Training Low Bitwidth Convolutional Neural Network on RRAM

Yi Cai, Tianqi Tang, Lixue Xia, Ming Cheng, Zhenhua Zhu, Yu Wang, Huazhong Yang
Dept. of E.E., Tsinghua National Laboratory for Information Science and Technology (TNList),
Tsinghua University, Beijing, China
e-mail: yu-wang@mail.tsinghua.edu.cn

***Abstract*—Convolutional Neural Networks (CNNs) have achieved excellent performance on various artificial intelligence (AI) applications, while a higher demand on energy efficiency is required for future AI. Resistive Random-Access Memory (RRAM)-based computing system provides a promising solution to energy-efficient neural network training. However, it's difficult to support high-precision CNN in RRAM-based hardware systems. Firstly, multi-bit digital-analog interfaces will take up most energy overhead of the whole system. Secondly, it's difficult to write the RRAM to expected resistance states accurately; only low-precision numbers can be represented. To enable CNN training based on RRAM, we propose a low-bitwidth CNN training method, using low-bitwidth convolution outputs (CO), activations (A), weights (W) and gradients (G) to train CNN models based on RRAM. Furthermore, we design a system to implement the training algorithms. We explore the accuracy under different bitwidth combinations of (A,CO,W,G), and propose a practical tradeoff between accuracy and energy overhead. Our experiments demonstrate that the proposed system perform well on low-bitwidth CNN training tasks. For example, training LeNet-5 with 4-bit convolution outputs, 4-bit weights, 4-bit activations and 4-bit gradients on MNIST can still achieve 97.67% accuracy. Moreover, the proposed system can achieve 23.0X higher energy efficiency than GPU when processing the training task of LeNet-5, and 4.4X higher energy efficiency when processing the training task of ResNet-20.**

I. Introduction

Convolutional Neural Networks (CNNs) have achieved excellent performance on a variety of artificial intelligence applications, such as image classification [1], video tracking [2], speech recognition [3], and natural language processing [4]. Since people are producing a large amount of valuable data on their devices, using these data to train or fine-tune CNN models for individuals is necessary and meaningful. While considering the cost of data communication and security of personal information, processing data at personal devices instead of at centralized servers is more likely to become a new way. But personal devices are usually energy limited, especially mobile devices such as phones, wearable devices or small intelligent robots. In this scenario, developing higher energy-efficiency training processors is of paramount importance.

In computing systems with von Neumann architecture, it's difficult to break through the bottleneck of "memory wall", if there exit a large amount of data movements between computing units and memories. CNN's training is exactly a type of data-movement-frequent task, as it usually requires thousands of iterations, and each iteration moves the training data and parameters. The emerging metal-oxide Resistive Random-Access Memory (RRAM)-based computing system has been proved as one of the most promising candidates for future CNN accelerators, because the RRAM crossbar can not only be used as memories, but also serve as a matrix-vector multiplier [5]–[8], thus the movements of weight parameters can be cut off. Thereby, training CNNs based on RRAM crossbars is potential to improve the speed and energy efficiency.

However, there still exist some challenges. Firstly, the data in CNN models are usually in high precision to achieve higher recognition accuracy, including the forward-propagated feature maps, the weights, and the backward-propagated gradients. But the RRAM-based system is hard to meet such requirements. On the one hand, the conductance precision of RRAM devices is usually limited. Only 7-bit precision is achievable on state-of-the-art RRAM device [9]. Therefore RRAM devices are unable to store high-bitwidth data. On the other hand, RRAM crossbars need Analog-to-Digital Converters(ADCs) and Digital-to-Analog Converters(DACs) to complete the data exchange with peripheral circuits. Previous work has proved that 8-bit AD/DA converters take up over 85% of system overhead [5]. Secondly, if the weights or feature maps deviate from expected values, the training may be greatly affected. Actually, non-ideal factors will cause a disturbance on RRAM's resistance [10], which leads to inaccurate stored value. It's important to find a way to reduce the impact of such disturbance on training. And low-bit CNN models are more likely to have a higher tolerance for the disturbance, as there are bigger quantization intervals between the numbers.

Recent researches have considerably reduced the precision of CNNs during training process by using low-bitwidth weights, activations and gradients, achieving comparable accuracy with networks which use full-precision numbers. Previous works like BNN [11] and XNOR-net [12] use binary weights and activations in convolutional layers, but with backpropagated gradients in floating point. Dorefa-net [13] has succeeded in quantizing the gradients to numbers with low bitwidth. While its training algorithm doesn't match well with RRAM-based system, as the it doesn't quantize the convolution outputs. Moreover, the quantization strategies in DoReFa-Net use some complicated floating-point operations, such as *tanh* functions, which are unachievable or in high cost in RRAM-based systems.

Previous works have also attempted to improve the algorithms and map them on RRAM-based systems. Tang proposed an RRAM-based binary convolutional neural network accelerator [14], but this work only accelerates the inference process, not training. Cheng proposed TIME [15], an architecture that can support RRAM-based deep neural networks training. His work focuses on the architecture design and non-ideal-factor-tolerant schemes, without considering the precision limit of interfaces.

In this paper, we propose an RRAM-based low-bitwidth CNN training system and discuss the structure in detail. Specifically, the main contributions of our work include:

1) We propose the method of training low-bitwidth convolutional neural networks, to enable a RRAM-based system to implement on-chip CNN training. And specific quantization and AD/DA conversion strategies are proposed to improve the accuracy of CNN model.

This work was supported by National Natural Science Foundation of China (61373026, 61622403, 61621091), National Key R&D Program of China (2017YFA0207600), Huawei Technologies Co. Ltd, and Joint Fund of Equipment Pre-Research and Ministry of Education (6141A02022608).

978-1-5090-0602-1/18/$31.00 ©2018 IEEE

2) We explore the configuration space of combinations of bitwidth of activations, convolution outputs, weights, and gradients by experiments of training LeNet-5 and ResNet-20 on proposed system, testing over the MNIST and CIFAR-10 datasets respectively. Moreover, a tradeoff of balancing between energy overhead and prediction accuracy is discussed.
3) We analyze the probability distribution of RRAM's stochastic disturbance and make experiments to explore the effects of the disturbance on CNN's training.

The remainder of this paper is organized as follows. Section II provides the background knowledge and the motivation of our work; Section III proposes the RRAM-based low-bitwidth CNN training system design; Section IV uses the case studies to analyze recognition accuracy and energy efficiency, and Section V concludes this paper and puts forward future works.

II. Preliminaries and Motivation

A. CNN and Training Process

Typical CNNs are usually constructed by a sequence of convolutional layers (Conv layers, optionally followed by batch normalization, non-linear and pooling functions) and fully-connected layers (FC layers). The feature maps are forwardly propagated through layers until the end. And the outputs of layers which will be sent to next layer are called *activations*. Each layer has learnable parameters, including the convolutional kernels of Conv layers and the weights of FC layers, which are usually named as *weights*. Weights are continually updated during training through the backpropagated *gradients*, according to certain optimization methods. In this paper, we add a new notation *convouts* for the convolution outputs of convolutional operations. A 3-phase cycle will be repeated in the training process: inference, backpropagation, and parameters update.

1) Inference: Inference is the phase by which conclusions are inferred from a well-trained neural network. These conclusions can be classification or detection results, or any other predictions corresponded to specific tasks. The inference of convolutional layers can be expressed as below:

$$fp_{\text{out}}(x,y,z) = \sum_{i=0}^{h-1}\sum_{j=0}^{w-1}\sum_{c=1}^{C_{\text{in}}} fp_{\text{in}}(x+i,y+j,c)\cdot k_z(i,j,c) \quad (1)$$

where fp_{in} represents the 3-dimensional input feature map with the size of $h_{\text{in}} \times w_{\text{in}} \times C_{\text{in}}$; fp_{out} represents the 3-dimensional convout with the size of $h_{\text{out}} \times w_{\text{out}} \times C_{\text{out}}$; k_z is the z^{th} convolution kernel with the size of $h \times w \times C_{\text{in}}$; and the remaining variables are all spatial coordinates of the feature maps and convolution kernels.

2) Backpropagation: Backpropagation is an effective method of training CNN and usually used in conjunction with an optimization algorithm, such as Stochastic Gradient Descent (SGD). When we get a conclusion from the *inference* phase, it's then compared to the target result (label), using a loss function to estimate the difference between inferred and true values. A training process always seeks for fitting parameters to minimize the loss function. A typical loss function is shown as Equ. 2.

$$J(W) = \frac{1}{2N}\sum_{i=1}^{N}\|h_W(x_i) - y_i\|^2 \quad (2)$$

where N is the number of a mini-batch of images, also named as "batch-size". x_i is the i^{th} input image, y_i is the i^{th} target result and $h_W(x)$ represents the equivalent hypothesis function of the network. The partial derivative of loss function as Equ. 2 are used to calculate the gradients w.r.t. neurons and w.r.t. weights in Conv layers. The equations are shown as Equ.3 and Equ. 4.

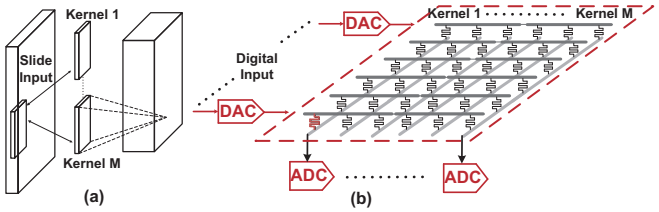

Fig. 1: Structure of the RRAM Crossbar [14].

$$\frac{\partial J(W)}{\partial a_i^l} = \sum_{j=1}^{n^{l+1}} \frac{\partial J(W)}{\partial a_j^{l+1}}\frac{\partial a_j^{l+1}}{\partial a_i^l} = \sum_{j=1}^{n^{l+1}} \frac{\partial J(W)}{\partial a_j^{l+1}} h'(z_j^{l+1}) w_{ij}^l \quad (3)$$

$$\frac{\partial J(W)}{\partial w_{ij}^l} = \sum_{j=1}^{n^{l+1}} \frac{\partial J(W)}{\partial a_j^{l+1}}\frac{\partial a_j^{l+1}}{\partial w_{ij}^l} = \sum_{j=1}^{n^{l+1}} \frac{\partial J(W)}{\partial a_j^{l+1}} h'(z_j^{l+1}) a_i^l \quad (4)$$

where a represents the activation of a layer, $h(x)$ represents the equivalent function of all non-linear transformations such as ReLU, BN or pooling. z represents the convout. l is the serial number of layers, and i,j are both the coordinates of neurons or elements in weight matrices.

3) Parameters update: After calculating the gradients of the loss function w.r.t. the weights, we use SGD to update the weight parameters-, attempting to minimize the loss. The updating function is shown as Equ. 5 and Equ. 6.

$$\Delta w_{ij}^l = \chi \cdot \frac{\partial J(W)}{\partial w_{ij}^l} + m \cdot \Delta w_{ijlast}^l \quad (5)$$

$$w_{ij}^l = w_{ij}^l - \Delta w_{ij}^l \quad (6)$$

where *Learning Rate* (χ) is used to control the speed and quality of training, and *Momentum* (m) is introduced to accelerate the optimization process.

By repeatedly performing the 3-phase cycle, we can make use of the training dataset to train a CNN model. It then can be applied to various artificial intelligence tasks to extract features.

B. RRAM Device and Crossbar

An RRAM device is a passive two-port element with a lot of resistance states, and multiple devices can be used to build the crossbar structure. As shown in Fig.1 , when the numbers of "matrix" can be mapped to the conductance of RRAM devices, and the numbers of "vector" can be represented by the voltage amplititudes, the RRAM crossbar will be able to perform the matrix-vector multiplication. The relationship of input voltages and output voltages [6] can be expressed as Equ. 7.

$$v_{\text{out}}(k) = \sum_{j=1}^{N} \frac{g(k,j)}{g_s} \cdot v_{\text{in}}(j) \quad (7)$$

Where $g(k,j)$ represents the conductance of RRAM which in k^{th} row and j^{th} column of the crossbar, g_s represents the load conductance which is used to read out the voltage. In Conv layer, a single convolutional kernel is mapped to one column of a crossbar, and different columns in one crossbar correspond to different Conv kernels.

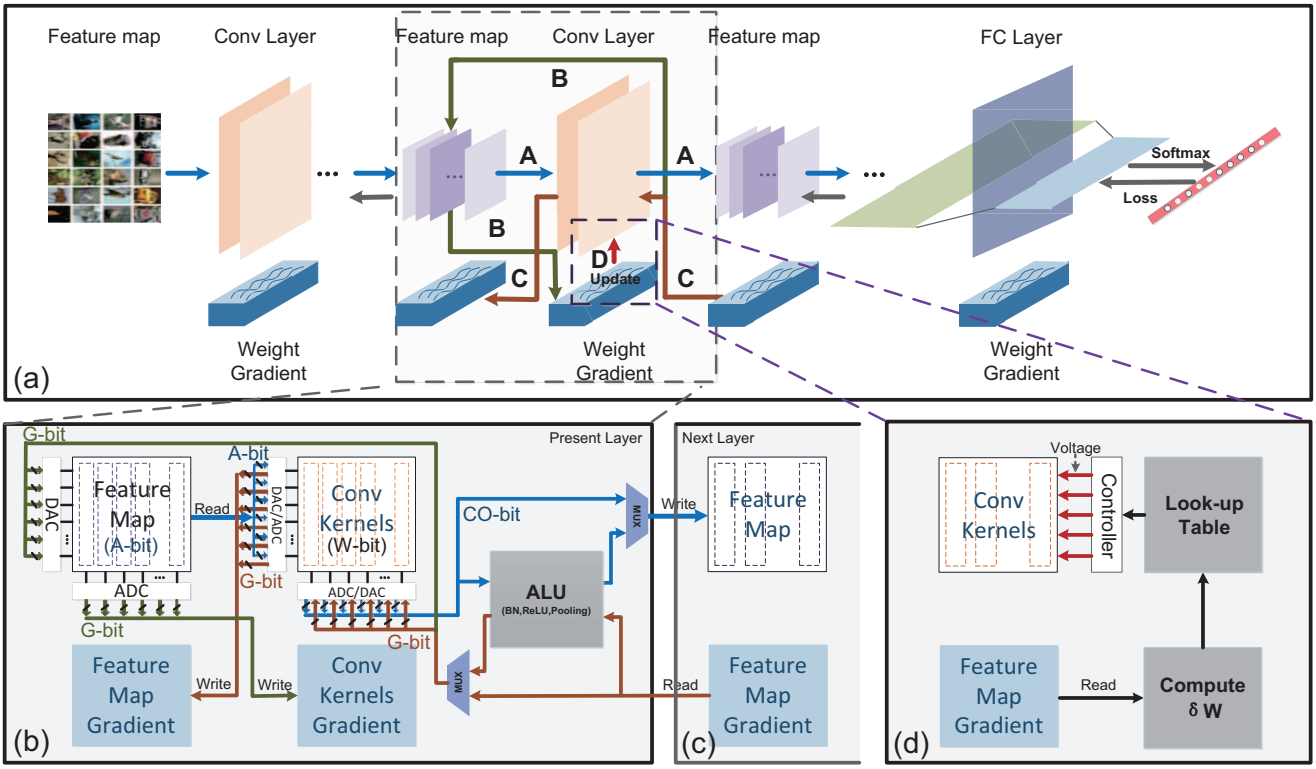

Fig. 2: Framework of RRAM-based low-bitwidth CNN training system.

III. RRAM-BASED LOW-BIT CNN TRAINING SYSTEM DESIGN

A. Framework and data path

Fig.2 shows the framework of proposed low-bitwidth CNN training system. The whole system is digital-analog mixed, the analog part of which is RRAM-based computing module and complete matrix-vector multiplications, and the digital part mainly includes the peripheral circuits around RRAM crossbars and arithmetic logic computing circuits which will finish complex calculations. The learnable weights (including convolution kernels, fully-connected weights, and bias) are stored in RRAM crossbars. To implement backpropagation basing on RRAM, we also store the activations on RRAM crossbars. The backpropagated gradients will also be buffered in RRAM, and fetched when updating the parameters in every iteration. The following paragraphs will give an explanation of the data path.

1) Inference: As shown in Fig. 2(a), line A shows the data path of inference. The inference is the process that the feature maps are forwardly propagated from the front layers to the back layers. If reshaping the convolution kernels and feature maps into matrices, the convolution operations are essentially matrix-matrix multiplications. The activations are mapped onto the input voltages, and the weights are mapped onto the conductance of RRAM devices. If using notations a to represent reshaped matrices of activations, K to represent reshaped matrices of convolutional kernels in the corresponding layer and l as an index of the layer, the mathematical expression of the forward process in Conv layer which is done on RRAM will be as Equ. 8.

$$\text{Forward\_conv}(a_{l-1}, K_l) = convout = a_{l-1} \cdot K_l \tag{8}$$

Then the convouts are sent to the ALU to implement complex arithmetic logic calculations to get the activations, or directly sent to next layer if there is no need to cascade additional functions, controlled by a multiplexer (MUX). After getting the activations, they are written into the RRAM crossbars of next layer which store feature maps.

2) Backpropagation: Line B and line C in Fig. 2 show the data path of backpropagation process. Gradients are calculated through partial derivative of loss functions and propagated from the back layers to the front layers. The first step of backpropagation in a layer is to read out the gradients w.r.t activations of the layer from buffers. Then it will be sent to the ALU to finish the calculations of backward BN, ReLU or Pooling. Then there are two operations in backpropagation process, one of which is calculating the gradients w.r.t. neurons (i.e. input feature maps) that is shown as line B, and the other is calculating the gradients w.r.t. weights that is shown as line C. The equations done on RRAM crossbars are respectively shown as Equ. 9,10.

$$\text{Backward\_conv}_{\text{input}}(\frac{\partial J}{\partial z_l}, K_l) = \frac{\partial J}{\partial a_{l-1}} = \frac{\partial J}{\partial z_l} \cdot K_l^T \tag{9}$$

$$\text{Backward\_conv}_{\text{weight}}(\frac{\partial J}{\partial z_l}, a_{l-1}) = \frac{\partial J}{\partial K_l} = a_{l-1}^T \cdot \frac{\partial J}{\partial z_l} \tag{10}$$

Where z represents the convout of Conv layer. In the matrix-matrix multiplication of Equ. 10, the activations are mapped on the RRAM crossbars which are stored in when forwardly propagated.

3) Parameters update: Line D in Fig. 2 shows the process of parameters update. After completing the calculations of backpropagation, the gradients w.r.t. weight parameters will be fetched from the buffers. Then the change amount of weights is calculated by Equ. 5. As the conductances of RRAM are changed by controlling the voltage pulses, we need firstly calculate the amplitude and duration of the applied voltages. Here we use a look-up table to determine

the writing voltages and schemes. The updating method is defined as below:

$$\mathrm{GetDeltaW}(\frac{\partial J}{\partial K}, \chi, m) = \Delta K = \chi \cdot \frac{\partial J}{\partial K} + m \cdot \Delta K_{last} \quad (11)$$

B. Quantization and conversion strategy

In our system, there are three types of data which need be converted or quantized: the input interfaces, the output interfaces of crossbars and the weights or feature maps stored in RRAM crossbars.

1) Quantization: Decreasing the bitwidth: As there exist variations on RRAM devices, to correctly read the resistance states of RRAM, certain intervals are set between adjacent resistance states to tolerate the variations. So if the parameters are mapped on the conductance of RRAM, we need firstly quantize them into low-bitwidth numbers. The quantizing strategy is to quantize a real number input $x \in [-1, 1]$ to a k-bit number, which is defined as below:

$$\mathrm{Quantize_k}(x) = \frac{1}{2^{k-1}-1}\mathrm{round}((2^{k-1}-1)\cdot x) \quad (12)$$

So before quantizing a number, we need to firstly linearly or nonlinearly transform it to be limited in the domain $[-1, 1]$. In this paper, scaling factors and truncation functions are used to limit the numbers to a specific range.

2) DACs: Conversion at input interfaces: As RRAM crossbar plays as an analog matrix-vector multiplier, it needs interfaces of DACs to be placed at input port to convert the digital signals to analog voltages. The digital data from peripheral circuits are fixed-point numbers. If the bitwidth of numbers exceeds k bits, we first truncate the low-weighted bits, then using the conversion strategy to converse the numbers as below:

$$\mathrm{DA\_convert_k}(d) = \alpha_{in} \cdot \sum_{i=0}^{k-1} 2^{-(i+1)} d[i] \quad (13)$$

Where k represents the bitwidth of digital numbers, $d[i]$ means the value of i^{th} bit satisfying $d[i] \in \{0, 1\}$, α_{in} is a scaling factor which can control the range of input voltage values.

3) ADCs: Conversion at output interfaces: The output port of RRAM crossbar also needs ADCs to transform analog current or voltage to digital fixed-point data. So the quantization and conversion strategy of output signals is shown as Equ. 14.

$$\mathrm{AD\_convert_k}(V_{out}) = \mathrm{Quantize_k}(\min\nolimits_1(\max\nolimits_{-1}(\frac{V_{out}}{\alpha_{out}}))) \quad (14)$$

where the function $\min_1(x)$ selects the smaller one in $\{1, x\}$, and $\max_{-1}(x)$ selects the larger one in $\{-1, x\}$, both of which are truncation functions to limit the number in the range of $[-1, 1]$. α_{out} is a scaling factor to scale the output voltages.

Through above strategies, the overall algorithm of RRAM-based training low-bitwidth CNN is stated as Algorithm 1.

C. Stochastic disturbance

RRAM's resistance is always affected by stochastic disturbance. Equ. 15 [16] shows the simplified model of non-linear $I - V$ relationship, where d is the average tunneling gap distance. V is the voltage across the RRAM device, and I is the current.

$$I = I_0 \cdot \exp(\frac{d}{d_0}) \cdot \sinh(\frac{V}{V_0}) \quad (15)$$

Simplifying this equation through Ohm's Law, Taylor series expansion and approximation (generally $V \ll V_0$), we can get the following equation:

$$R = \frac{V_0}{I_0}\exp(\frac{d}{d_0}) \quad (16)$$

Algorithm 1 Training a L-layer CNN with CO-bit convouts, W-bit weights and A-bit activations using G-bit gradients. The functions used in this algorithm is shown as above equations.

Require: a batch-size of training images, present weights $K^{W,t}$, learning rate χ, momentum m
Ensure: updated weights $K^{W,t+1}$

{1. Inference}
for $i = 1 : L$ **do**
  $a_{i-1}^{A} \leftarrow \mathrm{DA\_convert_A}\,(a_{i-1})$
  $z_i \leftarrow \mathrm{Forward\_conv}\left(a_{i-1}^{A}, K_i^{W,t}\right)$
  $z_i^{CO} \leftarrow \mathrm{AD\_convert_{CO}}\,(z_i)$
  Optionally cascade ReLU, Batch Normalization and pooling, $a_i \leftarrow h\left(z_i^{CO}\right)$.
end for
{2. Backpropagation}
for $i = L : 1$ **do**
  Back-propagate $g_{a,i}^{G}$ through ReLU, Batch Normalization or pooling if there is one, $g_{z,i} \leftarrow h_{backward}(g_{a,i}^{G})$.
  {2.1 bp1: gradients w.r.t activations}
  $g_{z,i}^{G} \leftarrow \mathrm{DA\_convert_G}\,(g_{z,i})$
  $g_{a,i-1} \leftarrow \mathrm{Backward\_conv_{input}}\left(g_{z,i}^{G}, K_i^{W,t}\right)$
  $g_{a,i-1}^{G} \leftarrow \mathrm{AD\_convert_G}\,(g_{a,i-1})$
  {2.2 bp2: gradients w.r.t weights}
  $g_{z,i}^{G} \leftarrow \mathrm{DA\_convert_G}\,(g_{z,i})$
  $g_{K_i^{W,t}} \leftarrow \mathrm{Backward\_conv_{weight}}\left(g_{z,i}^{G}, a_{i-1}^{A}\right)$
  $g_{K_i^{W,t}}^{G} \leftarrow \mathrm{AD\_convert_G}\left(g_{K_i^{W,t}}\right)$
end for
{3. Parameters update}
for $i = 1 : L$ **do**
  $\Delta K_i \leftarrow \mathrm{GetDeltaW}(g_{K_i^{W,t}}^{G}, \chi, m)$
  $K_i^{W,t+1} \leftarrow \mathrm{Quantize}_W(K_i^{W,t} - \Delta K_i)$
  $V_{pulse} \leftarrow \mathrm{LookUpTable}\left(K_i^{W,t+1}, K_i^{W,t}\right)$
  $\mathrm{WriteRRAM}\,(V_{pulse})$
end for

However, the gap distance is always affected by a random Gaussian disturbance $d_{disturb}$, satisfying a Gaussian distribution $d_{disturb} \sim \mathrm{N}(0, \sigma^2)$ [17]. Thus a real gap distance d satisfies $d = d_{expected} + d_{disturb}$. Thereby the resistance will also deviate from expected value. By Equ.16 we can get the deviation amount of resistance as below:

$$R_{expected} + R_{disturb} = R_0 \exp(\frac{d_{expected} + d_{disturb}}{d_0}) \quad (17)$$

$$\frac{R_{disturb}}{R_{expected}} = \frac{d_{disturb}}{d_0} \quad (18)$$

Where $R_{disturb}$ represents resistance deviation resulted from the disturbance, and $R_{expected}$ is the resistance value that we expect. As the parameters are mapped on the conductance of RRAM, so we further explore the disturbance of the conductance. Through the reciprocal relationship of resistance and conductance, we can get the following equation:

$$\frac{G_{disturb}}{G_{expected}} = -\frac{R_{disturb}}{R_{expected} + R_{disturb}} = -\frac{d_{disturb}}{d_0 + d_{disturb}} \quad (19)$$

From the above conclusion, as the gap distance d satisfies a Gaussian probability intense function $\mathrm{N}(0, \sigma^2)$, so the probability

TABLE I: The classification accuracy for MNIST test dataset with different combinations of bitwidth in LeNet-5. A, CO, W, G are bitwidth of activations, convouts, weights, and gradients.

A	CO	W	G	Accuracy without disturbance	Accuracy with disturbance
32[a]	32	32	32	0.9914	*
8	8	8	8	0.9828	0.9825
6	6	6	6	0.9745	0.9733
4	4	4	4	0.9767	0.9797
3	3	2	4	0.9687	0.9736
2	2	2	2	0.9670	0.9752
2	2	1	4	-[b]	-
2	2	2	1	0.9633	0.9647
1	1	2	1	0.9416	0.9375
1	1	1	1	-	-

[a] bitwidth=32 means 32-bit floating-point numbers.
[b] '-' means failing to train a convergent model under such bitwidth.

distribution of disturbance on conductance can be shown as below.

$$f_{\frac{G_{disturb}}{G_{expected}}}(x) = \frac{d_0}{\sigma\sqrt{2\pi}} \exp(-\frac{d_0^2 x^2}{2\sigma^2(1+x)^2}) \frac{1}{(1+x)^2} \qquad (20)$$

IV. Experimental Results

A. Experiment Setup

In this section, we constructed experiments to simulate the training of CNN models LeNet-5 and ResNet-20 on our system, respectively tested on MNIST and CIFAR-10 datasets. LeNet-5 is a simple model of a convolutional neural network, which is proposed by Yann LeCun [18] and usually applied in recognition of handwritten digits. ResNet-20 is a larger CNN model which proposed by MSRA [19], achieving great performance on complex computer vision tasks.

In experiments of training CNN with considering the disturbance, we set a uniformly distributed disturbance superimposed on the weights when each time they're updated. The disturbance satisfies a distribution of $K_{disturb}/K_{expected} \sim \mathrm{U}[-5\%, 5\%]$.

B. Accuracy: Evaluating the Performance Under Different Bitwidth

We explore the configuration space of combinations of the bitwidth of activations, convouts, weights and gradients by experiments of training LeNet-5 on the MNIST dataset and ResNet-20 on the CIFAR-10 dataset. We use the classification accuracy of the models to evaluate the efficacy. Furthermore, experiments of training with disturbance are also made to explore its impact on CNN's training.

Tab.I shows the prediction accuracy for MNIST test dataset and Fig.3 shows the accuracy curves for CIFAR-10 dataset. From the experimental results we can see that, in general, training with low-bitwidth (A,CO,W,G) will cause some degradation in recognition accuracy. But low-bitwidth CNN can reduce resource requirement, and be able to adapt to the constraints of a RRAM-based computing system.

Based on experimental results of LeNet-5 training, it can be figured out that there was no significant loss of accuracy when the bitwidth was going from 32-bit to 2-bit. While from the cases that failing to train an available model, we can observe that weights seems to be the most sensitive to bitwidth, and using weights with bitwidth $W = 1$ would lead to failed training, but the exact reason needs to be further explored in the future work. However, in more complex model ResNet-20, the accuracy significantly drops from 91.8% to 75.8% when the bitwidth going from $(32, 32, 32, 32)$ to $(32, 6, 6, 6)$. Nevertheless, under the bitwidth of $(32, 8, 8, 8)$, 86.26% classification accuracy is acceptable, as it's worthy dropping a few accuracy in exchange for higher energy efficiency.

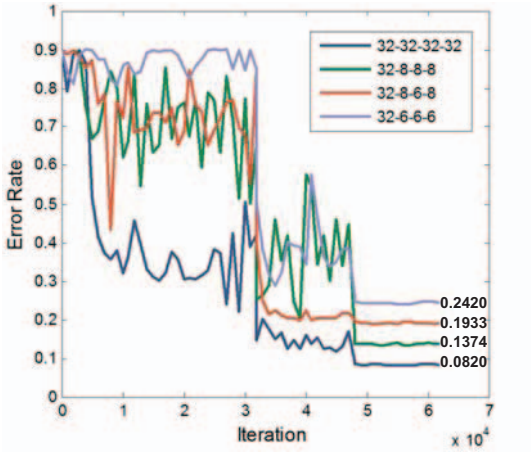

Fig. 3: Error Rate Curves of ResNet-20 on CIFAR-10: Accuracy Under Different Combinations of Bitwidth (A,CO,W,G).

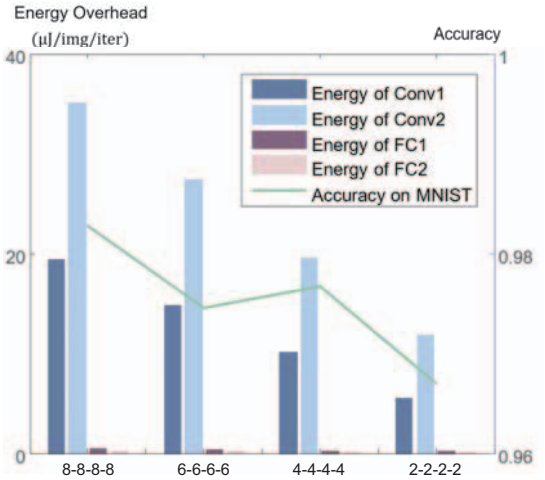

Fig. 4: Energy overhead estimation of RRAM Crossbars and accuracy of LeNet-5 on MNIST images under different combinations of bitwidth (A,CO,W,G). The bar diagram shows the statistical energy estimation, and the line shows the accuracy.

In the experiments of training with disturbance, we can conclude that the CNN's training has a great fault-tolerant ability. Although affected by the disturbance, there isn't significant loss on the accuracy, at most 0.41%. Moreover, in some cases of CNN training with low-bitwidth numbers, the accuracy even enhances. For example, the accuracy under bitwidth $(2, 2, 2, 2)$ increases from 96.7% to 97.52%. It's inferred that the disturbance somehow improves the generalization of the neural network. Previous works have made efforts to theoretically study the impact of noises and disturbances on training [20], [21].

C. Tradeoff: balancing between accuracy and energy overhead

Based above experimental results, we cannot achieve high accuracy and high energy efficiency at the same time. Therefore finding a balance between accuracy and energy overhead is important. Fig.4 shows the energy overhead estimation and simulated accuracy of LeNet-5 on MNIST dataset under selected combinations of bitwidth. As shown in the figure, the energy overhead increases with the

TABLE II: Energy Overhead Estimation of CNNs in Different Training Platforms

Database	Platform	CNN Model	Energy(μJ/img/iter)		
MNIST	CPU[a]	LeNet-5	Conv+FC	7997.6	88.7%
			Others	1015.6	11.3%
			All	9013.2	100.0%
	GPU[b]		Conv+FC	11824.9	96.0%
			Others	491.3	4.0%
			All	12316.2	100.0%
	RRAM		Conv+FC	44.96	8.4%
			Others	491.3	91.6%
			All	536.26	100.0%
CIFAR-10	CPU	ResNet-20	Conv+FC	262523.4	77.9%
			Others	74414.1	22.1%
			All	336937.5	100.0%
	GPU		Conv+FC	133066.9	79.4%
			Others	34465.2	20.6%
			All	167532.1	100.0%
	RRAM		Conv+FC	3653.7	9.6%
			Others	34465.2	90.4%
			All	38118.9	100.0%

[a] Intel(R) Core(TM) i7-6900K CPU @ 3.20GHz.
[b] NVIDIA TITAN X (Pascal).

bitwidth of numbers, and the recognition accuracy also conforms to the same trend. Based on the observations, if we want to save more energy, simultaneously achieving a comparable accuracy, it's recommended that choosing the combination of bitwidth as (A,CO,W,G)=$(4, 4, 4, 4)$, of which the accuracy can achieve 97.67% without disturbance and 97.97% with disturbance. While if the devices have very large demands on energy efficiency and are less sensitive to accuracy, $(2, 2, 2, 2)$ will be a better choice.

We compared the energy overhead of training LeNet-5 and ResNet-20 in different platforms, which are shown in Table.II. The overhead of CPU and GPU are estimated by $Power \times RunningTime$ which is measured by running the training tasks of full-precision models on corresponding machines. The energy consumption of RRAM cell is estimated by the formulas in Nvsim [22]. And the energy consumptions of digital arithmetic logic units and memory access are estimated as the same energy consumption of corresponding calculations of GPU.

From Table. II we can see, the energy overhead of processing per MNIST image per iteration on LeNet-5 model is estimated to be 536.26μJ, and the images of CIFAR-10 running on ResNet-20 network are estimated to be 38118.9 μJ/img/iter. The energy efficiency of proposed system has been greatly improved compared to traditional computing platforms, as the energy efficiency of proposed system is around $16.8\times$ higher than CPU and $23.0\times$ higher than GPU when processing LeNet-5 training task. And in a task of training ResNet-20 on CIFAR-10, the energy efficiency is $8.9\times$ and $4.4\times$ higher than CPU and GPU respectively.

We can also conclude from Table. II that the RRAM-based system achieve great improvement on energy efficiency compared with CPU and GPU when processing the computing of convolutional calculations and FC multiplications, around $36.4\times$ higher than GPU on ResNet-18 training. The other complex calculations which cannot be implemented by RRAM become the main factor which impedes the further improvement of energy efficiency.

V. Conclusions and Future Work

In this paper, we propose an RRAM-based low-bit width CNN training system, which is potentially applied in more complex tasks except MNIST and CIFAR-10 recognitions. We propose the algorithm of training low-bitwidth CNNs. We also explore the configurable space of combinations of bitwidth, and propose a tradeoff between the recognition accuracy and energy overhead. Besides, the stochastic disturbance existing on RRAM devices and its mathematical distribution is discussed. We evaluate the energy efficiency of proposed system. The estimated results show that the system can achieve significant improvement in energy efficiency for CNN training tasks.

Our future work will focus on improving the training algorithm to enhance accuracy, and make comparisons with more related work. Moreover, RRAM-based logic computing design is also intened to be achieved, as the neural computing includes many complex logic operations besides matrix-vector multiplications.

References

[1] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[2] J. Fan *et al.*, "Human tracking using convolutional neural networks." *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, 2010.

[3] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[4] A. Karpathy *et al.*, "Deep visual-semantic alignments for generating image descriptions," in *Computer Vision and Pattern Recognition*, 2015.

[5] B. Li *et al.*, "Merging the interface: Power, area and accuracy co-optimization for rram crossbar-based mixed-signal computing system," in *DAC*, 2015, p. 13.

[6] L. Xia *et al.*, "Selected by input: Energy efficient structure for rram-based convolutional neural network," in *DAC*, 2016.

[7] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, vol. 43, 2016.

[8] A. Shafiee *et al.*, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ISCA*, 2016.

[9] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[10] R. Degraeve *et al.*, "Causes and consequences of the stochastic aspect of filamentary rram," *Microelectronic Engineering*, vol. 147, pp. 171–175, 2015.

[11] M. Courbariaux *et al.*, "Binarized neural network: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[12] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," *arXiv preprint arXiv:1603.05279*, 2016.

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

[14] T. Tang *et al.*, "Binary convolutional neural network on rram," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 782–787.

[15] M. Cheng *et al.*, "Time: A training-in-memory architecture for memristor-based deep neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 2017, p. 26.

[16] Z. Jiang *et al.*, "A compact model for metal–oxide resistive random access memory with experiment verification," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 1884–1892, 2016.

[17] S. Yu *et al.*, "Scaling-up resistive synaptic arrays for neuro-inspired architecture: Challenges and prospect," in *Electron Devices Meeting (IEDM), 2015 IEEE International*. IEEE, 2015, pp. 17–3.

[18] Y. LeCun *et al.*, "Comparison of learning algorithms for handwritten digit recognition," in *International conference on artificial neural networks*, vol. 60. Perth, Australia, 1995, pp. 53–60.

[19] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[20] C. M. Bishop, "Training with noise is equivalent to tikhonov regularization," *Neural computation*, vol. 7, no. 1, pp. 108–116, 1995.

[21] A. F. Murray and P. J. Edwards, "Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Transactions on neural networks*, vol. 5, no. 5, pp. 792–802, 1994.

[22] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.