TIME:A Training-in-memory Architecture for Memristor-based Deep Neural Networks

Ming Cheng[1], Lixue Xia[1], Zhenhua Zhu[1], Yi Cai[1], Yuan Xie[2], Yu Wang[1], Huazhong Yang[1]
[1] Tsinghua National Laboratory for Information Science and Technology (TNList),
Department of Electronic Engineering, Tsinghua University, Beijing, China
[2]Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA
yu-wang@tsinghua.edu.cn

ABSTRACT

The training of neural network (NN) is usually time-consuming and resource intensive. Memristor has shown its potential in computation of NN. Especially for the metal-oxide resistive random access memory (RRAM), its crossbar structure and multi-bit characteristic can perform the matrix-vector product in high precision, which is the most common operation of NN. However, there exist two challenges on realizing the training of NN. Firstly, the current architecture can only support the inference phase of training and cannot perform the backpropagation (BP), the weights update of NN. Secondly, the training of NN requires enormous iterations and constantly updates the weights to reach the convergence, which leads to large energy consumption because of lots of write and read operations. In this work, we propose a novel architecture, TIME, and peripheral circuit designs to enable the training of NN in RRAM. TIME supports the BP and the weights update while maximizing the reuse of peripheral circuits for the inference operation on RRAM. Meanwhile, a variability-free tuning scheme and gradually-write circuits are designed to reduce the cost of tuning RRAM. We explore the performance of both SL (supervised learning) and DRL (deep reinforcement learning) in TIME, and a specific mapping method of DRL is also introduced to further improve the energy efficiency. Experimental results show that, in SL, TIME can achieve 5.3x higher energy efficiency on average compared with the most powerful application-specific integrated circuits (ASIC) in the literature. In DRL, TIME can perform averagely 126x higher than GPU in energy efficiency. If the cost of tuning RRAM can be further reduced, TIME have the potential of boosting the energy efficiency by 2 orders of magnitude compared with ASIC.

1. INTRODUCTION

Recently, neural network (NN) and deep learning constantly make breakthroughs in many fields [1]. Supervised learning (SL) and deep reinforcement learning (DRL) based on NN have been shown to be powerful tools for classification and learning policies. They even outperform human experts in some fields, such as the classification of objects in ImageNet [2] and playing Atari game [1].

Data movements between the processing units (PUs) and the memory have been the most critical performance bottlenecks in various computing systems. For example, the data transfer between CPUs and off-chip memory consumes two orders of magnitude more energy than a floating-point operation [3]. The state-of-the-art NN requires a large memory capacity to store the weights or intermediate data between layers. To address this challenge, some application-specific integrated circuits (ASIC) have adopted large on-chip memory to store the synaptic weights. For example, DadianNao [4] places large on-chip eDRAMs for both high bandwidth and data locality.

DAC '17, June 18 - 22, 2017, Austin, TX, USA

ISBN 978-1-4503-4927-7/17/06…$15.00

DOI: http://dx.doi.org/10.1145/3061639.3062326

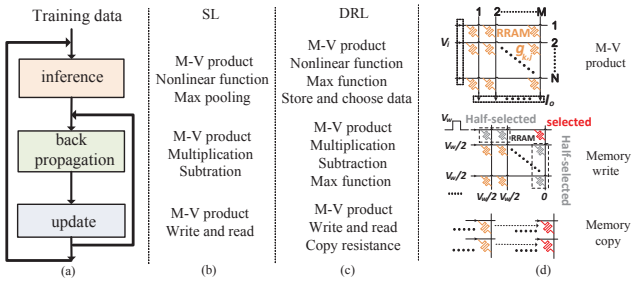


Figure 1: (a) The training flow of neural network (b) The basic operation of supervised learning and M-V product is the matrix-vector product (c) The basic operation of deep reinforcement learning (d) The basic operation on RRAM

Memristor provides promising alternative solutions to boost the performance [5]. Metal-oxide resistive random-access memory (RRAM), which is a sort of memristor devices, can be used to build a crossbar structure (shown in Fig.1 (d)) to efficiently perform the matrix-vector product with its unique multi-bit states. Recent work has demonstrated the potential of RRAM to realize the NN operations with 100x to 1000x energy efficiency gain comparing to CPU or GPU solutions [5], [6], [7], [8].

However, the current architecture, such as PRIME [6], can not realize the training of NN efficiently. As Fig.1 (a) shows, the training of NN contains three phases: inference, backpropagation (BP) and weights update. These 3 phases are repeated many iterations until convergence [5]. And the basic operation of inference is different from that of BP, weights update for both SL and DRL (shown in Fig.1 (b)(c)). Thus, the architecture supporting the training of NN is also different from the architecture of inference.

Another challenge lies in the huge energy required for tuning RRAM accurately during the training of NN. The first reason is that enormous training iterations are required and the weights are updated frequently. Thus, there exist a large amount of write operations. For example, training a small NN (784-100-10) for MNIST [9] needs about 7×10^3 iterations to converge. Hence each weight needs to be updated 7×10^3 times at least. Secondly, the cost of tuning RRAM is huge because of non-ideal factors, such as the variability of writing RRAM and the abrupt characteristic during the SET operation. All of the current architectures [5], [7], [10] do not consider the large cost of tuning RRAM when training NN.

In this paper, we propose a novel architecture, called TIME, to implement the training of NN in RRAM memory. Besides, a variability-free tuning scheme and gradually-write circuits are designed to reduce the cost of tuning RRAM. The main contributions of this paper are as follows:

1. We propose TIME (Training-In-MEmory), with modifications of peripheral circuits in PRIME to further support back-propagation and weights update operations. Experimental results show that TIME can improve the energy efficiency by 5.3x on average in SL compared with a recent ASIC solution [4]. A specific mapping method is proposed to support the training of DRL. TIME improves the energy efficiency by 126x compared with the GPU and the specific mapping method promotes 1.2x energy efficiency compared with direct mapping method.
2. We propose a variability-free tuning scheme to reduce the frequency of writing and reading RRAM, together with gradually-write circuits which are designed to overcome the asymmetry of writing RRAM. The results show the variability-free tuning scheme and the gradually-write circuits can improve the energy efficiency by 2.7x comparing to [11]. The energy efficiency can be improved to 2 orders of magnitude compared with the ASIC solution [4] if the energy of writing RRAM can be reduced.

2. PRELIMINARIES

2.1 RRAM Device Basics

Tuning RRAM includes 2 ways: SET and RESET. SET operation tunes the resistance of RRAM from high resistance state (HRS) to low resistance state (LRS). RESET operation is the reverse operation. According to [12], SET operation leads the resistance of RRAM to change abruptly while the RESET operation can gradually tune it. Besides, both of SET operation and RESET operation exist variability [12]. Thus, tuning RRAM into a target resistance value depends on iterative RESET operation and SET operation [11] but this method obviously increases the cost of tuning RRAM.

2.2 Supervised Learning

The training of SL is shown in Fig.1 (a) and it contains 3 phases: inference, BP, and update [7].

Inference of SL Fig.1 (b) shows that the inference [13] includes 3 operations: matrix-vector product, nonlinear function, which are represented as Eq. (1), and max pooling, which is represented in Eq. (2),

$$y = f(W * x + b) \tag{1}$$

where x and y are the input and the output of the NN, respectively. W is the weight of NN and b is a bias vector [11].

$$x = max(x_1, x_2, ..., x_n) \tag{2}$$

where $x_1, x_2, ..., x_n$ are the inputs of max pooling function and x is the output of max pooling function.

Backpropagation of SL Fig.1 (b) shows that BP includes 3 basic operations: matrix-vector product, multiplication, subtraction [13]. The BP process is expressed in sequence as Eq. (3), (4), (5),

$$\Delta y^{(l)} = y^{(l)} - y^{*(l)} \tag{3}$$

$$\delta^{(l)} = \Delta y^{(l)} .* y'^{(l)} \tag{4}$$

$$\Delta y^{(l-1)} = W^{T(l)} * \delta^{(l)} \tag{5}$$

where $y^{(l)}$, $y^{*(l)}$, $W^{(l)}$ represent the realistic output, the target output, weights of the lth layer, respectively. $\Delta y^{(l)}$ is the diff in lth between target output $y^{*(l)}$ and output $y^{(l)}$ in the lth layer. $\delta^{(l)}$ is the error of BP. Eq. (3) is performed in the final layer. Then, Eq. (4) and Eq. (5) are implemented in sequence iteratively.

Update of SL The update includes 3 operations [13]: write, read, matrix-vector product, which are expressed as Eq. (6),

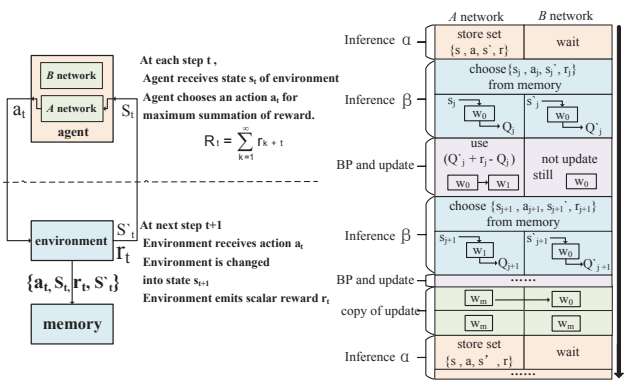


Figure 2: (a) the inference α (b) the process of DRL

$$\Delta W^{T(l)} = x^{(l)} * \delta^{T(l)} \tag{6}$$

where $\Delta W^{T(l)}$ is the variation of weights in the lth layer and $x^{(l)}$ is the input of the lth layer. After Eq. (4) is computed, we can compute Eq. (6) and use $\Delta W^{T(l)}$ to update the weight of NN.

2.3 Deep Reinforcement Learning

Fig.1 (c) shows that DRL also consists of 3 phases: inference, BP and update, but there exist differences compared with SL.

Inference of DRL Fig.1 (c) shows 4 basic operations in the inference : matrix-vector product, nonlinear function, max function, store and choose the data. Fig.2 (b) shows the detailed inference stage of DRL [1] and it includes 2 conditions, called inference α and inference β. The inference α can be represented in Fig.2 (a) and only the A network implements the computation in Eq. (1). Then, we choose the action a with the maximum output Q of A network in an incremental probability. Later, a is sent into the environment and its state is changed from s to s'. Finally, the data sets $\{s,a,r,s'\}$ are stored into memory. After several inference α, lots of data sets like $\{s,a,r,s'\}$ are stored in memory.

In the inference β, s, s', which are randomly selected from memory, denote the input of A network, B network, respectively, and the maximum output of A network, B network are denoted by Q and Q', respectively.

Backpropagation of DRL is shown in Fig.2 (b) [1] and the max function of DRL in Fig.1 is expressed in Eq. (7),

$$\Delta y = r + \gamma \max_{a'} Q'(s', a'; W_B) - Q(s, a; W_A) \tag{7}$$

where W_A and W_B are the weights of A network and B network, with Δy given by Eq. (3), and γ is a fixed parameter we set. Depending on the inference α and inference β, r, Q, Q' can be produced. Then, we implement Eq. (4) and Eq. (5) iteratively in the A network and keep the parameters of B network fixed.

Update of DRL Fig.2 (a)(b) shows the update of DRL [1]. The copy operation is to copy the weights of A network into that of B network, which is expressed as $W_B := W_A$.

After the BP process of DRL is implemented, Eq. (6) is calculated in A network and its weight is changed. After totally updating the parameters of A network for μ steps, which is a fixed parameter we define, the copy operation is implemented. Then we repeat the process in Fig.2 (b).

3. TIME ARCHITECTURE

TIME, proposed for training in memory, can support not only the inference, but also the BP and the update in the training process. Besides, TIME can support the training of SL and DRL. TIME consists of 3 subarrays: full function (FF) subarray, buffer subarray and memory (Mem) subarray. FF subarray not only realizes memory, but also supports all of the computations of NN, including the inference, BP, update. The Mem subarray only stores data. The buffer subarrays are the Mems that have the minimum distance to

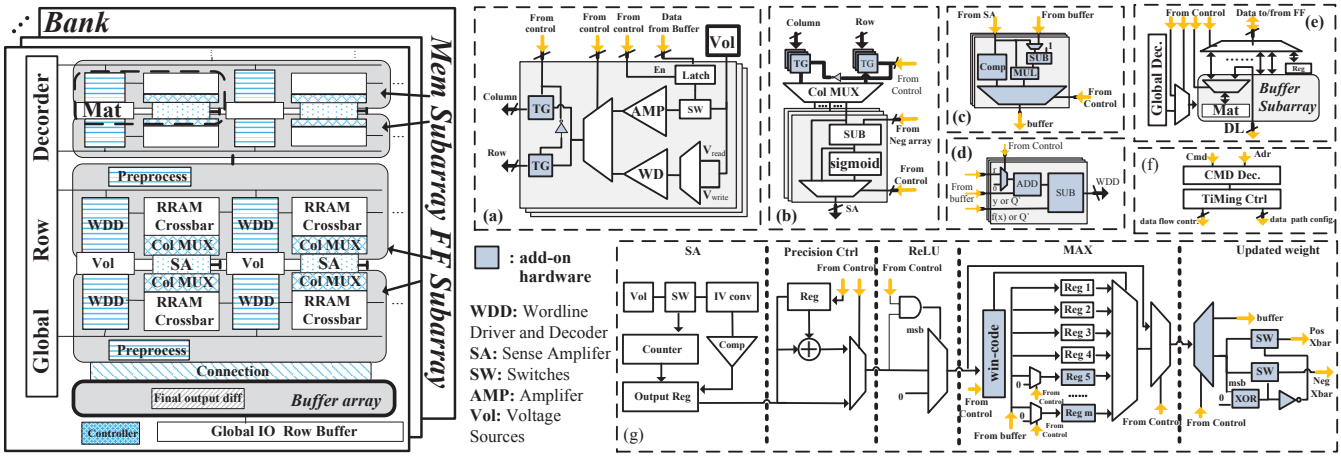


Figure 3: The TIME architecture. Left: bank structure. Right: functional blocks modified/added in TIME. (a) wordline driver with multi-level voltage sources; (b) column multiplexer with analog subtraction and sigmoid circuit; (c) preprocess for implementing the derivative of activation; (d) the final output diff for calculating the diff in the final layer; (e) connection between the FF and the Buffer subarrays; (f) TIME controller; (g) reconfigurable SA with counters for multi-level outputs, and added ReLU, max function and update weight

the FF subarrays, which can make use of the high bandwidth to store the intermediate data.

3.1 Design of Peripheral Circuit

To enable the training of NN in the FF subarray, as Fig.3 shows, we add preprocess and final output diff into FF subarray and we modify the wordline decoder and driver (WDD), column multiplexers (Col MUX), sense amplifiers (SA), and controller.

WDD Fig.3 (a) shows that WDD is for preparing the input data of NN and providing the voltage of writing and reading RRAM. WDD must support data from the wordline (WL) of RRAM crossbar or from the bitline (BL) of RRAM crossbar in order to reuse it in inference and BP. Thus, for isolating the BP and the inference, two transfer gates (TG) are connected with the wordline and bitline, respectively. By this low-cost design, the inference and the BP reuse the same WDD. Other parts of WDD are the same as the PRIME. The multi-level voltage sources are used to produce the multi-level voltages and switching circuit (SW) is to make sure all the input data can be fed into WL of RRAM crossbar simultaneously. Besides, the amplifier is for driving the WL and write driver (WD) is for modifying the resistance value of RRAM device.

Col MUX is shown in Fig.3 (b). Similar to the WDD, for reusing Col MUX, two transfer gates are added to the input of Col MUX to isolate the inference and BP of NN. Other parts are the same as PRIME. Sigmoid is used to realize the sigmoid function of NN and subtraction between 2 crossbars is to calculate the matrix-vector product. With the MUX, we can choose the output among sigmoid, subtraction, and the RRAM crossbar, which outputs the resistance value of RRAM.

SA is short of sense amplifier and implements the analog-digital conversion, precision control, ReLU function, max function and the selection of updating as shown in Fig.3 (g). Both of Eq. (7) of DRL and Eq. (2) of SL implement the max function. The pooling, which is calculated in Eq. (2), in PRIME usually finds the maximum value of 4 values but the DRL usually finds it in more than 4 values, such as 18 values in [1]. So we add more registers and corresponding MUX to switch the max functions in the pooling and in DRL. This change only appears in one SA because the max function of DRL is only computed in the final layer of NN. The updated weight is designed for choosing which crossbar needs to be updated according to the sign of the variation of weight ΔW. The detail of this circuit is discussed in Section 4. The sensing circuit is for realizing the analog-digital conversion and precision control is for overcoming the difficulty of precision. These two parts are the same as PRIME [6].

Final Output Diff Fig.3 (d) shows the design of final output diff and it is used for computing subtraction of Eq. (3) and Eq. (7). Therefore, a subtractor is used for the computation in final diff. Besides, there exists an adder to calculate the addition of Q' and r in Eq. (7) of DRL, which is for producing the target output in DRL. What we should note is that the final output diff only exists once so it causes a very low cost of energy and area. Compared with PRIME, the final output diff is a new circuit.

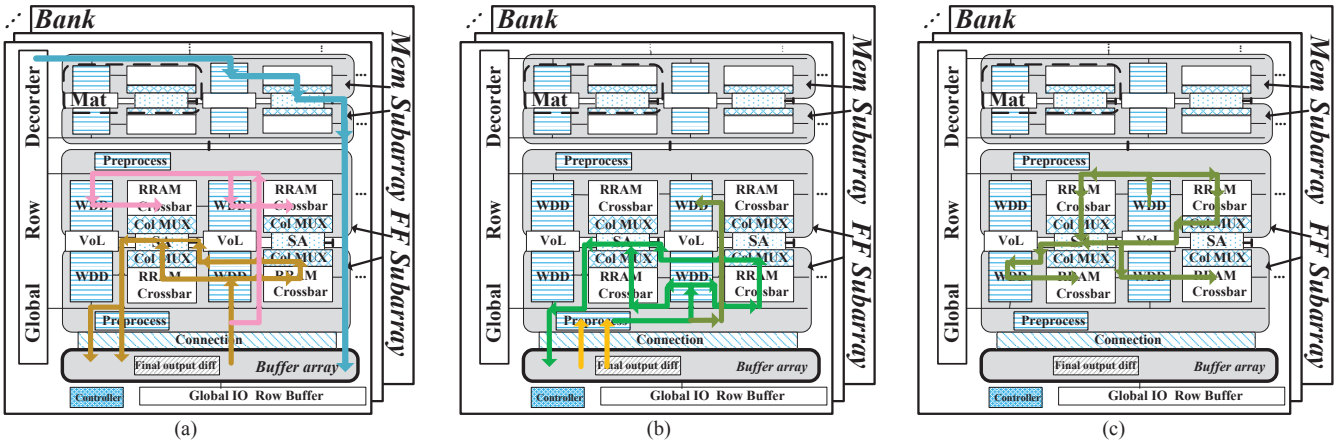
Preprocess As Fig.3 (c) shows, the preprocess is used to compute the derivative in Eq. (4). According to Eq. (4) and Eq. (6), the ΔW needs δ, which is the result of the multiplication between y' and Δy . For generating y' of sigmoid and ReLU, the preprocess is designed. According to [13], the comparator (Comp) can calculate the derivative of ReLU function. As for sigmoid function [13], the subtractor and multiplier can calculate the derivative of the sigmoid function. Besides, we reuse the multiplier in Fig.3 (c) to implement the multiplication of δy and y'. Preprocess is a new circuit compared with PRIME.

Controller can be seen in Fig.3 (f). It is mainly used to provide the control signals in FF subarray. The key function of the controller is to configure the FF subarrays among different modes. Compared with PRIME of two modes, which are memory and inference, the controller in TIME adds three modes: BP, update, DRL.

Connection As Fig.3 (e) shows, depending on the private connection between FF subarrays and the buffer subarrays, the intermediate data produced by FF subarrays can be stored in buffer subarrays with a high bandwidth and low cost of data movements.

3.2 Training Implementation of SL

Inference Fig.4 (a) shows the 1st layer of NN. At first, the blue line and arrow represent that sample (x^*, y^*) is fed into the Mem subarrays of TIME. Then, (x^*, y^*) is sent from Mem subarrays into the buffer subarrays. Next, the input x^* of the 1st layer is transferred into the FF subarrays through buffer subarrays. Then, x^* is sent into crossbar to implement matrix-vector product in Eq. (1). At the same time, as the pink line shows, x^* is written into the another crossbar ahead of implementing the operations in Eq. (6). Other layers are in similar condition. The input is sent to two crossbars for implementing inference of NN and it is stored into another FF subarrays at the same time. These 2 parallel operations



(a) (b) (c)

Figure 4: (a) Inference of NN. The blue line represents the flow of training data, the brown line represents the matrix-vector multiplication on RRAM and the pink line represents storing the input of every layer of NN (b) BP of NN. The yellow line represents the data flow from final output diff and buffer subarray to preprocess, the green line represents the backward matrix-vector multiplication and the deep green line represents preparing for the update (c) Updating of NN. The deep green line represents the update calculation and tuning the resistance value of RRAM

can accelerate the training.

As the brown line of Fig.4 (a) shows, the data are divided into 2 parts after Eq. (1). One part is sent into crossbars of the next layer of NN to continue the inference of NN, while the other part is fed into preprocess to compute y' in Eq. (4) before the results come out. These data flows also increase the speed of training.

Backpropagation Fig.4 (b) shows the final layer of NN. At first, the final output diff calculates the subtraction in Eq. (3). Then, the result Δy is sent into the preprocess part to perform the multiplication in Eq. (4) as the yellow line shows. After that, the results are sent into two places at the same time. One is sent to the crossbars that store weights of this layer to calculate Eq. (5) as the green line shows, and the other is to the crossbar that stores the input of layer for calculating Eq. (6) of update as the emerald green line shows. Thus, we parallel the BP and the update to boost the speed.

Update Fig.4 (c) shows the update of training. After realizing the operation of pink line in Fig.4 (a) and the operation of emerald green line in Fig.4 (b), the Eq. (6) can be implemented as shown in the emerald green line in Fig.4 (c). Then, the result is transferred into the corresponding crossbar to write the RRAM.

4. PROPOSED TUNING SCHEME

4.1 Variability-free Tuning Scheme

The existing method to overcome the variability of writing is to RESET, read and initialize the RRAM iteratively [11]. But this method consumes large energy because the frequency of writing and reading RRAM is high. For example, in a 784-250-100-10 artificial neural network (ANN), the energy of writing and reading RRAM dominates more than 90% energy when training the ANN in TIME. According to the Gaussian variation model [12], we propose a variability-free tuning scheme to reduce the frequency of writing, reading RRAM. The variation model is expressed in Eq. (8).

$$\Delta R_{real} \sim N(\Delta R_{accurate}, 0.09R) \tag{8}$$

where ΔR_{real} is the realistic change of resistance when we tune RRAM. $\Delta R_{accurate}$ is the resistance value compared with the target resistance and the R is the current resistance value. When the variance ($0.09R$) is larger than the mean value $\Delta R_{accurate}$, more number of RESET, read and initialization are needed. Thus, we find an optimal ratio λ to reduce the ΔR_{real} in order to reduce the frequency of writing and reading RRAM. Once the variance $0.09R$ is larger than the mean value $\Delta R_{accurate}$, we multiply ratio $\Delta R_{accurate}$ with λ to avoid unnecessary initialization, read and RESET. With variability-free tuning scheme, the cost of tuning RRAM is substantially reduced compared with that of method in [11].

4.2 Gradually-write Circuit

According to [12], there exists asymmetry between RESET operation and SET operation. The SET operation is abrupt while the RESET process is gradual, so it is risky to SET the RRAM. A weight of NN is expressed by two conductance in RRAM, called the positive RRAM and the negative RRAM respectively. Thus, we can only RESET the positive RRAM when reducing the weight and RESET the negative RRAM when adding the weight. This method can avoid the risk from the SET operation.

Therefore, we propose a circuit, called updated weight, in Fig.3 (g), so that we only RESET the resistance without SET operation in the computation of NN. According to the most significant bit (MSB) of ΔW, the crossbar needs to be tuned is chosen.

Besides, if we only RESET the RRAM, it would cause the resistance of RRAM to reach the boundary. [14] proposes a strategy that regularly calibrates the conductance to overcome this problem and this method is used in TIME.

5. DRL ON TIME

The training of DRL consists of 3 phases: inference, BP, update. The inference of DRL includes inference α and inference β. As shown in Fig.2 (a), the action a with the maximum output Q of A network is chosen in inference α. Besides, Eq. (7) also needs a max function to obtain Q' in inference β. These 2 operations are similar as the pooling in convolutional neural network (CNN). Thus, we design the max circuit in Fig.3 (g) to replace the max pooling circuit in PRIME to implement these 3 operations.

As Fig.2 (b) shows, there is a copy operation of update in DRL, which copies the weights of A network into B network. When we implement the copy operation, we need to read the resistance of RRAM of A network and write it into the resistance of RRAM of B network. However, the accurately tuning is costly on RRAM because of its writing variability. Thus, we propose a specific mapping method to cancel the tuning caused by copy operation of DRL. According to Fig.2 (b), B network is used to produce Q', which is used to calculate the Δy in Eq. (7), and its weights are same as the weights of A network of μ iterations before. In other words, the weights of B network are derived from A network and we use the output Q' and r from the same data set to train A network. Thus, we can use one network of RRAM crossbar to implement the 2 networks of DRL as shown in Fig.6. The inference α in one network is same as the one in Fig.2 (a). Then, we implement the inference β in

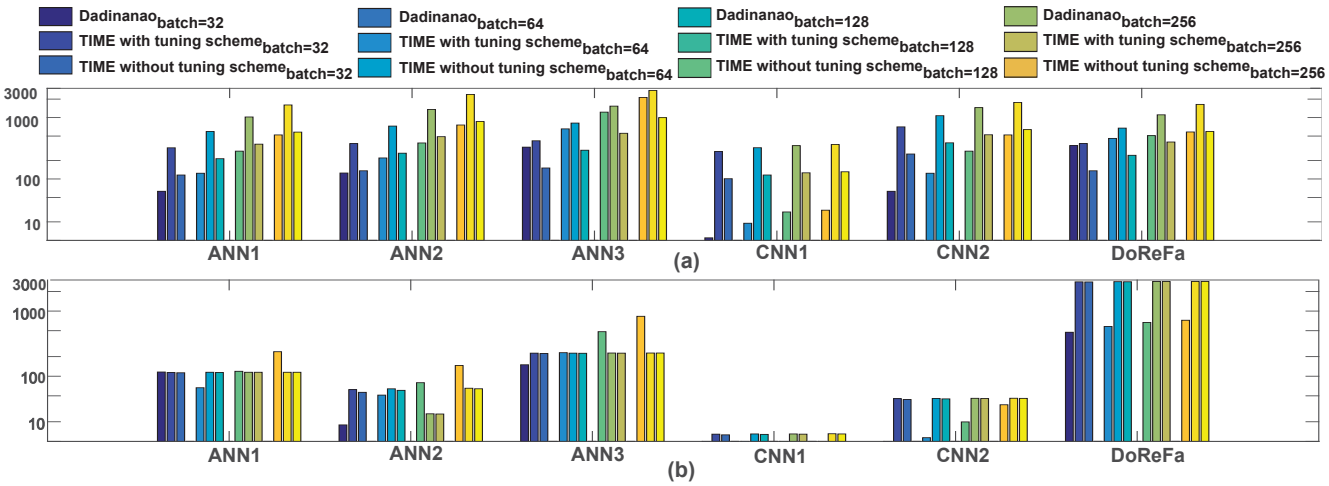


Figure 5: (a) The energy efficiency of SL. The vertical axis represents the GOP/J and horizontal axis represents different batch size in different NN. (b) The speed of SL. The vertical axis represents the GOP/s and horizontal axis represents different batch size

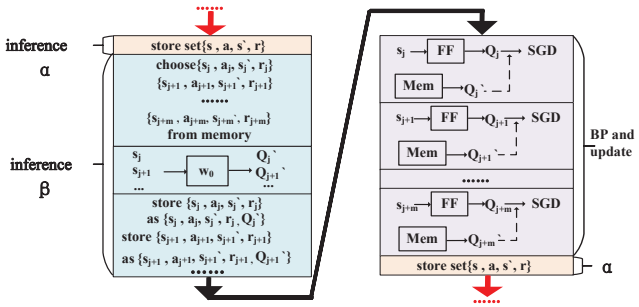


Figure 6: Specific mapping method of DRL to cancel the tuning of RRAM caused by copy of update

Table 1: Benchmark

Name	The structure of network
ANN-1	784-500-250-10
ANN-2	784-1000-500-250-10
ANN-3	784-1500-1000-500-10
CNN-1	conv5x5-pool2-720-70-10
CNN-2	conv7x10-pool2-1210-120-10
DoReFa Net	conv11x96-pool3-conv5x256-pool3-conv3x384-conv3x384-conv3x256-pool3-9216-4096-4096-1000
DRL net	conv8x16-conv4x32-2592-256-18

one network same times as the inference β in Fig.2 (b) and we can obtain the output Q' of one network, which is same as the output of B network in Fig.2 (b). We store all of Q' into the corresponding data set $\{s,a,r,s'\}$ and it is changed into $\{s,a,r,s',Q'\}$. Next, we also calculate Eq. (7) to operate the BP and implement Eq. (3) (4) (5) to update the one network. After μ iterations, one network continues to implement the inference α, inference β , BP and update. Thus, with the specific mapping method, we cancel the tuning operation caused by the copy operation of DRL. Besides, the scale of network is reduced to its half with this specific mapping method and the area of TIME is benefit from it.

6. EXPERIMENTAL RESULTS

6.1 Experiment Setup

The benchmarks include 7 NNs and their parameters are listed in TABLE I. The DRL net [1] is used to play atari game. DoReFa net is a large CNN for the ImageNet dataset [15] and contains about 6×10^7 synapses. Others are used to recognize the MNIST dataset.

TIME contains 64 banks in one chip and there are 2 FF subarrays and 1 buffer subarray per bank (totally 64 subarrays). In FF subarray, each mat includes one crossbar of 256×256 RRAM cells and 8 6-bit SAs. For the RRAM cell, according to the study we know, the weight of NN is of floating points in BP. Thus, we set

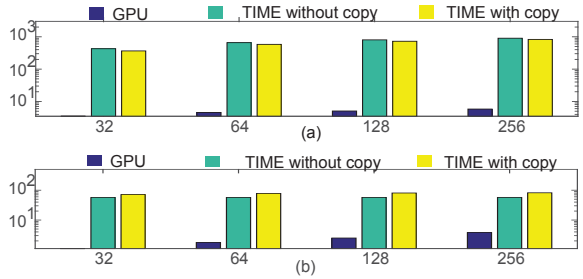


Figure 7: (a) Energy of DRL. The vertical axis represents the GOP/J and the horizontal axis represents the batch size. (b) Speed of DRL. The vertical axis represents the GOP/s and the horizontal axis represents the batch size

RRAM cell 5-bit during the inference and assume RRAM cell ideally consistent during BP for computation. The RRAM cell is 1-bit in memory mode.

The counterparts of TIME are GPU and Dadiannao. The K20c is adopted as the GPU counterpart and the data of Dadiannao are based on [4]. The simulation library is 65nm CMOS library. The detailed parameters are based on previous design and models, such as write driver [16], sigmoid and SA [17], RRAM model is in [12]. Other settings, such as the precision of RRAM, basic peripheral circuit parameters, etc, are based on PRIME.

6.2 SL Result

Energy Result Fig.5 (a) shows the detailed energy efficiency results of Dadiannao, TIME with the tuning scheme and TIME without the tuning scheme in different NN. Besides, the experiments are under different batch sizes, 32, 64, 128 and 256. The results show that the energy efficiency of TIME with the tuning scheme increases by 1.1x at least, 25.5x at most and 5.7x on average compared with Dadiannao. The comparison between TIME with tuning scheme and TIME with the scheme in [11] shows that the energy efficiency is boosted 2.7x on average. This result shows that the proposed tuning scheme is necessary to reduce the cost of tuning RRAM.

Speed Result Fig.5 (b) shows the detailed speed results of Dadiannao, TIME with the tuning scheme and TIME without the tuning scheme with different batch sizes. The speed of TIME increases at most 6.6x and on average 2.9x compared with Dadiannao. For ANN, the speed of Dadiannao is faster than TIME, such as the ANN3, while it is slower than TIME in CNN. And in the NN of larger scale, DoReFa net, TIME is 4.4x faster than Dadiannao.

According to Fig.5 (b), the speed of tuning scheme is almost the

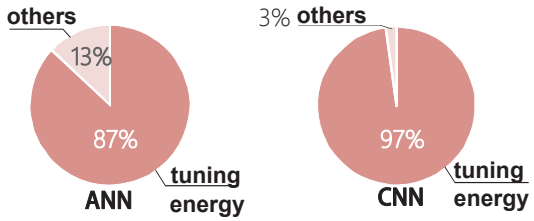


Figure 8: Energy distribution

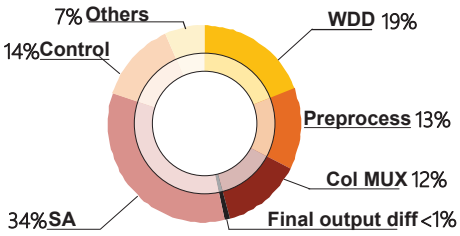


Figure 9: Area overhead of TIME

same as TIME with the method in [11]. Thus, the tuning scheme does not accelerate training NN in TIME.

6.3 DRL Result

Energy Result Fig.7 (a) shows the detailed energy efficiency results of GPU, TIME with the specific mapping method and TIME without the specific mapping method in DRL net under different batch sizes. The results show that the energy efficiency of TIME is at least 114x, at most 156x, and on average 126x compared with GPU. TIME with the specific mapping method can boost 1.2x energy efficiency on average compared with TIME without the specific mapping method.

Speed Result Fig.7 (b) shows the detailed speed results of GPU, TIME with the specific mapping method and TIME without the specific mapping method with different batch sizes. The GOP/s of TIME increases at most 70x and at least 15x compared with GPU in DRL net.

Besides, TIME without the copy and TIME with the copy are compared. The results in Fig.7 (b) show that the speed of the former is about 1.2x to 1.4x faster than the latter, while the energy efficiency of the former is about 1.2x lower than the latter. Thus, there exists a trade-off between energy efficiency and speed when we realize DRL in TIME.

6.4 Energy Distribution and Area Overhead

Energy Distribution Fig.8 shows the energy distribution in TIME. The tuning energy dominates 87% of the total energy in training ANN and 97% in training CNN. The write model in this paper is adopted from [18] and the write energy is 60.9 pJ. The current write energy in [12] can be reduced to about 1pJ. If we use 1 pJ as the write energy, TIME will improve the energy efficiency more than 50x, and boost energy efficiency to be 2 orders of magnitude higher compared with Dadiannao.

Area Overhead Fig.9 shows the area overhead of TIME. Our design only incurs 6.92% area overhead comparing with the whole RRAM memory according to [19]. The area overhead of SA is the largest part because we add ReLU function, max and weights update circuit in it. The design of WDD and COL MUX does not lead to significant increase in the area overhead. The control, which increases only 8% overhead in PRIME, increases 12% overhead in TIME because TIME can support the training of NN. The final output diff and preprocess occupy 14%.

7. CONCLUSION

In this paper, we propose a novel architecture, TIME, and peripheral circuit designs to enable the training of NN. TIME substantially improves the speed and energy efficiency for NN applications. Besides, a variability-free writing scheme and gradually-write circuits are designed to reduce the cost of tuning RRAM. A specific mapping method is proposed to improve the energy efficiency of DRL. We evaluate the energy distribution of TIME and find that the energy consumption mainly comes from tuning RRAM. TIME incurs an insignificant area overhead comparing with original R-RAM memory chips. The experimental results show that TIME can achieve a high speedup and significant energy saving for various NN applications.

8. ACKNOWLEDGMENTS

This work was supported by 973 project 2013CB329000, and National Natural Science Foundation of China (No.61373026, 61622403), and Huawei.

9. REFERENCES

[1] Volodymyr Mnih et al. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Kaiming He et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[3] Stephen W. Keckler et al. GPUs and the Future of Parallel Computing. *IEEE Micro*, 31(5):7–17, 2011.

[4] Yunji Chen et al. Dadiannao: A machine-learning supercomputer. In *IEEE/ACM ISM*, pages 609–622, 2015.

[5] Boxun Li et al. Training itself: Mixed-signal training acceleration for memristor-based neural network. In *ASPDAC*, pages 361–366, 2014.

[6] Ping Chi et al. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *ISCA*, pages 27–39, 2016.

[7] Ragib Hasan et al. Enabling back propagation training of memristor crossbar neuromorphic processors. In *IJCNN*, pages 21–28. IEEE, 2014.

[8] Lixue Xia et al. Switched by input: power efficient structure for rram-based convolutional neural network. In *DAC*, 2016.

[9] LeCun et al. The mnist database of handwritten digits, 1998.

[10] D Soudry et al. Memristor-based multilayer neural networks with online gradient descent training. *IEEE TNNLS*, 26(10):1, 2015.

[11] Boxun Li et al. RRAM-Based Analog Approximate Computing. *TCAD*, 34(12):1–1, 2015.

[12] Shimeng Yu et al. A neuromorphic visual system using RRAM synaptic devices with Sub-pJ energy and tolerance to variability: Experimental characterization and large-scale modeling. *IEDM*, 2012:10.4.1–10.4.4, 2012.

[13] Y. Lecun. A theoretical framework for back-propagation. In *Artificial Neural Networks: concepts and theory*, 1992.

[14] G. W. Burr et al. Large-scale neural networks implemented with non-volatile memory as the synaptic weight element: Comparative performance analysis (accuracy, speed, and power). In *IEDM*, 2015.

[15] Shuchang Zhou et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[16] Cong Xu et al. Understanding the trade-offs in multi-level cell reram memory design. In *DAC*, pages 1–6, 2013.

[17] Lixue Xia et al. MNSIM: Simulation platform for memristor-based neuromorphic computing system. In *DATE*, pages 469–474, 2016.

[18] H-S Philip Wong et al. Recent progress of phase change memory (PCM) and resistive switching random access memory (RRAM). In *IEEE IMW*, pages 1–5. IEEE, 2011.

[19] Xiangyu Dong et al. Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*, pages 15–50. Springer, 2014.