Efficient and Robust Neuromorphic Computing Design

by

Yandan Wang

Bachelor of Engineering, Chongqing University, 2010

Master of Engineering, North China Electric Power University, 2013

Submitted to the Graduate Faculty of the

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Yandan Wang

It was defended on

November 13, 2019

and approved by

Zhi-Hong Mao, Ph.D., Professor, Department of Electrical and Computer Engineering and Department of Bioengineering

Hai (Helen) Li, Ph.D., Associate Professor, Department of Electrical and Computer Engineering, Duke University

Jingtong Hu, Ph.D., Assistant Professor, Department of Electrical and Computer Engineering

Samuel J Dickerson, Ph.D., Assistant Professor, Electrical and Computer Engineering

Bo Zeng, Ph.D., Associate Professor, Industrial Engineering and Electrical and Computer Engineering

Dissertation Director: Hai (Helen) Li, Associate Professor,
Department of Electrical and Computer Engineering, Duke University

Efficient and Robust Neuromorphic Computing Design

Yandan Wang, PhD

University of Pittsburgh, 2019

In recent years, brain inspired neuromorphic computing system (NCS) has been intensively studied in both circuit level and architecture level. NCS has demonstrated remarkable advantages for its high-energy efficiency, extremely compact space occupation and parallel data processing. However, due to the limited hardware resources, severe IR-Drop and process variation problems for synapse crossbar, and limited synapse device resolution, it's still a great challenge for hardware NCS design to catch up with the fast development of software deep neural networks (DNNs).

This dissertation explores model compression and acceleration methods for deep neural networks to save both memory and computation resources for the hardware implementation of DNNs. Firstly, DNNs' weights quantization work is presented to use three orthogonal methods to learn synapses with one-level precision, namely, distribution-aware quantization, quantization regularization and bias tuning, to make image classification accuracy comparable to the state-of-the-art. And then a two-step framework named group scissor, including rank clipping and group connection deletion methods, is presented to address the problems on large synapse crossbar consuming and high routing congestion between crossbars.

Results show that after applying weights quantization methods, accuracy drop can be well controlled within negligible level for MNIST and CIFAR-10 dataset, compared to an ideal system without quantization. And for the group scissor framework method, crossbar area and routing area

could be reduced to 8% (at most) of original size, indicating that the hardware implementation area has been saved a lot. Furthermore, the system scalability has been improved significantly.

Table of Contents

List of Tables

List of Figures

Acknowledgments

Before diving into the details of this dissertation, I would like to express my deepest appreciation to my supervisor Hai (Helen) Li, an expert in neuromorphic computing and machine learning field. The research work in this dissertation can't be finished successfully without the careful guidance and financial support from my supervisor. And I would also give my deepest thankful to my co-supervisor Zhi-Hong Mao, an extremely admirable professor who is always caring for students and helping students at every moment.

I also would like to thank my committee members, Prof. Jingtong Hu, Prof. Samuel J Dickerson and Prof. Bo Zeng. Your advice during my dissertation meeting and your reputation in your field inspire me to dive deep into my research and solve many really interesting and meaningful topics.

1.0 Motivation

NCS demonstrates many important features including high computing efficiency, extremely low power consumption, and compact volume [1]. Integrating emerging technologies potentially enables a more compact and energy-efficient platform for information processing [2]. For instance, the two-terminal nonlinear memristor presents a series of advantages (Aojun Zhou 2017) of good scalability, high endurance and ultra-low power consumption [3]. Thus, it is taken as a promising candidate for neuromorphic computing system development.

The record-breaking classification performance of deep neural networks (DNNs) below below [4] in recent years has stimulated the fast-growing research on hardware design of NCS [5][6][6][7][8][9][2]. NCS utilizes device and circuit components to construct neural networks and therefore perform intelligent tasks, such as image classification, speech recognition and natural language processing. Circuit-level and architecture-level NCS designs using emerging memristor devices [10] and traditional CMOS technologies [6] are being explored. In software applications, the depth of DNNs rapidly grows from several layers to hundreds or even thousands of layers [11]. However, the scale of NCS hardware design falls far behind. A critical issue that obstructs the scaling-up of NCS is the limited synaptic connections (e.g., crossbar) in hardware implementation and induced heavy wire congestion (e.g., the routing between crossbars). Taking the memristor-based NCS as an example, due to IR-drop and process variations, both reading and writing reliability will be severely degraded when the size of a memristor-based crossbar is beyond 64x64 [12][13]. The similar scenario can be observed in CMOS-based conventional designs. For example, the IBM TrueNorth chip, as a pioneer in NCS design, limits the size of neurosynaptic crossbars to 256x256 [6]. It is inevitable to interconnect multiple crossbars to implement modern

large neural networks. The increasing scale of neural networks could quickly exhaust the resources of synapse crossbars and deteriorate the wire congestion [14][15]. Solutions have been explored to solve above issues. Akopyan et al. tend to map logically-connected cores to physically adjacent cores to reduce spike communications [15]. Such a core placement optimization cannot reduce the core number. Existing NCS optimization based on sparse neural networks can alleviate the wire congestion [15]. However, the separation of the software sparsification and hardware deployment makes the optimization very challenging.

At the same time, neuromorphic hardware implementations usually face a major challenge on system accuracy. TrueNorth, for example, allows only a few synaptic weights (e.g., 0, ±1, ±2). Accuracy degradation is inevitable when directly deploying a learned model to the system with limited precision [1]. The situation remains in memristor (or RRAM) based design. Theoretically, nanoscale memristor can obtain continuously analog resistance. While, a real device often can achieve only several stable resistance states [16]. The distinction between theoretical and actual properties results in significant accuracy loss. Extensive studies on learning low-resolution synapses have been performed to improve the accuracy of neuromorphic systems. Wen et al. presented a new learning method for IBM TrueNorth platform which biases the learned connection probability to binary states (0/1) to hinder accuracy loss [9]. Neural networks with binary resolution are more suitable for generic platforms [17][18][19]. BinaryConnect [18] as an example can achieve comparable accuracy in deep neural networks. However, neither TrueNorth nor BinaryConnect are pure binary neural networks: TrueNorth relies on the ensemble averaging layer in floating-point precision while the last layer of BinaryConnect is a floating-point L2-SVM.

1.1 Problem Statement

As stated before, NCS can fully utilize emerging devices and synapse crossbars to implement deep neural networks, which demonstrates lots of important features such as high computing efficiency, extremely low power consumption, and compact volume. However, due to the scaling up of DNN and the imperfect features of both synapse crossbar and single memristor, there still exists plenty of issues when mapping between software DNN and hardware implementation NCS design, which largely impedes the development of NCS. Furthermore, due to the limited resolution of hardware synapse device, classification accuracy can be severely affected when applying neural networks to NCS hardware implementation. Based on the above analysis, the main issues and challenges that should be addressed in this dissertation can be summarized as follows:

- **High crossbar area occupation:** As the scale of modern neural network grows from several layers to hundreds of or even thousands of layers, the hardware realization area of NCS implementation will be inevitable to grow fast. However, the hardware resources are always very limited, which can hardly catch up the scaling up speed of software level. In such as situation, the hardware resources will be easily and quickly exhausted. Thus, how to design an efficient NCS is extremely important in circuit level and architecture level when mapping the software level neural networks to hardware neuromorphic computing design.
- **Heavy routing congestion:** For both traditional CMOS based NCS design and emerging memristor NCS based design, they all suffer from severe writing and reading reliability degradation with the increase of crossbar size. Therefore, in the implementation of neural networks, we should divide the large neural layer into many small crossbars, whose size is

in the safe scope. As we need lots of small crossbars to implement a large neural network layer, huge of interconnection routing will be induced among different crossbars. Even though memristor is a nanoscale device and crossbar is compact manufactured, heavy routing connection will still occupy a large amount of area for the whole NCS design.

- **Accuracy degradation:** In algorithm level, weights are stored in floating-point type which can achieve a high classification accuracy for neural network system. However, in circuit level, weight representative devices usually have limited resolution, which can only represent several discrete number of weights. For example, in memristor based NCS, theoretically, memristor can perform analog continuous memristive and thus can represent floating-point synapse weights. However, in reality, memristor only can obtain two stable memristive states. Thus, when mapping algorithm level floating-point weights to circuit level representatives, there exists inevitable large classification accuracy degradation.

1.2 Research Contributions

Research contributions for this dissertation can be concluded as:

- A two-step framework named *group scissor* is proposed to overcome high crossbar area occupation and heavy routing congestion issues.
    - The first step, rank clipping, integrates low-rank approximation into the training process of neural networks. It targets to reduce the dimensions of connection arrays in a group-wise way and therefore reduce the consumption of synapse crossbars in NCS.

- The second step, group connection deletion, structurally deletes/prunes groups of connections. The approach directly learns sparse neural networks friendly to hardware and therefore deletes the routing wires between crossbars.

- Pure binary (1-level precision1) neural networks are proposed to address accuracy degradation issue caused by limited hardware synapse device resolution. While the realization of continuous analogue resistance states is still challenging, the 1-level precision is well supported by most of memory materials and architectures. Three orthogonal methods of leaning 1-level precision synapses and tuning bias to improve image classification accuracy are proposed:
  - Distribution-aware quantization discretizes weights in different layers to different values. The method is proposed based on the observation that the weight distributions of a network by layers.
  - Quantization regularization directly learns a network with discrete weights during training process. The regularization can reduce the distance between a weight and its nearest quantization level with a constant gradient.
  - Bias tuning dynamically learns the best bias compensation to minimize the impact of quantization. It can also alleviate the impact of synaptic variation in memristor based neuromorphic systems.
- Deformable quantization regularization method is presented to control the image classification accuracy loss under a negligible value for small and simple dataset such as MNIST or CIFAR-10, or under an acceptable value for larger and more complex dataset such as CIFAR-100, ImageNet, etc. This regularization method will combine the traditional l1-norm or l2-norm regularization method and the newly used quantization

regularization method. Therefore, this new deformable quantization regularization method will behave the characteristics and advantages of both l1-norm (l2-norm) and quantization regularization methods.

1.3 Dissertation Organization

This dissertation is organized as follows:

Chapter 2.0 introduces some related work of model compression and acceleration.

Chapter 3.0 introduces some background information for this dissertation, mainly about the neural network models, memristor technology and neuromorphic computing system.

Chapter 4.0 explores a framework named group scissor to overcome high crossbar area occupation and heavy routing congestion issues. Two steps are presented in group scissor including rank clipping and group connection deletion, which address the high crossbar area occupation and heavy routing congestion, respectively.

Chapter 5.0 presents three methods to improve image classification accuracy with one-level precision synapse for neuromorphic computing system. These three methods are distribution-aware quantization, quantization regularization and bias tuning.

Chapter 6.0 presents the deformable quantization regularization method to further control the image classification accuracy loss.

Chapter 7.0 presents a novel true random number generator design leveraging emerging memristor technology.

Chapter 8.0 concludes all the research works in this dissertation.

2.0 Related Work

In recent years, lots of excellent neural network models, such as AlexNet [4], VGG [31], GoogLeNet [32], ResNet [33], have been proposed and quickly becomes popular both in academia and industry, where excellent performance can be achieved in many artificial intelligence fields such as computer vision, image classification, speech recognition and natural language processing. All these outstanding models own one common feature that is large size and complex structure with a large amount of parameters, which is beneficial for model performance improvement, however on the contrary, it becomes a big obstacle to the hardware deployment for these models since portable devices usually have limited memory and computation resources. Although there exits lots of difficulties in neural network model deployment, we still face a rapid increasing on artificial intelligence applications especially in fields like mobile devices, drone, AR/VR devices and self-driving car. In order to meet the increasing demand of deployment for excellent performance but large size models, there is an explosive growth research on how to use low-precision representatives to represent full precision ones with small or negligible loss of performance. This low-precision learning process can be achieved by quantizing full precision representatives to low-precision ones.

Quantization operations can be applied to weights, activations, gradients or other representatives in neural network models. Low-precision representatives could be binary, ternary, 2bits, 4bits, 8bits or some other reasonable low bits. For example, BinaryConnect [34], Binarized Neural Networks [35] and XNOR-Net [36] all constrains weights or activations to +1 or -1. The very beginning work BinaryConnect encounters a relatively high accuracy loss on image dataset by quantizing only weights to +1 or -1. Based on BinaryConnect, XNOR-Net further introduces a

scaling factor to quantized weights, which largely improves the image classification accuracy. Besides the weights quantization done in BinaryConnect and XNOR-Net, Binarized Neural Networks constrained both weights and activations to +1 and -1, which inevitably results in a relatively high accuracy drop on more complex dataset like CIFAR-10. Some other work such as Trained Ternary Quantization [37], Deep Compression [38], fixed point quantization [39], fine-grained ternary quantization [40] all quantize weights or activations to multiple bits representatives. It's obvious that increasing quantized bits can improve the compressed model performance, however, at the same time, it can also consume more hardware resources. Unlike previous stated work, [41] proposes a novel incremental network quantization (INQ) method to quantize pre-trained full-precision convolutional neural network (CNN) model to low-precision one, where the quantized values are either powers of two or zero. While powers of two or zero are very friendly to hardware implementations, since the computation involving these numbers can be easily handled by shift operations.

Weights and activations quantization can speed up training and inference, save hardware memory and power resource requirements, however, there still exists a big issue that is nondifferentiable optimization in backpropagation stage during training [42]. To avoid this issue, some work like [39][43] conducts quantization operation after model training. Other work like [17][18][41] tries to solve this issue by utilizing a continues approximated function to approximate the quantized function during back propagation stage. What's more, a novel structure named relaxed quantization neural networks are put forward in [44], which introduces a differentiable quantization procedure for better gradient-based optimization.

In most recent one to two years, lots of other quantization work incorporating some emerging optimization method are also arising. For example, [45] put forward a new model

compression method, which combines weights quantization with novel distillation technology to compress the network model from larger “teacher” networks to smaller “student” networks. And in [38], trained quantization and weight sharing methods are applied based on the pre-pruned NN model. To address the gradient mismatch issue between forward and backward approximations, [42] proposes a half-wave Gaussian quantizer (HWGQ) for forward approximation and meanwhile utilizes batch normalization and activations statistics. From the aspect of bandwidth, [46] proposes an 8-bit approximation algorithms to compress 32-bit gradients and nonlinear activations to 8-bit approximations to better use the available bandwidth. To take inference stage into consideration, [47] proposes a new quantization method to use integer-only arithmetic, which can make inference more efficient compared with floating-point arithmetic operations.

3.0 Background

3.1 Neural Network Models

Neural networks (NNs) are a series of models inspired by biological neuron networks. The function can be formulated as:

$$\boldsymbol{y} = \boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b} \quad (1)$$

$$\boldsymbol{z} = \boldsymbol{h}(\boldsymbol{y}) \quad (2)$$

where the output neuron vector *z* is determined by the input neuron vector *x*, the weight matrix of connections $\boldsymbol{W}$ and the bias vector *b*. Usually, $h(\cdot)$ is a non-linear activation function and all the data in (1) and (2) are in floating-point precision.

3.2 Memristor Technology

Memristor, firstly introduced by Professor Leon Chua in 1971, is regarded as the fourth fundamental circuit element, representing the dynamic relationship between the charge $q(t)$ and the $\varphi(t)$ [20]. Most significantly, the total electric flux flowing through a memristor device can be "remembered" by recording it as its *memristance* (*M*). In 2008, HP Lab demonstrated the first actual memristor through a TiO_2 thin-film device and realized the memristive property by moving its doping front [10].

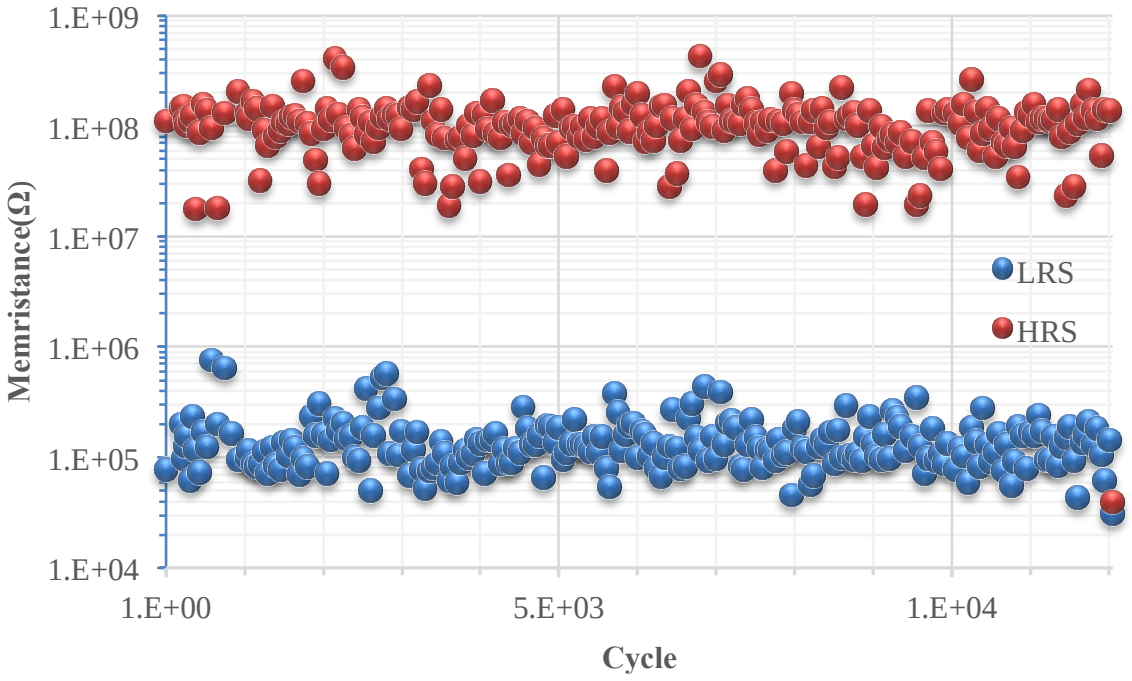

Figure 1 Statistical memristance distributions of a TiO2 device

Theoretically, a memristor device can achieve continuous analog resistance states. However, the imperfection of fabrication process causes variations and therefore memristance varies from device to device. Even worse, the memristance of a single memristor changes from time to time [21]. In most system designs, only two stable resistance states, high- and low-resistance state (HRS and LRS), are adopted. As the real statistical measurement data of a TiO2 memristor in Figure 1 shows, the distribution of HRS (LRS) follows an approximated lognormal *probability density function* (PDF) [16].

3.3 Neuromorphic Computing Systems

Neuromorphic computing systems (NCS) represents the hardware implementations of NNs by mimicking the neuro-biological architectures. For example, IBM TrueNorth chip is made of a network of neuro-synaptic cores, each of which includes a configurable synaptic crossbar connecting 256 axons and 256 neurons in close proximity [1]. The synaptic weight in the crossbar can be selected from 4 possible integers. Memristor based NCS has also be investigated [22]. Matrix-vector multiplication, the key operation in NNs, can be realized by memristor crossbar arrays as illustrated in Figure 2 [14]. The conductance matrix of memristor crossbar array is utilized as the weight matrix of NNs [22].

The synaptic weights in these neuromorphic computing systems usually have a limited precision, constrained either by design cost (e.g., the SRAM cells for each weight representation in TrueNorth) or current technology process (e.g., two or only a few resistance levels of memristor devices). As such, the classification accuracy loss could be very significant in NCS. To improve

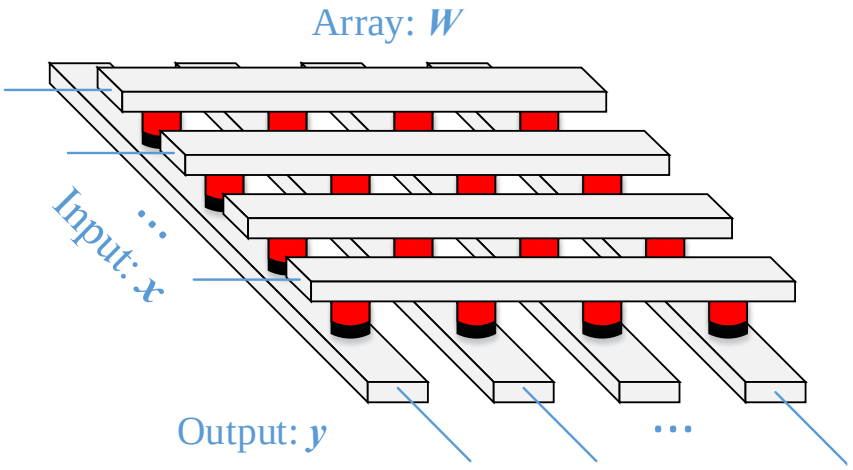

Figure 2 Mapping neural networks to memristor crossbar array

the classification accuracy, lots of research has been done [18][19][23]. Even though, some of them have floating-point layers and some ignore circuit design constraints. In this work, we focus on pure binary neural networks considering the constraints in NCS hardware implementation. Figure 3 (a) illustrates the implementation of a convolutional layer in neural network using memristor-based crossbars (MBC), where memristors (a.k.a. synapses) in each column encode the weights of one filter [24]. The implementation of a fully-connected layer utilizes the similar structure, but each column realizes the connections to one output neuron. As the size of crossbars is limited, implementing large neural networks requires a high volume of crossbars and the induced interconnection. Figure (b) depicts a circuit-level implementation of a large layer by tiling and interconnecting MBC [14]. As the scale of modern neural networks grows, the high crossbar area occupation and heavy routing congestion emerge as critical issues that obstruct the scalability of the hardware implementation.

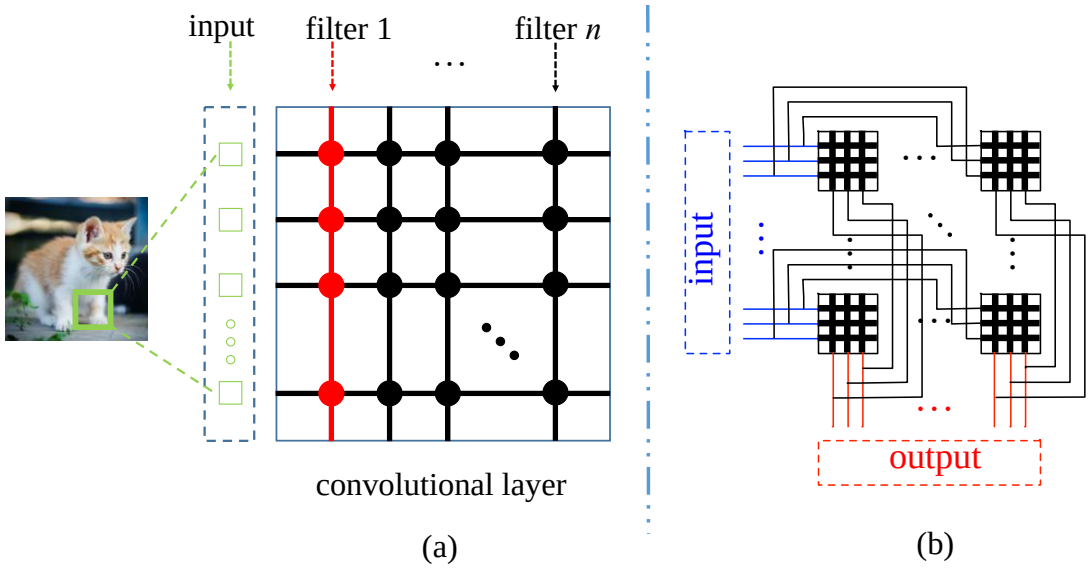

Figure 3 The NCS designs for (a) a small convolutional layer, and (b) a large layer

4.0 The Group Scissor Framework

In this work, we propose the Group Scissor framework to improve the scalability of neuromorphic computing design. The framework consists of two steps: rank clipping to reduce crossbar area occupation and group connection deletion for routing congestion reduction. The details of the proposed design are described in this section. Moreover, the estimations of circuit area and routing wires for MBC-based neuromorphic design are formulated.

4.1 Rank Clipping

As discussed above, the high crossbar area occupation and heavy routing congestion are the major challenges in realizing large neural networks. We propose to utilize low-rank approximation (LRA) to reduce the dimensions of weight (connection) matrices in large neural networks. LRA is a mathematical technique that uses the product of smaller matrices with reduced rank to approximate a given large matrix. Specifically, an original weight matrix $W \hat{I} \square^{N' M}$ can be approximated as:

$$W \approx U \cdot V^T = \widetilde{W} \quad (3)$$

Where $U \in \mathbb{R}^{N \times K}$, $V^T \in \mathbb{R}^{K \times M}$, and K is the rank of the approximation. When K << M, U and V are reduced to skinny matrices. The total crossbar area occupation can be reduced when K satisfies:

$$K < \frac{NM}{N+M} \quad (4)$$

There are various LRA techniques. Without losing generality, commonly used principal components analysis (PCA) [25] and singular value decomposition (SVD) [13] are adopted as the representatives in this work.

The PCA approach is formulated in Algorithm 1. Its essence is a linear projection from a high dimensional space ($\mathrm{w_n} \in \mathbb{R}^{\mathrm{M}}$) to a lower dimensional subspace ($\mathrm{u_n} \in \mathbb{R}^{\mathrm{K}}, \mathrm{K} \ll \mathrm{M}$) to minimize the reconstruction error of W, where $\mathrm{w_n}$ and $\mathrm{u_n}$ are the n-th row of W and U, respectively, and V is the basis of the subspace. The reconstruction error is

$$\mathbf{e_K} = \frac{\|\mathbf{W}-\widetilde{\mathbf{W}}\|^2}{\|\mathbf{W}\|^2} = \frac{\sum_{\mathbf{m=K+1}}^{\mathbf{M}} \boldsymbol{\lambda_m}}{\sum_{\mathbf{m=1}}^{\mathbf{M}} \boldsymbol{\lambda_m}} \quad (5)$$

where $\|\cdot\|$ is the Euclidean norm, namely Euclidean distance.

Algorithm 1: Principal Components Analysis (PCA)

Input: $N \times M$ matrix **W**, and rank K

1 Get mean of rows w_n $\forall n \in [1 \cdots N]$: $\mu = \frac{1}{N}\sum_{n=1}^{N} \mathrm{w}_n$,

2 Centralize the data: replace each w_n with $w_n - \mu$;

3 Calculate the $M \times M$ covariance matrix: $\mathrm{C} = \frac{\mathrm{W}^T\mathrm{W}}{N-1}$

4 Calculate the eigenvectors v_m and eigenvalues λ_m of covariance matrix C: $\mathrm{C}_{\mathrm{v}_{m=}} = \lambda_m \mathrm{v}_m, \forall m \in [1 \cdots M]$;

5 Project to subspace: $N \times K$ matrix U = WV, where $V = [\mathrm{v}_1, \cdots, \mathrm{v}_K]$ is a $M \times K$ matrix and $\mathrm{v}_1, \cdots, \mathrm{v}_K$ are eigenvectors corresponding to the largest K eigenvalues;

Output: $N \times M$ approximation matrix $\widetilde{\mathrm{W}}=\mathrm{U}\cdot\mathrm{V}^T$

Though LRA can approximately reconstruct the original weights, small perturbation in weights can deteriorate the classification accuracy. We compares the performance of the original baseline design (Original) and the low-rank networks which are directly decomposed by PCA

Table 1. Accuracy and ranks

Database	Net	Method	Accuracy		conv1†	conv2	conv3	fc1†	fc2
MNIST	*LeNet* [16]	*Original*	99.15%	‡	20	50	–	500	10
		Direct LRA	96.44%	Rank K	5	12	–	36	10
		Rank clipping	99.14%		5	12	–	36	10
CIFAR-10	*ConvNet* [1]	*Original*	82.01%		32	32	64	10	–
		Direct LRA	43.29%	Rank K	12	19	22	10	–
		Rank clipping	82.09%		12	19	22	10	–

† Conv1 denotes the first convolutional layer, fc1 is the first fully-connected layer, and so forth.
‡ The corresponding rank indicates the number of filters in convolutional layers or the number of output neurons in fully-connected layers.

(Direct LRA). The accuracy drops rapidly after applying Direct LRA. Fine-tuning (retraining) the low-rank neural networks can recover accuracy, but the optimal ranks in all layers are unknown. More importantly, it is very time-consuming to explore the entire design space by decomposing and retraining a wide variety of neural networks. We propose the LRA-based rank clipping that not only can successfully retain the accuracy but also can automatically converge to the optimal low ranks in all layers. Lower ranks are actually obtained by our rank clipping method as shown in the following table.

The key idea of rank clipping is illustrated in Figure 4 Rank clipping for crossbar area occupation reduction. Rather than direct LRA after training, we integrate LRA into the training process, carefully clip some ranks with small reconstruction errors, followed by a fixed number of training iterations, say, S iterations. The gentle clipping induces small reconstruction errors and thus slightly affect the classification accuracy, which could be recovered by the following S iterations. The iteration of clipping and training not only avoids irremediable accuracy degradation but also enables neural networks to gradually converge to the optimal ranks for all layers.

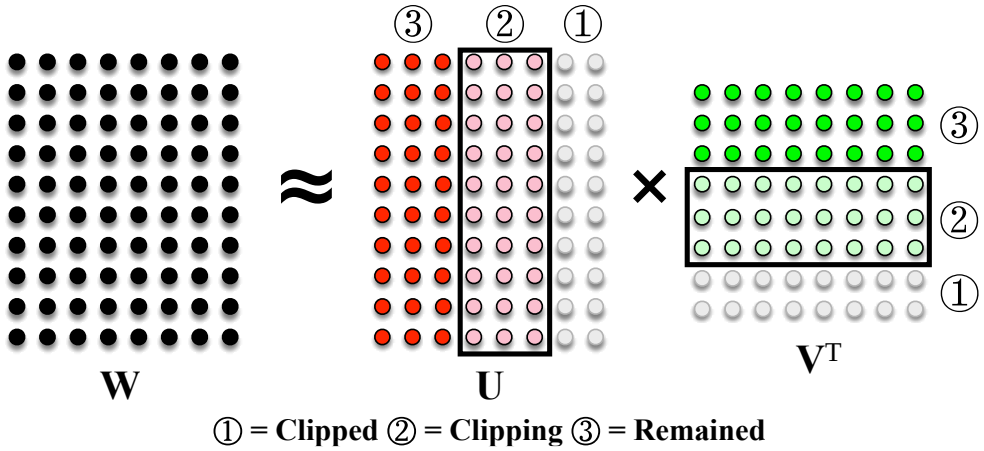

Figure 4 Rank clipping for crossbar area occupation reduction

Algorithm 2 describes the detailed operation of the rank clipping. The tolerable clipping error, ϵ, is the maximum allowable reconstruction error of each rank clipping. A gentle clipping can be enabled by setting a small ϵ, e.g., 0:01.

Algorithm 2: Rank Clipping

Input : Trained original neural network, ε, maximum training iteration I, clipping step S

for *each layer* l **do**
  PCA of weight matrix $\mathbf{W}_l = \mathbf{U}_l \cdot \mathbf{V}_l^T$ with full rank $K_l = M_l$;
end
while $i = 1; i < I; i = i + S$ **do**
  for *each layer* l **do**
    PCA of $\mathbf{U}_l = \hat{\mathbf{U}}_l \cdot \hat{\mathbf{V}}_l^T$ using the minimum rank $\hat{K}$ which satisfies $e_{\hat{K}} \leq \varepsilon$;
    if $\hat{K} < K_l$ **then**
      $K_l = \hat{K}$; $\mathbf{U}_l = \hat{\mathbf{U}}_l$; $\mathbf{V}_l^T = \hat{\mathbf{V}}_l^T \cdot \mathbf{V}_l^T$
    else
      continue;
    end
  end
  Train the neural network for S iterations;
end
Output: Clipped low-rank neural network with approximation $\mathbf{W}_l = \mathbf{U}_l \cdot \mathbf{V}_l^T$ for each layer l

Rank clipping starts with a full-rank LRA. It iteratively examines if the low-dimensional U can be further projected to a lower-rank subspace with only reconstruction error of ϵ. Note that PCA is used as the representative of LRA in Algorithm 2. Other LRA methods like SVD can also

be used. The only modification is to replace the approximation of weight matrix by other LRA methods.

Figure 5 plots the trends of rank reduction and accuracy retention of LeNet in Table 1, during PCA-based rank clipping. Rank clipping is examined every S = 500 iterations with $\epsilon = 0.03$. In the figure, the rank ratio is defined as the remained rank over full rank, i.e., K/M. The figure demonstrates that ranks are rapidly clipped at the beginning of iterations and converge to optimal low ranks. During the entire process, the accuracy fluctuations are limited within a small range.

As shown in Figure 5and Table 1, rank clipping successfully reduces the ranks in both convolutional layers and fully-connected layers without accuracy loss. The crossbar area occupation of the entire LeNet (ConvNet) reduces to 13.62% (51.81%). When applying SVD, the whole crossbar area can be reduced to 32.97% (55.64%) for LeNet (ConvNet), which indicates SVD is inferior to PCA. Therefore, we mainly conduct experiments using PCA approach. Note that the last layers of LeNet and ConvNet are not clipped because the rank (M = 10) is already very small so little improvement space exists.

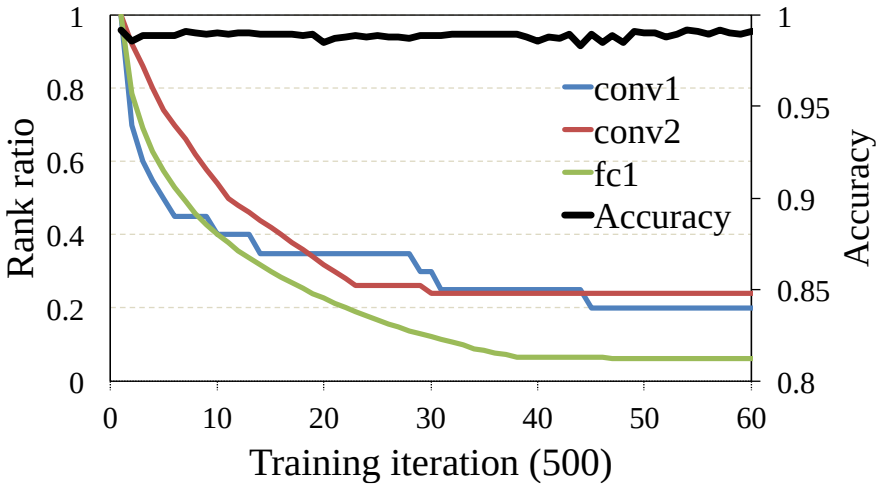

Figure 5 Rank ratio of each layer and accuracy during training with rank clipping

4.2 Group Connection Deletion

The rank clipping reduces the total number of required crossbars, while there are still a large number of crossbars to implement modern large neural networks. The second step of group scissor framework---group connection deletion aims to remove interconnections between synapse crossbars so as to reduce the circuit-level routing congestion and architecture-level inter-core communication for NCS.

Figure 6 gives the basic idea. An array of MBCs is connected to implement a large weight matrix $U \in \mathbb{R}^{N \times K}$. Suppose the elementary synapse crossbar has P inputs and Q outputs ($P \ll N, Q \ll K$), a $\left\lceil \frac{N}{P} \right\rceil \times \left\lceil \frac{K}{Q} \right\rceil$ array of crossbars must be interconnected to implement U as illustrated in Figure 6.

The implementation of matrix V follows the similar way. As memristors are densely manufactured in the crossbar and the area of each memristor cell is feature-size level, the routing wires dominate the circuit area [14]. Suppose a row group of connections in Figure 6 all have zero weights, implying that those connections are removable, we can delete/prune the wire routing to the input of this row group. Similarly, the wire routing from the output of a column group can be deleted when the column group of connections are all-zeros. Our group connection deletion method actively deletes those groups of connections during the learning of neural networks, meanwhile maintaining the classification accuracy at the similar level.

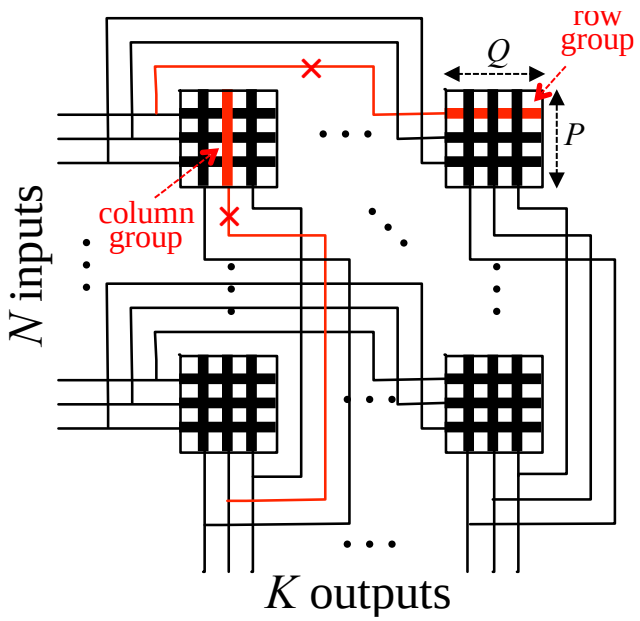

Figure 6 The group connection deletion

We harness group Lasso regularization to delete groups of connections. Group Lasso is an efficient regularization in the study of structured sparsity learning [26][27]. With group Lasso regularization on each group of weights, a high percentage of groups can be regularized to all-zeros. In group connection deletion method, weights are split into row groups and column groups as illustrated in the figure. And group Lasso regularization is enforced on each group. Mathematically, the minimization function for training neural network with group Lasso can be formulated as:

$$E(W) = E_D(W) + \lambda \cdot \left(\sum_{g=1}^{G(r)} \left\| W_g^{(r)} \right\| + \sum_{g=1}^{G(c)} \left\| W_g^{(c)} \right\| \right) \quad (6)$$

where W is the set of weights of neural network, ED(W) is the original minimization function when training traditional neural networks. G(r) and G(c) respectively denote the number of row groups and column groups, and $W_g^{(r)}$ and $W_g^{(c)}$ are the sets of weights in the g-th row group and column group, respectively. And

$$\bigcup_{g=1}^{G^{(r)}} W_g^{(r)} = \bigcup_{g=1}^{G^{(c)}} W_g^{(c)} = W \quad (7)$$

λ is the hyper-parameter to control the trade-o between classification accuracy and routing congestion reduction. A larger λ can result in lower accuracy but larger reduction of routing wires. During the back-propagation training with equation (6) , each weight w will be updated as

$$w \leftarrow w - \eta\left(\frac{\partial E_D(W)}{\partial w} + \frac{\lambda w}{\left\|W_i^{(r)}\right\|} + \frac{\lambda w}{\left\|W_i^{(c)}\right\|}\right) \quad (8)$$

where η is the learning rate, $i \in \left[1 \cdots G^{(r)}\right]$, $j \in \left[1 \cdots G^{(c)}\right]$, $w \in W_I^{(r)}$ and $w \in W_j^{(c)}$.

With group connection deletion, we disconnect all the zero-weighted connections and prune all the routing wires connecting to all-zero row groups or column groups. After deletion, we fine-tune (retrain) the structurally-sparse neural networks to improve accuracy. Figure 7 plots the trends of deleted routing wires (i.e., all-zero row/column groups) and the classification accuracy versus the iterations of group connection deletion. The deletion process starts with the low-rank LeNet in Table 1 that was already compressed by rank clipping. In Figure 7, we only delete the matrices of U and V whose dimensions are beyond the largest size of MBC. Even for low-rank neural networks, our method can delete the routing wires dramatically, e.g., 93.9% interconnection wires are removed in the crossbar array of fc1_v. Fine-tuning the deleted neural networks attains the baseline accuracy (99.1%).

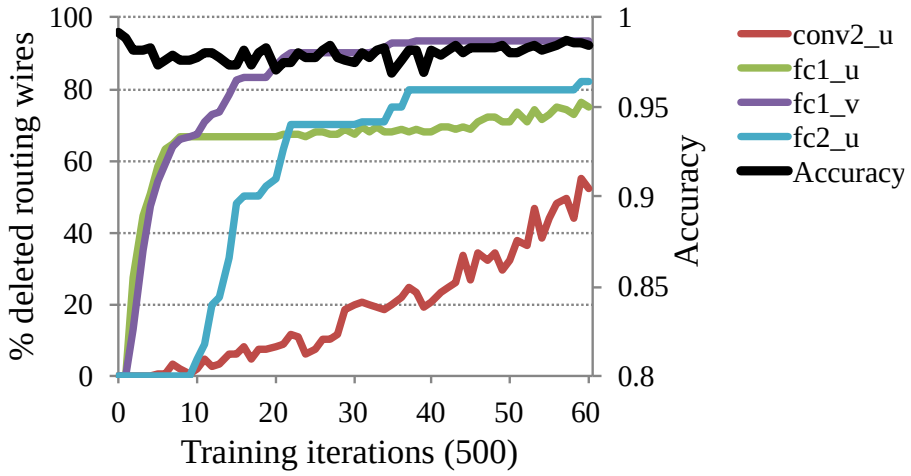

Figure 7 The percentage of deleted routing wires and accuracy during group connection deletion. fc1_u and fc1_v is the low-rank matrix U and V of fc1 after rank clipping, and so forth.

Note that compared with our method, it is more difficult to reduce the routing wires on traditional sparse neural networks. This is because its sparse weights are randomly distributed in the crossbar arrays and the corresponding routing wire must be preserved even there exists one nonzero weight in the row group or column group.

4.3 Area Estimation

This section formulates the area estimation method for hardware evaluation in this work.

MBC area estimation: The use of MBCs in NCS design has been extensively studied. As a critical component in such a system, MBCs occupy a significant proportion of whole design area. Each MBC is an ultra dense cross-point structure formed by a set of memristors and wires. The area of a memristor cell in MBC is 4F2 under the state-of-the-art technology [13], where F is the minimum feature size. Restricted by the technology limitations, a feasible MBC implementation only considers MBCs that are not larger than 64x64 [12]. To ensure the system reliability and

robustness, we only consider MBCs with dimensions constrained within 64x64 in the standard library. For those large weight matrices in neural networks, their connections can be distributed into several/many MBCs as demonstrated in Figure 1.

Routing area estimation: Assume that the metal width is Wm, the distance between two metals is Wd, and the length of i-th wire between crossbars is Li. The total routing area occupied by the wires can be roughly formulated as

$$A_r = (W_m + W_d) \sum_i^{N_w} L_I \quad (9)$$

Here Nw is total wire count including electrostatic shielding wires. Suppose the average wire length is linearly proportional to Nw, the routing area is estimated as

$$A_r = \alpha N_w^2 \quad (10)$$

where α is a scalar.

4.4 Experiments

This section describes the experiments that evaluate the effectiveness of the proposed rank clipping and group connection deletion methods. All the experiments are based on the NCS implemented by MBC. The related experiment parameters on memristor and MBC are summarized in Table 2. We mainly implement two neural networks--LeNet on MNIST and ConvNet on CIFAR-10. The detailed network structures can refer Table 1.

Table 2. Experiment parameters

Parameter	Value
memristor cell area	$4F^2$
maximum crossbar size	64×64
Wire length between two memristors	$2F$

4.5 MBC Area Reduction

In our experiments, we clip all the convolutional and fully-connected layers, except the last classier layer. The original rank in the last layer is determined by the number of classes so the further reduction is meaningless. The rank clipping method compresses each large weight matrix to two skinny matrices by reducing the rank. Figure 8 shows the final remained ranks with respect to the accuracy and tolerable clipping error ϵ for convolutional layers in LeNet. Here the original rank of conv1 and conv2 is 20 and 50, respectively, as denoted by upper markers on the stems. For each layer, the rank decreases as ϵ increases, and finally reaches to a very small value. It can be seen that the corresponding accuracy is well maintained. We also observe similar results in fc1. More specifically, the layer-wise ranks are reduced to 5, 12 and 36 without accuracy loss, and to 4, 6 and 6 with merely 1% accuracy loss.

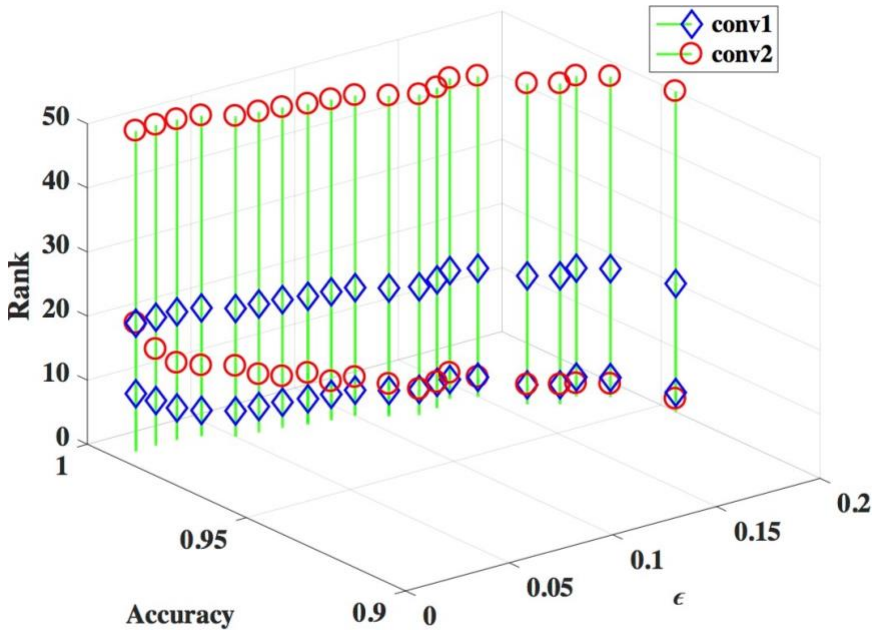

Figure 8 The remained ranks in convolutional layers of LeNet. fc1 is omitted for better visualization as its original rank 500 is out of chart.

Figure 9 (a, b) respectively plot the percentage of remained MBC area with respect to the classification error for LeNet and ConvNet. Routing area is excluded in this evaluation. The area of each layer is the sum of the areas of U and V. Total area includes the area of the last classier layer, i.e., fc2 in LeNet or fc1 in ConvNet. For both networks, the layer-wise areas of both convolutional layers and fully-connected layers rapidly reduce with small accuracy loss.

In summary, the rank clipping can reduce the total crossbar area of LeNet to 13.62% without sacrificing any accuracy loss. The crossbar area can be further reduced to 3.78% with merely 1% accuracy loss. For more complex ConvNet, no accuracy loss is observed when the crossbar area decreases to 51.81%. By paying 1% loss, the total crossbar area is reduced to 38.14%.

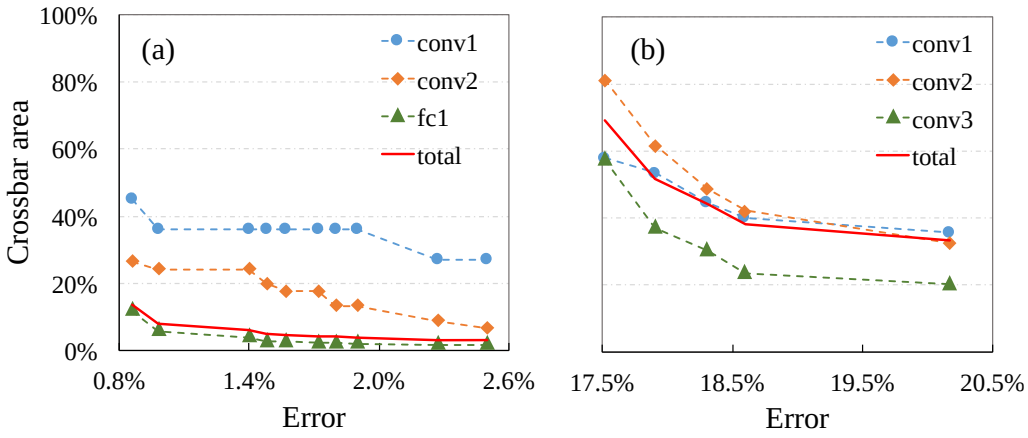

Figure 9 The MBC area for (a) LeNet and (b) ConvNet, after applying the rank clipping

4.6 Routing Area Reduction

To evaluate the routing congestion alleviated by group connection deletion, we use the number of routing wires and remained routing area of Eq. (10) as our metrics. Although the estimation of routing area in real circuit can be more complex, the real routing area reduction in hardware must be positively correlated to our results.

As aforementioned in Section 3.5, our standard library contains all types of memristor crossbars with dimensions constrained within 64x64. When implementing a $N \times K$ weight matrix U, the MBC sizes are selected based on the following criteria: (1) Implement U in a $N \times K$ MBC, when $N \ll 64$ and $K \ll 64$; (2) Implement U by an array of MBCs when $N > 64$ or $K > 64$, with the largest available MBC size $P \times Q$, where N and K is divisible by P and Q, respectively.

In the experiments, the group connection deletion starts with the rank-clipped LeNet or ConvNet without accuracy loss as presented in Table 1. Based on the MBC selection criteria, the

Table 3. MBC sizes and remained routing wires in large layers

Net	type	conv1_u	conv2_u	conv3_u	fc1_u	fc1_v	fc_last
LeNet	sizes	– †	50 × 12	–	50 × 36	36 × 50	50 × 10
	% wires	–	47.5	–	24.8	6.7	18.0
ConvNet	sizes	25 × 12	50 × 19	50 × 22	–	–	64 × 10
	% wires	83.3	40.5	74.4	–	–	81.9

† The weight matrix can be implemented by one crossbar.
conv1_v, conv2_v and conv3_v are omitted for the same reason.

sizes of MBC utilized in large layers are shown in Table 3. Matrices with sizes constrained by 64 x64 are omitted in the table, and no group Lasso regularization is enforced on those small matrices.

The experimental results of the remained routing wires after applying the group connection deletion without allowing accuracy loss are also presented in Table 3. The results for LeNet are remarkable. We achieve the same accuracy of the baseline, with routing wires being only 47.5%, 24.8%, 6.7% and 18.0% of the original ones in respective layer. This can reduce the layer-wise routing area to 8.1%, on average.

Table 3 also shows that, in ConvNet, our method on average reduces the layer-wise routing wires to 70.03% and therefore decrease the layer-wise routing areas to 52.06%, while achieving the same accuracy as the baseline. With an acceptable accuracy loss, the routing congestion can also be significantly alleviated. Figure 10 comprehensively studies the remained routing wires and routing area under different classification errors. With merely 1.5% accuracy loss, the routing area in each layer is reduced to 56.25%, 7.64%, 21.44% and 31.64%, respectively.

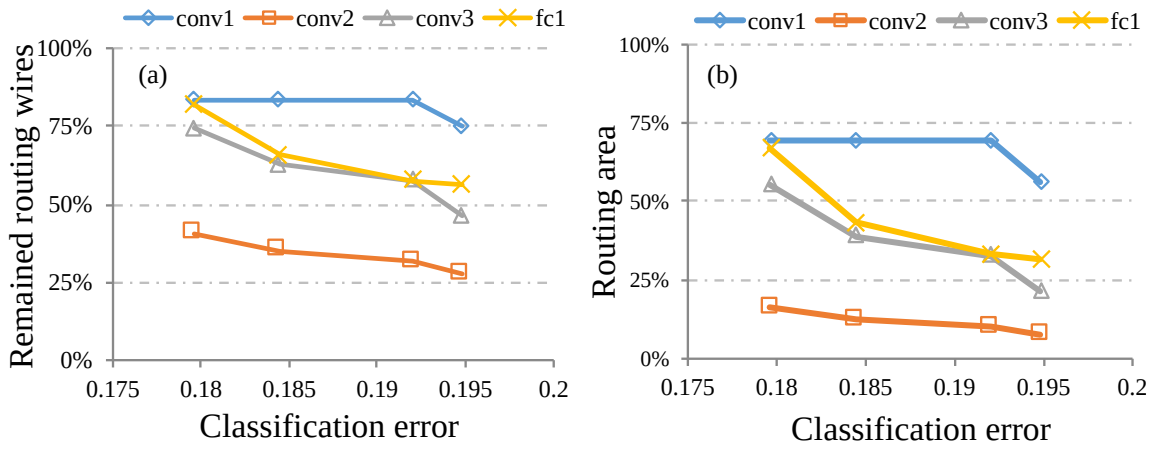

Figure 10 The routing wire (a) and routing area (b) w.r.t. the classification error in ConvNet

At last, Figure 11 shows the sparse weight matrices after group connection deletion for ConvNet in Table 3 without accuracy loss. Each blue/red block stands for a collection of weights, which are implemented by one crossbar in the NCS design. White regions indicate that there are no connections.

After applying the group connection deletion, the connections in crossbars become sparse. More importantly, the sparsity is structural instead of being randomly distributed in traditional sparse neural networks. In the figure, a high ratio of column groups in crossbars are regularized to all-zeros, such that interconnection wires routing from those crossbar columns can be removed. Impressively, as conv2_u and fc1 in the figure show, some blocks have no connections in the whole region, indicating that the entire crossbar can be removed in the NCS implementation. It is significant because not only routing congestion can be alleviated, but also crossbar area can be reduced. We also note that a crossbar with some zero columns/rows can be replaced by a smaller but dense crossbar after removing those zero groups, which can further reduce the crossbar area.

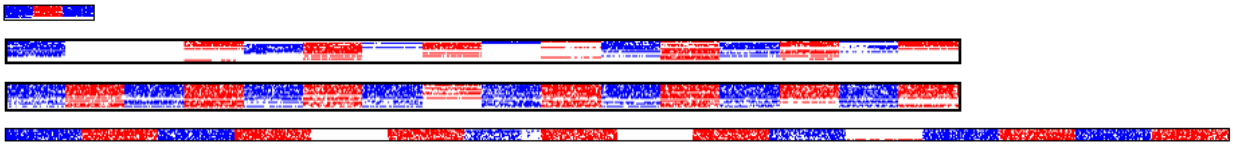

Figure 11. Weight matrices (transposed) after group connection deletion. The deletion starts from the rank-clipped ConvNet in Table 1. Matrices are plotted in scale in the order of conv1 u, conv2 u, conv3 u and fc1. White regions have no connections. And connections in each blue/red block are implemented in a crossbar.

5.0 Classification Accuracy Improvement for Neuromorphic Computing

5.1 Methodology

This paper aims at improving the classification accuracy of pure binary neural networks in all layers. Such neural networks can be naturally implemented on NCS, such as TrueNorth chip and memristor based design. Three novel classification accuracy improving methods are proposed in the work, namely, distribution-aware quantization, quantization regularization and bias tuning. The implementation of convolutional neural network (CNN) convolution operation in memristor crossbar array and a crossbar variation demo for accuracy improvement are also presented.

To explain our methodologies, in this section, we take LeNet [15] as the example of CNN trained on MNIST – a 28x28 handwritten digit database. Experiments and analysis on more neural networks and databases shall be presented in Section 4.2.

5.1.1 Distribution-aware Quantization

In training of neural networks, l2-norm regularization is commonly adopted to avoid over-fitting. With l2-norm regularization, the final distribution of learned weights in a layer approximately follows the normal distribution [28]. A naive quantization method in implementation is to quantify all weights to the same group of level selection. However, as shown in Figure 12 The blue and orange bars denote the original weight distribution of different layers and the learned discrete weights after quantization regularization (QR) in LeNet, respectively.

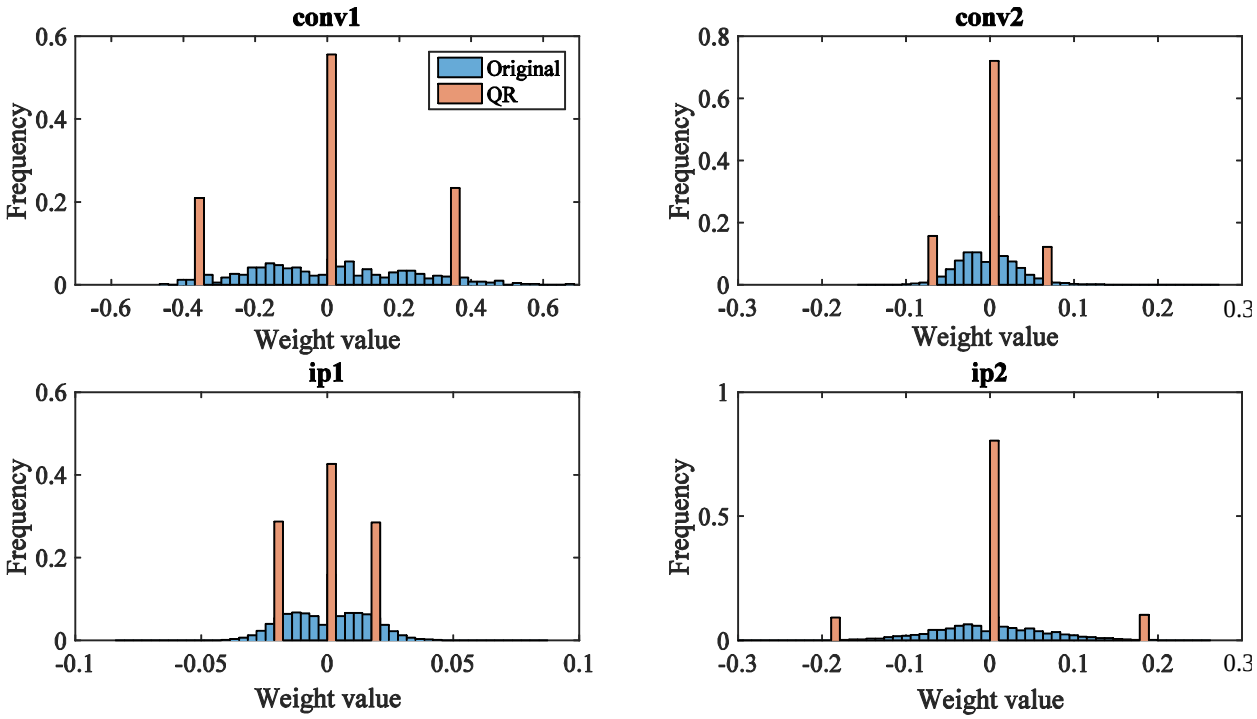

Figure 12 The blue and orange bars denote the original weight distribution of different layers and the learned discrete weights after quantization regularization (QR) in LeNet, respectively.

Let's taking LeNet as an example, the weight distribution varies from layer to layer: the first convolutional layer (conv1) has the most scattered distribution with a wider range scope, while the weights of second convolutional layer (conv2) and two fully connected layers (ip1, ip2) have concentrated to a relatively narrow scope. The data implies that a quantization optimized for one layer may result in a large information loss in another layer.

Here, we propose a heuristic method -- distribution-aware quantization (DQ) which discretizes weights in different layers to different values. In memristor-based NCS, this can be realized by programming the resistance states of each crossbar to different values [2]. Our experiments on LeNet show that when applying the aforementioned naive method, the test accuracy of 1-level quantization quickly drops from 99.15% to 90.77%, while our proposed

distribution-aware quantization can still achieve 98.31% accuracy. Note that without explicit mention, the quantization levels are selected by cross-validation [29].

5.1.2 Quantization Regularization

Distribution-aware quantization separates the training and quantifying processes and therefore it cannot avoid the accuracy loss once the quantization is completed. To further improve system performance, we propose quantization regularization (QR) which directly learns a neural network with discrete weights.

During the training of a network, a regularization term can be added to the error function to control the distribution of weights and avoid overfitting. For example, l2-norm regularization can learn weights with normal distribution and l1-norm is commonly utilized to learn sparse networks (Glorot and Bengio 2010). The total error function to be minimized with a generic regularization term can be formulated as

$$E(W) = E_D(W) + \lambda \cdot E_W(W) \quad (11)$$

where λ is the coefficient controlling the importance between data-dependent error $E_D(W)$ and regularization term $E_W(W)$. W is the set of all weights in neural networks. We propose a new quantization regularization as

$$E_W^q(W) = \mathrm{sgn}(W_k - Q(W_k)) \cdot (W_k - Q(W_k)) \quad (12)$$

where W_k is the k-th weight, $Q(W_k)$ is the quantization value nearest to W_k and $\mathrm{sgn}(\cdot)$ is the sign function. After forwarding and back propagation, the weight updating with learning rate η can be formulated as:

$$W_k \leftarrow W_k - \eta \cdot \frac{\partial E_D(W)}{\partial W_k} - \eta \cdot \mathrm{sgn}(W_k - Q(W_k)) \quad (13)$$

Through the third term on the right side of (13), our regularization descents (reduces) the distance between a weight and its nearest quantization level with a constant gradient (± 1). Compared with the l1-norm and l2 –norm regularization, our proposed regularization method can quantify learning weights to the desired discrete values more precisely, meanwhile properly control the weight distribution and overfitting. Figure 13 demonstrates and compares the three regularization methods. Zero is one of the targeted quantification values in this work, which is usually realized through l1-norm based neural network sparsification. In addition, our proposed method includes more discrete quantification values. Orange bars in Figure 1 correspond to the new weight distribution of LeNet after applying QR, indicating our method can efficiently learn weights around quantization levels. Compared with the naive 1-level quantization, including QR only can improve accuracy 6.21%. Combining with DQ, the accuracy drop from the ideal case is controlled within only 0.20% with 1-level quantization. More experiments will be discussed in section 5.2.

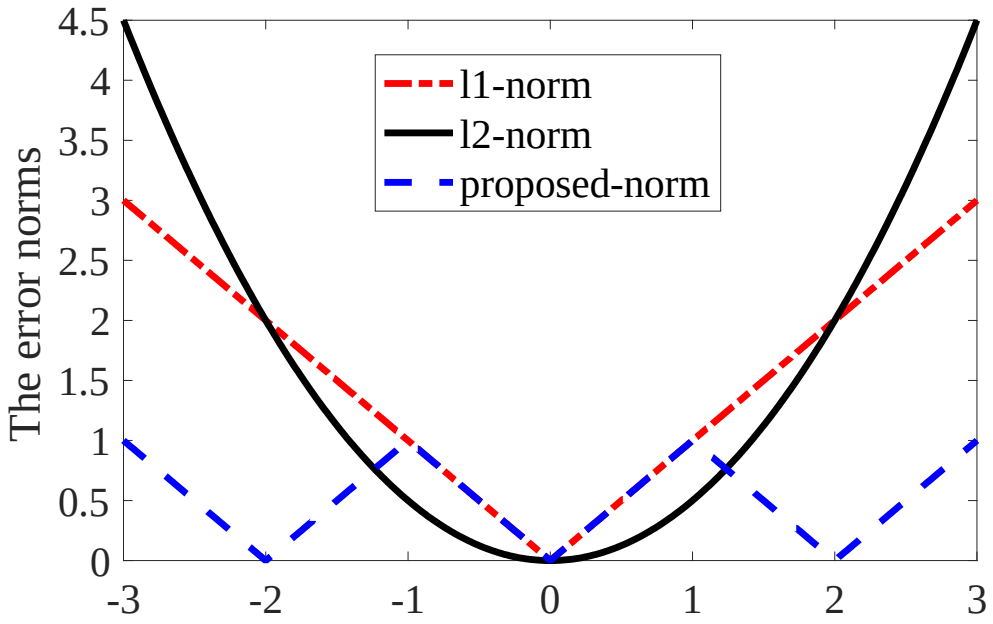

Figure 13 Comparison of l1-norm, l2-norm and our proposed regularization

5.1.3 Bias Tuning

The quantization of weights deviating the information can be formulated as

$$y_i + \Delta y_i = \sum_i (W_{ji} + \Delta W_{ji}) \cdot x_i + b_j \quad (14)$$

where W_{ji} is the weight connecting the i-th neuron in the previous layer to the j-th neuron in this layer. ΔW_{ji} and $\Delta y_i = \sum_i \Delta W_{ji} \cdot x_i$ are the deviation of weight and input of activation function, respectively, resulted from quantization. The deviation Δy_i propagates through layers toward the output classier neurons and deteriorates the classification accuracy.

In circuit design of neuron model, the bias usually is an adjustable parameter, e.g. the fire threshold in TrueNorth neuron model works as bias. Therefore, to compensate the deviation, we may adjust the neuron bias from b_j to $b_j + \Delta b_j$ such that

$$\Delta b_j = -\Delta y_i = -\sum_i \Delta W_{ji} \cdot x_i \quad (15)$$

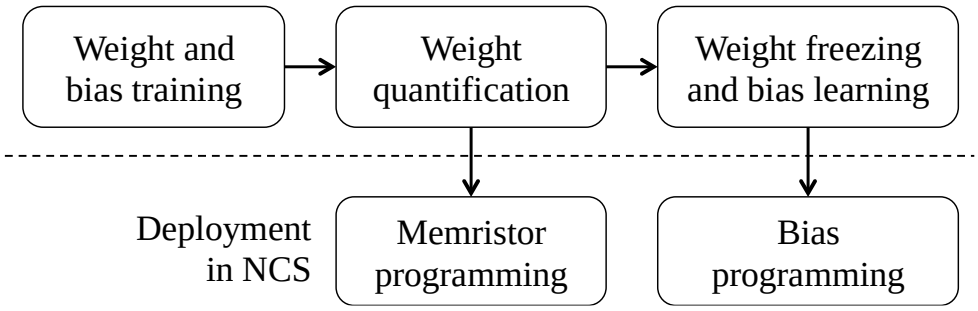

Figure 14 The framework of proposed bias tuning method

As such, the neuron activation can remain the original value before quantization. Unfortunately, the input x_i varies randomly with the input samples (e.g., images) and a unique bias compensation Δb_j cannot be identified.

We propose bias tuning (BT) which learns the optimal bias compensation to minimize the impact of quantization. Figure 14 The framework of proposed bias tuning method shows the framework of the bias tuning: first, both weights and biases are trained without quantization; second, weights are quantified and programmed into NCS; third, weights are frozen and biases are learned to improve classification accuracy; and finally, the tuned biases are programmed into NCS. Impressively, bias tuning method can achieve 7.89% classification improvement compared to the naive 1-level quantization baseline on LeNet. Combining with the above DQ and QR methods, the total accuracy drop can be reduced to merely 0.19%.

5.1.4 Convolution in Memristor Crossbar Array

The memristor crossbar structure can be naturally mapped to fully connected layers. Here, we extend its use to convolution layers. A pixel value (y) in a post feature map is computed by

$$\boldsymbol{y} = \sum_{\boldsymbol{k}} \boldsymbol{F_k} \cdot \boldsymbol{w_k} + \boldsymbol{b} \qquad (16)$$

where w_k is the k-th weight in the filter and F_k is the corresponding input feature. Because the essence of convolution is multiplication-accumulation, we can employ memristor crossbar array to compute. Figure 15 shows an example to compute the convolution of a 5-by-5 feature map with a 3-by-3 filter. At the time stamp t0, the green elements are converted to a vector and sent into a memristor array through word lines. And at t1, the pink elements are processed similarly to the green ones. As the filter shifts, the corresponding features in the previous layer are sent into the crossbar in a time-division sequence, such that the output features are computed by the bit line (blue) whose weights belong to the filter. As shown in the figure, each bit line is mapped to one filter in the convolutional layer. We note that the proposed DQ, DR and BT methods also work for weights in CNN.

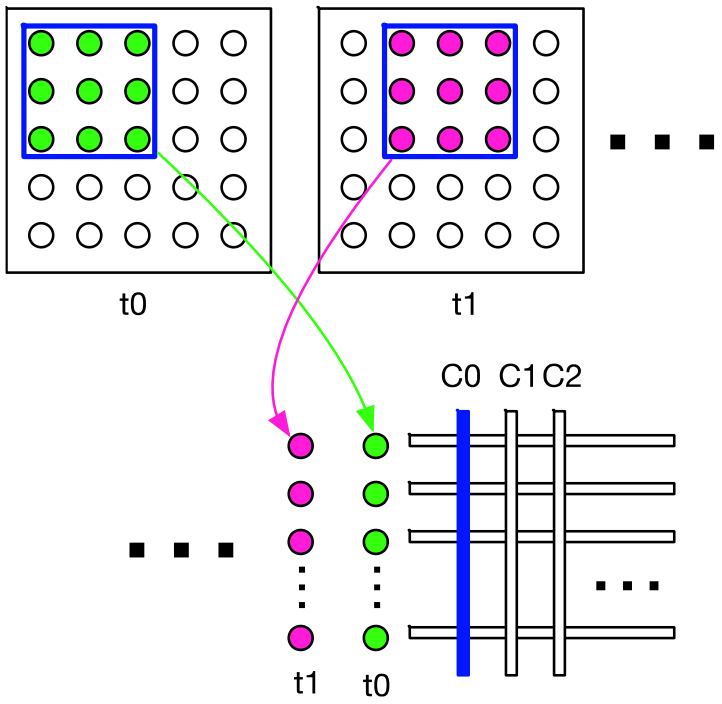

Figure 15 Convolution implementation in memristor crossbar array

5.2 Experiments

5.2.1 Experiment Setup

To evaluate the effectiveness of proposed methods, we conducted three experiments using multilayer perception (MLP) and CNN neural network structures on two datasets: MNIST and CIFAR-10 (a 32x32 color image database). The first two experiments are both conducted on MNIST dataset using a MLP and a CNN network, respectively. The third experiment is conducted on CIFAR-10 dataset using a CNN network. The adopted deep leaning framework is Caffe developed by the Berkeley Vision and Learning Center (BVLC) and community contributors (Krizhevsky, Sutskever, and Hinton 2012). Detailed network parameters and dataset are summarized in Table 4.

Table 4. Network and dataset

	Network 1	Network 2	Network 3
Dataset	MNIST	MNIST	CIFAR-10
Input	28×28	28×28	32×32
Conv1	–	20×5×5[§]	32×5×5
Conv2	–	50×5×5	32×5×5
Conv3	–	–	64×5×5
Ip1	784×500	800×500	1024×10
Ip2	500×300	500×10	–
Ip3	300×10	–	–

[§]20×5×5 means 20 filters with each filter size 5×5.

5.2.2 Function Validation of MLP on MNIST

Network 1 is a MLP network with a size of 784x500x300x10, which can't be directly implemented in NCS. Previously, we presented the hardware implementation of mapping a large network to small crossbar arrays [14]. Here, 784 corresponds to the 28x28 MNIST image input pattern; 500 and 300 are the neuron numbers of the first and second hidden layers, respectively; and 10 is the final classification outputs.

The baseline is set as the highest accuracy (all the layers quantified to 0.06) of all naive 1-level quantization situations without applying any proposed method. To explore the effectiveness of each single method and their combination situations, we conducted 8 separate experiments with combinations, the experiment results of which are summarized in Table 5.

Compared with the baseline accuracy, there is a large accuracy increase when applied only one of three accuracy improvement methods (1.52%, 1.26%, 0.4%, respectively). Applying any two of three methods will make the accuracy further increased. Combining all three methods together can achieve a highest accuracy with only 0.39% accuracy drop compared with the ideal value without any quantization. We note that, in some cases (e.g. DQ+QR+BT vs. DQ+BT), integrating more than one proposed methods does not improve accuracy much. This is because MNIST is a relative simpler database so the effectiveness of these methods on accuracy improvement quickly approaches to a saturated level. In more challenging CIFAR-10 database, experiments show that more methods of DQ, QR and BT are harnessed, higher accuracy can always be obtained by a large margin.

Table 5. The accuracy measurement for MLP on MNIST dataset

	DQ	QR	BT	Accuracy	Drop
Ideal[§]				98.39%	
0 (Baseline)				95.97%	2.42%
1	√*			97.49%	0.90%
2		√		97.23%	1.16%
3			√	96.37%	2.02%
4	√	√		97.91%	0.48%
5	√		√	98.00%	0.39%
6		√	√	97.23%	1.16%
7	√	√	√	98.00%	0.39%

[§]The ideal accuracy without quantization;
*√ denotes that the corresponding method is utilized.

5.2.3 Function Validation of LeNet

LeNet, which has strong robustness to image geometric transformations, is a much more popular network. We utilized it for MNIST and shows the results in Table 6. Compared with the MLP network, 1-level precision LeNet can achieve an even lower accuracy drop (0.19% compared with 0.39%) after combining all our methods. Remarkably, although the DQ method separates the training and quantifying processes, directly quantifying weights in each layer has accuracy loss less than 1%, without further fine-tuning. The orthogonality among DQ, QR and BT is also indicated by the results.

Table 6. The accuracy measurement for CNN on MNIST dataset

	DQ	QR	BT	Accuracy	Drop
Ideal				99.15%	
0 (Baseline)				90.77%	8.38%
1	√			98.31%	0.84%
2		√		96.98%	2.17%
3			√	98.66%	0.49%
4	√	√		98.96%	0.19%
5	√		√	98.68%	0.47%
6		√	√	98.75%	0.40%
7	√	√	√	98.96%	0.19%

5.2.4 Function Validation of CNN on CIFAR-10

We also evaluate the proposed methods in more challenging natural image dataset CIFAR-10 to verify their generality. The CNN in [4] is adopted without data augmentation. Table 7 presents the results of all the interested combinations. As expected, CNN has a large accuracy drop (64.32%) when applying the naive 1-level quantization while each our proposed technique can dramatically hinder the accuracy loss. However, unlike the experiments on MNIST, a sole method cannot improve the accuracy of CNN to a satisfactory level. Some combinations of two methods perform excellent accuracy improvement. For example, DQ+RQ makes the accuracy level to 74.43% BinaryConnect neural network in [18] performs state-of-the-art accuracy when the last layer utilizes L2-SVM. The parameters in the L2-SVM layer are floating-point and critical for accuracy maintaining. However, the SVM is not good for circuit implementation. Our work quantifies all weights to one level and controls the accuracy loss within 5.53% for more efficient circuit (e.g., memristor crossbar) design.

Table 7. The accuracy measurement for CNN on CIFAR-10 dataset

	DQ	QR	BT	Accuracy	Drop
Ideal				82.12%	
0 (Baseline)				17.80%	64.32%
1	√			32.92%	49.2%
2		√		66.88%	15.24%
3			√	46.54%	35.58%
4	√	√		74.43%	7.69%
5	√		√	57.74%	24.38%
6		√	√	67.22%	14.90%
7	√	√	√	76.59%	5.53%

5.2.5 Learned Filters

Figure 16 presents the learned floating-point and 1-level precision conv1 filters in LeNet and CNN on CIFAR-10, respectively. Our methods can efficiently learn the feature extractors similar to the corresponding original ones, even with 1-level precision. Furthermore, the number of input channels (RGB) of CIFAR-10 image is 3, such that each pixel in the filter has 33 possible colors. For filters with n channels, a 1-level precision filter still has a large learning space with $3^{n \cdot k \cdot k}$ possibilities, where k is the filter size. Those explain why our method can maintain the comparable accuracy.

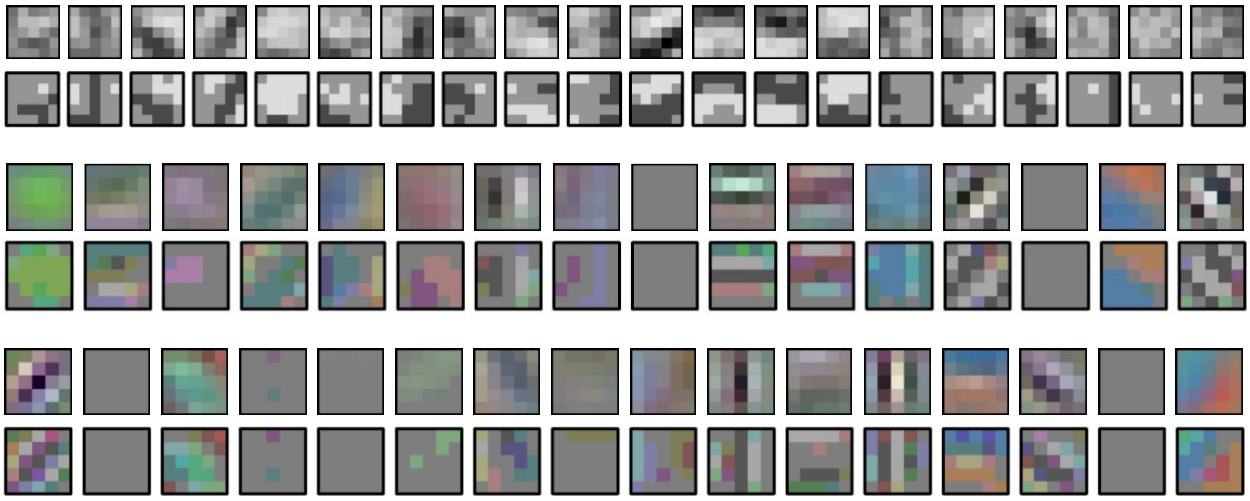

Figure 16 The learned floating-point (upper) and quantified (lower) conv1 filters in LeNet (the gray-scale ones) and CNN on CIFAR-10 (the color ones). A zero weight is mapped to pixel value 128, and negative (positive) weights are darker (brighter) ones.

5.2.6 Bias Tuning to Alleviate Crossbar Variation

As aforementioned, the memristive variations caused by fabrication imperfection can result in deviation of the programmed weights [4]. Our bias tuning method can also be extended to overcome memristor variation. After programming weights to memristors under the impact of variation, we read out the real programmed weights, then ne-tune the bias with weights frozen, and finally the tuned biases are reprogrammed to the circuit neuron models to compensate the impact of weight variation.

Figure 17 plots the accuracy vs. the variance of programming process. The entry 4 in Table III is taken as the baseline in this investigation on variation impact. The figure shows that the bias tuning method successfully hinders the negative impact of variation.

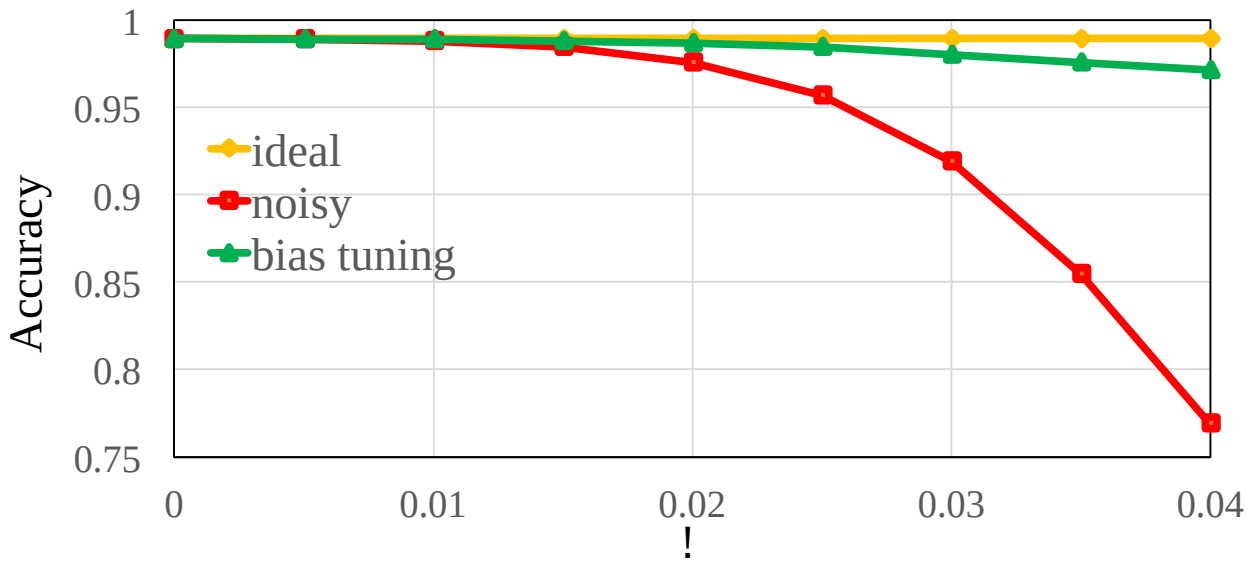

Figure 17 The bias tuning in LeNet. The yellow line denotes the accuracy after applying DQ and QR without noise; The red line is the baseline with quantization and noise; The green line denotes the accuracy recovered from the baseline after bias tuning; is the standard deviation of Gaussian noise.

5.2.7 Discussion

Our previous research study [5] species for spiking neural networks, where the probability distribution can only be biased to two poles (0 or 1). In this work, we extend the method to memristor-based neural networks adopted by state-of-the-art research and large-scale applications [30]. The proposed methods can regularize the weights to multiple levels with uniform or nonuniform quantization. For example, in our CIFAR-10 experiments, the quantization points in layer conv1, conv2, conv3 and ip1 are [-0.12, 0, 0.12], [-0. 08, 0, 0.08], [-0.02, 0, 0.02] and [-0. 008, 0, 0.008], respectively. Moreover, we discharge the reliance on the floating-point layer in [9] and explore a pure one-level precision solution. Comprehensive experiments and analyses on MLP and CNN using MNIST and CIFAR-10 datasets are conducted. Our experiments on MNIST shows negligible accuracy drop (0.19% in CNN), which is much better than the previous work like [9].

From the aspect of the system implementation, there are extensive research studies on binary neural networks deployed in traditional platforms such as CPUs, GPUs and FPGAs. However, those approaches may not suitable for the hardware characteristics of brain-inspired systems like memristor-based systems. For example, BinaryConnect [18] uses L2-SVM layer, which is very costly to be implemented by memristor hardware. In circuit design, bias has the characteristic of adjustability, which inspires our bias tuning method in this work. As shown in the paper, bias tuning can be used to control quantization accuracy as well as overcome the process variation of memristor technology.

6.0 Deformable Regularization Work

Quantization regularization method directly learns a network with discrete weights during training process and only brings with small accuracy loss for image classification task. Quantization regularization method can be demonstrated in the following figure. In the figure, each point falling on the X axis is one of the quantization values. As mentioned before, we can set as many discrete quantization values as we can and regularize synapse weights to the predefined quantization values.

As l1-norm and l2-norm regularization methods won't bring any accuracy loss in the neural network training and testing process, a new regularization method can be proposed to gradually change the regularization from l1-norm or l2-norm to our quantization regularization method. In

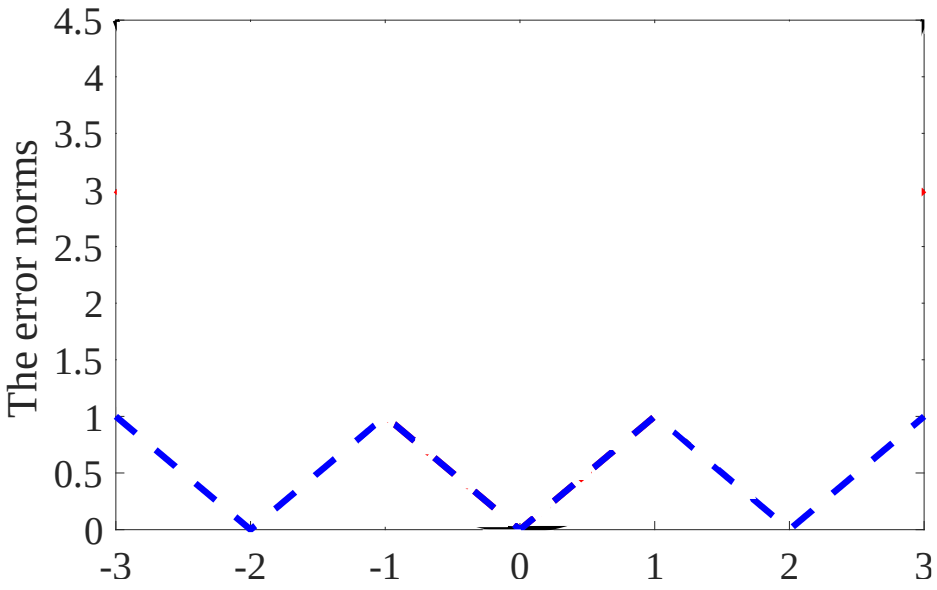

Figure 18 Quantization regularization method

this way, we can avoid accuracy loss as well as adopt the previous quantization regularization method. This process can be demonstrated in the following figure. The blue dashed line is Quantization regularization method, and black and red line are l1-norm and l2-norm, respectively. Green arrows show the changing process from l1-norm or l2-norm to quantization regularization norm.

To further improve the network model performance, the advanced deformable regularization (DR) method has been put forward to gradually learn the floating-point weights to desired quantized values, which guarantees both accuracy and quantifying results. At the beginning of training process, this approach targets to purely train the neural network model. While

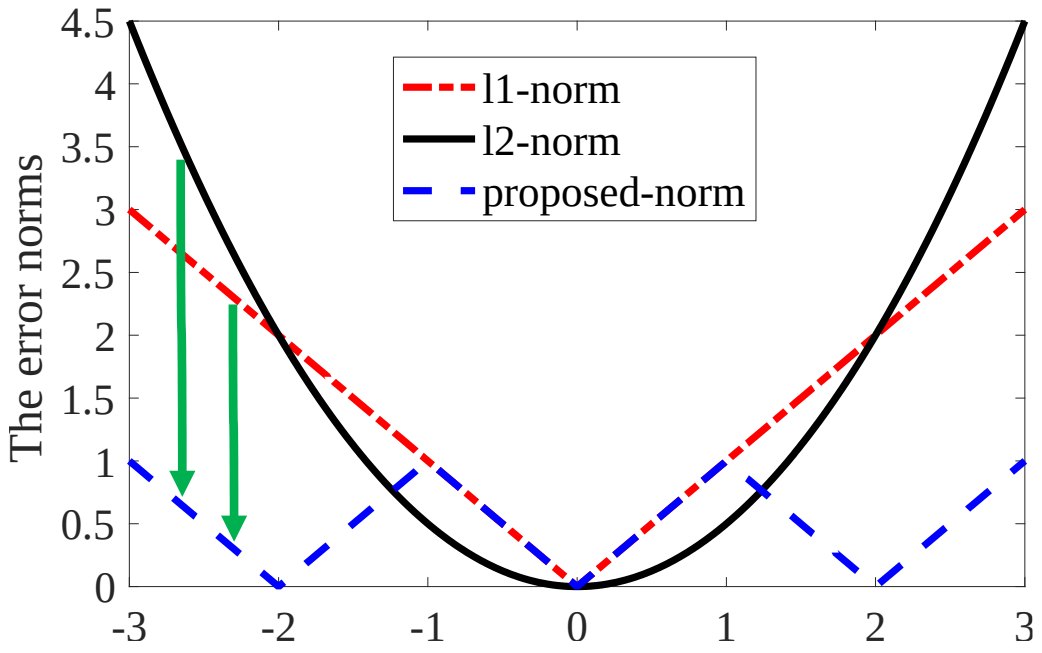

Figure 19 The difference among l1-norm, l2-norm and quantization regularization

at the end of training process, the main goal is to do quantization based on previous well trained model. However, during the middle stage, the regularization task will gradually shift from the training to quantization. Traditional commonly used l1-norm and l2-norm regularization methods won't bring any accuracy loss to neural network models. Thus, based on the equation (12) on quantization regularization method, deformable regularization can be formulated as:

$$E_W^d(W) = \alpha E_W^{l1}(W) + (1-\alpha)E_W^q(W) \tag{17}$$

$$E_W^d(W) = \alpha E_W^{l2}(W) + (1-\alpha)E_W^q(W) \tag{18}$$

Where equation (17) (18) describe the advanced deformable regularization method gradually changing from l1-norm and l2 -norm regularization respectively. α here represents the weighted parameter to control the balance between traditional regularization term and quantization regularization term. It can be easily inferred that when α is equal to 1, E_W^d is just the basic l1-norm or l2-norm term. While when α is equal to 0, it represents the quantization regularization instead. When α gradually changes from 1 to 0, the whole deformable regularization term will put more emphasize on traditional l1 -norm and l2-norm at the beginning and do more quantization work at the end. Here, α itself can also be changed in different ways, such as straight line, ellipse and even cosin. For example, when α is changing with a straight line, it can be denoted as:

$$\alpha = -\frac{1}{N}n + 1 \tag{19}$$

Or α changing like an ellipse or cosin function:

$$\alpha = \sqrt{1 - \frac{n^2}{N^2}} \tag{20}$$

$$\alpha = 0.5(1 + \cos\frac{\pi}{N}n) \tag{21}$$

Here in the above equations, N denotes the total training iterations, and n denotes the n-th training iteration along the whole training process. Figure 20 illustrates the three changing patterns

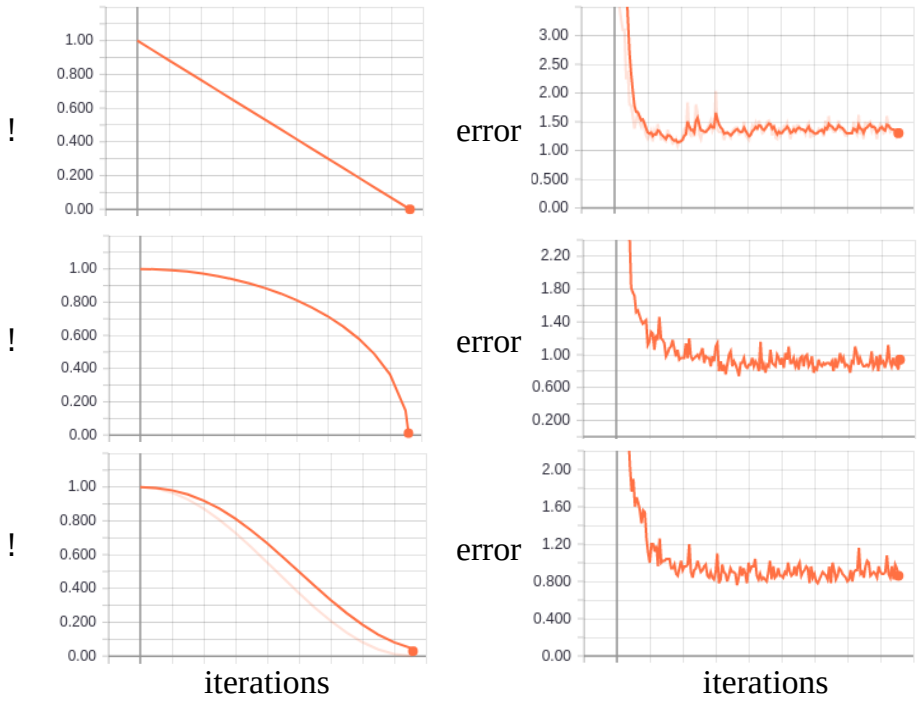

Figure 20 The left figures show three kinds of changing methods for parameter α, with straight line, ellipse and cosin, respectively. The right three figures show the corresponding error function doing experiments on MNIST dataset with LeNet neural network.

with the corresponding error function of doing experiments on LeNet neural network for MNIST dataset. Compared with straight line pattern, the latter two could have much more smooth error function curve and help the network model converge much faster.

Based on the above analysis, the total cost function can be summarized as (here we only take l2-norm as an example):

$$E(W) = E_D(W) + \lambda(\alpha E_W^{l2}(W) + (1-\alpha)E_W^{q}(W)) \tag{22}$$

Here $E_D(W)$, $E_W^{l2}(W)$ and $E_W^{q}(W)$ are the generic, l2-norm and quantization regularization term, respectively. Accordingly, the model synapse parameters can be updated by the following rule after forward and back propagation in each iteration:

$$W_k \leftarrow W_k - \eta\frac{\partial E_D(W)}{\partial W_k} - \eta\lambda(\alpha W_k + (1-\alpha)\mathrm{sgn}(W_k - Q(W_k))) \tag{23}$$

6.1 Incremental Quantization

In order to implement deformable regularization method more efficiently, we propose a novel incremental layer-wise quantization (ILQ) framework inspired by the INQ work presented by [41]. The key idea of INQ work is to use weights partition operation for group generating on the pre-trained neural networks model. Then quantization operation will be applied to one weight group while re-train the other groups to compensate the accuracy loss brought by quantization. In the following iterations, quantized weights will firstly be fixed, and weight partition, quantization and re-training will be repeated on the rest parts of weights. In our ILQ framework, weights partition is applied layer by layer, in other words, all weights on one layer will be partitioned as one single group. The whole ILQ framework is demonstrated in Figure 21. At the very beginning, the neural networks will be trained from scratch and the pre-trained model will be prepared for the follow-up operations. Deformable regularization methods together with distribution-aware quantization will be applied to the first convolutional layer in the 1st iteration and quantized weights can be got after this DR operation. Then quantized weights on first convolutional layer will be fixed and continue to apply DR to the second convolutional layer and so on forth. While at the end, before the hardware deployment of model, we will fix all quantized weights for all layers and apply a bias tuning to the whole model.

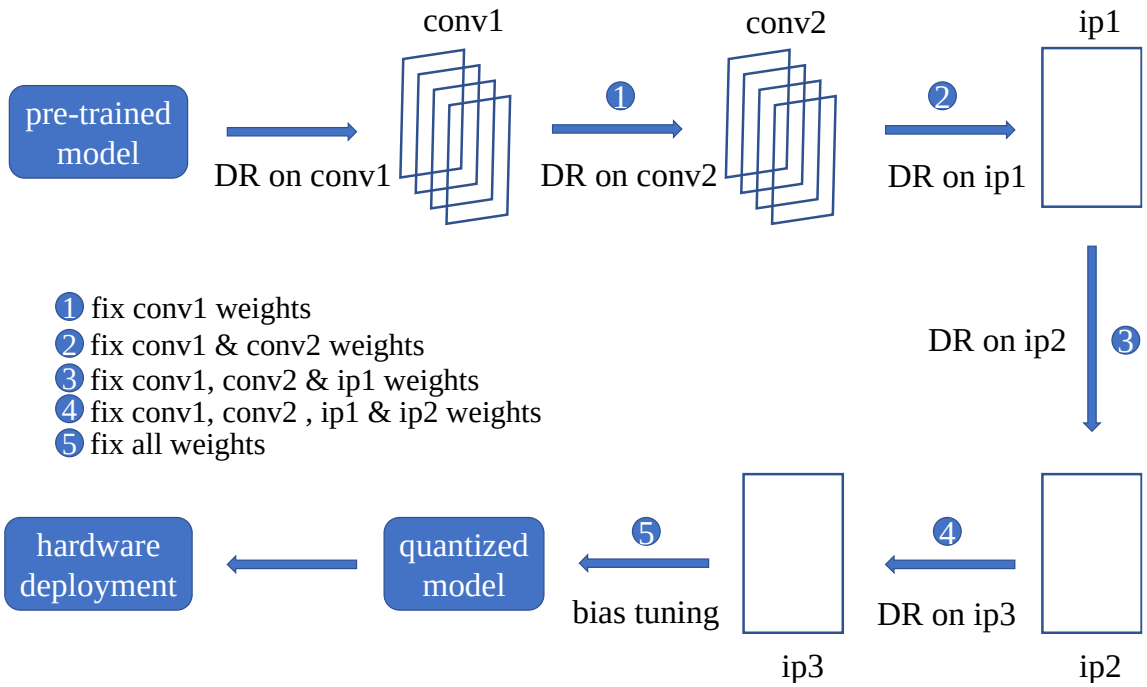

Figure 21 ILQ framework illustration. The pre-trained model will be fed into the neural network model and used for weights initialization. And then all quantized weights will be fixed after applying DR to the previous layer weights. When all weights are quantized, a final bias tuning operation will be applied to the whole neural networks.

ILQ framework together with DR method can efficiently learn a neural network and achieve a high accuracy. Figure 22 demonstrates the accuracy results after applying DR to each previous layer on CIFAR-10 dataset. We only achieve 3.6% accuracy loss by quantizing 32-bit floating-point weights to three levels compared with baseline model with the full-precision weights. As we can see from this figure, DR method outperforms the previous quantization regularization method (5.53% accuracy loss).

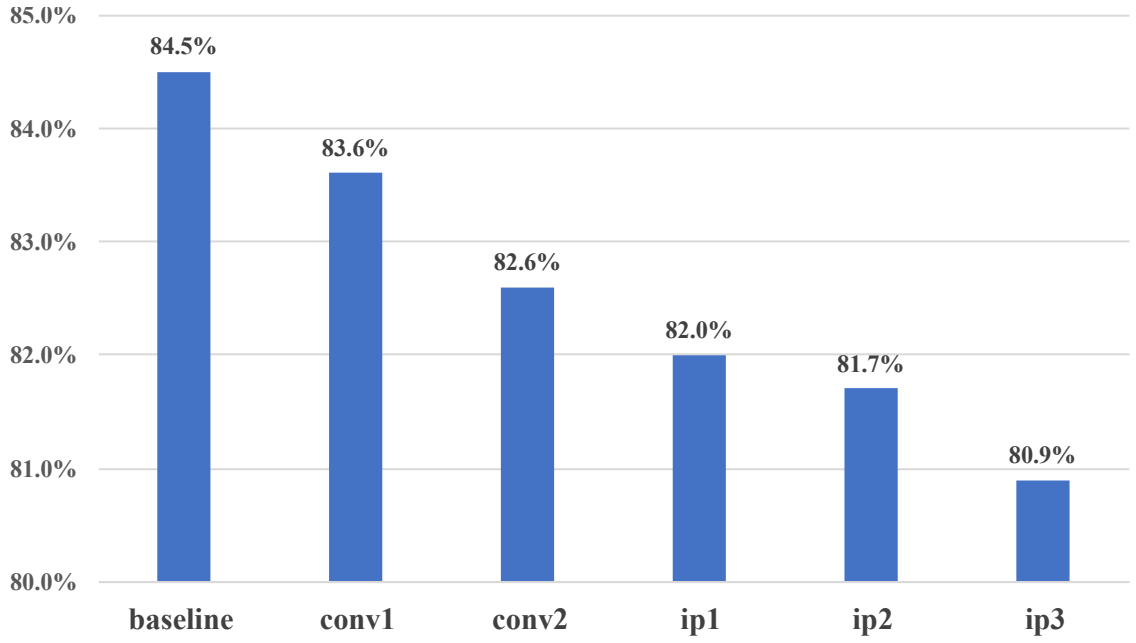

Figure 22 Classification accuracy results on CIFAR-10 dataset after implementing ILQ framework. Each columnar in the figure demonstrates the intermediate processing result by incrementally fixing the quantized weights of previous layers.

6.2 Function Validation of DR on MNIST

Experiments have been conducted on MNIST dataset using TensorFlow framework to evaluate the deformable regularization method together with distribution-aware quantization and bias tuning methods. Experimental results can be clearly visualized on TensorBoard, a suite of visualization tools accompanying with Tensorboard. Figure 23 and Figure 24 demonstrates the deformable quantization process on two convolutional layers and two fully-connected layers of LeNet-5-like CNN neural network structures. From these two figures starting from the very top to the very bottom along the y-axis, we can easily see that at the beginning of training, the neural network is more emphasize on training the network model, while at the end, the model is targeting

on quantifying weights parameters. During the whole training process, the model is gradually transforming from learning with full-precision parameters to quantized parameters. Also, we compared the deformable regularization with basic quantization regularization method on MNIST dataset by combining the distribution-aware quantization and bias tuning methods. Only 0.1% accuracy loss is observed for deformable regularization method, which outperforms the results obtained by utilizing quantization regularization method (0.19%). More detailed information is shown in Table 8.

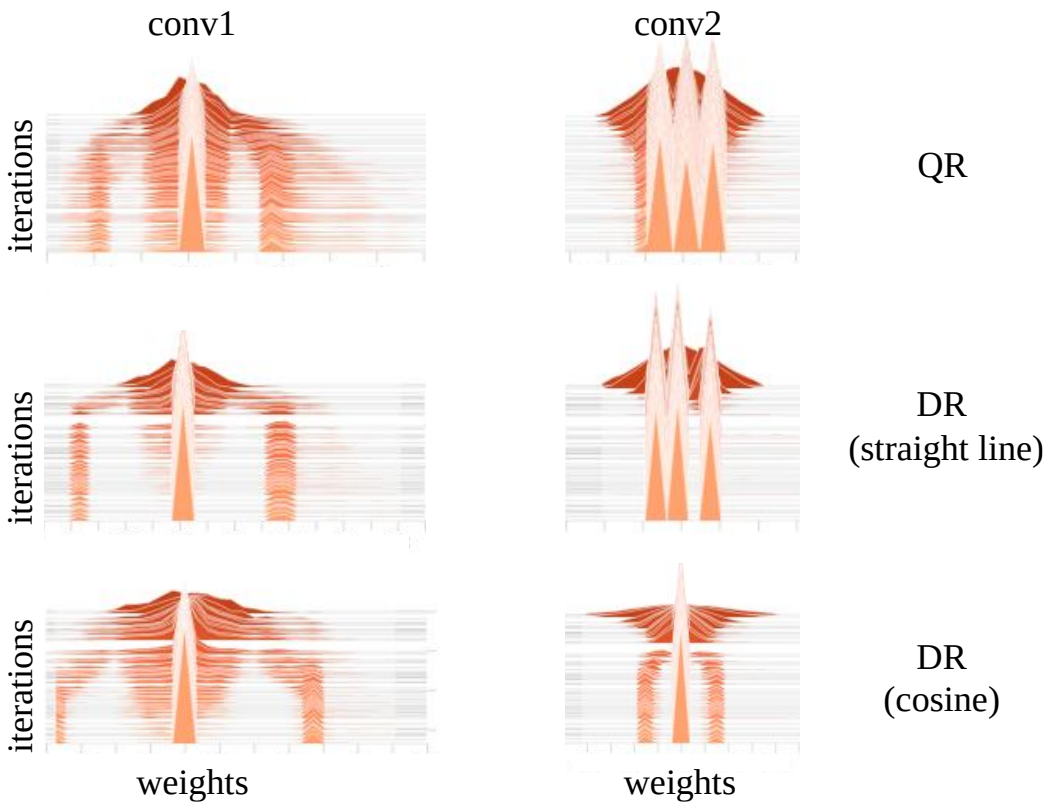

Figure 23 Deformable quantization process on two convolutional layers and the experiments are conducted on MNIST dataset using LeNet-5-like neural networks. The x-axis is the weights values and the y-axis is the training iterations. From the very top to the bottom along the y-axis, it shows the whole training process.

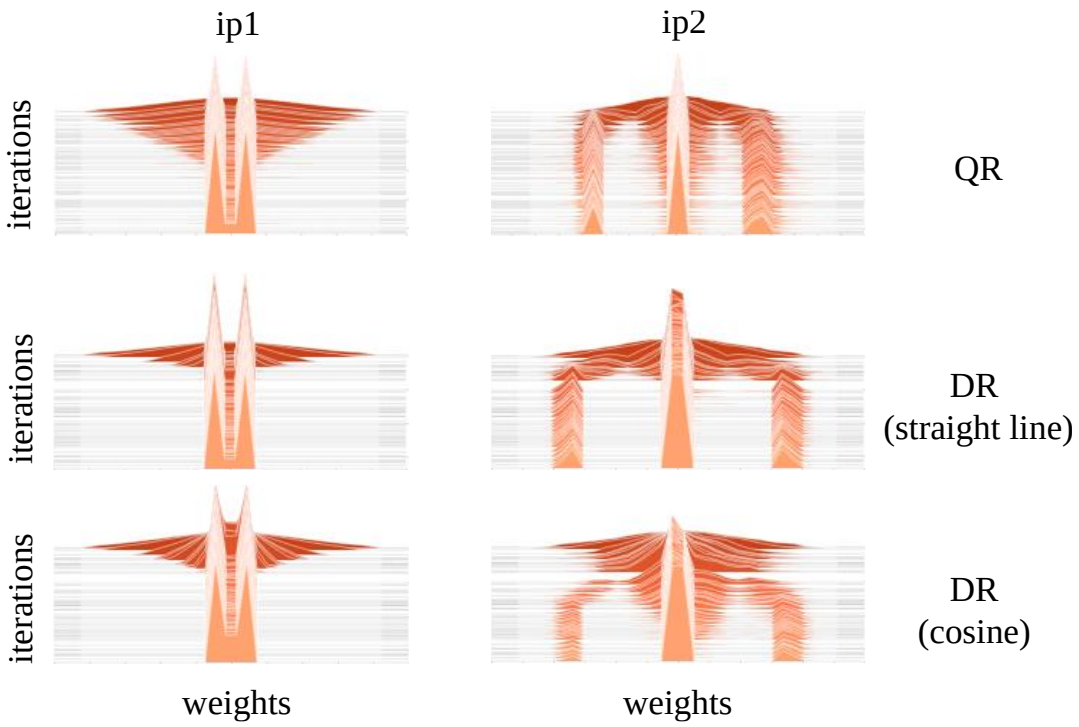

Figure 24 Deformable quantization process on two fully-connected layers and the experiments are conducted on MNIST dataset using LeNet-5-like neural networks. The x-axis is the weights values and the y-axis is the training iterations.

Table 8. The accuracy measurement for DR and QR on MNIST

DQ	QR	DR	BT	Baseline	Accuracy	Drop
√	√		√	99.15%	98.96%	0.19%
√		√	√	99.2%	99.1%	0.1%

6.3 Function Validation of DR on CIFAR-10

Since promising results have been achieved on MNIST dataset utilizing LeNet-5-like neural network, in order to explore its generality, we also implement this advanced quantization method on more complex and challenging CIFAR-10 dataset. Table 9 shows the comparison results between basic quantization regularization and advanced deformable regularization methods. As we can see that deformable regularization method can achieve less accuracy loss, which proves that deformable regularization method is more efficiently and friendly in the quantization process during training.

6.4 Discussion

Our previous research study specifies for spiking neural networks, where the probability distribution can only be biased to two poles (0 or 1). In this work, we extend the method to memristor-based neural networks adopted by state-of-the-art research and large-scale applications. The proposed methods can regularize the floating-point weights to multiple levels with uniform or nonuniform quantization. For example in our CIFAR-10 experiments, the quantization points in layer conv1, conv2, conv3 and ip1 are [0:12; 0; 0:12] , [0:08; 0; 0:08] , [0:02; 0; 0:02] and [0:008; 0; 0:008] , respectively. Moreover, we discharge the reliance on the floating-point layer and explore a pure ternary precision solution. Comprehensive experiments and analyses on MLP and CNN using MNIST and CIFAR-10 datasets arealso conducted. Our experiments on MNIST shows negligible accuracy drop (0.1% in CNN), which is much better than the previous work like.

Table 9. The accuracy measurement for DR and QR on CIFAR-10

DQ	QR	DR	BT	Baseline	Accuracy	Drop
√	√		√	82.12%	76.59%	5.53%
√		√	√	84.5%	80.9%%	3.6%

From the aspect of the system implementation, there are extensive research studies on binary neural networks deployed in traditional platforms such as CPUs, GPUs and FPGAs. However, those approaches may not suitable for the hardware characteristics of brain-inspired systems like memristor-based systems. For example, BinaryConnect uses L2-SVM layer, which is very costly to be implemented by memristor hardware. In circuit design, bias has the characteristic of adjustability, which inspires our bias tuning method in this work. As shown in the paper, bias tuning can be used to control quantization accuracy as well as overcome the process variation of memristor technology.

Furthermore, in our work, the quantization regularization method and deformable regularization method are directly applied into model training stage, which can well overcomes the big non-differential issue existing in quantization during feedforward and backforward propagation. This is because that the quantization regularization terms added to the generic error function for both methods are differential. Take the quantization regularization method as an example, it can descent the distance between a weight and its nearest quantization level with a constant gradient in each iteration. Quantifying weight parameters in training stage can largely alleviate the performance degradation for network model, one reason lies in that during train- ing, network model can be re-trained to tune weight parameters after quantization in later training iterations. The other important reason is that after quantization term is added to the generic cost

function term, both terms can be minimized during training stage which guarantees the quantization and performance degradation concurrently.

7.0 TRNG Design Leveraging Emerging Memristor Technology

7.1 Introduction

Random number generators (RNGs) are broadly used in various systems and applications where unpredictable data are required, such as communication systems, statistical sampling, computer simulation, and cryptography systems[48]. There are two types of typical RNG designs, pseudo random number generator (PRNG) and true random number generator (TRNG). PRNG generates a sequence of numbers by injecting an initial seed to a given computing algorithm. Because the initial seed is deterministic, the properties (correlation, probability distribution, etc.) of these numbers can only be an approximation of true randomness, that is, the number sequence is pseudo random. TRNG, instead, usually leverages unpredictable physical phenomenon, such as thermal noise, random telegraph noise (RTN), atmospheric noise, electromagnetic and quantum [49]. Random data plays a crucial role in system protection of many applications where the true stochastic characteristic is highly appreciated.

Thermal noise is an intrinsic noise induced by thermal agitation of charge carriers (usually the electrons) inside an electrical conductor at equilibrium, which occurs regardless of applied voltage. RTN refers to a kind of electronic noise in semiconductors: when applying discrete voltage or current levels on semiconductors, sudden step-like RTN signals can be generated. Traditional thermal-noise-based TRNG usually is composed of a stochastic signal source, multi-level amplifiers, A/D converter and post-processing circuits [50]. Recently, a TRNG based on RTN in contact resistive random access memory (CRRAM) was proposed in which the high- and low-resistance states (HRS and LRS) of CRRAM are subject to RTN and therefore the resistance

fluctuations can be converted to a stream of random bits [51]. Some TRNG designs leveraging the nanotechnologies have also been investigated. For example, Vivoli et al. presented a device-independent quantum TRNGs using a photon pair source based on spontaneous parametric down conversion (SPDC) which can gain both high entropy and high rate of random bit generation [52]. Spin dice is a spintronic-based TRNG that utilizes the stochastic nature of spin-torque switching in a magnetic tunnel junction (MTJ) to generate random numbers [53].

Memristors, as emerging two-terminal nonlinear dynamic electronic devices [54], have been extensively studied in recent years. Because of the advantages of good scalability, high endurance and ultra-low power consumption [55]. Memristors have been applied in non-volatile memory storage, logic implementation and neuromorphic computing systems [56][57][58]. Moreover, the memristive behaviors in various memristive devices have been thoroughly investigated, in which the stochastic processes have been clearly demonstrated [59][60]. For instance, the distribution of static memristances at HRS/LRS can be approximated with a lognormal probability density function, and the cumulative probability of dynamic switching from one static state to the other is also a lognormal function of the applied voltage. The standard deviation of the static stochastic behavior is negligible compared to the large gap between HRS and LRS, making memristor as an ideal component for binary data storage. Due to the big variance of physical materials and the flexible configuration in programming operation, the dynamic switching of memristive devices demonstrates a very large scalability. The state-of-the-art switching performance in real tantalum-oxide based memristors showed the cycling endurances of over 1012 cycles and fast switching speed below 10ns [61]. Moreover, the sub-nanosecond switching time has been demonstrated through tantalum-oxide based memristors with durations of 105 and 120ps for low- and high-memristance switching, respectively [62].

In this work, we propose a novel memristor-based true random number generator (MTRNG) design by leveraging the stochastic behaviors of memristor. By modulating the width and amplitude of programming pulses applied on memristor devices, the zero-versus-one distribution and the sampling rate of bit streams can be flexibly adjusted. More importantly, the adoption of memristor technology effectively simplifies the structure of TRNG, offering a compact, fast and energy-efficient design. To further improve the entropy of random bit streams, we propose to enhance the design by integrating two basic (1-branch) MTRNGs through an XOR gate. The circuit simulations show that the clock of 1-branch and 2-branch designs based on TiO2 memristors [67][68] can reach at 1.05GHz and 0.96GHz with the power assumptions of 31.1μW and 80.3μW, respectively.

7.2 Preliminary

7.2.1 Memristor

As the fourth fundamental component besides resistor, capacitor and inductor, memristor describes the dynamic relationship between charge (q) and flux (φ) [63]. Particularly, it can "remember" the total electric flux flowing through the device and represent it as the memristance (M).

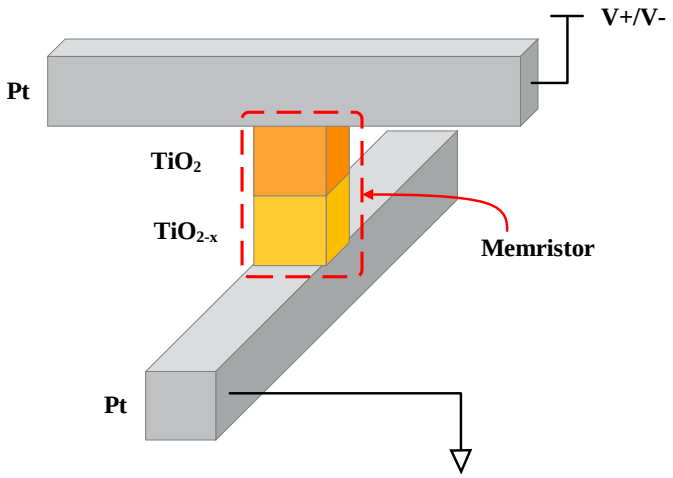

Figure 25 The structure of a TiO2 memristor

Figure 25 illustrates the structure of a TiO_2 memristor sandwiched between two metal wires. The device consists of two titanium dioxide layers: the doped layer TiO_{2-x} is filled with oxygen vacancies and therefore has a high conductivity; the pure TiO_2 (undoped layer), in contrast, has the character of insulator. While there is a positive bias voltage (V+) applying to the device, the oxygen vacancies will be forced into the undoped area and therefore the total resistance (or memristance) continuously reduces. On the contrary, a reversed bias voltage (V–) will force the vacancies back to its original position and raise the memristance. Without enough external voltage, the oxygen vacancies within the structure remain so as that the memristance maintains [64].

For ease of explanation, we define the following terminologies and variables that will be referred in this paper:

- Static states – the state in which the equivalent resistance is high (R_{off}) or low (R_{on}). OFF state and ON state denote the states of R_{off} and R_{on}, respectively.

- Dynamic switching – the process of switching from one static state to the other. OFF switching refers to the process switching from ON to OFF, while ON switching corresponds to the opposite operation.
- Programming pulse – the voltage pulse applied on the memristor to trigger the dynamic switching process.

7.2.2 Stochastic Behaviors of Memristors

Stochastic behaviors have been widely observed in metal oxide based memristor devices, including the variations in static states and dynamic switching processes.

Static stochastic behavior: The final resistance value of a memristor during a programming operation is not deterministic but a stochastic variable related to the voltage amplitude and duration of the programming pulse. The randomness of Ron and R_{off} is denoted as the static stochastic behavior of memristors. The distributions of Ron and R_{off} usually follow the lognormal probability density functions [65][66].

Dynamic stochastic behavior is resulted by means of the complicated stochastic oxide electroforming process during ON/OFF switching [65] in which the successful switching probability monotonically increases along with the increase of the amplitude and/or duration of programming pulse. More specific, the cumulative probability function of a successful switching between Ron and R_{off} follows a lognormal distribution [67].

7.3 Methodology

In this work, we propose a new memristor-based true random number generator (MTRNG) design. The reconfigurable dynamic stochastic behavior of memristors provides a flexible design space for various applications with different sampling rate requirements. Though the memristance value of each programming is not deterministic due to the static stochastic behavior, the stability of the design can still be promised by the large gap between the high and low memristance states. Moreover, we design and customize a 2-branch MTRNG which integrates two pieces of basic 1-branch MTRNGs. Markov chain analysis shows that the 2-branch scheme further maximizes the entropy of the random number sequence. Our work not only presents a novel circuit to generate random number streams but also can be generalized to a statistical methodology for memristor-based design.

7.3.1 Stochastic Model of TiO2 Memristor

Because of the static stochastic behavior, the memristor resistance in ON or OFF state is not deterministic but random, even for a single identical device. In a TiO_2 memristor, the distributions of static state resistance Ron and R_{off} both can be approximated to the lognormal probability density function (pdf) such as [67]:

$$f_x(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(\ln x/\mu)^2}{2\sigma^2}\right), \quad x > 0 \qquad (24)$$

where, μ is the normal mean and σ is the standard deviation of the normal distribution of the initial barrier width of the memristor device. Certainly, the parameters of μ and σ for Ron and R_{off} are different. Figure 26 presents the real measurement data of a TiO_2 memristor [68].

Giving E[R_{on}] and E[R_{off}] as the means of R_{on} and R_{off}, respectively; and their standard deviations are D[R_{on}] and D[R_{off}], respectively. The device demonstrated in Figure 26 has E[R_{on}] ≈ 105Ω and E[R_{off}] ≈ 108Ω. Both D[R_{on}] and D[R_{off}] are more than 2 orders smaller than the gap between the means (E[R_{off}] − E[R_{on}]). Such a highly isolated binary characteristic in memristors guarantees an ideal physical mechanism for MTRNG design. Details shall be presented and discussed in Section 7.4.

The dynamic stochastic behavior refers to the successful switching probability between ON and OFF state. Under an external programming pulse, the switching probability is determined by the voltage amplitude and the pulse width (duration) t. The cumulative distribution can be approximated by lognormal distribution [67]:

$$F(t;\tau,\sigma_t)=\int_0^t \frac{1}{\sqrt{2\pi}\sigma_t T} e^{-\left(\frac{\ln T}{\sqrt{2}\tau\sigma_t}\right)^2} dT = \frac{1}{2} erfc\left(-\frac{\ln t}{\sqrt{2}\tau\sigma_t}\right). \quad (25)$$

Where, τ is the mean of the switching time, which has an exponential dependency on the applied voltage amplitude, while its deviation σt only has a weak dependence on the voltage.

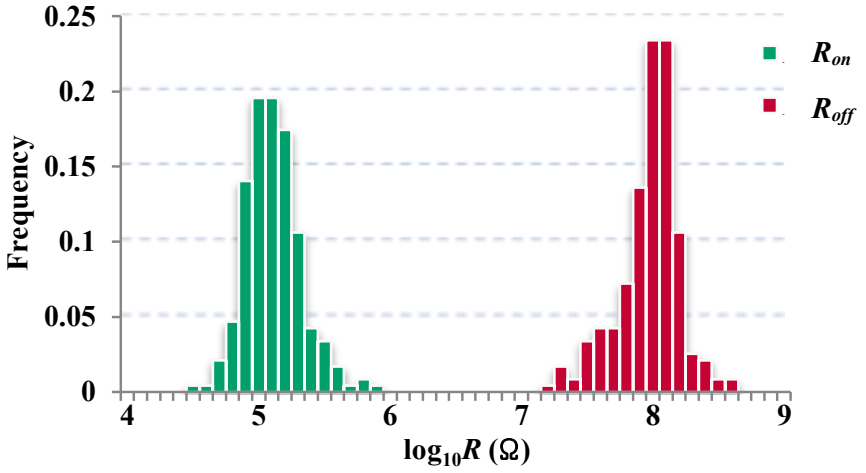

Figure 26 Static stochastic behavior

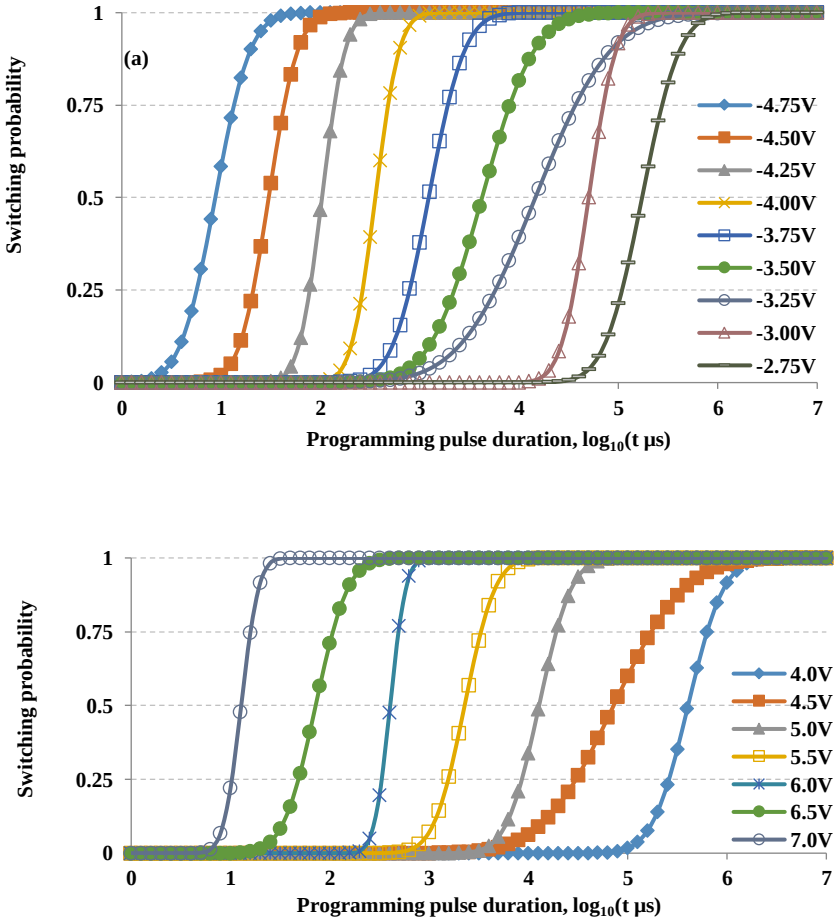

Figure 27 Cumulative switching probability distribution for ON (a) and OFF (b) switching under different applied voltage amplitude

Figure 27 shows the cumulative switching probability distributions of ON and OFF switching. Both results reveal that increasing the programming duration of a constant-amplitude pulse can increase the switching probability. Moreover, a larger voltage amplitude decreases the required programming duration to reach a given switch probability.

7.3.2 The MTRNG Design

Our proposed MTRNG design switches between the programming mode and the reading mode to generate the random bit stream. In the programming mode, a programming pulse is applied on the memristor to trigger a dynamic switching between the ON and OFF states. In the reading mode, the programmed binary resistance is converted to a binary bit. In the design, the selection of the programming pulse amplitude determines the maximal allowable sampling rate of the bit stream. We can control the ratio of the probability of 0's and 1's by modulating the programming duration. Ideally, a uniform distribution of 0/1 bit stream can be obtained by aligning the pulse width to the switching probability of 0.5 under a given pulse voltage (refer Figure 27).

Figure 28 depicts the proposed MTRNG circuit with the following key control and internal signals: V_{dc_r}, V_{dc_on} and V_{dc_off} are the DC voltage sources used in reading mode, the ON switching and the OFF switching programming, respectively.

V_{read} is the control signal to enable the reading mode to detect the state of the memristor. V_{p_on} and V_{p_off} are used for program the memristor to ON and OFF states, respectively.

V_d is the bias voltage representing the state of memristor. It determines the generated output bit of the MTRNG.

Vg is used to modulate V_d for bit generation.

Clk is the clock signal to control the data capture at D flip-flop.

The sequence of control signals is also illustrated in Figure 28. V_{p_on} and V_{p_off} are turned on alternatively to enable the ON and OFF switching. Under the ideal condition with the sufficient programming voltage and pulse duration, the memristor can always be programmed, that is, the device switches between ON and OFF states. By properly controlling the programming voltage amplitude together with the pulse duration corresponding to the required bit distribution, the

switching of the memristor becomes more random. In our design, following every programming period is a read operation enabled by V_{read}. The ON and OFF states of the memristor will be transferred to 1 or 0, respectively, under appropriate Vg setup. Here, a D flip-flop is used to recover distorted binary signal resulted by stochastic memristance values. More details of our design configurations and the experimental results of simulation shall be conveyed and discussed in Section 7.4.

The simple MTRNG in Figure 28 can be used to generate a stream of random bits. However, the scheme cannot obtain the maximal entropy because the memristor will keep at the ON or OFF state if the previous switching fails. Take the signal sequence in Figure 28 as an example and assume the previous state of the memristor is OFF: if an ON switching triggered by V_{p_on} fails so that the memristor remains as OFF, the following OFF switching initialized by V_{p_off} does not affect the state of the memristor. In such a case, this OFF switching is not a stochastic process.

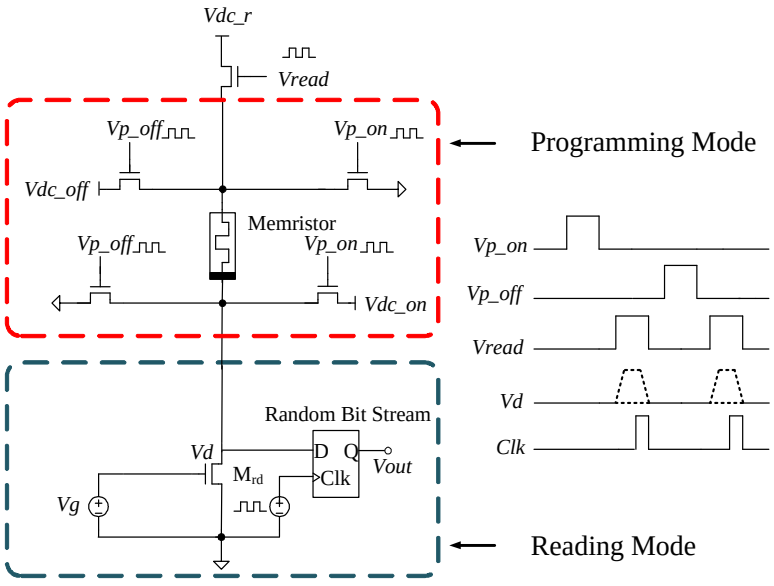

Figure 28 The scheme of the basic 1-branch MTRNG design

To improve the entropy of the random bit stream, we further enhance the design. As illustrated in Figure 29, it integrates two basic (1-branch) MTRNGs through an XOR gate. Because the stochastic switching of one memristor is independent to the other, the entropy of the random bit stream through the XOR function can be maximized under appropriate dynamic switching probability. We name this scheme as 2-branch MTRNG design.

7.3.3 MTRNG Markov Chain Analysis

Here, we will give a detailed probability analysis for both the basic 1-branch and the enhanced 2-branch MTRNG designs based on the Markov chain analysis. The variables used include:

- $P_{even}(i)$ – the probability of an even bit in the random bit stream as logic state $i \in (0,1)$ after ON switching operation.
- $P_{odd}(i)$ – the probability of an odd bit in the random bit stream as logic state $i \in (0,1)$ after OFF switching operation.
- P_{on} – the ON switching probability to which the V_{p_on} cumulates, which is also the successful switching probability from OFF to ON state shown in Figure 27 (upper).
- P_{off} – the OFF switching probability to which the V_{p_off} cumulates. It is equivalent to the successful switching probability from ON to OFF state shown in Figure 27 (lower).
- S – the state space of a bit in the random stream, $S = \{S_{mn} \mid m=0,1 \text{ and } n=0,1\}$. m and n denote the position and value of the bit, respectively. The bit is the even-th (odd-th) one in the stream if m=0 (m=1) and its value is n. n=0 (n=1) corresponds to R_{off} (R_{on}).

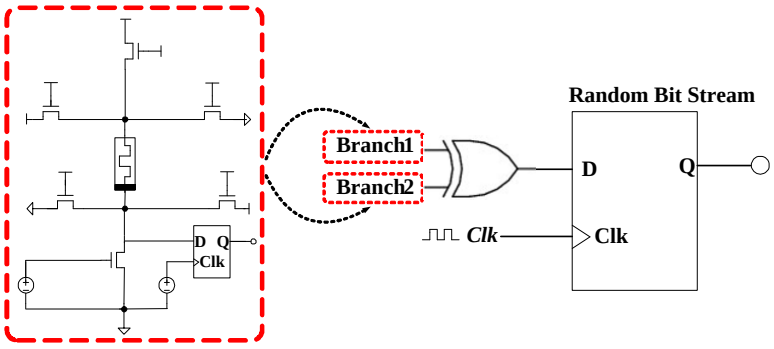

Figure 29 The scheme of the enhanced 2-branch MTRNG design

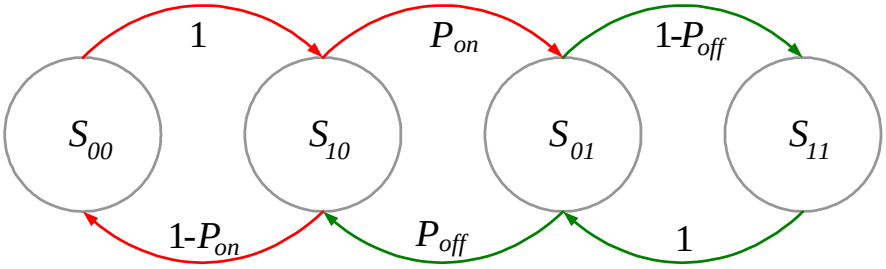

Figure 30 The state transition diagram

- $\mathbf{P}^{(2)}_{even}$ – the two-step transition matrix between two sequential even bits.
- $\mathbf{P}^{(2)}_{odd}$ – the two-step transition matrix between two sequential odd bits.
- $P_{2\text{-branch}}(i)$ – the probability distribution of the output of 2-branch MTRNG (i=0,1).

Figure 30 summarizes the state transition diagram. As aforementioned in Section 7.3.2 , the transition probability of 1 exists because of the invalid ON (OFF) switching operation on ON (OFF) state.

The stochastic process of generating the random bit stream is a first-order Markov chain. To simplify the Markov chain analysis, we separately calculate the 0/1 probability distributions of the even and the odd bits, such as:

$$\mathbf{P}_{even}^{(2)} = \begin{bmatrix} 1 & 0 \\ P_{off} & 1-P_{off} \end{bmatrix} \cdot \begin{bmatrix} 1-P_{on} & P_{on} \\ 0 & 1 \end{bmatrix}, \tag{26}$$

$$\mathbf{P}_{odd}^{(2)} = \begin{bmatrix} 1-P_{on} & P_{on} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ P_{off} & 1-P_{off} \end{bmatrix}. \tag{27}$$

Given $0 < P_{off} < 1$ and $0 < P_{on} < 1$, every element in $\mathbf{P}_{even}^{(2)}$ and $\mathbf{P}_{odd}^{(2)}$ is larger than 0. As such, the Markov chains with the transition matrixes of Eqs. (26) and (27) have stationary distributions $\boldsymbol{\pi}_{even}$ and $\boldsymbol{\pi}_{odd}$, denoting the stationary 0/1 distributions of even and odd bits, respectively. They satisfy

$$\begin{cases} \boldsymbol{\pi}_{even} \cdot \mathbf{P}_{even}^{(2)} = \boldsymbol{\pi}_{even} = \left[P_{even}(0), P_{even}(1) \right] \\ \boldsymbol{\pi}_{odd} \cdot \mathbf{P}_{odd}^{(2)} = \boldsymbol{\pi}_{odd} = \left[P_{odd}(0), P_{odd}(1) \right] \end{cases}. \tag{28}$$

Given P_{on} and P_{off}, the solution of the equation set is

$$\begin{cases} P_{even}(0) = 1 - P_{even}(1) = \dfrac{P_{off} - P_{on} \cdot P_{off}}{P_{on} + P_{off} - P_{on} \cdot P_{off}} \\ P_{odd}(0) = 1 - P_{odd}(1) = \dfrac{P_{off}}{P_{on} + P_{off} - P_{on} \cdot P_{off}} \end{cases}. \tag{29}$$

To maximize the Shannon entropy of random bit stream generated by MTRNG, the probability should be uniformly distributed:

$$P_{odd}(0) = P_{odd}(1) = P_{even}(0) = P_{even}(1) = 0.5. \tag{30}$$

Note that Eq. (7) cannot be a solution of Eq. (6), indicating that the basic 1-branch MTRNG design cannot generate an entropy-maximized random number sequence. Only skewed probability distribution can be produced where $P_{even}(i) \neq 0.5$ or $P_{odd}(i) \neq 0.5$.

The enhanced 2-branch MTRNG design, in contrast, can obtain the maximized entropy by appropriately setting P_{on} and P_{off}, e.g., aligning V_{p_off} of a branch to V_{p_on} of the other branch and setting $P_{even}(0) = P_{even}(1) = 0.5$, uniformly distributed $P_{2\text{-}branch}(i)$ can be satisfied because

$$\begin{cases} P_{2-branch}(0) = P_{even}(0) \cdot P_{odd}(0) + P_{even}(1) \cdot P_{odd}(1) = 0.5 \\ P_{2-branch}(1) = P_{even}(0) \cdot P_{odd}(1) + P_{even}(1) \cdot P_{odd}(0) = 0.5 \end{cases} . \quad (31)$$

In this case,

$$P_{on} = \frac{P_{off}}{1 + P_{off}} . \quad (32)$$

P_{off} and P_{on} shall be carefully selected for the enhanced 2-branch design. From the one hand, smaller P_{off} and P_{on} are more preferable because the circuit can operate under a faster sampling rate. From the other hand, we tend to avoid the steep slope of switching probability curve because a tiny fluctuation of programming duration can result in a large drift of the switch probability.

7.4 Experiment

We evaluate the proposed MTRNG designs through circuit simulations in Cadence Virtuoso environment. The 180nm CMOS technology and the memristor device parameters in [67] were adopted. Here, we first discuss the design configuration followed by the simulation of MTRNGs and the probability distribution of random bits. At the end, the speed and power consumptions of the proposed designs are evaluated and analyzed.

7.4.1 The Selection of Gate Voltage Vg

The gate voltage of transistor M_{rd} (V_g) in Figure 28 is a crucial parameter to modulate the bias voltage V_d and the finally output V_{out}. As aforementioned in Section 3.1, the distributions of Ron and Roff are approximated to the lognormal probability density function. Based on the real measured resistance distribution of a TiO2 memristor in Figure 2, the means of the high and low resistance states, E[R_{on}] and E[R_{off}], are about 105Ω and 108Ω, respectively. Even considering the worst situation where Ron is 106Ω and R_{off} is 107Ω, R_{off} is still one order higher than Ron. Comparing the difference between E[R_{on}] and E[R_{off}] and the noise margin of CMOS transistors, we are able to map the static memristor resistances to binary code by constraining Vg within a critical range.

We start the evaluation with the typical condition when Ron=105Ω and R_{off} =108Ω. To find the critical range for Vg, the memristor resistance is fixed and Vg is scanned from 0V to V_{dc}=1.8V. The simulation results in Figure 31 (upper) show that V_{out} falls from high to low when Vg is higher than a critical voltage and therefore the equivalent resistance of M_{rd} is smaller than a threshold. More specific, under the typical situation when R_{off} =108Ω, V_{out} drops to low as Vg approaches to 0.34V. For Ron=105Ω, the critical falling point is around 0.66V. Thus, the ON and OFF states of memristor can be respectively mapped to HIGH and LOW of V_{out} if setting Vg within the range from 0.34V to 0.66V.

We also verify the circuit stability under the worst scenario condition when R_{on}=106Ω and R_{off} =107Ω. Figure 31 (lower) presents the simulation results. A similar trend as Figure 31 (upper) can be observed except that the allowable range of V_g reduces to 0.43V ~ 0.53V. The narrower critical range indicates the degraded circuit stability. Even though, the inclusion relationship of the critical ranges in Figure 31 (upper) and (lower) shows that Vg in the intersection set can guarantee

our MTRNG functions properly even under the worst scenario condition. Based on the analysis, we set the gate voltage Vg to 0.5V in the following simulations.

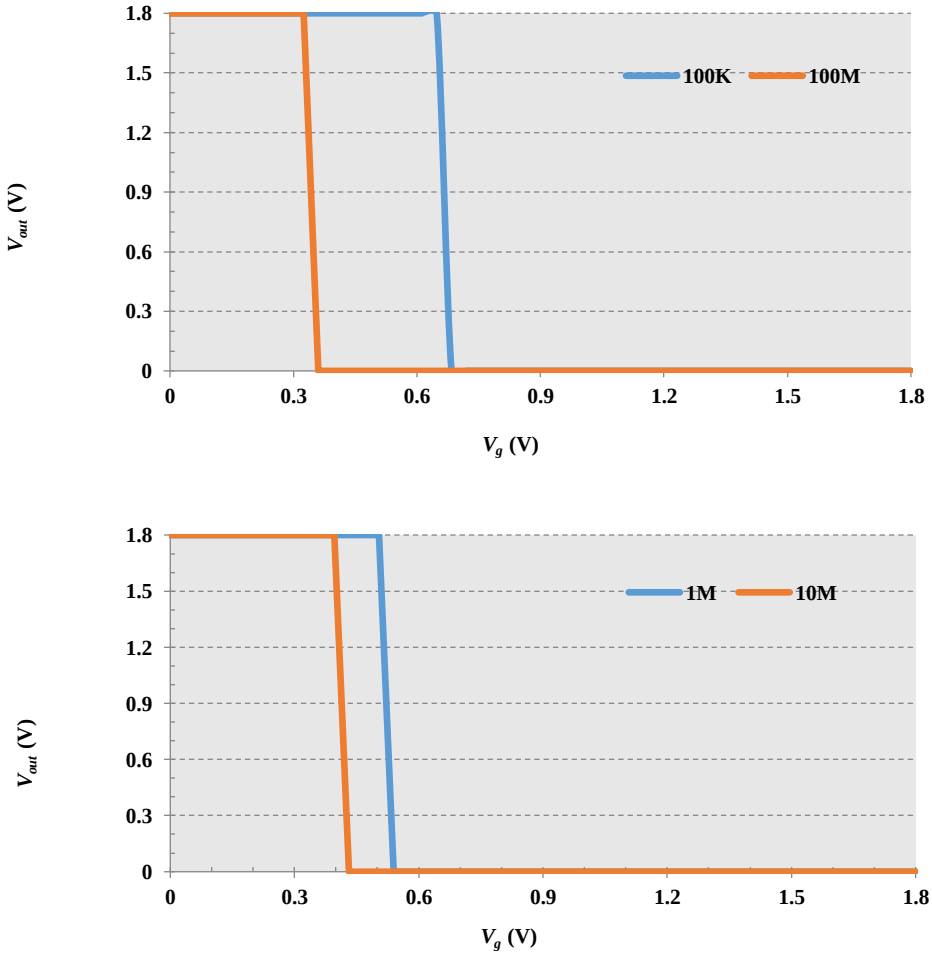

Figure 31 Vg vs. Vout: (a) under the means of the high and low resistance states, as Ron=105Ω and Roff =108Ω; (b) at the worst condition when Ron=106Ω and Roff =107Ω.

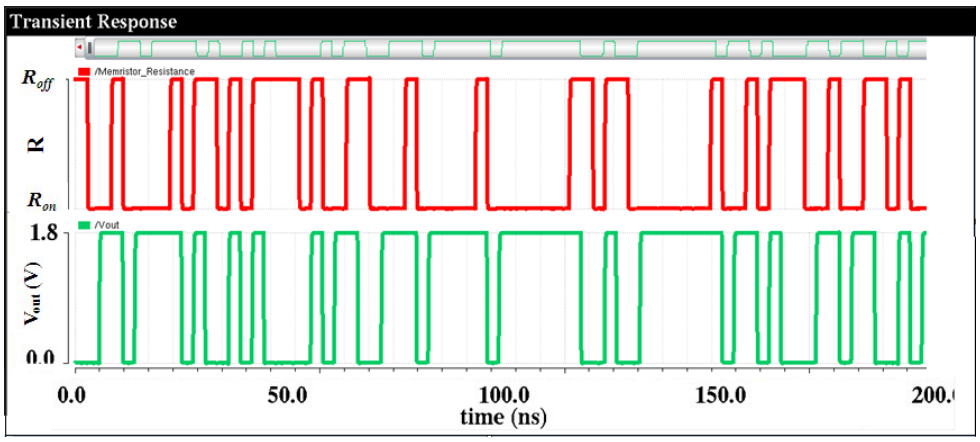

Figure 32 Simulation of 1-branch MTRNG (Ron=105Ω and Roff =108Ω)

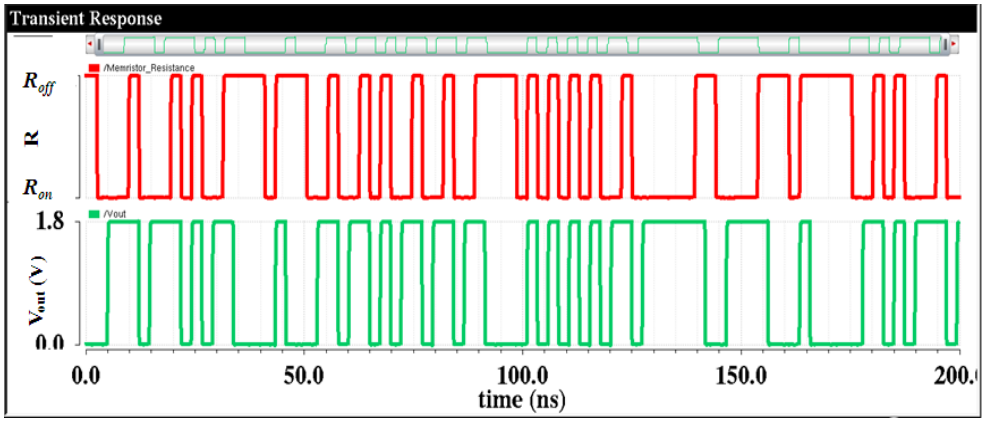

Figure 33 Simulation of 1-branch MTRNG (Ron=106Ω and Roff =107Ω)

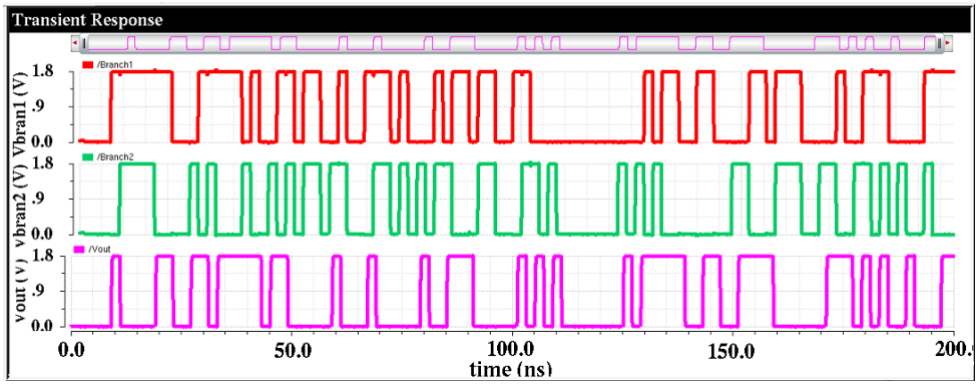

Figure 34 Simulation of 2-branch MTRNG

7.4.2 MTRNG Simulation

Figure 32 and Figure 33 show the simulation results of the basic 1-branch MTRNG at the typical (R_{on}=105Ω and R_{off} =108Ω) and the worst-case (R_{on}=106Ω and R_{off} =107Ω) conditions, respectively. The simulations show that stochastic binary states of memristor can be successfully converted to random bit stream. Even in the extreme situation when R_{off} is very close to Ron, the basic 1-branch MTRNG design still functions properly. Figure 33 shows the simulation result of the enhanced 2-branch MTRNG, the output random bit stream of which is dependent on the signals of two bit sequences generated by the two 1-branch MTRNGs.

To analyze the probability distribution of the 1-branch and 2-branch MTRNG designs, the memristor ON switching and OFF switching probabilities are set to P_{on}=1/4 and P_{off}=1/3, respectively. To ease the explanation, we show the probability distributions of the first 100 bits generated by 1-branch and 2-branch MTRNG in Figure 35. Here, each point represents the probability of logic 1 at the bit. Simulation shows that both MTRNG schemes rapidly converge towards their stationary distributions after only a few steps because of the ergodicity of the Markov chain. The fast convergence of the Markov chain guarantees that the bit probability approaches to the desired distributions quickly.

For the 1-branch MTRNG design, the probability distribution of the odd-th bits is non-uniform. The situation can be solved by passing two bit streams of the 1-branch design through an XOR gate. Consistent to the theoretical analysis in Section 7.3.3, a uniformly distributed random bit stream can be generated via the 2-branch MTRNG design.

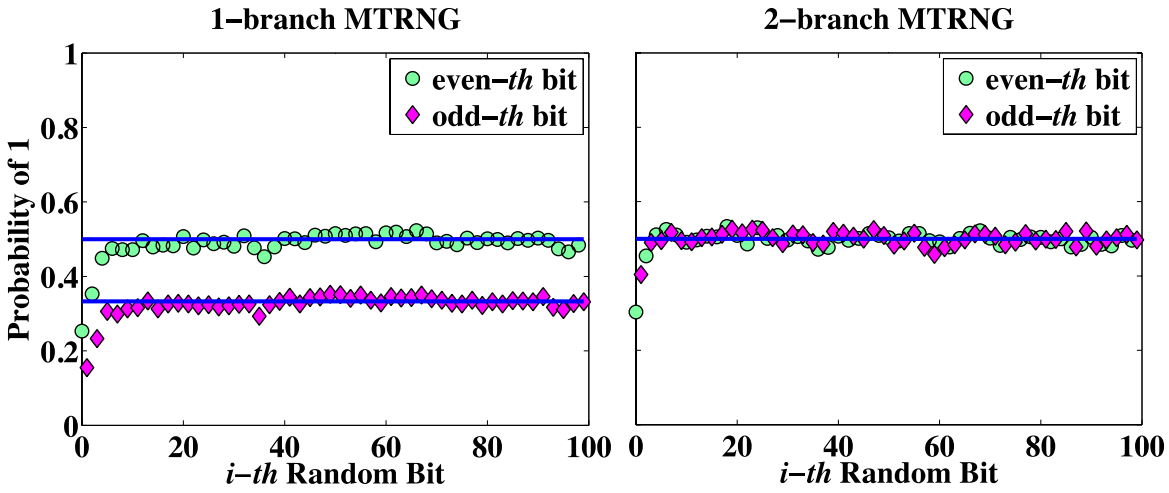

Figure 35 The probability distribution of random bit in the stream generated by 1-branch (left) and 2-branch (right) MTRNG

7.4.3 The Design Evaluation

Traditional thermal noise based TRNGs usually require multistage voltage amplifiers to magnify the weak signals, resulting in high design complexity and cost [68]. The latest random telegraph noise (RTN) based TRNG requires an analogy comparator to convert RTN to binary code [51]. For the reason, its sampling rate is relatively low at only 11.4Hz. Thus, its applications are limited to low-speed systems such as encryption system [51].

Amplifier is not necessary in our MTRNGs for the large bias voltage V_d. The design is realized in a much simpler form: the basic 1-branch MTRNG consists of only one memristor, six access control transistors, and one D flip-flop. Determined by the memristor programming voltage and duration, the proposed MTRNGs can operate under a large frequency range varying from Hz to GHz. Our simulations show that the minimal reading periods of the 1-branch and 2-branch

designs are only 0.95ns (1.05GHz) and 1.04ns (0.96GHz), respectively. Moreover, Figure 36 shows the relationship between the random bit stream sampling period T and the voltage of programming pulse for the 2-branch MTRNG when setting P_{off}=1/3 and P_{on}=1/4. The log function of sampling period approximately linearly decreases with the voltage amplitude.

The detailed power consumption results of the 1-branch and 2-branch MTRNGs are summarized in Table 10. Benefiting from the simple structure and the ultra-low energy characteristic of memristors, MTRNGs obtain low power consumption of tens of μW regardless of 1-branch or 2-branch design styles.

Table 10. Power consumption of MTRNGs

	HRS (μW)	LRS (μW)	Average (μW)
1-branch	16.5	45.6	31.1
2-branch	44.6	115.9	80.3

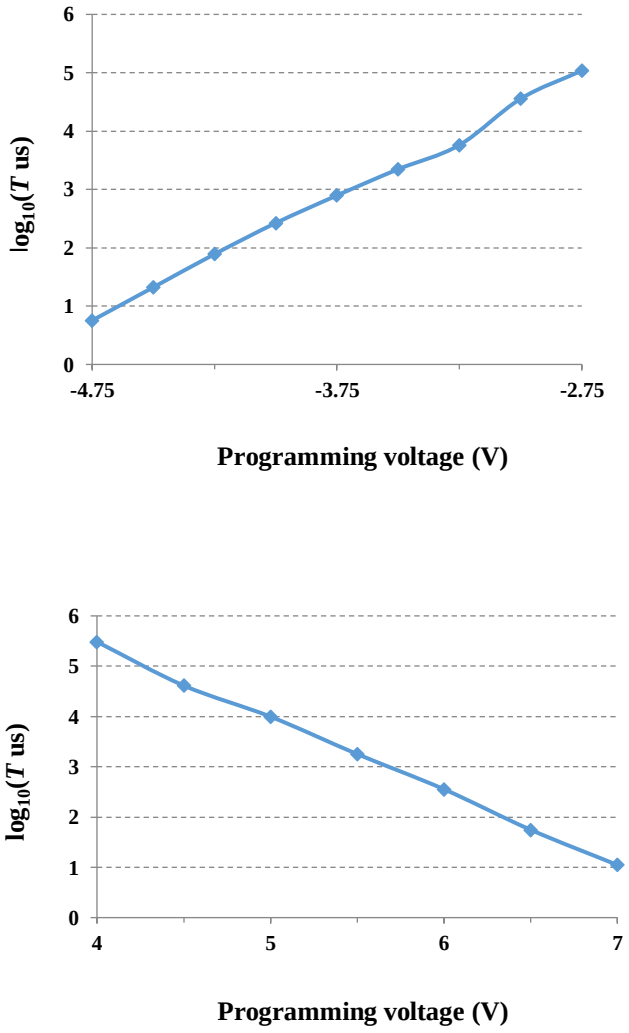

Figure 36 Dependence of programming voltage for random bit stream sampling period in ON switching (upper) and OFF switching (lower)

8.0 Conclusions

Firstly, a framework named group scissor that aims to alleviates the impact of hardware limitations on the NCS implementation of large neural networks has been introduced. Specifically, rank clipping and group connection deletion methods are proposed to reduce area consumption of synapse crossbars and routing area between crossbars, respectively. The experiments show that our methods can reduce crossbar area and routing area to 13.62% and 8.1%, respectively, with no accuracy loss for LeNet. Furthermore, for implementation of more challenging ConvNet, we can safely reduce the crossbar and routing areas to 51.81% and 52.06% respectively without losing classification accuracy. The proposed framework can significantly save hardware area and improve system scalability.

Secondly, in the weight quantization work, we first analyze the impact on accuracy degradation of low-resolution synapses (weights) in neuromorphic hardware implementations theoretically. In order to maintain the high image classification accuracy for neural network model with full precision weights and minimize the performance degradation during NCS deployment, we propose three orthogonal methods (distribution-aware quantization, quantization regularization and bias tuning) to learn synapses with ternary levels. What's more, based on quantization regularization method, we further propose an advanced deformable regularization method with incremental lay-wise quantization framework, which can further improve the low-precision network model performance. We firstly applied three orthogonal methods and their combinations to MLP on MNIST, CNN on MNIST and CNN on CIFAR-10 database, comparable state-of-the-art achievements are obtained: only 0.39%, 0.19%, and 5.53% accuracy loss, respectively. Our work will be more suitable for memristor-based neural networks. And then we applied advanced

deformable method and their combinations to CNN on MNIST and CNN on CIFAR-10 database, comparable state-of-the-art achievements are obtained: only 0.39%, 0.1%, and 3.6% accuracy loss, respectively. Even though our work is conducted based on the theory of memristor devices, all proposed methods in this paper are general solutions and can be applied to any other low-precision NCS design.

At last, a memristor-based true random number generator (MTRNG), which leverages the stochastic behavior of memristors and converts the programmed resistances to random binary bit stream, has been proposed in this work. Besides the basic 1-branch MTRNG, we also enhance the design to 2-branch scheme which can obtain the identical generating probability of bit 1 and bit 0, promising the maximum entropy of random number generation. Sampling rate of our designs can reach at GHz with minimum power consumption of 31.1μW. The proposed MTRNG designs exhibit characteristics of simple structure, compact area, high frequency, low power and flexible configurability.

References

[1] Cassidy, Andrew S., Paul Merolla, John V. Arthur, Steve K. Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta et al. "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores." In The 2013 International Joint Conference on Neural Networks (IJCNN), pp. 1-10. IEEE, 2013.

[2] Hu, Miao, Hai Li, Yiran Chen, Qing Wu, Garrett S. Rose, and Richard W. Linderman. "Memristor crossbar-based neuromorphic computing system: A case study." IEEE transactions on neural networks and learning systems 25, no. 10 (2014): 1864-1878.

[3] Gaba, Siddharth, Phil Knag, Zhengya Zhang, and Wei Lu. "Memristive devices for stochastic computing." In 2014 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2592-2595. IEEE, 2014.

[4] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In Advances in neural information processing systems, pp. 1097-1105. 2012.

[5] Jo, Sung Hyun, Ting Chang, Idongesit Ebong, Bhavitavya B. Bhadviya, Pinaki Mazumder, and Wei Lu. "Nanoscale memristor device as synapse in neuromorphic systems." Nano letters 10, no. 4 (2010): 1297-1301.

[6] Esser, Steve K., Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur et al. "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores." In The 2013 International Joint Conference on Neural Networks (IJCNN), pp. 1-10. IEEE, 2013.

[7] Xu, Cong, Xiangyu Dong, Norman P. Jouppi, and Yuan Xie. "Design implications of memristor-based RRAM cross-point structures." In 2011 Design, Automation & Test in Europe, pp. 1-6. IEEE, 2011.

[8] Li, Boxun, Yuzhi Wang, Yu Wang, Yiran Chen, and Huazhong Yang. "Training itself: Mixed-signal training acceleration for memristor-based neural network." In 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 361-366. IEEE, 2014.

[9] Wen, Wei, Chunpeng Wu, Yandan Wang, Kent Nixon, Qing Wu, Mark Barnell, Hai Li, and Yiran Chen. "A new learning method for inference accuracy, core occupation, and performance co-optimization on TrueNorth chip." In 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1-6. IEEE, 2016.

[10]Strukov, Dmitri B., Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. "The missing memristor found." nature 453, no. 7191 (2008): 80.

[11]He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[12]Liang, Jiale, and H-S. Philip Wong. "Cross-point memory array without cell selectors—Device characteristics and data storage pattern dependencies." IEEE Transactions on Electron Devices 57, no. 10 (2010): 2531-2538.

[13]Liu, Beiye, Hai Li, Yiran Chen, Xin Li, Tingwen Huang, Qing Wu, and Mark Barnell. "Reduction and IR-drop compensations techniques for reliable neuromorphic computing systems." In Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design, pp. 63-70. IEEE Press, 2014.

[14]Wen, Wei, Chi-Ruo Wu, Xiaofang Hu, Beiye Liu, Tsung-Yi Ho, Xin Li, and Yiran Chen. "An EDA framework for large scale hybrid neuromorphic computing systems." In 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1-6. IEEE, 2015.

[15]Akopyan, Filipp, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam et al. "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34, no. 10 (2015): 1537-1557.

[16]Hu, Miao, Yu Wang, Qinru Qiu, Yiran Chen, and Hai Li. "The stochastic modeling of TiO 2 memristor and its usage in neuromorphic system design." In 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 831-836. IEEE, 2014.

[17]Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." In European Conference on Computer Vision, pp. 525-542. Springer, Cham, 2016.

[18]Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." In Advances in neural information processing systems, pp. 3123-3131. 2015.

[19]Courbariaux, Matthieu, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1." arXiv preprint arXiv:1602.02830 (2016).

[20]Chua, Leon. "Memristor-the missing circuit element." IEEE Transactions on circuit theory 18, no. 5 (1971): 507-519.

[21]Yi, Wei, Frederick Perner, Muhammad Shakeel Qureshi, Hisham Abdalla, Matthew D. Pickett, J. Joshua Yang, Min-Xian Max Zhang, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. "Feedback write scheme for memristive switching devices." Applied Physics A 102, no. 4 (2011): 973-982.

[22]Hu, Miao, Hai Li, Qing Wu, and Garrett S. Rose. "Hardware realization of BSB recall function using memristor crossbar arrays." In Proceedings of the 49th Annual Design Automation Conference, pp. 498-503. ACM, 2012.

[23]Kim, Minje, and Paris Smaragdis. "Bitwise neural networks." arXiv preprint arXiv:1601.06071 (2016).

[24]Song, Linghao, Xuehai Qian, Hai Li, and Yiran Chen. "Pipelayer: A pipelined reram-based accelerator for deep learning." In 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 541-552. IEEE, 2017.

[25]Wold, Svante, Kim Esbensen, and Paul Geladi. "Principal component analysis." Chemometrics and intelligent laboratory systems 2, no. 1-3 (1987): 37-52.

[26]Yuan, Ming, and Yi Lin. "Model selection and estimation in regression with grouped variables." Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68, no. 1 (2006): 49-67.

[27]Wen, Wei, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. "Learning structured sparsity in deep neural networks." In Advances in neural information processing systems, pp. 2074-2082. 2016.

[28]Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249-256. 2010.

[29]Golub, Gene H., Michael Heath, and Grace Wahba. "Generalized cross-validation as a method for choosing a good ridge parameter." Technometrics 21, no. 2 (1979): 215-223.

[30]Tang, Tianqi, Lixue Xia, Boxun Li, Rong Luo, Yiran Chen, Yu Wang, and Huazhong Yang. "Spiking neural network with rram: Can we use it for real-world application?." In Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, pp. 860-865. EDA Consortium, 2015.

[31]Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[32]Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

[33]He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[34]Courbariaux, Matthieu, Yoshua Bengio, and Jean-Pierre David. "Binaryconnect: Training deep neural networks with binary weights during propagations." In Advances in neural information processing systems, pp. 3123-3131. 2015.

[35]Hubara, Itay, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. "Binarized neural networks." In Advances in neural information processing systems, pp. 4107-4115. 2016.

[36]Rastegari, Mohammad, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. "Xnor-net: Imagenet classification using binary convolutional neural networks." In European Conference on Computer Vision, pp. 525-542. Springer, Cham, 2016.

[37]Zhu, Chenzhuo, Song Han, Huizi Mao, and William J. Dally. "Trained ternary quantization." arXiv preprint arXiv:1612.01064 (2016).

[38]Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).

[39]Lin, Darryl, Sachin Talathi, and Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks." In International Conference on Machine Learning, pp. 2849-2858. 2016.

[40]Mellempudi, Naveen, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. "Ternary neural networks with fine-grained quantization." arXiv preprint arXiv:1705.01462 (2017).

[41]Zhou, Aojun, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. "Incremental network quantization: Towards lossless cnns with low-precision weights." arXiv preprint arXiv:1702.03044 (2017).

[42]Cai, Zhaowei, Xiaodong He, Jian Sun, and Nuno Vasconcelos. "Deep learning with low precision by half-wave gaussian quantization." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5918-5926. 2017.

[43]Vanhoucke, Vincent, Andrew Senior, and Mark Z. Mao. "Improving the speed of neural networks on CPUs." (2011).

[44]Louizos, Christos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. "Relaxed quantization for discretized neural networks." arXiv preprint arXiv:1810.01875 (2018).

[45]Polino, Antonio, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization." arXiv preprint arXiv:1802.05668 (2018).

[46]Dettmers, Tim. "8-bit approximations for parallelism in deep learning." arXiv preprint arXiv:1511.04561 (2015).

[47]Jacob, Benoit, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2704-2713. 2018.

[48]Blackwell, Trevor Leslie. Applications of randomness in system performance measurement. Harvard University, 1998.

[49]Lewis, Peter A. W., Allan S. Goodman, and James M. Miller. "A pseudo-random number generator for the System/360." IBM Systems Journal 8, no. 2 (1969): 136-146.

[50]Fujita, S., Ken Uchida, S. Yasuda, R. Ohba, H. Nozaki, and T. Tanamoto. "Si nanodevices for random number generating circuits for cryptographic security." In 2004 IEEE International Solid-State Circuits Conference (IEEE Cat. No. 04CH37519), pp. 294-295. IEEE, 2004.

[51]Huang, Chien-Yuan, Wen Chao Shen, Yuan-Heng Tseng, Ya-Chin King, and Chrong-Jung Lin. "A contact-resistive random-access-memory-based true random number generator." IEEE Electron Device Letters 33, no. 8 (2012): 1108-1110.

[52]Vivoli, V. Caprara, P. Sekatski, J. D. Bancal, C. C. W. Lim, A. Martin, R. T. Thew, H. Zbinden, N. Gisin, and N. Sangouard. "Device-independent quantum random number generator with a photon pair source." arXiv preprint (2014).

[53]Fukushima, Akio, Takayuki Seki, Kay Yakushiji, Hitoshi Kubota, Hiroshi Imamura, Shinji Yuasa, and Koji Ando. "Spin dice: A scalable truly random number generator based on spintronics." Applied Physics Express 7, no. 8 (2014): 083001.

[54]Yang, Yuchao, and Wei Lu. "Nanoscale resistive switching devices: mechanisms and modeling." Nanoscale 5, no. 21 (2013): 10076-10092.

[55]Gaba, Siddharth, Phil Knag, Zhengya Zhang, and Wei Lu. "Memristive devices for stochastic computing." In 2014 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 2592-2595. IEEE, 2014.

[56]Kim, Sungho, ShinHyun Choi, and Wei Lu. "Comprehensive physical model of dynamic resistive switching in an oxide memristor." ACS nano 8, no. 3 (2014): 2369-2376.

[57]Chua, Leon. "Resistance switching memories are memristors." Applied Physics A 102, no. 4 (2011): 765-783.

[58]Hu, Miao, Hai Li, Yiran Chen, Qing Wu, Garrett S. Rose, and Richard W. Linderman. "Memristor crossbar-based neuromorphic computing system: A case study." IEEE transactions on neural networks and learning systems 25, no. 10 (2014): 1864-1878.

[59]Vincent, Adrien F., Jerome Larroque, W. S. Zhao, N. Ben Romdhane, Olivier Bichler, Christian Gamrat, J-O. Klein, Sylvie Galdin-Retailleau, and Damien Querlioz. "Spin-

transfer torque magnetic memory as a stochastic memristive synapse." In 2014 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1074-1077. IEEE, 2014.

[60]Gaba, Siddharth, Patrick Sheridan, Jiantao Zhou, Shinhyun Choi, and Wei Lu. "Stochastic memristive devices for computing and neuromorphic applications." Nanoscale 5, no. 13 (2013): 5872-5878.

[61]Lee, Myoung-Jae, Chang Bum Lee, Dongsoo Lee, Seung Ryul Lee, Man Chang, Ji Hyun Hur, Young-Bae Kim et al. "A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta 2 O 5− x/TaO 2− x bilayer structures." Nature materials 10, no. 8 (2011): 625.

[62]Torrezan, Antonio C., John Paul Strachan, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. "Sub-nanosecond switching of a tantalum oxide memristor." Nanotechnology 22, no. 48 (2011): 485203.

[63]Chua, Leon. "Memristor-the missing circuit element." IEEE Transactions on circuit theory 18, no. 5 (1971): 507-519.

[64]Stanley Williams, R. "How we found the missing memristor." In Chaos, CNN, Memristors and Beyond: A Festschrift for Leon Chua With DVD-ROM, composed by Eleonora Bilotta, pp. 483-489. 2013.

[65]Hu, Miao, Yu Wang, Qinru Qiu, Yiran Chen, and Hai Li. "The stochastic modeling of TiO 2 memristor and its usage in neuromorphic system design." In 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 831-836. IEEE, 2014.

[66]Yu, Shimeng, Bin Gao, Zheng Fang, Hongyu Yu, Jinfeng Kang, and H-S. Philip Wong. "Stochastic learning in oxide binary synaptic device for neuromorphic computing." Frontiers in neuroscience 7 (2013): 186.

[67]Medeiros-Ribeiro, Gilberto, Frederick Perner, Richard Carter, Hisham Abdalla, Matthew D. Pickett, and R. Stanley Williams. "Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution." Nanotechnology 22, no. 9 (2011): 095702.

[68]Yi, Wei, Frederick Perner, Muhammad Shakeel Qureshi, Hisham Abdalla, Matthew D. Pickett, J. Joshua Yang, Min-Xian Max Zhang, Gilberto Medeiros-Ribeiro, and R. Stanley Williams. "Feedback write scheme for memristive switching devices." Applied Physics A 102, no. 4 (2011): 973-982.