Defending Bit-Flip Attack through DNN Weight Reconstruction

Jingtao Li, Adnan Siraj Rakin, Yan Xiong, Liangliang Chang, Zhezhi He, Deliang Fan, Chaitali Chakrabarti
School of Electrical Computer and Energy Engineering
Arizona State University, Tempe, AZ, 85287
{jingtao1, asrakin, yxiong35, lchang21, zhezhihe, dfan, chaitali}@asu.edu

***Abstract*—Recent studies show that adversarial attacks on neural network weights, aka, Bit-Flip Attack (BFA), can degrade Deep Neural Network's (DNN) prediction accuracy severely. In this work, we propose a novel weight reconstruction method as a countermeasure to such BFAs. Specifically, during inference, the weights are reconstructed such that the weight perturbation due to BFA is minimized or diffused to the neighboring weights. We have successfully demonstrated that our method can significantly improve the DNN robustness against random and gradient-based BFA variants. Even under the most aggressive attacks (i.e., greedy progressive bit search), our method maintains a test accuracy of 60% on ImageNet after 5 iterations while the baseline accuracy drops to below 1%.**

***Index Terms*—Bit-Flip Attack, Row-Hammer Attack, Security of Deep Neural Network**

I. Introduction

Neural networks have facilitated many real-world applications, including security-critical applications such as self-driving and authentication systems. Consequently, the security of neural networks has emerged as a very important problem. It has been shown that neural network accuracy can be severely compromised by adversarial input attacks; an adversary can maliciously add input noise to the image to cause targeted or un-targeted misclassification [1]. Very recently, it has been shown that a small variation on the model parameters can also draw similar devastating effects. For instance, it is demonstrated in [2], [3] that the target DNN can be easily crushed via naively flipping the exponential bit of weights stored in 32-bit floating-point representation. A possible method to mitigate it is by constraining the weight magnitude through fixed-point quantization [4], [5]. However, a successive work [6] presented a greedy progressive bit search algorithm which can precisely identify the vulnerable bits of 8-bit quantized DNN. It showed that a ResNet-18, with 93 million weight bits, can be fully malfunctional with merely 13 bit-flips.

Bit-flip Attacks (BFAs) can be easily launched on the DNN weights stored in DRAM through the well known Row-Hammer Attack (RHA) [7], [8] causing a significant accuracy drop [4], [6]. This can cause catastrophic consequences in security-critical applications such as object detection for autonomous driving [9]. Existing approaches to mitigate RHA include PARA [7], and Error-Correction Coding (ECC). However, it has been pointed out PARA needs modifications in the memory controller and ECC scheme is still vulnerable to attacks [10]. Recent research [11] creates a safety region for the Page-Entry Table (PTE) via pure software updates.

Our approach is a low-cost method to reconstruct DNN weights, including the attacked weights, in a way such that the change in weight value caused by the BFA is either dramatically reduced or is diffused to its neighboring weights. We achieve this through averaging over a grain of weights followed by an appropriate combination of quantization and clipping of weight values in a grain. The proposed reconstruction is done on the weights prior to inference computation in a three-step manner. We show that our method retains the clean model information with high probability and thereby reduces the effect of the weight perturbation caused by the BFA. The reconstruction process is embedded in the forward pass of training to achieve minimal accuracy degradation.

We evaluate the accuracy and loss degradation under three BFA variants, namely random BFA, gradient-based BFA including one-shot BFA and progressive BFA [6]. Our proposed scheme consistently achieves significantly higher accuracy compared to the baseline scheme. For instance, on the ResNet-18 ImageNet model, we achieve around 60% accuracy after 5 iterations of progressive BFA while the baseline accuracy drops to below 1%. Finally, through experiments on a gem-5 based multi-core system, we demonstrate our method has very small computation overhead. Our main contribution can be summarized as:

- We present a technique to significantly mitigate adversarial weight attack (a.k.a BFA). To the best of our knowledge, this is the first paper that presents a low-cost countermeasure solution to BFA.
- Through extensive experimentation, we show that our proposed weight reconstruction method greatly mitigates effect of BFA. On the ImageNet dataset, we show that our accuracy is around 60% while the baseline has only about 1% under PBFA attack.
- We show that the computation overhead of our scheme is very small through Gem5 cycle-accurate simulations.

The rest of this paper is organized as follows: In section II, we provide the background on BFA and the threat model. In section III, we present the detailed weight reconstruction method. In section IV, we present experimental results on the effect of weight reconstruction against BFA. In section V, we conclude our work.

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

II. Background

A. Random Bit-Flip Attack

In this version of the BFA, the attacker randomly selects a bit in any layer of a DNN model. It requires no knowledge of the model. This attack needs flipping of at least 10% of the weights to be able to breakdown the model as reported in [12].

B. Gradient-based Bit-Flip Attack

To conduct a Gradient-based BFA, the attacker requires access to a part of the dataset, DNN architecture and network parameters (i.e., weights, biases). This is similar to the traditional white-box attack [13], [14] in a related security domain.

First, we present the One-shot Bit-Flip Attack (OBFA), which identifies the bits with the highest gradient based on one-time inference loss. Given a n_q-bit quantized DNN weight parameterized by binary bits, define tensor $\{\mathbf{B}_l\}_{l=1}^{L}$, where $l \in \{1, 2, ..., L\}$ is the layer index. The OBFA identifies multiple bits with high gradient (by ranking elements in $|\nabla_{\mathbf{B}_l}\mathcal{L}|$) as vulnerable bit candidates across all the layers of the DNN, where $\mathcal{L}$ is the inference loss. Then OBFA flips several top candidates from each layer of the DNN based on gradient ranking.

Finally, we test our defense strategy against Progressive Bit-Flip Attack (PBFA), which is an effective attack algorithm presented in [6]. PBFA identifies vulnerable bits with a greedy gradient-based ranking per layer followed by a progressive search across layers. It performs intra-layer search similar to OBFA, but only identifies the bit with highest gradient ($\arg\max_{\mathbf{B}_l} |\nabla_{\mathbf{B}_l}\mathcal{L}|$) as vulnerable bit candidate. The inter-layer search is followed by an intra-layer search which compares the bit candidates selected by the intra-layer search using the increase in loss function for each bit candidate. The bit searching in iteration i can be formulated as an optimization process:

$$\begin{aligned} &\max_{\{\hat{\mathbf{B}}_l^i\}} \ \mathcal{L}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}_l^i\}_{l=1}^{L}\big), \tilde{\boldsymbol{t}}\Big) \\ &\text{s.t.} \ \ \tilde{\boldsymbol{t}} = f(\boldsymbol{x}; \{\mathbf{B}_l\}_{l=1}^{L}); \ \sum_{l=1}^{L} \mathcal{D}(\hat{\mathbf{B}}_l^i, \mathbf{B}_l) \in \{0, 1, ..., N_b\} \end{aligned} \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{t}$ denotes the selected input mini-batch and ground-truth labels. $\hat{\mathbf{B}}_l^i$ is the quantized bit tensor of l-th layer perturbed by BFA in i-th iteration. $f(\boldsymbol{x}; \{\mathbf{B}\}_{l=1}^{L})$ compute the outputs of DNN parameterized by $\{\mathbf{B}_l\}_{l=1}^{L}$. $\tilde{\boldsymbol{t}}$ is the output of clean model as the soft-label, which replaces the ground-truth $\boldsymbol{t}$ to perform the attack. $\mathcal{L}(\cdot, \cdot)$ computes the loss. The attack efficacy can be measured by the Hamming distance (i.e., effective bit-flips) between prior- and post-attack model parameters $\{\hat{\mathbf{B}}_l^i\}_{l=1}^{L}$ and $\{\mathbf{B}_l\}_{l=1}^{L}$ given by $\sum \mathcal{D}(\hat{\mathbf{B}}_l^i, \mathbf{B}_l)$. Finally the attacker's optimization goal is to cause the DNN to malfunction with least number of bit-flips (i.e., $\min \sum \mathcal{D}(\hat{\mathbf{B}}_l^i, \mathbf{B}_l)$).

C. Threat Model

A neural network model consists of parameters such as weight, bias and batch-normalization parameters. The weights are stored in DRAM and loaded in cache prior to computation. We assume that the attacker has knowledge of the model architecture, model parameters, and partial training dataset.

Fig. 1 shows the threat model in this study. The attacker can identify the vulnerable bits using gradient-based BFA. Once the vulnerable bits are identified, the attacker performs RHA. To perform RHA, the attacker requires knowledge of victim physical address, partial knowledge of the vulnerable templates, methods to bypass cache, and OS behavior. We eliminate the possibility that an attacker can use RHA to attack the page table entry, by using the technique in [11]. We also assume that the batch-normalization and bias parameters cannot be attacked by RHA since they are difficult to precisely target because of their small size (less than 1 MB) [4].

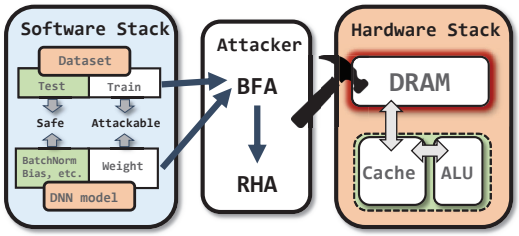

Fig. 1. Threat model, where the attackers use gradient-based BFA to identify the vulnerable bits and RHA to attack those bits

III. Proposed Method

A. Tackling Gradient-based BFA

To better understand the effect of gradient-based BFA, we first investigate the gradient distribution of weights near the target weight (similar to target bit), where a target weight is a candidate weight whose bits will be flipped by PBFA attack. Fig 2 shows the gradient distribution near the target weight of a standard ResNet-18 model (baseline). The target weight has much larger gradient compared to that of its neighboring weights. After 5 iterations, the magnitude of gradient becomes even larger. This is the main reason why PBFA, which targets weights with large gradients, is much more effective than other BFAs.

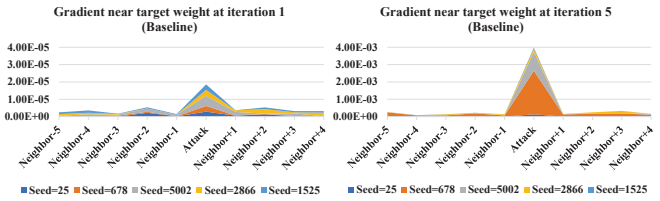

Fig. 2. Gradient distribution near the target weight of a standard ResNet-18 model (Baseline) for PBFA iteration number 1 and 5. Note around 200x increase in gradient values in iteration 5.

Consider a 8-bit weight with gradient $\nabla_{\mathbf{B}_i}\mathcal{L}$ that is picked as the target by gradient-based attack. The attacker induces a change of Δ_W on the target weight. The overall effect on the loss can be represented by:

$$|\Delta_{loss}| = F(|\nabla_{\mathbf{B}_i}\mathcal{L}|, |\Delta_W|) \tag{2}$$

where we assume that F is a a monotonically increasing function. This is corroborated by our evaluation presented in later sections. Thus, to reduce the overall increase in loss, we present a weight reconstruction method capable of reducing the Δ_W caused by a potential bit-flip. We show that reducing Δ_W helps mitigate the effect of the BFA.

B. Weight Reconstruction Method

The goal of weight reconstruction is to reduce the change in weight caused by the BFA. The proposed technique has three main steps: averaging, quantization and clipping. The main purpose of the averaging operation is to spread the effect of $|\Delta_W|$ on a group (or grain) of size G such that it only causes a small $|\Delta_W/G|$ change on the mean.

The purpose of the quantization step is to cancel the effect of $|\Delta_W/G|$ change on the quantized mean (Qmean). For an 8-bit weight, the average value before the attack could be any value between -128 and 127. Hence, if the gap between two quantization levels (Qlevel) is very large, the probability that the quantization step will cancel the $|\Delta_W/G|$ is high.

In Fig. 3 (a), we use hyperparameters P and K to control the Qlevel, and vary P and K to get different Qlevel settings. The Qlevel is shown as white bars between [-128, 127]; P is the ratio of the active Qlevel range compared to the full range (255); K is the number of Qlevels. If there is a change in the value of $|\Delta_W/G|$ on the mean before quantization, for the $P = 0.8, K = 2$ setting with large gap in Fig. 3 (a), the Qmean has a higher probability to stay unchanged compared to the $P = 0.8, K = 4$ setting.

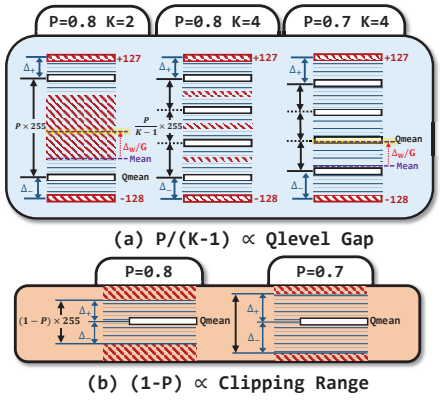

Fig. 3. Effect of range control hyperparameter P and number of quantization levels, K on weight reconstruction.

After the attack, if the Qmean information of the clean model is retained, it can be used as a base value and help achieve reduction in perturbation caused by the BFA. The clipping operation restricts all the weights to a small range around the Qmean, that is, the maximum deviation of target weight from the cleaned model is at most the whole clipping range ($\Delta_+ - \Delta-$). The clipping range of Qlevels with different P values is in shown in in Fig. 3, where the $P = 0.8$ case has a smaller clipping range compared to the $P = 0.7$ case. The blue-line area shows the allowed value range of the final reconstructed weight.

The detailed operation is presented in Algorithm 1.

Algorithm 1 Pseudo-code for weight reconstruction

Input: 8-bit fixed point weight W_b in integer format, number of weights in a grain G, ratio P of quantization range to the full weight value range of [-128, 127], and number of quantization levels K.

Output: W_r (reconstructed weight)

function RECONSTRUCT(W_b, G, P, K)
  `Level [1:K]` $\leftarrow P \times (-128) : \frac{P \times 255}{K-1} : P \times 127$
  $\Delta_- \leftarrow -(1-P) \times 128$
  $\Delta_+ \leftarrow (1-P) \times 127$
  `Iterator:` $w_b \leftarrow G$
  for w_b `in` W_b **do**
    `Mean` $\leftarrow$ `average(`w_b`)`
    `Qmean` $\leftarrow$ `round(Mean) to Level [1:K]`
    for `item in` W_b **do**
      if `item <` `Qmean` $+ \Delta_-$ **then**:
        `item` $\leftarrow$ `Qmean` $+ \Delta_-$
      else if `item >` `Qmean` $+ \Delta_+$ **then**:
        `item` $\leftarrow$ `Qmean` $+ \Delta_+$
      end if
    end for
  end for
  $W_r \leftarrow$ `stack(`w_r`)`
  return W_r
end function

We illustrate the reconstruction procedure with an example in Fig. 4 (a) for $G = 4$, $P = 0.8$ and $K = 4$. The original 8-bit weights are presented as integer values for better demonstration. In step 1, the average value of the weights in the grain (circled by gray dash-line) is computed. The four quantization levels are -0.8 x 128 = -102 to 102 with step size of 68, so the four levels are -102, -34, +34 and +102. The mean is rounded to its nearest quantization level at -34 in step 2. Since $\Delta-$ is -26 and $\Delta+$ is +25, the clipping is between lower boundary of -34 - 26 = -60 and upper boundary of -9. The first and second elements -74 and -105 are lower than -60 and get clipped to -60. The third and fourth elements do not exceed the boundary and so stay unchanged.

Fig. 4 (b) and (c) show two possible attacking scenarios. In Fig. 4 (b), the MSB of the third weight in the first row is flipped, and thus the value is changed from -10 to +118. The change of +128 on the third weight induces a change of +32 on the mean. But thanks to the quantization step, the Qmean remains unchanged. After clipping, the targeted weight becomes -9 (-10 $\rightarrow$ +118 $\rightarrow$ -9) while all three other weights in the same grain are unaffected. Thus, the effect of the attack is greatly mitigated.

In Fig. 4 (c), the MSB of the fourth weight in the second row is flipped by the BFA. Here the Qmean changes from +34

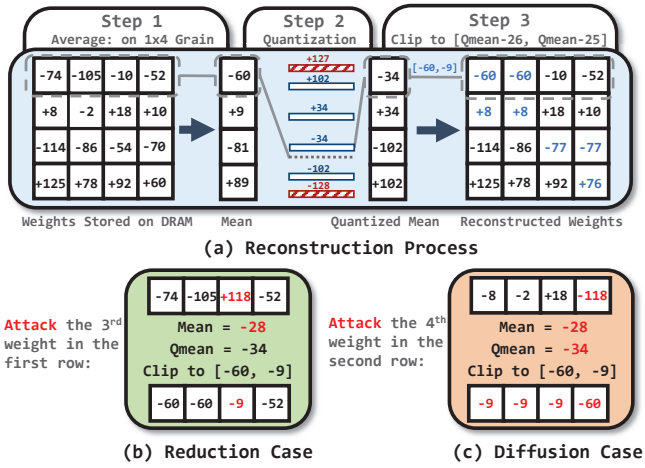

Fig. 4. DNN Weight Reconstruction Process. (a) Three steps of the reconstruction process. Two cases of bit-flip: (b) A direct reduction case, (c) A diffusion case.

to -34, and the average value affected by the attack goes to another quantization level. After clipping around the shifted Qmean, the first three neighboring weights are shifted by -17, -17, -27, respectively, and the targeted weight is shifted by -70. The value reduction in the target weight is nearly equal to the total value increase on its neighboring weights, so we call this case "diffusion case". This case is less favorable than (b) because the change in the weight values is likely to affect the accuracy. It is still better than the original case as the change in value of the target weight is reduced by nearly half. The small value change in its neighbors, which have smaller gradient, have a minor effect on the loss, which is supported by our gradient distribution study in 2.

The reduction case demonstrated in Fig. 4(b) reduces the change in value induced by BFA due to several reasons. First, since the weights in a grain have similar value, the average value is very close to its original value. Next, the quantization step cancels the change in the mean value. Finally, the clipping operation clips the value inside a small range near the Qmean, which ensures that the weight will always be very close to the average value.

Effect of parameters G, P and K: The choice of hyper-parameters G, P and K affect the accuracy and resiliency to BFA. We find that for different neural network architectures, the optimal hyperparameters are different. However, there are certain characteristics that are true for all networks.

By increasing grain size G, we can achieve higher reduction on the change in mean value, which is linked to better performance. However a large G may have a regularization effect on the weight distribution, to the point that the average value is close to zero, leading to poor accuracy. The gap between two quantization levels is a linear function of P and $1/K$. A larger gap implies a higher probability of canceling the error at the expense of reduced accuracy of weight representation.

The modification of weights during inference implies that re-training or fine-tuning has to be done to achieve high model accuracy. Our training algorithm is as follows.

Algorithm 2 Proposed Training Method

Setting: Function Reconstruct(W_b, G, P and K). L is the number of layers, C is the loss function for a mini-batch and η is the learning rate. a_l is the activation value in layer l. W^{t-1} represents the full-precision weight reference, and the 8-bit fixed point weights W_b^{t-1} and biases b^{t-1} of iteration $t-1$ are reconstructed as weights W_r^{t-1} for standard forward pass.

1. Forward Propagation:

$W_b^{t-1} \longleftarrow \text{Quantize}(W^{t-1})$

$W_r^{t-1} \longleftarrow \textbf{Reconstruct}(W_b^{t-1}, G, P, K)$

for $l = 1 : L$

  Compute a_l given a_{l-1}, W_r^{t-1} and b^{t-1}

2. Backward Propagation:

Initialize output layer's activation gradient $\frac{\partial C}{\partial a_L}$

for $l = L$ to 2

  Compute $\frac{\partial C}{\partial a_{l-1}}$ given $\frac{\partial C}{\partial a_l}$ and W_r^{t-1}

3. Parameter Update:

Compute $\frac{\partial C}{\partial W_r^{t-1}}$ and $\frac{\partial C}{\partial b^{t-1}}$ given $\frac{\partial C}{\partial a_l}$ and a_{l-1}

$W^t \longleftarrow W^{t-1} - \eta \frac{\partial C}{\partial W_r^{t-1}}$

$b^t \longleftarrow b^{t-1} - \eta \frac{\partial C}{\partial b^{t-1}}$

In the training algorithm, the full-precision weights are used as a reference for training only. The back-propagation through non-differentiable functions (such quantization and clipping) is done by using straight-through estimator [15] to perform standardized quantized neural network training [16]. After training converges, the full precision weights are quantized to 8-bit format and the quantized weights are stored for inference.

IV. Experimental Results

A. Experimental Setting

Dataset. To evaluate our defense strategy, we used two popular visual datasets for image recognition: CIFAR-10 [17] and ImageNet [18]. CIFAR-10 contains 60k RGB images with 32×32 size evenly sampled from 10 classes, with 50k images as training-set and 10k images as test-set. ImageNet dataset contains 1000 distinct classes with 1.2M training samples. To conduct Progressive BFA we randomly select a sample test batch of size 128 for CIFAR-10 and size 256 for ImageNet, respectively.

DNN model. We use ResNet-20 model for CIFAR-10 dataset and ResNet-18 model for ImageNet [19]. ResNet-20 model uses a batch size of 128. It is trained from scratch for 200 epochs using Adam [20], a learning rate of 0.01 and learning rate decay of 0.0001. For ImageNet, we do fine-tuning on the trained model for 20 epochs using stochastic gradient descent (SGD) [21] method and batch size of 256.

Platform. All experiments are conducted on a machine equipped with an NVIDIA RTX 2080ti GPU. The training and inference for the DNN models are implemented using Pytorch.

B. BFA Settings

For each set of bit-flip injection experiments, we repeat 5 times to get the average accuracy and loss. For RBFA, we randomly inject 20, 40, 80, 160, 320 bit-flips on each layer of the network. In OBFA, the adversary needs to pinpoint the targeted location to perform the attack, and so we inject fewer bit-flips. With 10 bit-flips on each layer, the baseline accuracy drops to a very low value, and so we pick GBF-2, 4, 6, 8, 10 in the experiments. In PBFA, the attack needs multiple rounds of inference of target system, which adds to the time cost and difficulty. The accuracy of baseline drops more quickly, so we only perform iteration of progressive search till the baseline accuracy drops to around 20% for CIFAR-10 and around 1% for ImageNet.

C. Defending against RBFA/OBFA

For the ResNet-20 model on CIFAR-10, we train the baseline model (8-bit fixed point standard model) and our proposed weight reconstruction based model (WRecon). We select $G = 4$, $P = 0.8$, and $K = 8$. The test accuracy of the reconstruction model with no attack (clean model) is 90.83%, which is 1% lower than the baseline model of 91.89%.

RBFA: We perform RBFA with different numbers of bit-flips. The test accuracy and loss versus number of bit-flips are shown in Fig. 5. The accuracy drop and increase in loss are much slower for WRecon. With 320 bit-flips in each layer, the baseline reduces to below 60% while WRecon still has a 75% accuracy.

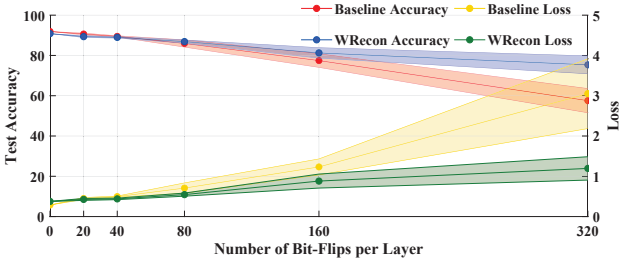

Fig. 5. RBFA: Comparison of test accuracy and loss of the baseline and proposed WRecon for ResNet-20 model on CIFAR-10 (the standard deviation is represented by the shaded area).

OBFA: We perform OBFA on the baseline model and the WRecon model with $G = 4$, $P = 0.8$, and $K = 8$, and vary the number of bit-flips from 2 to 10 in each layer. The test accuracy and loss values are shown in Fig. 5. For the baseline, the accuracy degradation is much more severe when the number of bit-flips increases, compared to the RBFA case in Fig. 5. With 10 bit-flips in each layer, the accuracy of the baseline model reduces to around 20% while WRecon has a 45% accuracy.

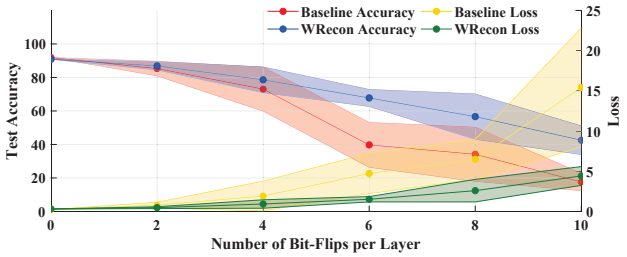

Fig. 6. OBFA: Comparison of test accuracy and loss of the baseline and proposed WRecon for ResNet-20 model on CIFAR-10 (the standard deviation is represented by the shaded area).

D. Defending against PBFA

For ResNet-20 model on CIFAR-10, we run PBFA for up to 20 iterations. We use the WRecon model with $P = 0.8$, $G = 4$ and $K = 4$. Fig. 7 shows the test accuracy and loss value as a function of the number of iterations for the two models. The accuracy and loss of the proposed WRecon model change slowly compared to the baseline model. After 10 iterations, the accuracy of the baseline model reduces to below 40%, while the proposed WRecon has 75% accuracy. After 20 iterations, the baseline model accuracy degrades further to below 20% while WRecon has around 45% accuracy.

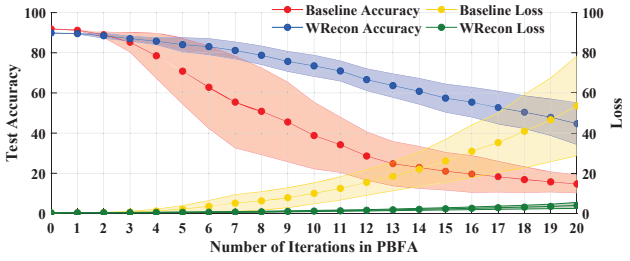

Fig. 7. PBFA: Comparison of test accuracy and loss of the baseline and model with proposed weight reconstruction for ResNet-20 model on CIFAR-10 (the standard deviation is represented by the shaded area).

Next, we present results for the ResNet-18 model on ImageNet. We perform PBFA for only 5 iterations for both the baseline model and the reconstruction model using $P = 0.8$, $G = 16$ and $K = 8$. The test accuracy of the WRecon model with no attack (clean model) is 69.23%, which is 0.5% lower than the baseline model of 69.79%. After 5 iterations of PBFA, the baseline accuracy degrades to below 1%, while the WRecon model has an accuracy of 60%.

E. Weight gradient analysis

As already demonstrated in Fig. 2, PBFA attacks weights with large gradients and causes a disastrous drop in accuracy for the baseline model. This is the main reason for the effectiveness of PBFA on the baseline model. For the WRecon model, the gradient distribution near the target weight is shown in Fig. 9. At iteration 1, compared with the baseline shown in Fig. 2, the gradient of WRecon model does not show much difference. While at iteration 5, the gradient of models with

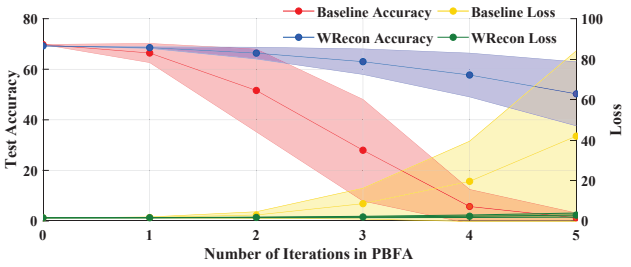

Fig. 8. PBFA: Comparison of test accuracy and loss of the baseline and model with proposed weight reconstruction for ResNet-18 model on ImageNet (the standard deviation is represented by the shaded area).

weight reconstruction increases at a much lower rate (increase by 10x compared to baseline by 200x).

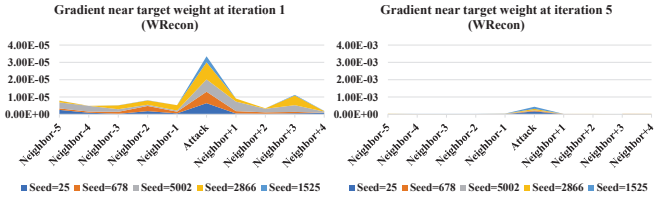

Fig. 9. Gradient distribution near the target weight of ResNet-18 models with weight reconstruction (G16P0.8K8) at PBFA iteration of 1 and 5.

F. Overhead Analysis

For each layer, we compute the inference of one input and collect the execution time of each kernel for a multi-core 14nm node simulated in Gem5. For the case when the fully connected layer has 128 input neurons and 1024 output neurons and weight matrix is of size 128x1024, the execution time is 2.691ms and the reconstruction process costs another 0.013ms. For convolutional layer, we consider the input size of 32x32x3 with zero-padding, output size of 32x32x16 and kernel size of 3x3. The computation time of 2D convolution is 0.363ms while the reconstruction time costs another 0.009ms. Thus in both cases, the overhead introduced by reconstruction process is very small, and this cost will further decrease when the input is processed in batches.

V. Conclusion

In this work, we present a weight reconstruction method to mitigate several types of BFAs, including the powerful progressive BFA (PBFA), on neural networks. We assume that the attacker meets the white-box assumption and is capable of implementing different types of BFA through perform RHA. Our low overhead method reduces the effect of these attacks through changing the value of the attacked weight through a combination of averaging over a grain of weights followed by appropriate quantization and clipping. On a ResNet-18 model, the proposed method achieves 60% accuracy after 5 iterations of PBFA while the baseline accuracy drops to 1%.

References

[1] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57.

[2] M. A. Hanif, F. Khalid, R. V. W. Putra, S. Rehman, and M. Shafique, "Robust machine learning systems: Reliability and security for deep neural networks," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pp. 257–260.

[3] X. Hou, J. Breier, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Experimental evaluation of deep neural network resistance against fault injection attacks." *IACR Cryptology ePrint Archive*, vol. 2019, p. 461, 2019.

[4] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," *arXiv preprint arXiv:1906.01017*, 2019.

[5] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 131–138.

[6] A. S. Rakin, Z. He, and D. Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220.

[7] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[8] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 245–261.

[9] S. Hecker, D. Dai, and L. Van Gool, "Failure prediction for autonomous driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1792–1799.

[10] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 55–71.

[11] X.-C. Wu, T. Sherwood, F. T. Chong, and Y. Li, "Protecting page tables from rowhammer attacks using monotonic pointers in dram true-cells." in *ASPLOS*, 2019, pp. 645–657.

[12] J. Li, M. Mao, and C. Chakrabarti, "Improving reliability of reram-based dnn implementation through novel weight distribution," in *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 189–194.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[14] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[15] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[17] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255.

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

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.