Fault-tolerant Deep Learning using Regularization

(Invited Paper)

Biresh Kumar Joardar*, Aqeeb Iqbal Arka†, Janardhan Rao Doppa†, Partha Pratim Pande†

*University of Houston, Houston, TX, email: bjoardar@central.uh.edu

†Washington State University, Pullman, WA, email: {aqeebiqbal.arka, jana.doppa, pande}@wsu.edu

ABSTRACT

Resistive random-access memory has become one of the most popular choices of hardware implementation for machine learning application workloads. However, these devices exhibit non-ideal behavior, which presents a challenge towards widespread adoption. Training/inferencing on these faulty devices can lead to poor prediction accuracy. However, existing fault tolerant methods are associated with high implementation overheads. In this paper, we present some new directions for solving reliability issues using software solutions. These software-based methods are inherent in deep learning training/inferencing, and they can also be used to address hardware reliability issues as well. These methods prevent accuracy drop during training/inferencing due to unreliable ReRAMs and are associated with lower area and power overheads.

CCS CONCEPTS

• Hardware • Robustness • Fault tolerance • System-level fault tolerance

KEYWORDS

Deep learning, ReRAM, Reliability, Regularization

1 Introduction

Deep learning algorithms are employed in a wide variety of real-world applications, e.g., self-driving cars, medical diagnosis, and face recognition. Both training and inferencing of these deep models are computationally demanding tasks and are typically deployed on the cloud. However, there is a growing necessity to implement deep learning on edge platforms due to privacy and security concerns [1], the need for user-specific customization [2], low latency and real-time requirements (such as in augmented/virtual reality applications). However, implementing these applications on edge devices is challenging due to area and energy constraints. Addressing this necessitates suitable high-performance and energy efficient hardware support.

Emerging resistive random-access memory (ReRAM) technology can accelerate both CNN training and inferencing and are suitable for edge devices [3][4]. Existing ReRAM-based architectures, e.g., Pipelayer [3], ISAAC [4] and AccuReD [5], outperform GPUs for training/inferencing of CNNs while consuming significantly less energy. ReRAM-based architectures have also been used to accelerate other deep learning applications such as RNNs, GNNs, transformers, etc. [16][22][23]. ReRAM-based systems are more area-efficient compared to their GPU counterparts and do not require expensive off-chip memory access due to the "in-memory" nature of the ReRAM-based computation [3][4][5]. Despite these advantages, existing ReRAM-based architectures are not reliable. ReRAMs are susceptible to different types of defects and noise. These non-idealities hamper the widespread adoption of large-scale deep learning algorithms on ReRAM crossbar-based accelerators [6]. Prior work has demonstrated that CNN training and inferencing in the presence of these non-idealities leads to unacceptably low accuracy of the models [5][7][8]. These challenges must be addressed to unlock the tremendous potential of ReRAM-based architectures for both training and inferencing using various types of deep models.

Several techniques have been proposed in the literature to counter the effect of faults in ReRAM-based architectures [7][9][10][11][12][13]. Some of the most common methods for fault tolerance include the use of error-correction codes (ECC), fault-aware remapping, and retraining. However, these solutions tend to have implementational challenges as we outline later in this paper. A practical fault-tolerant scheme for on-chip training/inferencing must be fast, and with low performance and hardware overhead. To address these challenges, we present an alternative direction of software-based solutions to address the reliability issues in ReRAM-based architectures. These software methods are native to deep learning algorithms, i.e., they are inherently used by ML practitioners during training/inferencing to improve predictive accuracy. Hence, these methods are often associated with lower hardware overheads and are also easy to implement. Experimental results demonstrate that using these software-based solutions training/inferencing of deep learning workloads can achieve near-ideal accuracy even when several of the available ReRAM cells are faulty/noisy.

2 Summary of Existing Fault-tolerant Methods

In this section, we discuss the different sources of non-idealities in ReRAM crossbars and how they impact training and inferencing of deep learning models. Next, we highlight the challenges associated with existing fault-tolerant methods to motivate the need for new types of solutions.

2.1 Reliability in ReRAM-based architectures

ReRAMs can suffer from both "hard" faults and "soft" faults [6]. Hard faults prevent the resistance of a ReRAM cell from being updated, resulting in write failures. Hard faults can be further classified into pre-deployment and post-deployment faults based on when the fault appears for the first time. Pre-deployment faults

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561120

appear before use, i.e., at $t = 0^-$ and are caused by manufacturing defects such as short defects, over-forming defects or reset failure. These faults cause the affected ReRAM cell's resistance to be stuck at the low/high resistance state, resulting in stuck-at-faults (SAFs), which are permanent in nature. Even if a crossbar passes manufacturing test, new faults can appear over time as the crossbar is utilized (i.e., at $t > 0$) [14]. These faults, referred as post-deployment faults, can be attributed to the limited write endurance of ReRAMs or due to the application of multiple consecutive write-0 (or write-1) pulses [6]. Faults due to multiple write operations to the same cell can result in SAFs that are permanent in nature, whereas faults due to multiple write-0 (or write-1) pulses appear only for a limited duration (i.e., the underlying faulty cells in this case are recoverable).

In addition to hard faults, ReRAMs also suffer from soft faults. These faults lead to erroneous outputs due to small changes in the ReRAM cell's behavior (it does not result in SAFs). These faults can be attributed to noise, process variations, IR drop, etc. [5][6]. The resistivity of ReRAM cells depends on the operating temperature. Hence, if the temperature changes, the resistance of the ReRAM cell will change. Since data on ReRAM cells is represented as resistance, any change in resistance will cause loss of stored data and lead to erroneous outputs. Similarly, process variations can cause cells to behave differently under similar operating conditions. This can lead to variations during read and write operations. All these factors interfere with normal operations of ReRAM-based architectures, which can lead to poor accuracy during training/inferencing of deep learning workloads.

2.2 Effect of non-idealities on deep learning

In this sub-section, we show the effect of ReRAM non-idealities on various deep learning algorithms.

Soft faults: As mentioned earlier, there are several sources of soft faults in ReRAMs. Here, we focus on one of them as an example: thermal noise. It is well known that ReRAM behavior is temperature dependent [5]. The relationship between temperature and the conductance of ReRAM cells has been studied in prior work (e.g., [15]). The ON- and OFF-state conductance for an ReRAM cell (G_{ON} and G_{OFF} respectively) is a function of temperature, which in turn affects the output current. As temperature increases, the conductance range (i.e., gap between G_{ON} and G_{OFF}), reduces significantly [15]. This causes the output current to change (resulting in erroneous outputs). The noise margin between the states also decreases gradually with increasing temperature as the conductance range (i.e., gap between G_{ON} and G_{OFF}), reduces significantly. As a result, ReRAM-based architectures are very susceptible to noise at higher temperature.

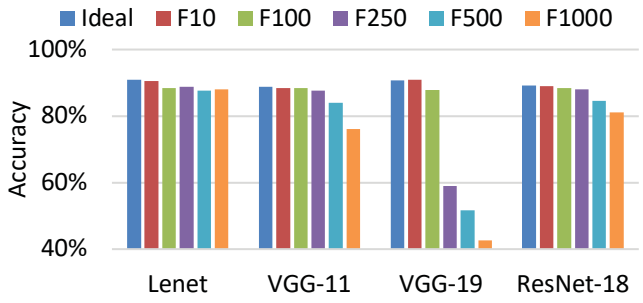

Fig. 1: Prediction accuracy of four CNNs trained on a non-ideal noisy ReRAM-based manycore system at different operating frequencies.

Fig. 1 shows how thermal-induced non-ideal effects, affect the accuracy of the trained model for four CNNs at different operating frequencies. The thermal noise is modeled following [7]. Here, the ReRAMs store 4-bits per cell following [4]. The operating frequency is gradually increased from 10 MHz (F10 in Fig. 1) to 1 GHz (F1000), where Fx refers to an ReRAM operating frequency of x MHz to study the effect of noise at different frequencies. As shown in Fig. 1, most CNNs fail to train in the presence of thermal noise (especially when higher frequencies are used). Clearly, this problem must be solved to enable on-chip training on ReRAM-based architectures.

Hard faults: Next, to demonstrate the effect of hard faults on CNN training/inferencing, we train six well-known CNNs: LeNet, AlexNet, VGG-11/16/19, and ResNet-18. LeNet is trained on the FashionMNIST dataset. AlexNet, VGG-11/16/19 and ResNet-18 are trained on the CIFAR-10 dataset. Fig. 2 shows the inferencing accuracy when these six CNNs are trained using both ideal ReRAMs and with fault densities of 0.1% and 2%, respectively. Here, we define "fault density" as the percentage of ReRAM cells that are faulty in the ReRAM-based architecture. 'Fault-x' in Fig. 2 refers to x% fault density. For instance, a fault density of 2% indicates that 2% of all available ReRAM cells are faulty. No fault-tolerant design strategy is adopted here to first assess the severity of the problem. As we can see from Fig. 2, most CNNs fail to train successfully (i.e., reach near-ideal accuracy) in the presence of faults. The problem is more acute for CNNs with more layers (such as VGG-19 and ResNet-18) and for higher fault density. For instance, ResNet-18 fails to train at even 0.1% fault density.

This problem is not specific to CNNs only. Other deep learning application workloads also encounter accuracy drop in the presence of these non-ideal effects. As an example, we show how hard faults affect GNN training. GNNs also consist of matrix multiplications, which can be accelerated using ReRAMs [16]. Fig. 3 shows the accuracy when three GNNs are trained with different fault densities. 'Fault-x' in Fig. 3 refers to x% fault density. Here, we train three GNN models for node classification on the Cora, PPI, and Amazon2M datasets. Unlike some of the CNNs, GNNs can train in the presence of faults but reach noticeably lower accuracy

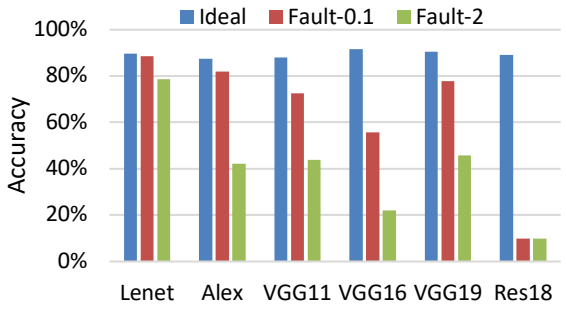

Fig. 2: Accuracy of trained model when different CNNs are trained using ReRAM crossbars with Ideal, 0.1%, and 2% fault densities (referred to as Ideal, Fault-0.1 and Fault-2 respectively).

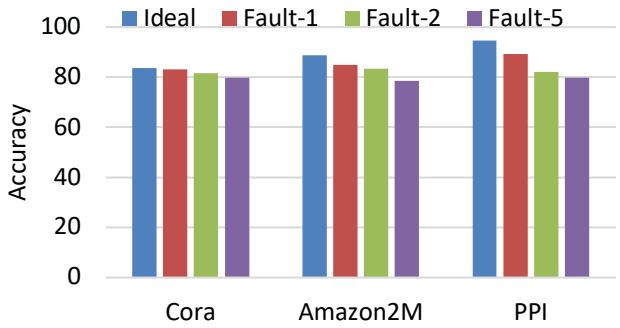

Fig. 3. Accuracy comparison of trained GNN model, with different fault densities (1%, 2%, and 5% faults)

than their ideal counterparts. This happens as GNN models tend to be shallower than CNNs. The GNNs used here are up to five layers deep only, while CNNs such as ResNet-18 has 18 layers. However, even at 1% fault density all three GNN models suffer from accuracy loss of 0.6%, 3.7%, and 5.4% for Cora, Amazon2M and PPI, respectively. As expected, the drop in accuracy increases at higher fault density for all GNN models. For 2% and 5% fault density, there is an average accuracy drop of 6.7% and 9.5% respectively.

Summary of key observations: To analyze the accuracy drop, Fig. 4 explains the effects of faults on CNN weights. Here, we study the accuracy drop due to hard faults as it is more severe; the accuracy drop due to soft faults can also be attributed to the same findings. The training becomes unstable when the weights become extremely large; this leads to poor training accuracy. This explosion in some weights happens due to the distributed nature of the mapping of weights to ReRAM cells [4]. Fig. 4 shows an example of how weights are mapped to ReRAM cells. Note that ReRAM-based architectures most commonly utilize a 16-bit fixed-point representation. However, storing all 16-bits in one cell is practically impossible due to noise and area concerns. Groups of bits are mapped to different ReRAM cells as shown in Fig. 4. In Fig. 4, we assume that 4-bits are stored per cell for the sake of illustration following [3]. Note that, 2-bits and 1-bit per cell are also common. The partial outputs (y_i) need to be accumulated using a shift-and-add operation to obtain the final output (Y), which can be mathematically expressed as:

$$Y = \sum_{i=1}^{4} 16^{4-i} * y_i \quad (1)$$

This shift-and-add operation due to the distributed mapping of weights to ReRAM cells, presents an interesting problem. From Equation (1), we note that faults at different locations will have varying amounts of impact on the final output (Y). For instance, an error due to a fault in y_4 (LSB bits) will be multiplied by a factor of 16^0 whereas the error due to a fault in y_1 (MSB bits) will be multiplied (magnified) by a factor of 16^3. Hence, faults near the MSB can artificially introduce an exploding/vanishing gradients problem, which results in faulty feedback during backpropagation. The explosive gradients add a positive reinforcement in the training loop, i.e., some of the weights explode after repeatedly accumulating these large gradients over multiple iterations of the weight update step. This leads to the rapid increase in the weight values, which in turn results in poor model accuracy after training for both CNNs and GNNs.

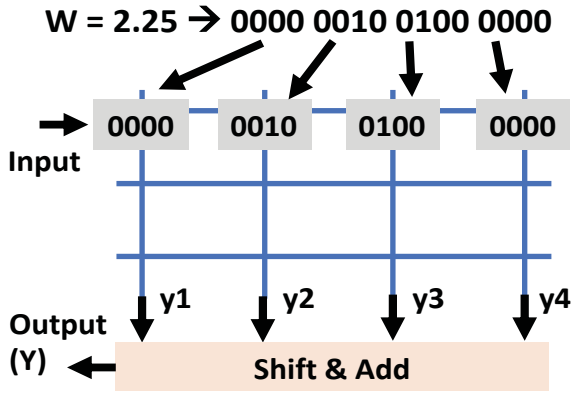

Fig. 4: Distributed mapping of weight (w) bits to multiple ReRAM cells. The shift-and-add block accumulates partial outputs (y_i) to compute the output (Y) following Equation (1)

2.3 Existing fault-tolerant methods

Several fault-tolerant ReRAM-based architectures have been proposed in prior work to address both soft and hard faults. Remapping of CNN weights to non-faulty ReRAM cells has been proposed in [10]. The remapping-based scheme uses the inherent sparsity of a neural network to make the ReRAM-based design robust against stuck-at-0 (SA0) faults specifically for inferencing [10]. By re-ordering the columns/rows, the zeros in the weight matrices are mapped to ReRAM cells with SA0 faults. However, this method requires solving a Knapsack-based formulation, which is an NP-hard problem. To implement the genetic algorithm-based Knapsack formulation (as described in [10]), we will need additional hardware, which leads to performance and area overheads. In addition, this method is not suitable for the purpose of training where the weights keep changing after each update.

Error-correcting codes (ECC) can be used to detect and correct errors in ReRAM-based architectures. Data aware AN code and LDPC are two such schemes that are equally effective for CNN inferencing [9][17]. However, the encoding and decoding involved in any ECC-based scheme introduce additional power and performance overheads. Moreover, the area overhead of ECC-based methods is relatively high. For instance, the AN code method introduces 6.3% area overhead to ReRAM tiles [9].

Error compensation has been proposed as another effective fault-tolerant mechanism for ReRAM-based architectures [7]. The compensation scheme relies on a one-time profiling using fault-detection techniques. Next, the difference between the non-ideal output caused by faults and the ideal output (referred as compensation) is calculated using a digital co-processor such as a CPU/GPU. Finally, the compensation is added to the faulty crossbar output. However, this methodology requires an additional digital co-processor besides the ReRAM-based system (hence, high area and power overhead). In addition, it requires prior profiling, which is not possible for training of deep models. Moreover, this

methodology may not be able to compensate for post-deployment faults as it relies on a priori fault detection.

Redundancy is another popular fault-tolerant scheme for ReRAMs. Prior work has proposed using both redundant ReRAM cells [12] and redundant CNN neurons [13] as fault tolerant methods. The methodology proposed in [11] uses triple modular redundancy (TMR) to achieve reliable operation. However, redundancy schemes such as TMR are prohibitively expensive in terms of the hardware overhead.

Retraining on faulty ReRAMs has been proposed for inferencing [11]. Here, the aim is to make the pre-trained weights aware of the hardware faults by re-training the weights. However, this method is not applicable for the purpose of training from scratch.

To summarize, existing fault tolerant solutions suffer from a combination of the following drawbacks. They are often slow, impose high performance, power, and area overheads, and require a priori fault detection; these shortcomings make prior solutions unattractive for training CNNs/GNNs on faulty ReRAMs. An effective fault-tolerant scheme for CNN/GNN training must be fast, while introducing negligible overhead.

3 Software solutions to mitigate faults

In this section, we will present software solutions that are native to ML algorithms as an alternative for reliable training/inferencing for CNNs and GNNs.

3.1 Batch Normalization

It is well known that training deep CNNs is difficult due to the vanishing/exploding gradients problem [18]. Batch normalization (BN) is a key CNN layer that addresses this problem. BN regularizes gradient from distraction to outliers. Prior work has discovered that by viewing BN as an implicit regularizer, it can be decomposed into population normalization and gamma decay as an explicit regularization [24]. Without BN, training deep CNNs on ReRAM-based architectures can result in: (a) no meaningful training, or (b) significant loss of prediction accuracy. Specialized initialization schemes (e.g., Xavier initialization [19]) have been proposed to train CNNs in the absence of BN. However, these methods require careful hyper-parameter selection (i.e., expert domain knowledge) and yet, are not effective all the time. The use of BN reduces the hyper-parameter dependencies and improves CNN training. Hence, it is widely used by ML practitioners. Interestingly, BN layers can also be used to solve some reliability issues on ReRAM-based architectures. For instance, BN layers can reduce the impact of thermal noise during CNN training.

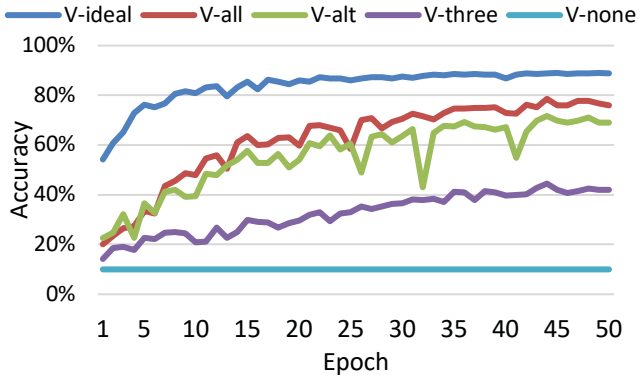

Fig. 5: Accuracy of trained models using noisy ReRAMs with different number of normalization layers.

To study the impact of normalization layers on noise resilience, we vary the number of normalization layers used during training. Fig. 5 shows the prediction accuracy with varying number of Normalization layers. For this experiment, we consider four different flavors of CNN architectures: VGG-11 with (a) all layers followed by a Normalization (V-All), (b) Normalization after alternate layers (V-Alt) and (c) Normalization after every 3 layers (V-Three), and (d) no Normalization case (V-none). Fig. 5 shows the accuracy when VGG-11 is trained with these configurations mentioned above along with the ideal training case (V-Ideal). Here, we assume all ReRAMs to operate at 1GHz where the effect of noise is the highest (Fig. 1). From Fig. 5, we note that V-All performs the best among the four nonideal configurations (accuracy of 78.6%). The model accuracy gradually falls due to the non-ideal nature of ReRAMs as the number of Normalization layers are reduced (69% for V-Alt, 42% for V-Three, 10% for V-None). This happens as the effect of thermal noise gets amplified, resulting in an artificial exploding gradient scenario. Normalization layers counter this problem. Therefore, having Normalization layers can improve the reliability of CNN training in presence of some hardware non-idealities. The AccuReD architecture introduces normalization support in ReRAM-based systems [5]. Since normalization layers are computationally inexpensive, the increase in execution time due to their addition is relatively negligible.

3.2 Weight clipping

Besides thermal noise, ReRAMs are prone to hard faults (both pre- and post-deployment). Unlike noise, these defects result in SAFs, which can be permanent or temporary in nature. First, we attempt to solve this problem using BN layers only. However, our analysis indicates that BN layers fail to solve the accuracy drop due to hard faults completely. This happens as a subset of the ReRAM hard faults are permanent in nature. Consequently, some of the exploding gradients are permanent, which result in repeatedly accumulating errors in the weights; this results in extremely large weights that make the training unstable. As a result, the scaling due to BN is ineffective here as the errors keep getting accumulated after every epoch unlike in the case of noise.

To prevent these exploding weights, we can use weight clipping [8]. Clipping these exploding weights to a relatively lower value ϵ, where $\epsilon > 0$, will enable the CNNs to train successfully. The clipping operation can be mathematically expressed using the following equation:

$$|w| = \begin{cases} |w|, \text{if } |w| < \epsilon \\ \epsilon, \text{otherwise} \end{cases} \quad (2)$$

Here, we can determine ϵ in two ways: (a) Static threshold based on prior profiling, and (b) Adaptive threshold, where the value of ϵ is determined during runtime without user intervention. The static threshold can be determined based on prior profiling. The threshold is 'static' as it remains constant throughout the training process.

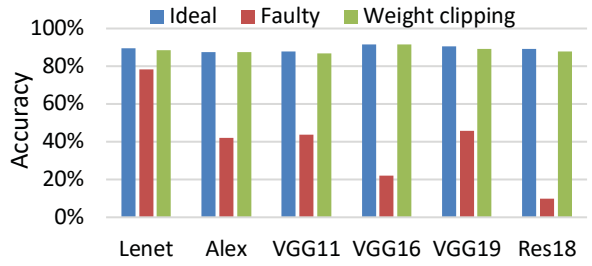

Fig. 6. Accuracy comparison of trained CNN models with/without weight clipping at different fault densities.

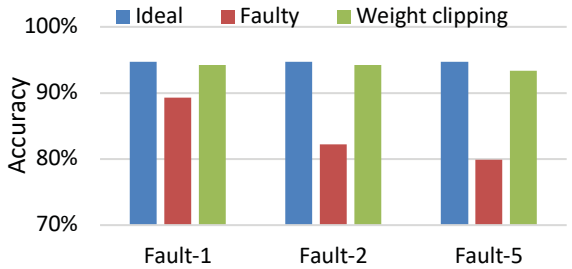

Fig. 7. Accuracy comparison of trained GNN models with/without weight clipping at different fault densities.

Adaptive threshold, on the other hand, enables us to determine ϵ during run-time without any prior knowledge. The threshold can be calculated adaptively using the mean and variance of weights like the computations in the BN layers. For instance, a threshold of $\mu + 3\sigma$ is used in [8]. The μ and σ calculations are already done in BN layers; hence the threshold calculation can be done in similar manner as the BN layers. We have introduced a ReRAM/GPU based heterogeneous manycore architecture called AccuReD, which has been shown to outperform sole GPU-based counterparts for training CNNs [5]. AccuReD consists of multiple planar layers of ReRAMs and GPUs, connected using a 3D structure. In AccuReD, BN layers are implemented using GPUs as they require full precision support and more complex mathematical operations (such as division and square-root) [5]. The adaptive threshold calculation is also implemented on the GPUs of AccuReD as it requires similar set of operations like BN. The GPUs are used for the BN computations during training (forward and backward phase). However, clipping occurs only during the weight update stage after a batch of data is processed. During the weight-update stage, new weights are written to ReRAM cells while GPUs remain idle. Hence, the clipping operation can be implemented using GPUs (which would remain idle otherwise during weight updates) without affecting performance.

Weight clipping can address the accuracy loss due to faults as clipping the large weights stops the CNN training from becoming unstable [8]. As a result, the backpropagation algorithm has a much better chance to train the remaining weights and compensate for the ones mapped to the faulty cells. In addition, weight clipping acts as an implicit regularizer and reduces the sensitivity of the loss function to various types of distortions [20] (ReRAM faults in our case). Regularization encourages the optimization algorithm to find simple models using the training data, which leads to better generalization accuracy. We can view weight clipping as a form of regularization for deep learning. For more intuition, we explain this phenomenon through the lens of proximal optimization. In its most general form, the overall optimization objective for CNN training with a regularizer is as follows:

$$w^* = \arg\min_w \{L(w) + R(w)\} \quad (3)$$

Here w stands for the weights of the deep neural network, w^* represents the optimal set of weights, L(w) is the loss function defined over the training data, and R(w) is a regularizer (e.g., L2-norm of the weight vector). The above optimization (Equation (3)) is commonly solved using the standard stochastic gradient descent (SGD) algorithm. The weight update equation for SGD algorithm is $w_{new} = w_{old} - \gamma \frac{\partial L}{\partial w_{old}}$, where γ is the learning rate. In proximal optimization, we apply a proximal operator as follows: $w_{new} = \text{prox}_R\left(w_{old} - \gamma \frac{\partial L}{\partial w_{old}}\right)$. For the weight clipping approach, the proximal operator is defined as follows: $\text{prox}_R(w) = \max\{\min\{w, \epsilon\}, -\epsilon$, where *max* and *min* functions are applied over each element of the weight vector individually and ϵ represents the clipping threshold. Therefore, the corresponding regularizer for weight clipping in the optimization objective for CNN training (the parameter R(w) in Equation (3)) is given as:

$$R(w) = \begin{cases} 0, \text{if } \|w\|_\infty \leq \epsilon \\ \infty, \text{otherwise} \end{cases} \quad (4)$$

Intuitively, due to this regularizer (Equation (4)), the optimization algorithm will avoid selecting weight vectors that lie outside the ϵ-ball of l_∞ norm because of the large penalty. Empirically, in the case of non-convex optimization problems arising in training deep neural networks, this regularization reduces the sensitivity of the loss function objective $L(w)$ to distortions. This property has been used to apply different non-linear distortions to CNN weights with minimal accuracy loss [20]. The CNN variants with weight clipping were found to be more robust to non-linear distortions than those without clipping. To map this idea for our hardware architecture, the distortions arise due to faults in ReRAM crossbars. Fig. 6 shows the accuracy after training in the presence of faults using weight clipping. As shown in Fig. 6, weight clipping can recover almost all the lost accuracy due to faults. This shows how we can ensure reliable training in an otherwise faulty device using a simple regularization method.

Weight clipping is not specific to CNNs only. We have also tested its efficacy on other deep learning algorithms such as GNNs. GNNs consist of sparse matrix multiplication operation which can also be implemented using ReRAM-based architectures [16][21]. Hardware faults also affect the final accuracy of the trained GNN model as shown in Fig. 3 earlier. Fig. 7 illustrates the accuracy for the PPI dataset with varying fault densities (1%, 2%, and 5%) and using weight clipping. Here, we consider the PPI dataset as it experienced the highest accuracy drop in Fig. 3. As shown in Figure 7, weight clipping restores the accuracy loss in the presence of hard faults. When the fault density is below 2%, weight clipping achieves near-ideal accuracy. As an example, with 2% fault density, weight clipping improves accuracy by 11.9%. Weight clipping also helps restore accuracy within 1.5% of the fault free case even when 5% of the available cells are faulty. We make similar observations for other GNN datasets such as Cora and Amazon2M as well. These results and analysis demonstrate that weight clipping (a

regularization method) can be used to enable training even when the underlying hardware is faulty.

4 Conclusion

In this paper, we have discussed the different sources of non-idealities in ReRAM-based computing architectures. We have shown that these non-idealities can lead to unreliable training/inferencing of various deep neural networks. Existing fault-tolerant schemes tend to have implementation overheads. We present software solutions, namely normalization and weight clipping, to address this important problem. Normalization is a key CNN layer that is often used for training deep CNNs. We have demonstrated that Normalization adds significant amount of robustness in presence of ReRAM non-idealities. In addition, we have presented weight clipping, (another regularization scheme). We have shown that weight clipping can also improve the fault-tolerance of deep learning algorithms at negligible hardware cost. Overall, these methods present an alternate direction of fault-tolerant solutions that are yet to be investigated thoroughly.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1955353, CNS-1955196, Semiconductor Research Corporation under task ID 3012.001 and task ID 3014.001, and by the USA Army Research Office grant W911NF-17-1-0485. Biresh Kumar Joardar was also supported in part by NSF Grant # 2030859 to the Computing Research Association for the CIFellows Project.

REFERENCES

[1] California consumer privacy act home page. https://www.caprivacy.org/. Online; accessed 14/02/2021

[2] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," Google Research Blog, vol. 3, 2017

[3] L. Song, X. Qian, H. Li and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 541-552

[4] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 14-26

[5] B. K. Joardar, J. R. Doppa, P. P. Pande, H. Li and K. Chakrabarty, "AccuReD: High Accuracy Training of CNNs on ReRAM/GPU Heterogeneous 3-D Architecture," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 5, pp. 971-984, May 2021

[6] A. Chaudhuri and K. Chakrabarty, "Analysis of Process Variations, Defects, and Design-Induced Coupling in Memristors," IEEE International Test Conference (ITC), Phoenix, USA, 2018, pp. 1-10

[7] Z. He, J. Lin, R. Ewetz, J. Yuan and D. Fan, "Noise Injection Adaption: End-to-End ReRAM Crossbar Non-ideal Effect Adaption for Neural Network Mapping," 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2019, pp. 1-6

[8] B. K. Joardar, J. R. Doppa, H. Li, K. Chakrabarty, and P. P. Pande, "Learning to Train CNNs on Faulty ReRAM-based Manycore Accelerators," in ACM Transactions on Embedded Computing Systems (TECS), 20, 5s, Article 55, 2021.

[9] B. Feinberg, S. Wang and E. Ipek, "Making Memristive Neural Network Accelerators Reliable," 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, Austria, 2018, pp. 52-65

[10] L. Xia, M. Liu, X. Ning, K. Chakrabarty and Y. Wang, "Fault-Tolerant Training Enabled by On-Line Fault Detection for RRAM-Based Neural Computing Systems," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 9, pp. 1611-1624, Sept. 2019

[11] C. Liu, M. Hu, J. P. Strachan, and H. Li, "Rescuing Memristor-based Neuromorphic Design with High Defects," In Proceedings of the 54th Annual Design Automation Conference 2017 (DAC '17). Association for Computing Machinery, New York, NY, USA, Article 87, 1–6

[12] C. Lee, H. Lin, C. Lien, Y. Chih and J. Chang, "A 1.4Mb 40-nm embedded ReRAM macro with 0.07um2 bit cell, 2.7mA/100MHz low-power read and hybrid write verify for high endurance application," 2017 IEEE Asian Solid-State Circuits Conference (A-SSCC), Seoul, Korea (South), 2017, pp. 9-12

[13] B. Zhang, N. Uysal, D. Fan, and R. Ewetz. 2020. Redundant Neurons and Shared Redundant Synapses for Robust Memristor-based DNNs with Reduced Overhead. In Proceedings of the 2020 on Great Lakes Symposium on VLSI (GLSVLSI '20). Association for Computing Machinery, New York, NY, USA, 339–344

[14] E. Esmanhotto et al., "High-Density 3D Monolithically Integrated Multiple 1T1R Multi-Level-Cell for Neural Networks," 2020 IEEE International Electron Devices Meeting (IEDM), San Francisco, CA, USA, 2020, pp. 36.5.1-36.5.4

[15] C. Walczyk et al., "Impact of Temperature on the Resistive Switching Behavior of Embedded HfO2-Based RRAM Devices," IEEE Trans. Electron Devices, vol. 58, no. 9, 2011

[16] A. I. Arka, et. al., "Performance and Accuracy Tradeoffs for Training Graph Neural Networks on ReRAM-Based Architectures," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 29, no. 10, pp. 1743-1756, Oct. 2021.

[17] Q. Lou, et. al., "Embedding error correction into crossbars for reliable matrix vector multiplication using emerging devices," In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '20). Association for Computing Machinery, New York, NY, USA, 139–144

[18] B. K. Joardar, et. al., "High-Throughput Training of Deep CNNs on ReRAM-Based Heterogeneous Architectures via Optimized Normalization Layers," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 5, pp. 1537-1549, May 2022.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in AISTATS, pp. 249-256, 2010

[20] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K. Esser, Dharmendra Modha, 2016, Deep neural networks are robust to weight binarization and other non-linear distortions, in arXiv:1606.01981, 2016

[21] A. I. Arka, B. K. Joardar, J. R. Doppa, P. P. Pande, and K. Chakrabarty, "ReGraphX: NoC-enabled 3D Heterogeneous ReRAM Architecture for Training Graph Neural Networks," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1667-1672, doi: 10.23919/DATE51398.2021.9473949.

[22] Y. Long, T. Na and S. Mukhopadhyay, "ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 12, pp. 2781-2794, Dec. 2018, doi: 10.1109/TVLSI.2018.2819190.

[23] X. Yang, B. Yan, H. Li and Y. Chen, "ReTransformer: ReRAM-based Processing-in-Memory Architecture for Transformer Acceleration," 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2020, pp. 1-9.

[24] P. Luo, X. Wang, W. Shao, Z. Peng, "Towards Understanding Regularization in Batch Normalization," in ICLR 2019