# Long Live TIME: Improving Lifetime for Training-In-Memory Engines by Structured Gradient Sparsification

Yi Cai*†‡, Yujun Lin*†‡, Lixue Xia†‡, Xiaoming Chen¶, Song Han‖, Yu Wang†‡, Huazhong Yang†‡

†Department of Electronic Engineering, Tsinghua University, Beijing, China
‡Beijing National Research Center for Information Science and Technology (BNRist), Beijing, China
¶State Key Laboratory of Computer Architecture, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
‖Department of EECS, Massachusetts Institute of Technology, Cambridge, MA, US
yu-wang@tsinghua.edu.cn

## ABSTRACT

Deeper and larger Neural Networks (NNs) have made breakthroughs in many fields. While conventional CMOS-based computing platforms are hard to achieve higher energy efficiency. RRAM-based systems provide a promising solution to build efficient Training-In-Memory Engines (TIME). While the endurance of RRAM cells is limited, it's a severe issue as the weights of NN always need to be updated for thousands to millions of times during training. Gradient sparsification can address this problem by dropping off most of the smaller gradients but introduce unacceptable computation cost. We proposed an effective framework, SGS-ARS, including Structured Gradient Sparsification (SGS) and Aging-aware Row Swapping (ARS) scheme, to guarantee write balance across whole RRAM crossbars and prolong the lifetime of TIME. Our experiments demonstrate that 356× lifetime extension is achieved when TIME is programmed to train ResNet-50 on Imagenet dataset with our SGS-ARS framework.

## 1 INTRODUCTION

While deeper and larger neural networks (NNs) achieve better performance in many fields[9, 11], the training of state-of-the-art NNs usually consumes weeks of time. Meanwhile, as the *memory wall* exits in the von Neumann architecture, and the feature size of integrated circuit technology is approaching the physical limit [15], it is difficult for conventional CMOS-based processors to achieve significant improvements in energy efficiency or performance. People are now seeking for more energy-efficient and faster platforms for NN training.

Resistive Random-Access Memory (RRAM) provides a promising solution to build Training-In-Memory Engines (TIME) for accelerating NN computing, due to its traits of high density, low power consumption, fast read&write speed, and suitability for implementing the crossbar structure to perform matrix-vector multiplications efficiently [16]. For instance, the Training-In-MEmory [5] architecture and PipeLayer [13] are both proposed as RRAM-based accelerators for the training and inference of CNNs, achieving over 100× energy efficiency improvement and 42.45× speedup than GPU-based implementations, respectively.

However, the endurance of state-of-the-art RRAM devices covers a wide range from $10^6$ to $10^{12}$, while high-endurance devices are usually bipolar and used for only storage, and devices for multi-valued computing generally have much lower and disappointing endurance [2, 4, 8]. Limited endurance challenges almost all the RRAM-based architectures or systems for NN acceleration. For the widely-used Stochastic Gradient Descent (SGD) optimizer, updates of the weight parameters are needed in every iteration, yielding a write operation on each RRAM cell in each update cycle. For example, ResNet-50 [6] needs $5\times10^5$ iterations to be fully trained on the *Imagenet* dataset [12]. With an endurance limit of $5\times10^6$, RRAM crossbars can only be programmed for $(5\times10^6)/(5\times10^5) = 10$ times. What's more, state-of-the-art NNs (such as GoogLeNet [14]) usually require millions of iterations to train. Such endurance is not sufficient to support long-term, large-scale NN training.

Recent researches have demonstrated the feasibility of optimizing NN models with gradient sparsification (GS) [1]. GS reserves smaller gradients, and only uses larger ones to update the weights. Deep Gradient Compression proposed in [10] can drop off 99.9% of the gradients, and only use the top-0.1% gradients with the largest magnitude, without any accuracy loss. This enlightens us that through gradient sparsification, the number of write operations to RRAM devices can be reduced by three orders of magnitudes. Ideally, the lifetime of RRAM devices can be extended to 1000× longer.

However, in our experiments with conventional GS, we observe a severe unbalanced update distribution among different positions of the weight matrix. The possible reasons include that *frequently-updated* weights have more significant effects on feature extraction. Since an update on a weight value corresponds to a write operation on an RRAM cell, the frequently-written RRAM cells will, undesirably, wear out much quicker than the rest. Xia et al. [17] has proved that only with 10% broken RRAM cells, it will lead to substantial degradation on the NN performance. Therefore, the write-balance across RRAM crossbars is of paramount importance.

*: Both authors contributed equally to this work.


*DAC '18, June 24-29, 2018, San Francisco, CA, USA*

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196071

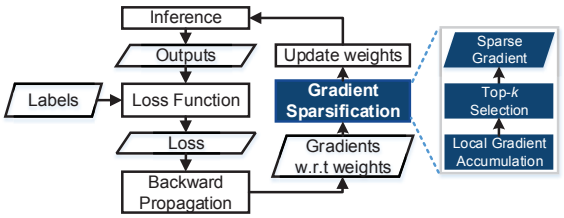



**Figure 1: The flow of training neural networks with gradient sparsification.**

Moreover, GS introduces additional computation overhead, mainly introduced by top-$k$ selection. To find the gradients with the top-$k$ largest magnitude out of $n$ elements, it usually has the time complexity of $O(n \log_2 k)$. While in large neural networks, if all gradients are involved, such time consumption will greatly slow down the pace of training. This drives us to find a faster and low-overhead way to select the important gradients for weight updates.

This paper aims to improve the lifetime of TIME. Specifically, the main contributions of this paper are listed as follows,

- We propose Structured Gradient Sparsification (SGS) by selecting structured gradients to update weights. Row-wise and element-wise sparsification are introduced for gradient matrices with different number of rows.
- We propose an Aging-aware Row Swapping (ARS) method to balance the write count across all rows in crossbars, by which the write unbalance can be efficiently mitigated.
- We propose an effective training system framework based on SGS and ARS, referred as the SGS-ARS framework. our experiment demonstrate that the lifetime of TIME can be extended to 356× the original if programmed to process the training of ResNet-50 on Imagenet.

## 2 PRELIMINARIES

### 2.1 RRAM-based Neural Computing

An RRAM cell is a passive two-port element which has multiple resistance states, and multiple cells can construct crossbars to perform efficient analog matrix-vector multiplications. If we map a *matrix* on the conductances of RRAM cells in the crossbar, and transform a *vector* as the input voltage signals, the RRAM crossbar can perform analog matrix-vector multiplications efficiently. The relationship between the input voltages and output voltages can be represented as: $i_{\text{out}}(z) = \sum_{j=1}^{M} g(z,j) \cdot v_{\text{in}}(j)$, where $v_{\text{in}}$ is the input voltage vector (denoted by $j = 1, 2, ..., M$), $i_{\text{out}}$ is the output current vector (denoted by $z = 1, 2, ..., N$), $g(z,j)$ represents the conductance of RRAM which in $z^{\text{th}}$ row and $j^{\text{th}}$ column of the crossbar ($N \times M$). By mapping the weights on RRAM crossbars, and the feature on the input voltages, RRAM crossbars can implement fast and energy-efficient neural computing.

### 2.2 Gradient Sparsification

To resolve the communication bottleneck in neural network distributed training, researchers have proposed gradient sparsification (GS) to reduce the communication data size by sending only the important gradients for the weight update [1, 7, 10], as shown in Fig.1. Two simple heuristics for importance are the gradient magnitude [1, 10] and the ratio of gradient magnitude to the weight magnitude

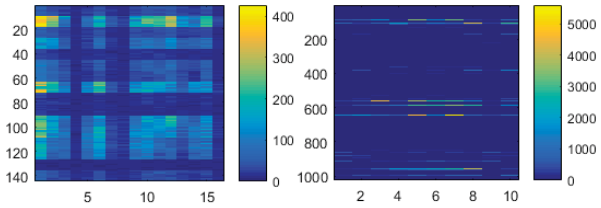



**Figure 2: The overall write distribution of RRAM crossbars. Left: the second CONV layer of ResNet-20. Right: the last FC layer of VGG-11. Total training iteration counts are both 64000.**

[7]. To avoid losing information, the rest of the gradients are accumulated locally and eventually become large enough to transmit. GS for distributed training is also practical for neural network training with a single node. Stochastic Gradient Descent (SGD) performs the following update:

$$F(w) = \frac{1}{|\chi|} \sum_{x \in \chi} f(x, w), \quad w_{t+1} = w_t - \eta \frac{1}{b} \sum_{x \in \mathcal{B}_t} \triangledown f(x, w_t) \tag{1}$$

where $\chi$ is the training dataset, $w$ are the weights of a network, $f(x, w)$ is the loss computed from samples $x \in \chi$, $\eta$ is the learning rate, and $\mathcal{B}_t$ is a mini-batch of size $b$ sampled from $\chi$ at iteration $t$. Consider the weight value $w^{(i)}$ of $i$-th position in flattened weights $w$. After $T$ iterations, we have

$$w_{t+T}^{(i)} = w_t^{(i)} - \eta T \cdot \frac{1}{bT} \left( \sum_{\tau=0}^{T-1} \sum_{x \in \mathcal{B}_{t+\tau}} \triangledown^{(i)} f(x, w_{t+\tau}) \right) \tag{2}$$

Equation (2) shows that local gradient accumulation can be considered as increasing the batch size from $b$ to $bT$ (the first summation over the iteration $\tau$), where $T$ is the length of the sparse update interval between two iterations at which the gradient of $w^{(i)}$ is adopted. Local gradient accumulation ensures the convergence of training with sparse gradients. The sparsity of gradients is able to achieve 99.9% without any loss of accuracy [10].

### 2.3 Fault-tolerant Training on RRAM

Previous work has also explored methods to prolong the lifetime of RRAM-based training systems. Xia et al. [17] proposed a fault-tolerant training method to reduce the impact of faults in RRAM cells, so that the system availability can still be guaranteed even if errors occur during training, thereby extending the lifetime. They achieved 10× lifetime extension, which is, however, not satisfactory enough for long-term stability.

## 3 MOTIVATIONAL EXAMPLE

Theoretically, the conventional gradient sparsification seems very likely to reduce the total number of write operations. However, it indeed occurs two undesired phenomena in our simulation of training ResNet-20 and VGG-11 on the CIFAR-10 dataset, which are highly unfriendly to the RRAM crossbar and make the improvement on RRAM lifetime far less appealing than expected.

### 3.1 Unbalanced Writes

Different positions of fully-connected (FC) weight matrices or convolutional (CONV) kernels usually do not share the equal chance to be updated in the conventional gradient sparsification. After mapping

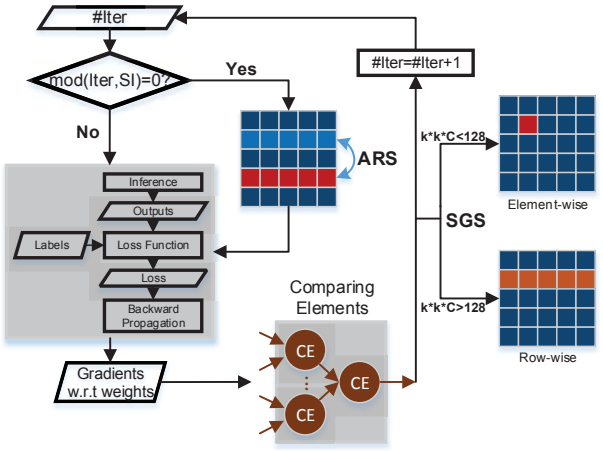



**Figure 3: Proposed the SGS-ARS framework for improving lifetime of TIME with (1) Structured Gradient Sparsification (SGS); (2) Aging-aware Row Swapping (ARS).**

weights to RRAM crossbars, it leads to the unbalanced writes on RRAM cells. Fig.2 shows the overall write distributions of two sample RRAM crossbars, each one corresponding to a weight matrix of a layer. Both of the two models are trained for 64000 iterations. The left one demonstrates the write distribution of second convolutional layer of ResNet-20 ($3 \times 3 \times 16 \times 16$, reshaped as $144 \times 16$), and the right shows the last fully-connected layer of VGG-11 ($1024 \times 10$).

The distribution maps show a severe unbalance in the total write times throughout the whole matrix. With 99.9% gradient sparsity, the expected write times shall be $64000 \times 0.1\% = 64$. Though, in ResNet-20, some cells are written for up to 376 times, and some are written even less than 10 times. It gets even worse in the FC layer of VGG-11 as the maximum of write times surges to 5595. In this occasion, frequently-written RRAM cells will wear out much more quickly than expected, following by soft faults or stuck-at-faults (SAFs). Therefore, a write-balanced solution is required.

## 3.2 Overhead of Gradient Top-$k$ Selection

Taking the largest convolution layer of VGG-11 as an example, the convolutional kernel is shaped as $3 \times 3 \times 512 \times 512$. If all $n = 2359296$ gradients are involved for searching the top 0.1% gradients ($k = n \times 0.1\% = 2359$) with the largest magnitude, both the time complexity and computation amount increase by $O\left(n \log_2 k\right) = O\left(10^7\right)$. Such large extra cost drives us to design a better sparsification method.

# 4 THE SGS-ARS FRAMEWORK FOR TIME

To extend the lifetime of TIME, we propose a simple but effective framework, referred as the SGS-ARS framework, as shown in Fig.3. Structured Gradient Sparsification (SGS) and Aging-aware Row Swapping (ARS) are carefully designed to reduce the write times and mitigate the unbalanced-writes, which will be introduced in Sec.4.1 and Sec.4.2 respectively in detail.

The whole process goes as follows. At the beginning of each iteration, the decision to perform ARS is made based on whether the current iteration number is a multiple of ARS intervals ($SI$), following by the inference and backpropagation pass. The gradients with respect to the weights are then calculated and sent to Comparing Elements (CEs) to select one with the largest magnitude. Subsequently, SGS is applied to update the weights. A *row count threshold* ($RCT$) is used to partition the neural network layers into two types, which is set to 128 in our implementation. If the row count of the weight matrix of a layer is less than $RCT$, the element-wise sparficiation will be applied; otherwise, the row-wise sparsification will be adopted to update weights. These steps will be repeated until the iteration number reaches the configured maximum number of training iterations.

## 4.1 Structured Gradient Sparsification

As observed in Fig.2, write unbalance introduced by concentration of sparse updates on weight matrices can be considered as a dual character, since it exhibits a structured pattern. This structured concentration of updated locations facilitates the re-mapping of weights. In the meantime, structured write operations on RRAM crossbar can be fully parallelized. Inspired by these nature characteristics of gradient sparsification and RRAM crossbar structure, we propose the Structured Gradient Sparsification (SGS) to overcome the drawbacks of directly applying conventional GS in RRAM-based training systems. SGS not only adapts well to RRAM by sparsifying the gradients in a RRAM-friendly structured pattern, but also significantly reduces the complexity of top-$k$ selection by changing the way to select the gradients in need for the weights update. Fig.4(a) illustrates the channel-wise, row-wise and element-wise SGS. To ensure sufficient sparsity, two parts are introduced in this paper: the row-wise sparsification and the element-wise sparsification.

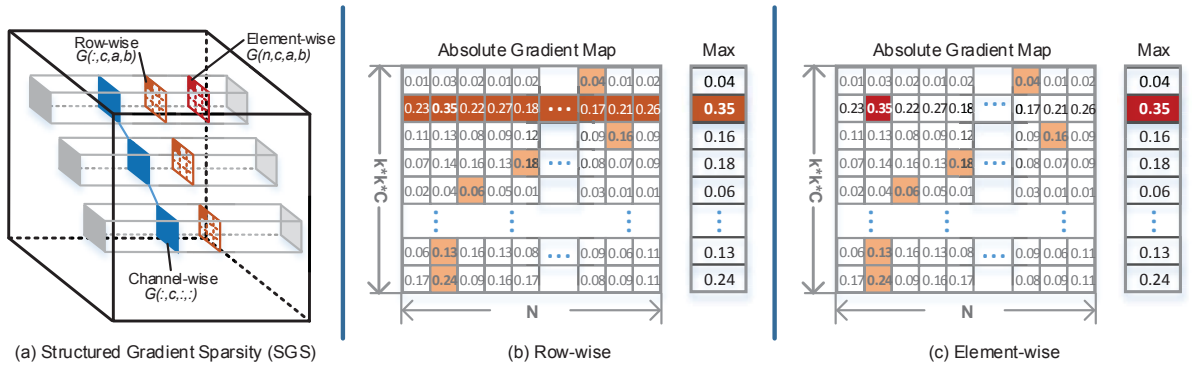
**Row-wise Sparsification.** Fig.4(b) illustrates the row-wise structured sparsification process. When mapping on RRAM crossbars, both FC and CONV layer are treated as matrix-matrix multiplication, since the kernels of CONV layer are reshaped from 4-dimension tensors ($k \times k \times C \times N$) to matrices ($k^2C \times N$), where $k$ is the kernel size, $C$ is the number of input channels and $N$ is the number of output channels.

Row-wise SGS only selects one row of weight matrix where the max gradient magnitude lies. Due to backpropagation computation trait of RRAM-based training systems, only one row of gradients is obtained each cycle [5], and these gradients are calculated in parallel. Then the maximum gradient magnitude in the row will be popped out. After the last row of gradients is finished, we immediately get the index of the row which contains the maximum magnitude of whole gradient matrix, and update the corresponding row of weights. Accordingly, the sparsity of row-wise SGS will be $1 - 1/(k^2C)$.

Row-wise sparsification is naturally favorable for RRAM structure, since RRAM supports writing cells in one row in parallel [3]. It helps significantly reduce the writing cycles for the weight update and enforces the write distribution to be uniform in rows.

**Element-wise Sparsification.** In small weight matrices with very few rows, the sparsity of row-wise sparsification will be much lower than desired. For example, the first CONV layer of most CNNs has kernel tensors shaped as $k \times k \times 3 \times N$, where $k$ usually ranges from 3 to 7, as the input images commonly have 3 channels (RGB). In this scenario, the gradient sparsity of row-wise sparsification will be $1 - 1/(3k^2)$. When $k = 3$, only 96.3% sparsity is achieved.

To increase the sparsity, a finer-grained sparsification scheme should be exploited. As shown in Fig.4(c), the element-wise sparsification follows the similar process to the row-wise. The difference is that the index of the column where the maximum gradient magnitude lies is also calculated along with the index of the row, and




(a) Structured Gradient Sparsity (SGS) (b) Row-wise (c) Element-wise

**Figure 4: The proposed Structured Gradient Sparsification (SGS). Gradients can be split into multiple groups. (a) illustrates the channel-wise, row-wise and element-wise gradient sparsification. (b-c) shows the row-wise and element-wise selection process.**

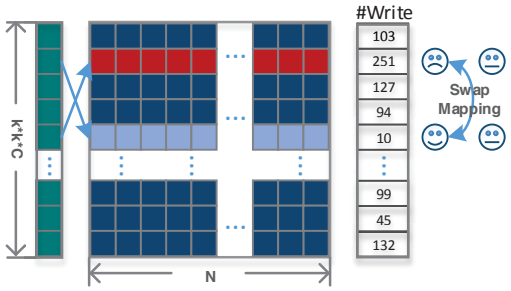



**Figure 5: Basic process of Aging-aware Row Swapping (ARS).**

we only update the corresponding location in the weight matrix. Consequently, the gradient sparsity will rise to $1 - 1/(k^2C \times N)$.

**Complexity Analysis.** The complexity of row-wise and element-wise SGS mainly concentrate on maximum value selection. With a kernel size of $k^2C \times N$, selecting the gradients with the largest magnitude in all rows will be operated for $k^2C$ times; and selecting the largest one from these gradients will be operated for one time. Thus, the total operation count will be $k^2C \times N + k^2C$. If operating serially, the time complexity will be $O(k^2C \times (N+1)) \approx O(n)$ ($n = k^2C \times N$), which is far less than $O(nlog_2 s)$ as conventional gradient sparsification. And if operating in parallel, the time complexity will be reduced to $O(log_2 n)$.

## 4.2 Aging-aware Row Swapping

Although the row-wise SGS ensures the cells in one row to share the same write times, the write distribution is still extremely unbalanced among rows. However, the row-wise operation fits the horizontal stripe pattern of the write distribution shown in Fig.2. It intensifies the concentration of writes and thus is favorable for row swapping to balance the write-load. Therefore, we propose the Aging-aware Row Swapping (ARS) approach when processing the training tasks, to dynamically adjust the weight mapping at a small extra overhead.

**Basic process.** In the training with sparse gradients, if some locations are updated much more often than others, it is reasonable to assume that they will be updated frequently in the following training iterations. So if we conduct a re-mapping, swapping the rows which are mostly written with the rows which are least written, the write times across whole crossbars will be more balanced.

Fig.5 shows the basic process of row swapping. Registers are placed to count the write times of all cells. As the cells in one row share the same write times, only $M$ registers are needed to record the ages in a crossbar of size $M \times N$. If the maximum training iteration number is set to $T$, the bit-width of a counter register only need to be $\log_2(T)$ at most. Thus, the total memory requirement of write counters will be $M \log_2(T)$. Moreover, ARS interval (noted as $SI$) is set to control the frequency of row swapping, and the variable $R$ is set to decide the number of swapped rows in each ARS. Every $SI$ iterations, the most-written $R/2$ rows and the least-written $R/2$ rows are picked out respectively; then swap the largest one with the smallest one, the second largest one with the second smallest one, and so forth.

**Trade-offs.** As stated above, the process of ARS is in control of two hyper-parameters, the ARS Interval $SI$ and the number of swapped rows $R$ in each ARS operation, and will introduce extra overhead. Firstly, the swapping of two RRAM crossbar rows needs to read out the original weights on these rows, and then write them back to each other's positions respectively. In other words, a read operation and a write operation will be performed on each cell in the selected rows. Secondly, performing the ARS during training requires not only an interruption, but also the re-scheduling of the input data addressing, since the weight mapping has been changed.

Intuitively, performing ARS more frequently certainly results in more balanced writes. Since each row swapping needs to re-write the weights mapped on corresponding rows, it will stop being profitable to continue to decrease the interval $SI$, when the $SI$ reaches a threshold value. Moreover, the configuration of $R$ also should take into consideration the trade-off between hardware overhead and write-load balance. In our implementation, we experimentally find a optimal pair of $(SI, R)$, which is presented in Sec.5.3.

# 5 EXPERIMENTAL RESULTS

## 5.1 Experiment Setup

We evaluate our SGS-ARS framework on VGG-16 and ResNet-20 neural networks with the CIFAR-10 dataset for framework performance exploration, and on ResNet-50 with the ImageNet dataset for large-scale training analysis. The proposed framework "SGS-ARS" is compared to three other algorithms. The "Baseline" method is training without gradient sparsification; the "Conventional GS" method is the conventional gradient sparsification; the "FT-Train" method is the Fault-Tolerant Training method proposed in [17].

**Table 1: Experimental Results**

| Model | Dataset | Classification Accuracy | | | | Sparsity of SGS | #Writes | | Lifetime Extension | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | SGS | | Baseline | | | SGS-ARS | Baseline | SGS-ARS | FT-Train [17] |
| | | Top-1 | Top-5 | Top-1 | Top-5 | | | | | |
| VGG-16 | Cifar10 | 91.0% (-1.5%) | - | 92.5% | - | 99.7% | 489 | 78200 | 160× | 15× |
| ResNet-20 | Cifar10 | 91.7% (+0.0%) | - | 91.7% | - | 99.9% | 316 | 64124 | 177× | - |
| ResNet-50 | ImageNet | 75.1% (-1.0%) | 92.4% (-0.5%) | 76.1% | 92.9% | 99.8% | 1264 | 450450 | 356× | - |

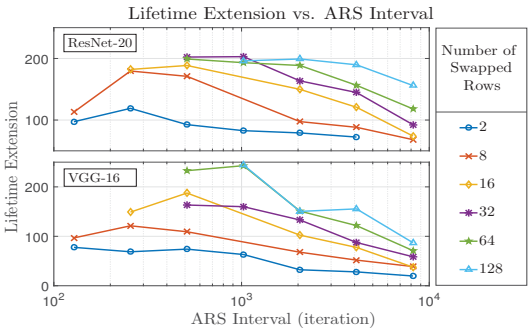



**Figure 6: The inverted U-shaped curve of lifetime extension vs. ARS interval. Under different numbers of ARS swapped row, the lifetime extensions first rise, then saturate, and eventually decline, as the ARS interval increases.**

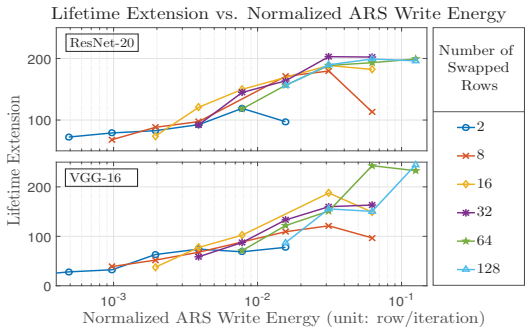



**Figure 7: The S-shaped curve of lifetime extension vs. normalized ARS write energy. Normalized ARS write energy is the average energy consumption over iterations caused by writing swapped rows, comparing to the energy consumed by writing one row. Under different numbers of ARS swapped row, the lifetime extensions first rise, and finally saturate, as the normalized ARS write energy increases.**

## 5.2 Accuracy

We demonstrate the performance of SGS-ARS methodology by evaluating the classification accuracy of trained models. Table.1 shows that there is no loss of top-1 accuracy on ResNet-20 and a small accuracy loss of 1.5% on VGG-16 when applying SGS-ARS compared to the baseline in experiment on CIFAR-10. Even when it comes to the large scale dataset and deeper neural network, the experiment of ResNet-50 only displays 1.0% loss of top-1 accuracy. Therefore, SGS-ARS is able to acquire very close performance as the baseline.

## 5.3 Trade-offs

We explore the trade-offs by performing experiments with different configuration parameter sets ($SI$, $R$). The search space of ARS interval $SI$ ranges from $10^2$ to $10^4$, and the number of swapped rows $R$ in each ARS ranges from 2 to 128. Fig.6 and Fig.7 respectively show the curves of lifetime extension versus ARS interval and normalized ARS write power under different configurations.

As is shown in Fig.7, the curves of lifetime extension versus $SI$ are inverted U-shaped. To maximize the expected lifetime extension, the sets of configuration within the flat stage of the curves are preferred. However, while maintaining the lifetime extension, the larger $SI$ is and the smaller $R$ is, the less overhead will be introduced by ARS. Meanwhile, Fig.7 shows S-shaped curves of lifetime extension versus normalized ARS write energy. The optimal set should also be within the flat stage of the curves but with less energy consumption. Therefore, we choose a relatively optimal set of ($SI$, $R$) as (1024,32). This is a near-optimal generic configuration, although the performance may slightly vary in different models. All the following experiments utilize this set of configuration.

## 5.4 Write distribution

The effectiveness of the proposed SGS-ARS approach in mitigating unbalanced writes is evaluated from two aspects: the statistical distribution of write times, and the trend of maximum write time of layers during training, compared to Conventional GS and Baseline.

**Statistical Distribution.** As shown in Fig.8, The box plot is adopted here to statistically analyze the distribution of the write times of all cells in CONV and FC layers of VGG-16 after 78200 iterations training. Common statistical information is illustrated in the plot: first quartile, median, third quartile, ±2.7 variance, and outliers. For CONV layers, both the median and variance of the write times are much smaller with SGS-AGS. For FC layers, the median of write times is smaller, but third quartile is slightly larger, which indicates the distribution of write times becomes more left skewed to fewer writes. The range of outliers of both CONV and FC layers trained with SGS-AGS dramatically declines from hundreds and thousands to dozens, and even vanishes in some layers. This demonstrates that SGS-AGS not only significantly reduces the write times, but also effectively resolves the unbalanced writes issue.

**Maximum write times.** Fig.9 shows the maximum cell write times at each of the 78200 iterations. The sample layers are respectively CONV (conv5.2) layer and FC (fc.0) layer of VGG-16. The write time of Baseline increases linearly, because without gradient sparsification, all cells will be written in every iteration. The maximum write time of Conventional GS increases much slower than the baseline, but still much faster than SGS. At the last iteration, the maximum write time of FC layer by Conventional GS is more

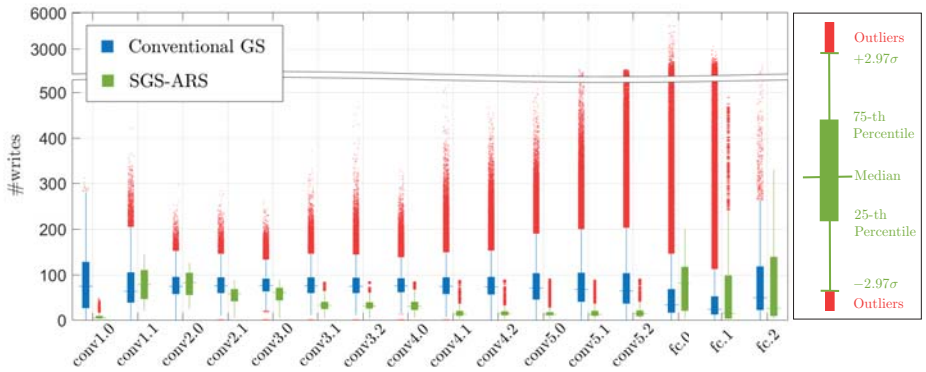



**Figure 8: The box plot of write times of all layers when training VGG16. The color-filled box indicates the range of the centered half of statistics, and the whiskers show the 99.3% coverage if normally distributed. The outliers in red describe the degree of the unbalanced distribution.**

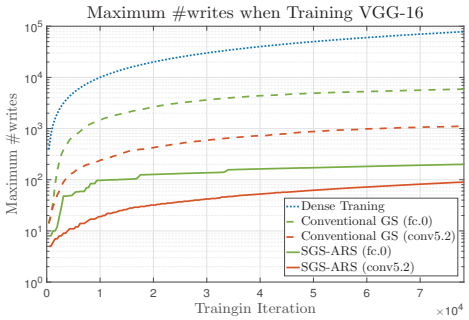



**Figure 9: Maximum write times of FC (fc.0) and CONV (conv5.2) layer when training VGG16 on Cifar10 dataset. Conventional Gradient Sparsification (Conventional GS) reduce the write times by 10-fold, while, Structured Gradient Sparsification with Aging-aware Row Swapping (SGS-ARS) achieves a reduction of writes by more than two orders of magnitude.**

than 29 times larger than by SGS. Overall, SGS reduces the maximum write time by 160× and 12× compared to the baseline and Conventional GS, respectively.

### 5.5 Lifetime Extension

Table.1 also shows the experimental results of lifetime extension under different models. When TIME is programmed for VGG-16 trained on CIFAR-10, at least 160× longer lifetime will be achieved, and 177× for ResNet-20. Compared to FT-Train, we also get over 10× extension on training of VGG-16. Moreover, on the larger model ResNet-50, 356× lifetime extension is achieved, which is in accordance with our expectation, since the sparsity of row-wise SGS will increase remarkably as the row number of weight matrix increases in the larger neural networks.

## 6 CONCLUSION

We have presented an effective framework, SGS-ARS, to improve lifetime of TIME with Structured Gradient Sparsification (SGS) and Aging-aware Row Swapping (ARS), reducing the overall write times and ensuring the write balance throughout RRAM crossbars simultaneously. Experimental results show the proposed methods extend the lifetime of TIME for approximately two orders of magnitude while maintaining almost the same neural network performance.

## 7 ACKNOWLEDGEMENTS

This work was supported by National Key R&D Program of China 2017YFA0207600, National Natural Science Foundation of China (No. 61622403, 61621091), Joint fund of Equipment pre-Research and Ministry of Education (No. 6141A02022608), and Beijing National Research Center for Information Science and Technology (BNRist).

## REFERENCES

[1] Aji et al. 2017. Sparse Communication for Distributed Gradient Descent. In *Empirical Methods in Natural Language Processing (EMNLP)*.
[2] Karsten Beckmann et al. 2016. Nanoscale Hafnium Oxide RRAM Devices Exhibit Pulse Dependent Behavior and Multi-level Resistance Capability. *Mrs Advances* 1 (2016), 1–6.
[3] G. W. Burr et al. 2015. Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power). In *IEEE International Electron Devices Meeting*. 4.4.1–4.4.4.
[4] C. H Cheng et al. 2010. Novel Ultra-low power RRAM with good endurance and retention. In *VLSI Technology*. 85–86.
[5] Ming Cheng et al. 2017. TIME: A Training-in-memory Architecture for Memristor-based Deep Neural Networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 26.
[6] Kaiming He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
[7] Kevin Hsieh et al. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 629–647.
[8] Hsu et al. 2013. Self-rectifying bipolar TaOx/TiO2 RRAM with superior endurance over 1012 cycles for 3D high-density storage-class memory. In *VLSI Technology*. T166–T167.
[9] Andrej Karpathy et al. 2015. Deep visual-semantic alignments for generating image descriptions. In *Computer Vision and Pattern Recognition*.
[10] Yujun Lin et al. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training. *International Conference on Learning Representations* (2018).
[11] Wei Liu et al. 2016. SSD: Single Shot MultiBox Detector. In *European Conference on Computer Vision*. 21–37.
[12] Olga Russakovsky et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.
[13] Linghao Song et al. 2017. PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *IEEE International Symposium on High PERFORMANCE Computer Architecture*. 541–552.
[14] Christian Szegedy et al. 2015. Going deeper with convolutions. In *Computer Vision and Pattern Recognition*. 1–9.
[15] M. M. Waldrop. 2016. The chips are down for Moore's law. *Nature* 530, 7589 (2016), 144.
[16] Yu Wang et al. 2016. Low power Convolutional Neural Networks on a chip. In *IEEE International Symposium on Circuits and Systems*. 129–132.
[17] Lixue Xia et al. 2017. Fault-Tolerant Training with On-Line Fault Detection for RRAM-Based Neural Computing Systems. In *Design Automation Conference*. 1–6.