable multi-precision networks. Second, it brings redundant data transfer and storage by directly expanding the convolution operation into a MAC operation. In view of this, [19] proposes a new data mapping scheme to reduce data transfer overhead, and [12] fully exploits the network sparsity to reduce unnecessary storage. Third, RRAM crossbars cannot support NN training. [20] implements back propagation and weight update in RRAM crossbars, which greatly reduces the energy consumption required in NN training. Fourth, it cannot deal with different crossbar sizes. To solve this problem, [21] proposes an overlapped mapping method (OMM) and mixed-size crossbar-based RRAM CNN accelerator (MISCA) to improve area utilization. [15] presents a novel methodology for mapping BNN operations on RRAM crossbars of arbitrary sizes.

Our pattern representation improves the direct weight matrix representations of the convolutional layers and the fully-connected layers in single-bit RRAM crossbars. It is orthogonal to the four improvements mentioned before. That is to say, first, we can further improve the area efficiency of previous studies with our representation. Second, our representation makes an equivalent conversion to the direct representations. It does not change the outputs of these two types of layers and has negligible effect on the network accuracy.

8 Conclusions

In this work, we proposed an area efficient weight representation framework for BNNs on RRAM. Unlike two mainstream representations that directly map the weight matrix onto the crossbars, our pattern representation extracts patterns from the original weight matrix and then maps patterns onto the crossbars to reduce the redundant 0s and 1s as many as possible. Experimental evaluation showed that our framework saves more than 20% of the crossbar area compared with the pos-neg representation and the XNOR representation.

The pattern representation currently only supports BNNs. However, there is also some redundancy in the weight matrices of CNNs. Therefore, in the future, we would like to apply the novel representation to general CNNs and multi-level RRAM cells.

References

[1] Hinton G, Deng L, Yu D *et al.* Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process Mag.*, 2012, 29(6): 82-97. DOI: 10.1109/MSP.2012.2205597.

[2] Akinaga H, Shima H. Resistive random access memory (ReRAM) based on metal oxides. *Proc. IEEE*, 2010, 98(12): 2237-2251. DOI: 10.1109/JPROC.2010.2070830.

[3] Chi P, Li S, Xu C *et al.* PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *Proc. the 43rd International Symposium on Computer Architecture*, Jun. 2016, pp.27-39. DOI: 10.1109/ISCA.2016.13.

[4] Chen L, Li J, Chen Y *et al.* Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2017, pp.19-24. DOI: 10.23919/DATE.2017.7926952.

[5] Liu C, Yan B, Yang C *et al.* A spiking neuromorphic design with resistive crossbar. In *Proc. the 52nd Design Automation Conference*, Jun. 2015. DOI: 10.1145/2744769.2744783.

[6] Rastegari M, Ordonez V, Redmon J, Farhadi A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proc. the 14th European Conference on Computer Vision*, Oct. 2016, pp.525-542. DOI: 10.1007/978-3-319-46493-0_32.

[7] Alemdar H, Leroy V, Prost-Boucle A, Pétrot F. Ternary neural networks for resource-efficient AI applications. In *Proc. the International Joint Conference on Neural Networks*, May 2017, pp.2547-2554. DOI: 10.1109/IJCNN.2017.7966166.

[8] Tang T, Xia L, Li B, Wang Y, Yang H. Binary convolutional neural network on RRAM. In *Proc. the 22nd Asia and South Pacific Design Automation Conference*, Jan. 2017, pp.782-787. DOI: 10.1109/ASPDAC.2017.7858419.

[9] Ni L, Liu Z, Song W *et al.* An energy-efficient and high-throughput bitwise CNN on sneak-path-free digital ReRAM crossbar. In *Proc. the 2017 IEEE/ACM International Symposium on Low Power Electronics and Design*, Jul. 2017. DOI: 10.1109/ISLPED.2017.8009177.

[10] Sun X, Yin S, Peng X, Liu R, Seo J, Yu S. XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks. In *Proc. the Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2018, pp.1423-1428. DOI: 10.23919/DATE.2018.8342235.

[11] Sun X, Peng X, Chen P Y, Liu R, Seo J, Yu S. Fully parallel RRAM synaptic array for implementing binary neural network with (+1, -1) weights and (+1, 0) neurons. In *Proc. the 23rd Asia and South Pacific Design Automation Conference*, Jan. 2018, pp.574-579. DOI: 10.1109/ASPDAC.2018.8297384.

[12] Wang P, Ji Y, Hong C, Lyu Y, Wang D, Xie Y. SNrram: An efficient sparse neural network computation architecture based on resistive random-access memory. In *Proc. the 55th ACM/ESDA/IEEE Design Automation Conference*, Jun. 2018. DOI: 10.1109/DAC.2018.8465793.

[13] Chi C C, Jiang J H R. Logic synthesis of binarized neural networks for efficient circuit implementation. *IEEE Trans. Comput. Des. Integr. Circuits Syst.*. DOI: 10.1109/TCAD.2021.3078606.

[14] Garey M R, Johnson D S, Stockmeyer L. Some simplified NP-complete problems. In *Proc. the 6th ACM Symposium on Theory of Computing*, Apr. 30-May 2, 1974, pp.47-63. DOI: 10.1145/800119.803884.

[15] Kazemi A, Alessandri C, Seabaugh A C, Sharon H X, Niemier M, Joshi S. A device non-ideality resilient approach for mapping neural networks to crossbar arrays. In *Proc. the 57th ACM/IEEE Design Automation Conference*, Jul. 2020. DOI: 10.1109/DAC18072.2020.9218544.

[16] Song L, Qian X, Li H, Chen Y. PipeLayer: A pipelined ReRAM-based accelerator for deep learning. In *Proc. the International Symposium on High Performance Computer Architecture*, Feb. 2017, pp.541-552. DOI: 10.1109/HPCA.2017.55.

[17] Shafiee A, Nag A, Muralimanohar N *et al.* ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Comput. Archit. News*, 2016, 44(3): 14-26. DOI: 10.1145/3007787.3001139.

[18] Zhu Z, Sun H, Lin Y *et al.* A configurable multi-precision CNN computing framework based on single bit RRAM. In *Proc. the 56th ACM/IEEE Design Automation Conference*, Jun. 2019, Article No. 56. DOI: 10.1145/3316781.3317739.

[19] Peng X, Liu R, Yu S. Optimizing weight mapping and dataflow for convolutional neural networks on processing-in-memory architectures. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 2020, 67(4): 1333-1343. DOI: 10.1109/TCSI.2019.2958568.

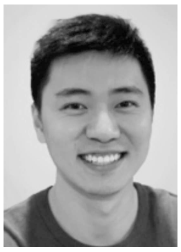
[20] Cheng M, Xia L, Zhu Z *et al.* TIME: A training-in-memory architecture for RRAM-based deep neural networks. *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, 2019, 38(5): 834-847. DOI: 10.1109/TCAD.2018.2824304.

[21] Zhu Z, Lin J, Cheng M *et al.* Mixed size crossbar based RRAM CNN accelerator with overlapped mapping method. In *Proc. the International Conference on Computer-Aided Design*, Nov. 2018, Article No. 69. DOI: 10.1145/3240765.3240825.

Feng Wang received his Ph.D. degree in computer science from Peking University, Beijing, in 2021. He is currently a research scientist in LEDA Technology. His research interests include processing-in-memory and EDA algorithms.

Guo-Jie Luo received his Ph.D. degree in computer science from University of California, Los Angeles, in 2011. He is an associate professor at Peking University, Beijing. His research interests include design automation for emerging computer architectures. He was a recipient of the 2013 ACM SIGDA Outstanding Ph.D. Dissertation Award in Electronic Design Automation and the 2017 ASP-DAC Ten-Year Retrospective Most Influential Paper Award.

Guang-Yu Sun received his Ph.D. degree in computer science from Pennsylvania State University, University Park, in 2011. He is an associate professor at Peking University, Beijing. His current research interests include energy-efficient memory architectures, storage system optimization for new devices and acceleration systems for deep learning applications.



Yu-Hao Wang received his Ph.D. degree in electrical computer engineering from Nanyang Technological University, Singapore, in 2015. He is a research scientist in Pingtouge, Alibaba Group, Hangzhou. His research interests mainly lie in the intersection of emerging non-volatile memory, processing in memory architecture, and hardware/algorithm co-designs. He has authored or co-authored two monographs, 16 scientific publications and holds two US patents.

Di-Min Niu received his Ph.D. degree in computer science from the Pennsylvania State University, University Park, in 2012. He is a research scientist in Pingtouge, Alibaba Group, Hangzhou. Prior to joining Alibaba, he was a staff memory architecture in Memory Solutions Laboratory at Samsung Semiconductor Inc.. His current research interests include computer architecture, memory architecture, storage system, process-in-memory, and domain specific architectures.

Hong-Zhong Zheng received his Ph.D. degree in computer engineering from the University of Illinois at Chicago in 2009. He is a leading research scientist in the field of new memory and storage system, processing in memory technology and architecture in Pingtouge, Alibaba Group, Hangzhou. Prior to joining Alibaba, he was a director in Memory Solutions Laboratory at Samsung Semiconductor Inc.. His current research interests include computer architecture, memory architecture, storage system, process-in-memory, and domain specific architectures.