Quantization-Aware In-situ Training for Reliable and Accurate Edge AI

João Paulo C. de Lima and Luigi Carro
Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
{jpclima,carro}@inf.ufrgs.br

***Abstract*—In-memory analog computation based on memristor crossbars has become the most promising approach for DNN inference. Because compute and memory requirements are larger during training, memristive crossbars are also an alternative to train DNN models within a feasible energy budget for edge devices, especially in the light of trends towards security, privacy, latency, and energy reduction, by avoiding data transfer over the Internet. To enable online training and inference on the same device, however, there are still challenges related to different minimum bitwidth needed in each phase, and memristor non-idealities to be addressed. We provide an in-situ training framework that allows the network to adapt to hardware imperfections, while practically eliminating errors from weight quantization. We validate our methodology with image classifiers, namely MNIST and CIFAR10, by training NN models with 8-bit weights and quantizing to 2 bits. The training algorithm recovers up to 12% of the accuracy lost to quantization errors even under high variability, reduces training energy by up to 6×, and allows for energy-efficient inferences using a single cell per synapse, hence enhancing robustness and accuracy for a smooth training-to-inference transition.**

***Index Terms*—synaptic devices, deep neural networks, in-situ training, inference, process variation.**

I. Introduction

Neural Network (NN) models have become larger, deeper, and more diverse to satisfy a wide range of applications. With the increase in size and demand, the number of arithmetic operations and memory references posed a concern for designing sustainable AI systems using conventional Von-Neumann architectures. In the particular case of NN processing, the crossbar arrays of memristors, whose conductance states can be used to store NN weights, allowed the design of novel accelerators that can perform matrix-vector multiplication in the analog domain in $O(1)$ time complexity [1]. Especially for edge devices (e.g., IoT and mobile devices), which suffer more from limited memory, computing performance, and hard constraints in energy consumption, Memristor Crossbar Arrayss (MCAs) are seen as the most promising approach for achieving enormous acceleration and energy efficiency in the execution of machine learning applications [2].

Although a lot has already been investigated to make MCAs viable for inference accelerators, there are still challenges like process variation, the precision of peripheral interface, and the intrinsic endurance. The situation is even worse when one tries to use the same device for both in-situ training and inference, given that training requires more bits and precision than inference. From the performance and energy efficiency perspective, in-situ MCA-based training is estimated to outperform GPU and its digital counterpart training accelerators [1]. Nonetheless, in-situ training can also mitigate some MCA imperfections, such as stochastic variability, IR drop, and state-stuck issues, better than ex-situ approaches because it can thoroughly map and adjust weights by taking into consideration the actual values of device- and circuit-level non-idealities [3].

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

However, it is widely known that the precision of inputs, weights, and activation required during the training phase is much higher than the precision needed for accurate inference [4]. Yet, algorithmic optimizations for efficient inference, such as weight pruning, quantization, and sharing, are carried out solely in software before mapping NN models onto crossbar arrays [5], [6] or even ignored [1], [7]. Also, current in-situ training strategies still fall short of achieving a training-to-inference transition suitable for such optimizations. The importance of having training and inference phases in the same device lies in how current models and methods either rapidly drain the battery of edge devices to train NN in the field or increase the Total Cost of Ownership in data centers and the costs of over-the-air updates to meet the increasing and huge demand for deep learning tasks.

Up to now, the effects of quantization, pruning, hardware- and variation-aware training for MCAs were only investigated under the traditional offline approach, [5], [8], [9]. This study aims to fill a gap between training and inference in the same device by presenting online training and quantization methods that assimilate device- and circuit-level non-idealities within tolerable accuracy loss. Our specific contributions are:

- We present a fine-tuning method that allows low-precision training to have larger gradients, lower quantization underflow, and smoother conductance variation, thus speeding up the training process in Section III-A.
- We introduce a Quantization-aware Training (QAT) strategy to avoid local minima in low-bitwidth models and further aggressively quantize models with negligible accuracy loss in Section III-C.
- We show that QAT allows more reliable and accurate training under quantization errors and device non-idealities, recovering up to 12% of the accuracy in Section IV-C.
- Finally, we show how our work reduces training energy by up to 6× and maintains optimized inference numbers compared to state-of-the-art methods in Section IV-D.

978-3-9819263-6-1/DATE22/©2022 EDAA

II. Background and Motivation

Until recently, Deep Neural Network (DNN) training was considered an energy-intensive task only performed by CPUs and GPUs with high-precision arithmetic circuits and abundant memory. As a consequence, most of the existing techniques today only focus on the deployment of a well-trained compressed network in dedicated hardware, thus typically requiring training and inference to be processed in separate devices [4]. This observation is also valid for MCA-based NNs, where training is usually offloaded onto powerful GPUs that run frameworks adapted to take into consideration the MCA architecture and its hardware imperfections [5], [6], [8], [10]. However, by using the same device for inference and training, one can reduce (or even eliminate) communication cost and, hence, obtain a smaller training time and energy [2].

To support on-chip training, several works have presented a comprehensive hardware implementation of the backpropagation algorithm or Equations 1–4, which are, respectively, feed-forward, error-calculation, weight-gradient-calculation, and weight-update step [1]. Fig. 1 shows a possible implementation of matrix operations present in the back-propagation on a MCA.

$$I_j = \sum_{i}^{n} w_{ij} \cdot V_i, \quad V_i^{l+1} = \sigma(I_i^l) \tag{1}$$

$$\xi = Loss(V^L, y) \tag{2}$$

$$\delta_j^l = \begin{cases} \frac{\partial \xi}{\partial I_j^l}, & \text{if } l = L; \\ \frac{\partial \sigma}{\partial I_j^l} \sum_i w_{ij}^l \ \delta_i^{l+1}, & l \leq L \end{cases} \tag{3}$$

$$\Delta W^l = \eta \sum_{n=1}^{B} v^l(n) \ \delta^l(n)^\top \tag{4}$$

In the feed-forward step (blue arrows in Fig. 1), a vector-matrix multiplication is performed by MCAs using the first part of Eq.1, where I is the readout current from the bottom electrodes, w is the weight matrix of layer l and v is the input voltage vector applied to the top electrodes of the MCA. In the second part of Eq. 1, V^{l+1} is the input voltage vector to the next layer obtained by applying the activation function σ to the readout current vector. This step is performed layer by layer sequentially, i.e., beginning with the input voltage vector to the first layer coming from the dataset and finishing with an output current that can be interpreted as the classification result. Eq. 2 computes the output error between the predicted (V^L) and actual (y) values of the output layer, which is not necessarily performed by analog circuits.

In the backward pass step (brown arrows in Fig. 1 and Eq. 3), a transpose vector-matrix multiplication is performed by MCAs by applying the error vector from the previous layer (δ^{l+1}) as voltage on the columns and reading the current from the rows of the crossbar (δ^l) to compose the error vector to the current layer. Hence, this step propagates the gradient of the loss function with respect to every training weight to compute the errors at each layer. Finally, the weight update (ΔW) for each layer is calculated using Eq. 4, where η is the learning rate, v is the input voltage vector, δ the output error vector

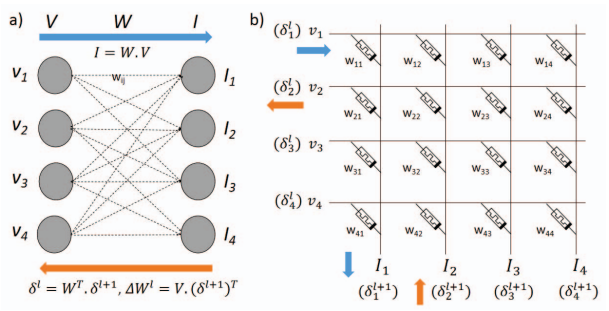


Fig. 1: a) Forward and backward pass in a fully connected layer, b) matrix operations (MVM and $M^\top VM$) on a MCA.

for the l^{th} layer, and B is the batch size, which is, again, not necessarily made by analog circuits.

A. Challenges of backpropagation training in MCAs

Here, we summarize the main challenges to train models in MCAs with good enough accuracy. Solutions for them generally face limitations regarding area and energy and make the efficiency less interesting than expected.

- **HW non-idealities:** memristors suffer from various non-ideal effects, including nonlinear and asymmetric conductance update, conductance range variation, write endurance, Device-to-device (DTD) and Cycle-to-cycle (CTC) variation [3]. They affect the training accuracy by making it difficult to precisely stimulate the controlled change in conductance by simple schemes and by restricting the number of conductance levels per cell.
- **Weight bitwidth**: each memristor is programmed into multiple conductance levels to represent a value. However, due to non-ideal characteristics, the bitwidth of the value a memristor cell can store is rather low (e.g., 2∼3 bits [8], [11]) and each weight is conventionally segmented into multiple subwords. The more accurate a weight is, more cells and crossbars will be needed to deploy a NN, thus impacting area and energy. Also, training needs more bits than inference, which is related to underflow issues throughout the training progress.
- **Gradient underflow**: at the beginning of the training, a model can even learn with low-precision parameters. As training loss and gradients decrease, quantization underflow happens more often in the weight-update step, thus slowing down and stagnating the training progress [12]. Although floating point is less prone to underflow issues, only low-precision integer arithmetic like WAGE [4] suits MCAs due to the aforementioned challenges.

Putting all the challenges together, quantization methods remain indispensable because analog computing on MCAs and their peripheral interfaces will continue to face severe limitations in data precision even with future material improvements.

B. Related Work

Studies on in-situ training have focused on demonstrating the training capabilities without concern for speed optimization

[7], more efficient training accelerators without variation effects [1], [13], evaluating the impact of non-idealities [3], [14], or overcoming non-idealities without inference optimization [15].

1) Quantization in offline training: Also, various solutions have been proposed to integrate hardware-aware quantization methods in offline training flow. Cai et al. [11] introduce splitting and quantizing methods to favor low-bitwidth weight and small ADCs. In [8], a quantization-aware and mapping framework eliminates the need for per-layer full-custom peripheries. In [6], non-uniform quantization of activation data to reduce ADC resolution and a weight regularization to reduce crossbar computing energy are employed. In [10], quantization flow powered by deep learning finds the best parameters for weights, inputs, and ADC output. However, [6], [8], [10], [11] do not consider hardware non-idealities. In [5], the accuracy loss due to conductance variation is eliminated by introducing a trainable quantizer with non-uniform conductance levels. Although the models optimized by [5], [6], [8], [10], [11] are suitable for edge devices, these quantizers would introduce a high overhead when computed in-situ because they depend on high-precision model in the backward pass. Other than that, edge devices cannot afford computing-intensive or time-consuming methods to compute the quantization error because the overhead cost is not compatible with the limited hardware resources.

2) Training to Inference Schemes: Other works implement the backpropagation algorithm directly in MCAs using low-bitwidth data and quantization, which get closer to an optimized setup for inference. In [15], the impact of hardware non-idealities in the WAGE algorithm is evaluated and variability issues are partially overcome by using momentum. Although 4-bit ADCs are enough for multi-bit per cell to maintain inference accuracy [11], the training accuracy in [15] is highly dependent on high-resolution ADCs (i.e., at least 8-bit ADCs) and comes with a loss of 10% for 4-bit weight quantization. Luo and Shimeng [16] introduce hybrid precision synapses combining eNVM and capacitor arrays to improve training efficiency and reliability. However, these improvements come at the cost of a higher area, retention issues that get worse in larger NN models, and the dependence of high-precision ADCs.

III. Techniques for optimized training and inference

In this section, we describe techniques for tackling the challenges addressed in Section II-A by building upon the on-chip training architecture modeled by [17]. At the top level of the chip, the architecture consists of multiple tiles, global buffer, accumulation, activation, pooling, and weight gradient computation unit. The tiles, in turn, are composed of input/output buffers, accumulators, and processing elements, each one containing synaptic arrays and their peripheral circuits, such as ADCs, decoder, adders, and registers. Although this architecture can operate with various device technologies, we target MCA and each array can be used in both the feed-forward and error calculation steps. In the forward pass, the input vector voltages are provided to the BLs, and the weighted sum currents are read out from the SLs in parallel. In the backward pass, the array is used for $M^\top VM$ by switching the input vector and output between BL and SL. Similarly, ADCs can be shared in MVM and $M^\top VM$ operations simply by multiplexing their inputs.

A. Accumulation of underflowed gradients

Mini-batch training with B batches is commonly used, which means that B activations during feed-forward and B computation of errors obtained in backpropagation of the entire network will be stored. Moreover, larger batch sizes not only reduce the batch sampling error in general training, but they also mitigate the variation effects in the weight update of in-situ schemes [3], [15]. Therefore, the number of intermediate data that need to be used and stored is huge if one tries to improve accuracy. Originally, gradient accumulation was proposed to imitate a large batch size in large DNN models by accumulating gradient values and proceeding to the next batch, instead of updating the weights on every batch. In addition to the benefits of large batch size training, we introduce the Underflowed Gradient Accumulation (UGA) to obtain larger gradients and less frequent quantization underflow without the storage overhead present in the former approach.

The calculation of weight gradients of layer l takes the error vector from layer $l+1$ and the activations from layer l to do element-wise multiplication and accumulation across the batch to composed the delta weight. Since this accumulation would require frequent write operations, we do this computation layer-by-layer to alleviate the off-chip memory traffic. After calculating ΔW, we obtain the underflowed part by applying $U = \Delta W - Q_N(\Delta W)$, where Q_N is the quantization function and N the bitwidth of the current layer. The underflowed value is accumulated (U_{Acc}), and stored in the off-chip memory if less than the minimum resolution ϵ of the weight of layer l. If $U_{Acc} \geq \epsilon$, ΔW is updated accordingly and the underflowed part of U_{Acc} is stored. Thus, we must keep in the storage a single floating-point or fixed-point data for each weight, which is far more efficient than storing errors and activations for each image on the batch.

B. Weight splitting across arrays

Instead of distributing each N-bit synaptic weight in N/w w-bit cells located in the same row, we distribute the weight bits across N/w crossbars. Each array column is a partial output that must be properly shifted by its significance, summed in the accumulation unit, and then applied the activation function. While weight splitting across crossbars has previously been used to reach high-precision operations [1], we make a substantially different use by aiming to ease the calculation of quantization metrics and allow for transparent, lightweight transition from training to inference by gradually turning off the less significant crossbars.

C. Quantization methodology

A common approach for QAT is to train in floating-point simulating the quantization effects in the forward pass [18]. Although this approach using floating-point is unfeasible for MCAs, one can still mimic the simulation of lower bitwidth

quantization. Thus, by taking advantage of the weight splitting, we perform additional feed-forward passes ignoring the crossbars of lesser significance to compute quantization metrics, which are then used to decide if the bitwidth of the layer l will be lowered in the next iterations.

The decision of when the quantization should start can be defined by either a threshold for the training loss or by a specific number of epochs. When allowed to start, our algorithm evaluates the quantization metric layer-by-layer in a bottom-up fashion. Layers in the same network may have different bitwidth requirements; thus we must save the bitwidth configuration of each layer. The algorithm can be called on intervals of multiple batches, but this must be ideally long intervals to favor the global training performance. We depict just an overview of simulated quantization and the adjustment of bitwidth in Algorithm 1, where the vector BW is the bitwidth configuration and $Loss_{batch}$ the loss of the last trained batch. Apart from that, we apply a post-training quantization to any layer that could not be quantized to the target bitwidth.

Algorithm 1 Simulated quantization and bitwidth adjustment.

Input: $Loss_{batch}, BW_{0...L}, batch, bits\_per\_cell$
Output: $BW_{0...L}$
Initialisation : $set\_size = round(batch\_size/10)$
S = generate_random_subset($batch, set\_size$)
for $l = 0$ to L **do**
  if $(BW[l] > bits\_per\_cell)$ **then**
    $BW[l] = BW[l] - bits\_per\_cell$
  end if
  $Loss_{avg}$ = feed_forward(S, BW) / size(S)
  if $(Loss_{avg} \geq Loss_{batch})$ **then**
    $BW[l] = BW[l] + bits\_per\_cell$
  end if
end for
return BW

IV. Experimental setup and Evaluation Results

To evaluate the proposed training scheme, we build a simulation framework for on-chip training based on Tensorflow and DNN+NeuroSim [17]. The framework models the crossbar computation, including converters, memristor non-idealities, and weight splitting/mapping, and can train and test NN models considering such properties with flexible network structures and training configurations. Originally, DNN+NeuroSim did not support weights composed of several RRAM cells, so we made such arrangements on it to simulate weight splitting into several crossbars.

A. Benchmarks

We evaluate the present techniques on the MNIST dataset with a small 2-layer model from [7] summing up 4K parameters and also with a second small, popular model of LeNet-300-100 summing up 266K parameters. The small model consists of $8x8$ sized images, 54 hidden and ten output neurons, and the LeNet-300-100 model consists of $28x28$ images. After being trained with FP32 through 40 epochs, they achieved 2.75% and 1.65% error rates, respectively. We also experiment on a VGG-like network from [8] to classify images from CIFAR-10. The model consists of 3x2 2D Convolutional (conv2d) blocks and a Fully Connected (FC) classification layer summing up 307K parameters and achieves 97.35% test accuracy after 40 epochs in traditional FP32 training.

B. Simulation parameters

All experiments regarding training accuracy were performed using $batch\_size = 64$ and momentum as weight update rule ($\gamma = 0.9$ and $\alpha = 0.1$). The layers from all three models are followed by the ReLu activation function, except for the output layer that is followed by a softmax. All models were initialized with a normal distribution within the limits of R_{on} and R_{off}. During the training phase, the non-ideal properties of RRAM devices are introduced in the weight update. The parameters for asymmetric and non-linear conductance tuning are set to 1.94/-0.61, DTD variability to 2%, and CTC, unless declared in the text, is varied.

The hardware performance estimation on NeuroSim considers RRAM cells at 32nm, $R_{on} = 100k\Omega$ and on/off ratio = 10 providing 120 levels with 3.7% CTC variation [19]. We set the sub-array size to be 128×128 and investigated two schemes with 2-bit per cell and 8-bit per cell. For 2-bit, the 8-bit weights are conveniently split into 4 and 2 crossbars so they can run in parallel, thus avoiding extra latency for accumulation along the rows and, more importantly, ease the implementation of the quantization algorithm and the powering off of some crossbars afterward. We employed a linear ADC quantization and parallel processing of dot-products, where multiple rows are activated simultaneously by a switch matrix and the current summation is read out by a 5-bit ADC [17]. The WAGE8 setup consider 8-bit RRAM cells, and WAGE8+UGA and W8QX setups consider 4x2-bit RRAM cells to represent 8-bit weights.

C. Effect of CTC variability on the proposed techniques

In this section, we train the three aforementioned models under four scenarios: first, using WAGE8 scheme [4] adopted in recent in-situ training accelerators [15], [16], then incorporating the UGA scheme on WAGE8, and, finally, the proposed quantization to 4 and 2 bits (i.e., W8Q4 and W8Q2) while still using UGA. In Fig. 2, we present the classification accuracy on the test set after the 40th epoch under several distributions of CTC variation by ranging σ from 0% to 8%, i.e., the relative random variation between the target and the actual conductance values. For W8Q4, we run 35 epochs using 8-bit weights followed by five epochs using 4-bit weights. For W8Q2, we run 30 epochs using 8-bit, five epochs quantized to 4 bits, and, finally, five epochs using 2 bits.

For all models in Fig. 2, WAGE8 fails to reach similar software accuracy. The UGA scheme makes it possible to recover a significant part of underflowed gradients, thus achieving classification errors at $\sigma = 0$ only 2.7% and 2.1% higher than FP32's error for MNIST on the 2-layer and LeNet models. As quantization errors in ADCs and weights are also at stake and contribute to this accuracy loss, 6% of the accuracy loss in the 2-layer model can be attributed solely to gradient underflow.

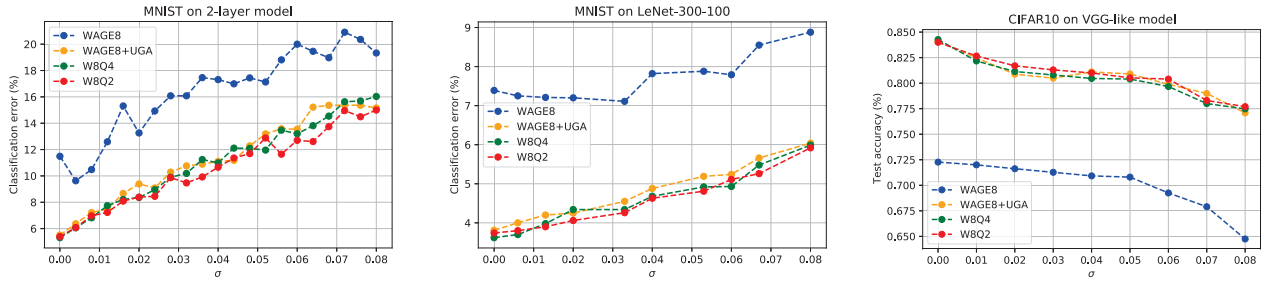


Fig. 2: Effect of CTC variation, where σ is the standard deviation of the probability distribution, on MNIST classification error on 2-layer and LeNet-300-100 models and CIFAR10 accuracy on VGG-like model after the 40th training epoch. WAGE8 represents the baseline execution, W8+UGA represents the contribution of the technique proposed in Section III-A and W8QX the quantization scheme from Section III-C.

For CIFAR10 on the VGG-like model, the accuracy loss of WAGE8 is even higher (25%), and UGA is solely responsible for improving accuracy by 11.8%. Training accuracy of CIFAR10 with WAGE8 scheme in Fig.2 is compatible with the results reported by [8] when asymmetry/non-linearity, ADC quantization error, and DTD variation are ignored.

As σ increases from 0 to 8%, the accuracy of models drops almost proportionally with σ for all scenarios. W8Q4 and W8Q2 setups, which employ weight splitting, achieve accuracy equivalent to WAGE8+UGA for LeNet and VGG models at $\sigma \leq 0.08$ and MNIST on 2-layer model at $\sigma < 0.05$. The higher the CTC variation in the small model, the greater the difference between WAGE8+UGA and W8QX error rates is, reaching up to 2.6% at $\sigma < 0.064$. The accuracy curves for the 2-layer model are more irregular, non-linear than the other plots because weights in small models are more sensitive to inaccurate changes. In other words, larger models can recover more quickly from highly deviated conductance updates.

The degree of CTC variation's impact on the training accuracy depends on the number of conductance levels in the synaptic device. Considering real device variations, only 2-bit RRAMs could be programmed at low variability ($< 2\%$), while synaptic devices with more than 5 bits suffer from $\sigma \geq 0.05$ [17]. As a matter of comparison, [3] found that CTC variation of 5% significantly degrades the CIFAR10 accuracy to be less than 40%. It means that our method can improve the robustness of the trainable parameters under real conductance variation, thus improving in-situ training accuracy in comparison to state-of-the-art methods and practically eliminating errors from aggressive weight quantization.

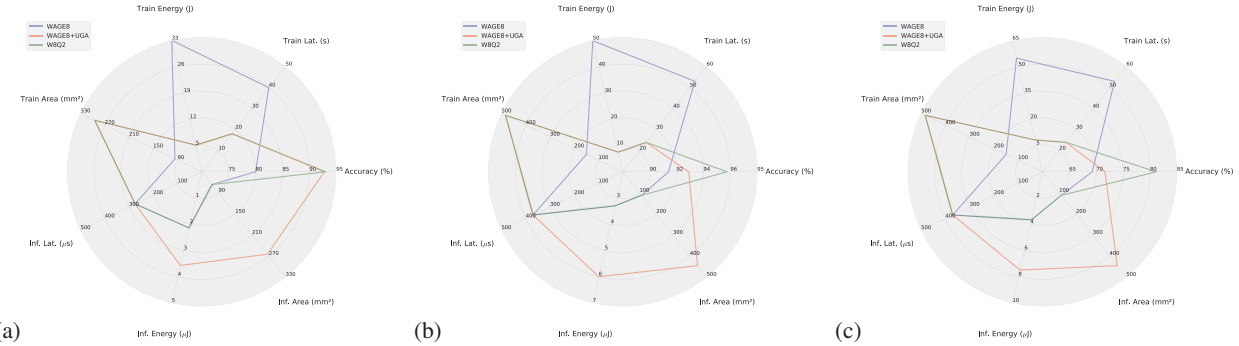
D. Training and Inference Performance

In this section, we compare area, latency, and energy requirements for in-situ training and the resultant inference performance of the trained models with the support from Fig. 3. The figure also presents the test accuracy considering 2-bit devices with 1% CTC variation in WAGE8+UGA and W8Q2 setups, and 8-bit synapses with 7.4% variation in WAGE8 setup. We assume that the 120 conductance levels of RRAM cells reported by [19] can be scaled to provide 2-bit configuration with 1% CTC variability using multiples pulses per step and a narrow conductance range (i.e, $1/6$ of the original on/off ratio), while 8-bit scheme considers a theoretical scenario with a doubled variability. For all models in Fig. 3, the W8Q2 scheme presents the best accuracy numbers. The reported latency and dynamic energy refer to numbers per epoch at the 10th epoch.

In Fig. 3, the WAGE8 scheme suffers from more expensive weight updates. The reason for that lies in the exponential number of conductance levels per cell, since average magnitude of weight updates requires more pulses in 8-bit cells than in 2-bit ones. The WAGE8+UGA, in turn, improves accuracy and makes weight updates $\sim 3.2\times$ more efficient by dividing the operation into four cells, each one with few conductance levels per cell under a much lower variability. Also, by operating in a narrower conductance range with lower conductance values, the energy spent on arrays can potentially be reduced by up to $6\times$. At the 10th epoch, the latency and energy numbers of the W8Q2 setup are precisely the same as WAGE8+UGA and a slight improvement is seen only when quantized to 4 or 2 bits.

Despite the reduced training energy and latency, the WAGE8+UGA increases inference energy by approximately $2\times$ in comparison to WAGE8 due to the additional synaptic arrays. However, the W8Q2 makes it possible to use a single cell as one synapse again, thus reducing energy at the end of the training, as well as inference energy and area. As we consider all peripheral circuits in chip area and performance, the increase in the number of arrays and ADCs is not necessarily proportional to the required area and energy. Also, as the inference phase does not depend on training-specific components, they can be turned off and not count as inference area. Thus, on average, we observe an area overhead of $3.1\times$ due to the quadruplication of synaptic arrays and ADCs during WAGE8+UGA and W8Q2 training. However, the area overhead for WAGE8+UGA inference is only $2.5\times$ larger than using a single synaptic device per weight.

Finally, regarding inference performance, the latency remains the same even for WAGE8+UGA. In this scheme, the activation bits corresponding to different weight significances are com-



(a) (b) (c)

Fig. 3: Accuracy and main aspects of training and inference performance under three scenarios. Subfigure a), b) and c) refer to the 2-layer, LeNet-300-100 and VGG-like models, respectively.

puted in parallel and summed together without penalty in the accelerator's pipeline, hence not impacting execution time. The main takeaway from Fig. 3 is that, by using our methods jointly (i.e., the W8Q2 scheme), DNN accelerators can bring together the best of WAGE8, weight splitting, and UGA and provide energy-efficient on-chip training and inference at the cost of additional crossbars.

V. Conclusion

In this work, we identify important sources of accuracy degradation when training NN models in MCAs and introduce some techniques to tackle the challenges of learning and making inferences with low-bitwidth data while still tolerating non-ideal device properties. By introducing the UGA, we partially recover the accuracy lost by quantization errors without significant storage overhead, thus avoiding stagnation and speeding up the training process. By employing weight splitting across several synaptic arrays, we enable energy-efficient weight updates and better tolerate CTC variation. Finally, we present an aggressive quantization method that eliminates quantization errors compared to 8-bit models without doubling the model size. Therefore, we improve on-chip training accuracy compared to state-of-the-art methods, i.e., WAGE8, and further optimize the trained model for efficient inference.

References

[1] A. Ankit, I. El Hajj, S. R. Chalamalasetti, S. Agarwal, M. Marinella, M. Foltin, J. P. Strachan, D. Milojicic, W.-M. Hwu, and K. Roy, "Panther: A programmable architecture for neural network training harnessing energy-efficient reram," *IEEE Trans. on Computers*, vol. 69, no. 8, pp. 1128–1142, 2020.

[2] M. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–37, 2021.

[3] X. Sun and S. Yu, "Impact of non-ideal characteristics of resistive synaptic devices on implementing convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 570–579, 2019.

[4] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," *arXiv preprint arXiv:1802.04680*, 2018.

[5] C. Zhang and P. Zhou, "A quantized training framework for robust and accurate reram-based neural network accelerators," in *2021 26th Asia and South Pacific Design Automation Conf. (ASP-DAC)*. IEEE, 2021, pp. 43–48.

[6] H. Sun, Z. Zhu, Y. Cai, X. Chen, Y. Wang, and H. Yang, "An energy-efficient quantized and regularized training framework for processing-in-memory accelerators," in *2020 25th Asia and South Pacific Design Automation Conf. (ASP-DAC)*. IEEE, 2020, pp. 325–330.

[7] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature communications*, vol. 9, no. 1, pp. 1–8, 2018.

[8] F. García-Redondo, S. Das, and G. Rosendale, "Training dnn iot applications for deployment on analog nvm crossbars," in *2020 Int. Joint Conf. on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[9] G. Yuan, X. Ma, C. Ding, S. Lin, T. Zhang, Z. S. Jalali, Y. Zhao, L. Jiang, S. Soundarajan, and Y. Wang, "An ultra-efficient memristor-based dnn framework with structured weight pruning and quantization using admm," in *2019 IEEE/ACM Int. Sym. on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

[10] S. Huang, A. Ankit, P. Silveira, R. Antunes, S. R. Chalamalasetti, I. El Hajj, D. E. Kim, G. Aguiar, P. Bruel, S. Serebryakov *et al.*, "Mixed precision quantization for reram-based dnn inference accelerators," in *2021 26th Asia and South Pacific Design Automation Conf. (ASP-DAC)*. IEEE, 2021, pp. 372–377.

[11] Y. Cai, T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Low bit-width convolutional neural network on rram," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1414–1427, 2019.

[12] T. Huang, T. Luo, M. Yan, J. T. Zhou, and R. Goh, "Rct: Resource constrained training for edge ai," *arXiv preprint arXiv:2103.14493*, 2021.

[13] A. Gural, P. Nadeau, M. Tikekar, and B. Murmann, "Low-rank training of deep neural networks for emerging memory technology," *arXiv preprint arXiv:2009.03887*, 2020.

[14] S. Roy, S. Sridharan, S. Jain, and A. Raghunathan, "Txsim: Modeling training of deep neural networks on resistive crossbar systems," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 730–738, 2021.

[15] S. Huang, X. Sun, X. Peng, H. Jiang, and S. Yu, "Overcoming challenges for achieving high in-situ training accuracy with emerging memories," in *2020 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 2020, pp. 1025–1030.

[16] Y. Luo and S. Yu, "Accelerating deep neural network in-situ training with non-volatile and volatile memory based hybrid precision synapses," *IEEE Trans. on Computers*, vol. 69, no. 8, pp. 1113–1127, 2020.

[17] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. of the IEEE Conf. on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[19] W. Wu, H. Wu, B. Gao, P. Yao, X. Zhang, X. Peng, S. Yu, and H. Qian, "A methodology to improve linearity of analog rram for neuromorphic computing," in *2018 IEEE Sym. on VLSI Technology*. IEEE, 2018, pp. 103–104.