HARDWARE-RESTRICTION-AWARE TRAINING (HRAT) FOR MEMRISTOR NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Memristor neural network (MNN), which utilizes crossbar arrays for vector matrix multiplication, has huge advantages in terms of scalability and energy efficiency for neuromorphic edge computing. Weights of MNN are usually trained offline and then deployed as memristor conductance through a sequence of programming voltage pulses. Although weight noise caused by memristor process variation has been addressed in prior training algorithms, efficient design and training of MNNs has not been systematically explored to date. In this work, we propose Hardware-Restriction-Aware Training (HRAT), which takes into account various non-negligible limitations and non-idealities of memristor circuits and systems. Hardware restrictions include weight mismatch due to process variation, weight quantization noise, non-destructive threshold voltage of memristors, saturation voltage of operational amplifiers, and finite resolution of digital-to-analog converters (DACs). HRAT simulates MNN behavior and restrictions during offline training, thereby bridging the gap between offline training and in-situ deployment. We present a new BN fusing strategy as a part of HAT, the strategy aligns the scale of noise between training and non-BN inference. After fusing BN, we not only obtain an improvement in model performance but also save the hardware resource of BN during deployment. Due to the limited operating voltage of the memristor, most normal scale signals will be limited in amplitude. To reduce the distortion of the input signal, a scale factor is learned to decrease the signal intensity. The learned scale can be merged into the parameters of linear operation along with fusing BN, so no extra signal scale circuits are needed. The software and hardware simulations show that the models trained by the proposed HAT can be directly deployed on our memristive neural network processor with negligible performance degradation and without any device calibration and on-chip training. This method will promote the development of commercial memristive computing.

1 INTRODUCTION

memristor chip demonstration: Naqi et al. (2022), Li & Ang (2021), Yao et al. (2020),Kim et al. (2021) In this section, we discuss why it is necessary to conduct off-chip training.

The file `iclr2023_conference.pdf` contains these instructions and illustrates the various formatting requirements your ICLR paper must satisfy. Submissions must be made using LaTeX and the style files `iclr2023_conference.sty` and `iclr2023_conference.bst` (to be used with LaTeX2e). The file `iclr2023_conference.tex` may be used as a "shell" for writing your paper. All you have to do is replace the author, title, abstract, and text of the paper with your own.

The main contributions of this work are:

First, we model the random programming noise of memristor crossbars and integrate it into the training process, realizing noise-aware training.

Second, our method does not depend on any assumption of statistical knowledge about memristor devices. Our method enables large massive production of deep memristor neural networks.

Third, we conduct experiments verifying that in several network architectures, including VGG-16-BN on CIFAR-10 and CIFAR-100 datasets.

2 RELATED WORK

Variation-aware memristor NN training. Prior work on memristor NN training has predominately focused on memristor variation-aware training. Liu et al. (2015) address memristor conductance variation by introducing an additional term called "penalty of variation" into the training constraints. Then, an upper bound for the variation penalty is estimated and used during training. This method improves the tolerance of trained weights to memristor variation by applying tighter training constraints. Zhu et al. (2020) propose statistical training to deal with weight programming noise resultant from memristor variation. Random variables are used to represent memristor variation, and network weights are modeled as linear functions of these random variables. Then, subsequent network computations are modified to propagate the effects of weight programming noise. The effectiveness of this method is validated only on small NNs. Gao et al. (2021) propose a variation-aware and defect-aware training framework, where the developed analytical weight models are used as the priori and constraint for memristor NN training. Yang et al. (2021) propose a hardware-aware training method to deal with effects of stochastic noise, which includes thermal, shot, random telegraph and programming noise. These stochastic noises are injected during training process to improve the inference accuracy of memristor NNs. Later, based on a measured profile of memristor conductance variation, Mao et al. (2022) propose defect-aware training to include the effects of memristor conductance variation, relaxation, and failure. Büchel et al. (2022) address the issue of memristor parameter variation via adversarial regularization. Their algorithm attacks the weight space during training to enhance the robustness of memristor devices to programming noise.

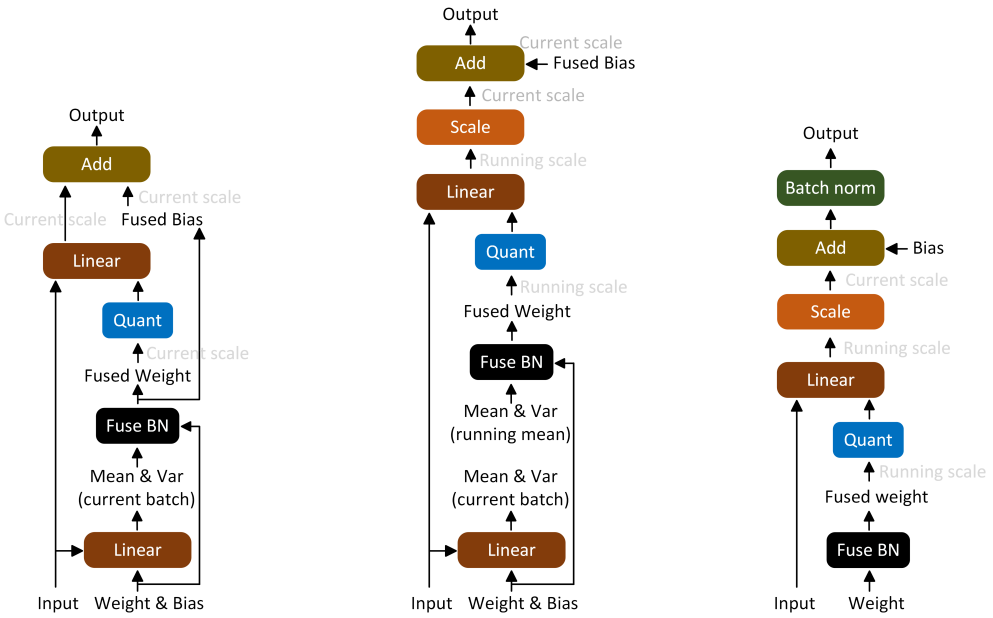
From the above-mentioned discussion, it is clear that existing memristor NN training algorithms do not fully address the requirement of highly effective training. It is attractive to develop new to overcome the limitations of. **Consideration of hardware constraint during training:** Training considerations regarding hardware limitations are primarily concerned with parameter mismatch between training and deployment. Researchers focus their attention on the robustness of neural networks in face of weight perturbation. Other constraints of the memristor neural network system have not been investigated.

Strategy of fusing Batch Normalization: BN can accelerate the convergence of training and improve performance of trained neural networks. The scale and shift operations of BN can be merged into the weight and bias term of previous linear operation after training Ioffe & Szegedy (2015). As a result, the overhead of BN can be saved during deployment. Training model with BN and fusing BN (also known as folding BN) during deploy dose not draw attention in the field of neuromorphic computing yet. In the field of edge computing, there are some constraints produced by hardware limitations, *e.g.*quantization of low-precision representation of parameters and memristor variations caused by manufacturing process non-idealities. So, the hardware constraints should be aligned before and after BN fusing. Two issues make it hard to align the constraints: (1) If we fuse BN before training, statistics of BN are not updated which will lead a non-BN model and a bad training performance. And (2) If we add the constraints on the linear operation, the added constraints will be changed after fusing BN. There are several existing BN fusing strategies, specifically the fixed-point represented neural networks, to deal with the aforementioned problem. Shown as Fig. 2,

Strategy 1: Introduced by Jacob et al. (2018), the linear operation will be executed twice to train a linear+BN combination. The first time calculation of linear operation uses original parameters to compute the statistics of BN. Then the mean and variance of the current batch are used to merge into the original parameters of linear operation, and the fused weight and bias are obtained and also their quantized counterpart. After that, the quantized fused parameters are used for the second time in linear operation, and the output of linear+BN is obtained.

Strategy 2: Introduced in Krishnamoorthi (2018), the linear operation will be executed twice. The first time calculation of linear operation uses original parameters to compute the statistics of BN. Then the moving average mean and variance are used to obtain the fused weight, and the quantized fused weight is used to compute the second time of linear operation. Since the fused weight is obtained from moving average mean and variance, a scale correction is needed to correct the scale of output to the scale of the current batch.

Strategy 3: The above strategies almost double the calculation of training. PyTorch introduced a one-time version in their quantization codes [1]. This method firstly uses running statistics to obtain



(a) Jacob et al. (2018) (b) Krishnamoorthi (2018) (c) PyTorch

Figure 1: Strategies of fusing BN for quantization-aware-training.

the quantized fused weight, and the output of the linear operation. Then the output is corrected to the original scale. At last, a normal BN is executed.

All the above strategies have updated the BN statistics. The scale of quantization noise has aligned between training and inference. However, the constraints of our hardware system of memristor neural networks are significantly different from the fixed-point represented neural networks. In addition to quantization, our memristor neural network system has constraints such as parameter mismatch, voltage limitation, input signal limitation, and DAC resolution. A new BN fusing strategy need to be developed for our memristor neural networks.

Hardware implementation of neuromorphic computing

3 Memristor NN Architecture and Restrictions

3.1 Memristor NN Hardware Overview

3.2 Hardware Restrictions of Memristor NN Systems

Weight Programming Error: ds
Memristor non-destructive threshold voltage:

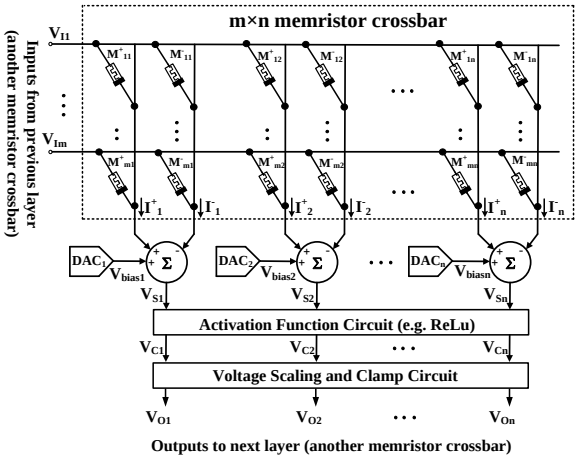
Fuse BN in memristor:

Different memristors have different non-destructive threshold, such as Yan et al. (2018)

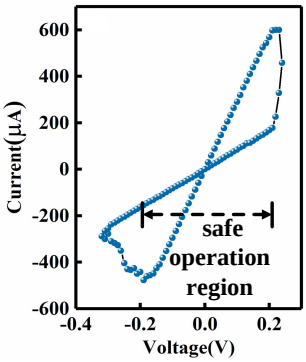
Saturation Voltage of Operational Amplifiers: In this section, we discuss power supply limits and memristor threshold limits.

Bias Error Due to Finite-Resolution Digital-to-Analog Converters (DACs):

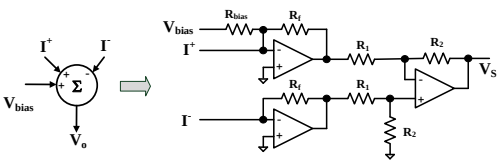
[1] https://github.com/pytorch/pytorch/blob/master/torch/nn/intrinsic/qat/modules/conv_fused.py

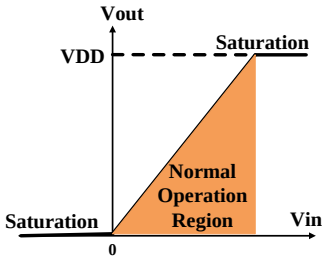


(a) Schematic of a one-layer memristor neural network for forward pass. This crossbar implements a layer of neurons. The input voltages of each crossbar (e.g., V_{I1}…V_{In}, V_{O1}…V_{On}) must not exceed its non-destructive threshold voltage as shown in (b). All analog circuits, such as amplifiers and digital-to-analog converters (DACs) operate from a single supply voltage and have a limited normal operation region as shown in (c).



(b) Measured memristor current-voltage sweep curve from Yan et al., (2019), whose non-destructive threshold voltage is 0.2V.





(c) The output voltages of amplifier circuits and DACs are limited between 0V (ground) and VDD (the supply voltage)

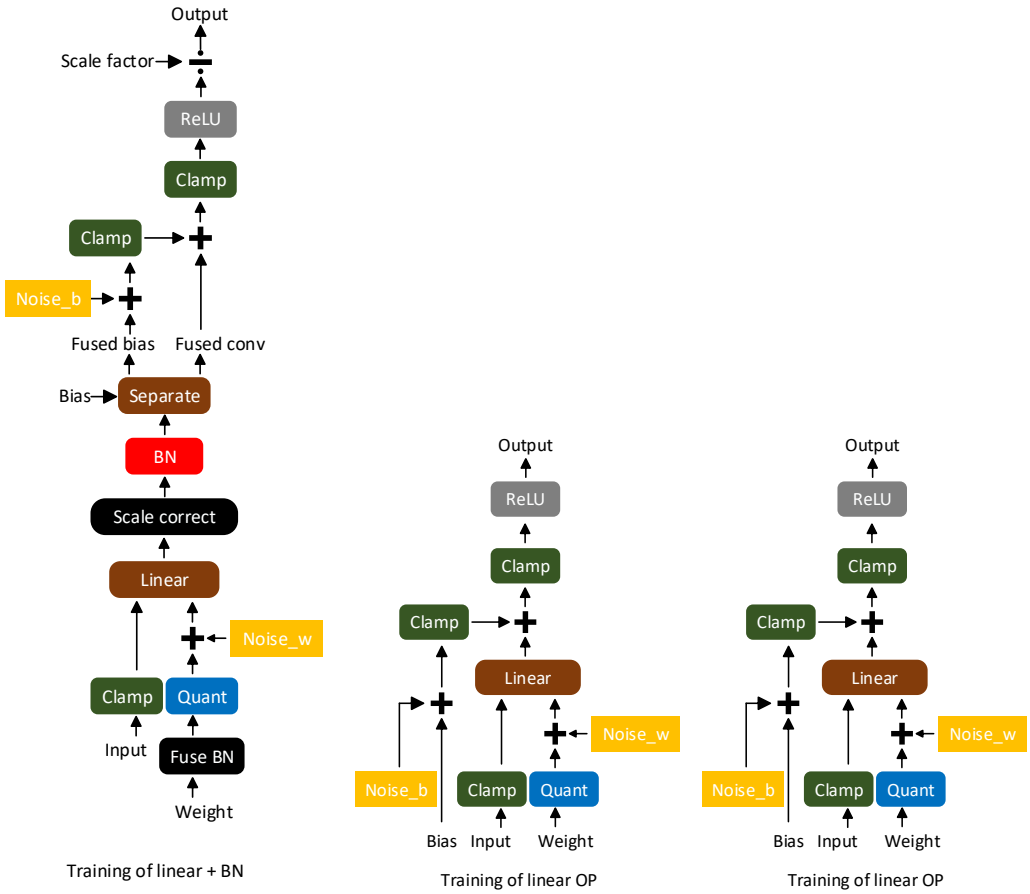
Figure 2: Hardware overview of memristor neural networks: (a) Schematic of one-layer memristor neural network, consisting of a memristor crossbar, DACs design for fully connected layer. Yan et al. (2019)

DACs are analog circuits widely used in bias generation applications, due to their high precision, low drift, and great flexibility. The output resolution of a DAC is usually specified in bits, representing the smallest output increment that can be produced. For example, the resolution of an n-bit DAC is $VDD/2^n$ when it has an output range of 0V to VDD. When providing bias voltages to neurons in memristor NN, quantization error is introduced by a DAC as a result of its finite resolution. This quantization error will influence and degrade the inference accuracy of memristor NN.

4 Noise&Quantization&Hardware Limitation-Aware Training for Memristor NN

Memristor programming Noise-Aware Training: Murray & Edwards (1994) add Gaussian noise to parameter during forward pass to increase the model robustness to parameter noise. For simplicity, we follow the strategy for dealing with noise. Although there are some possibilities that the aforementioned approaches would perform better on the problem of parameter mismatch, we found that simple injection of noise for dealing with parameter mismatch dose not become the bottleneck of the system. Inspired by Murray & Edwards (1994), we add a truncated Gaussian noise to the weights for robustness. The gradients of noisy weights are used to update the original weights, *i.e.*the STE trick is used. Then the obtained neural networks are insensitive to the deployment noise of weights.

BN-version model training must follow the principle of aligning the scale of deployment noise before and after training.



(a) Training of linear+BN (b) Testing (c) Training of linear layer

Figure 3: Off-chip training of memristor neural networks.

4.1 WEIGHT QUANTIZATION

To simplify conductance programming of memristor crossbars, weights are quantized into a finite number of status. Low-precision weights also make model insensitive to deployment noise. In this work, we uniformly quantizate weights using the following quantization function:

$$\mathcal{Q}(\boldsymbol{v}) = \mathcal{C}_w(\lfloor \frac{\boldsymbol{v}}{\boldsymbol{s}} \rceil \cdot \boldsymbol{s}) \tag{1}$$

where s denotes the quantization step size, $\lfloor \cdot \rceil$ denotes the rounding operation and $\mathcal{C}_w$ clamps quantized weights to a limited range. Due to the non-differentiation of rounding operation, weights and quantization function cannot be trained through gradient-based optimizers. To address this issue, Eq. (1) is converted into a differentiable form:

$$\mathcal{Q}(\boldsymbol{v}) = \mathcal{C}_w((\frac{\boldsymbol{v}}{\boldsymbol{s}} + \delta_{rounding}) \cdot \boldsymbol{s}) \tag{2}$$

where the rounding noise $\delta_{rounding} = \lfloor \frac{\boldsymbol{v}}{\boldsymbol{s}} \rceil - \frac{\boldsymbol{v}}{\boldsymbol{s}}$. Instead of computing the gradient of $\delta_{rounding}$, we assume that $\delta_{rounding}$ is quantization-independent. Therefore, we can easily train memristor neural networks with gradient-based optimizers.

4.2 NOISE&QUANTIZATION-AWARE TRAINING AND DEPLOYMENT

Output of a neuron is calculated as:

$$\boldsymbol{V}_{o_n} = \mathcal{C}_p(\mathcal{C}_{p \cdot \boldsymbol{\alpha}_n}(\sum_m \mathcal{C}_t(\boldsymbol{V}_n)\mathcal{Q}(\boldsymbol{W}_{:,n})) + \mathcal{C}_p(\boldsymbol{V}_{bias_n} + \delta)) \tag{3}$$

Then the trained $\boldsymbol{W}$ and $(V)_{bias}$ can be used for deployment. The memristor conductance can be calculated from the value of trained weight.

When BN is applied on the network, we will deploy the BN-fused model to reduce the cost of independent BN. To aviod the mismatch of deployment. Let $_{fused} = \frac{\gamma}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}\boldsymbol{W}$ and $\boldsymbol{b}_fused = \boldsymbol{\beta} + \frac{\gamma}{\sqrt{\boldsymbol{\sigma}^2+\epsilon}}(\boldsymbol{b} - \boldsymbol{\mu})$.

5 Baseline Network Architectures and Experimental Setup

We choose a four-layer fully connected NN (FC-4), LeNet-5 LeCun et al. (1998), and VGG-16 Simonyan & Zisserman (2015), as our baseline architectures. We design the miniature model FC-4 for rapid verification of algorithms. FC-4 consists of only four fully connected layers, namely three hidden layers and one classification layer. The three hidden layers have 512, 128 and 32 features respectively. Input images are flattened into 1-dimensional vectors before being fed into the FC-4 model. The convolutional NN models LeNet-5 and VGG-16 are implemented from the original papers, only slightly different to accommodate different datasets. LeNet-5 consists of two convolutional layers and three fully connected layers. VGG-16-BN consists of thirteen convolutional layers and three fully connected layers. FC-4 and LeNet-5 are baselines for experiments on the MNIST dataset, while LeNet-5 and VGG-16 are baselines for experiments on the CIFAR-10 and CIFAR-100 datasets. For experiments on the CIFAR datasets, the number of features in the hidden layers of LeNet-5 is increased by a factor of 5. For fast convergence and better performance, we also implement batch normalization (BN) layers in each baseline model. All features are normalized via BN when training these BN-version models except VGG-16. According to torchvision [2], BN is not applied to the fully connected layers of the VGG-16 model.

Memristor programming noise, bias noise, setting

6 Experimental Results and Comparison

Do not change any aspects of the formatting parameters in the style files. In particular, do not modify the width or length of the rectangle the text should fit into, and do not change font sizes (except perhaps in the References section; see below). Please note that pages should be numbered.

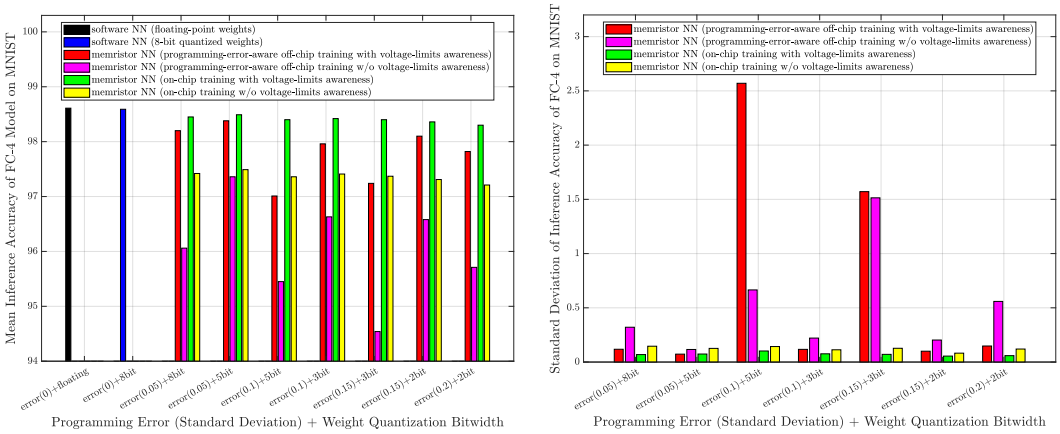


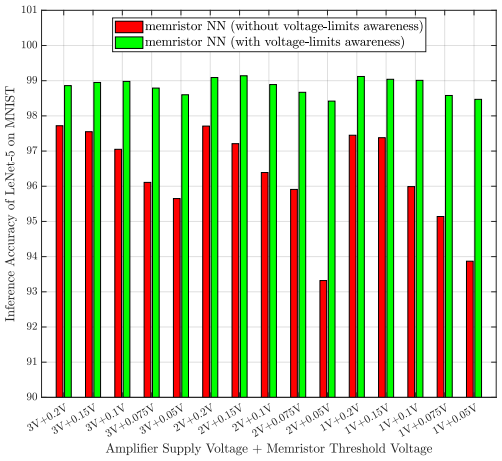
Figure 4: Mean and standard deviation of accuracy for FC-4 on MNIST dataset, respectively.

6.1 FC-4

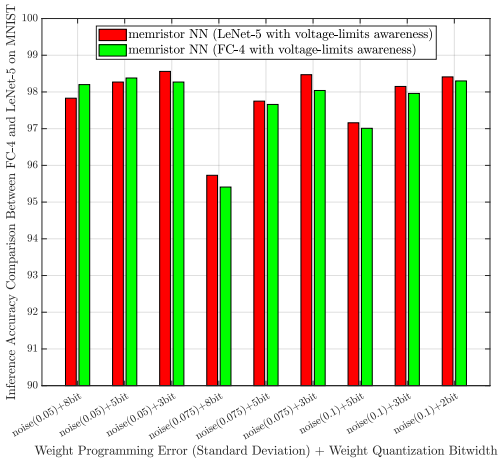
99

[2] https://github.com/pytorch/vision/blob/main/torchvision/models/vgg.py

6.2 LENET-5



(a) LeNet-5



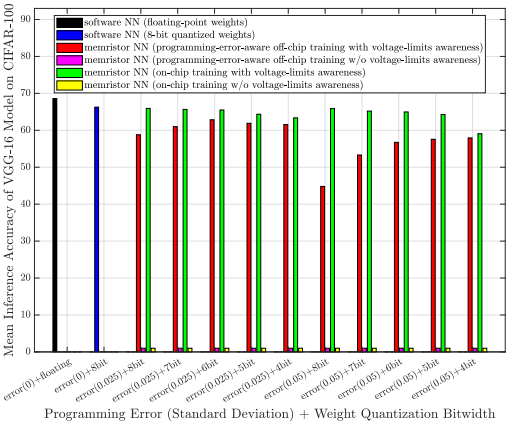
(b) LeNet-5

Figure 5: (a) Inference accuracy of LeNet-5 with or without voltage-limits awareness on MNIST., (b) Inference accuracy for LeNet-5 on MNIST dataset, respectively.

6.3 HARDWARE IMPLEMENTATION OF FC-4 AND LENET-5

6.4 VGG-16

Please prepare PostScript or PDF files



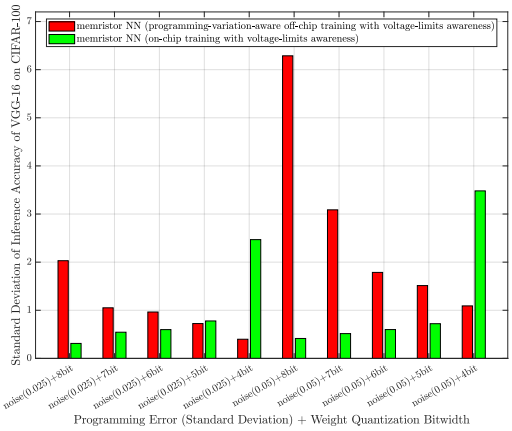


Figure 6: Mean and standard deviation of accuracy for VGG-16 on CIFAR-100 dataset, respectively.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

Julian Büchel, Fynn Faber, and Dylan R Muir. Network insensitivity to parameter noise via adversarial regularization. In *ICLR*, 2022.

Table 1: Best Performance Comparison among various standard deviations of weight programming errors. Network model is VGG-16 and training is voltage-limits awareness. Ideal DAC is used.

	CIFAR-10			CIFAR-100		
Weight Error (STD)	**On-Chip**	**This Work**	**Degradation**	**On-Chip**	**This Work**	**Degradation**
0.0	91.48	91.48	0	66.27	66.27	0
0.025	91.30	90.33	0.97	65.93	62.85	3.08
0.05	90.72	87.24	3.48	65.88	57.94	7.94
0.075	90.70	84.70	6.00	66.10	55.64	10.46

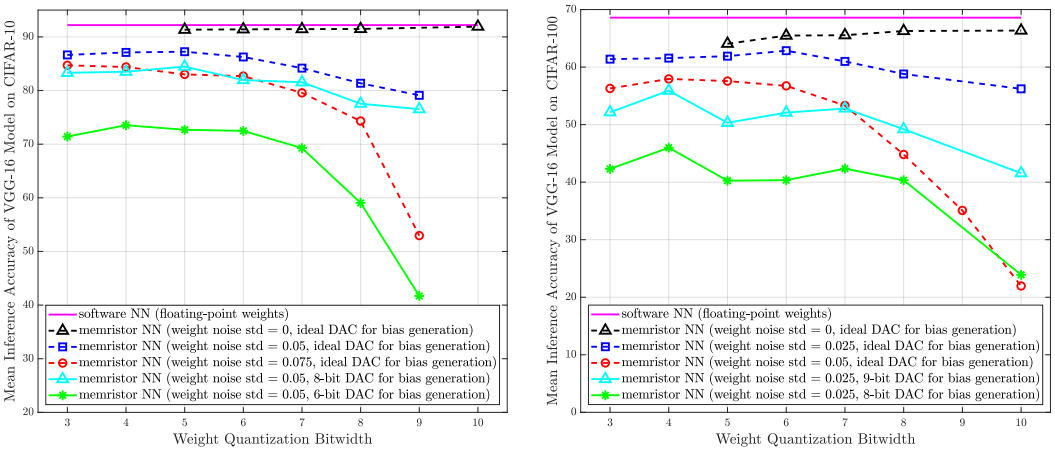
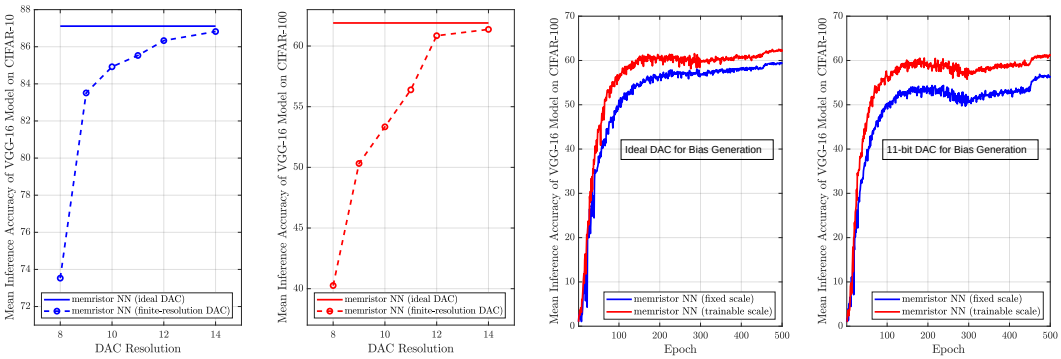


Figure 7: Mean and standard deviation of accuracy for VGG-16 on CIFAR-100 dataset, respectively.



(a) DAC resolution (b) scale

Figure 8: Mean and standard deviation of accuracy for VGG-16 on CIFAR-100 dataset, respectively.

Di Gao, Grace Li Zhang, Xunzhao Yin, Bing Li, Ulf Schlichtmann, and Cheng Zhuo. Reliable memristor-based neuromorphic design using variation-and defect-aware training. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Benoit Jacob, 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.

Hyungjin Kim, MR Mahmoodi, H Nili, and Dmitri B Strukov. 4k-memristor analog-grade passive crossbar circuit. *Nature communications*, 12(1):1–11, 2021.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yesheng Li and Kah-Wee Ang. Hardware implementation of neuromorphic computing using large-scale memristor crossbar arrays. *Advanced Intelligent Systems*, 3(1):2000137, 2021.

Beiye Liu, Hai Li, Yiran Chen, Xin Li, Qing Wu, and Tingwen Huang. Vortex: Variation-aware training for memristor x-bar. In *Proceedings of the 52nd Annual Design Automation Conference*, pp. 1–6, 2015.

Ruibin Mao, Bo Wen, Mingrui Jiang, Jiezhi Chen, and Can Li. Experimentally-validated crossbar model for defect-aware training of neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(5):2468–2472, 2022.

Alan F Murray and Peter J Edwards. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on neural networks*, 5(5):792–802, 1994.

Muhammad Naqi, Min Seok Kang, Taehwan Kim, Seungho Baek, Arindam Bala, Changgyun Moon, Jongsun Park, Sunkook Kim, et al. Multilevel artificial electronic synaptic device of direct grown robust mos2 based memristor array for in-memory deep neural network. *npj 2D Materials and Applications*, 6(1):1–9, 2022.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Xiaobing Yan, Lei Zhang, Huawei Chen, Xiaoyan Li, Jingjuan Wang, Qi Liu, Chao Lu, Jingsheng Chen, Huaqiang Wu, and Peng Zhou. Graphene oxide quantum dots memristors with progressive conduction tuning for artificial synaptic learning. *Advanced Functional Materials*, 114(28): 1803728, 2018.

Xiaobing Yan, Hui Li, Lei Zhang, Chao Lu, Jianhui Zhao, Zhenyu Zhou, Hong Wang, Jingjuan Wang, Xiaoyan Li, Yifei Pei, et al. Density effects of graphene oxide quantum dots on characteristics of zr0. 5hf0. 5o2 film memristors. *Applied Physics Letters*, 114(16):162906, 2019.

Xiaoxuan Yang, Syrine Belakaria, Biresh Kumar Joardar, Huanrui Yang, Janardhan Rao Doppa, Partha Pratim Pande, Krishnendu Chakrabarty, and Hai Helen Li. Multi-objective optimization of reram crossbars for robust dnn inferencing under stochastic noise. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2021.

Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577 (7792):641–646, 2020.

Ying Zhu, Grace Li Zhang, Tianchen Wang, Bing Li, Yiyu Shi, Tsung-Yi Ho, and Ulf Schlichtmann. Statistical training for neuromorphic computing using memristor-based crossbars considering process variations and noise. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1590–1593. IEEE, 2020.