SPARSEBFA: ATTACKING SPARSE DEEP NEURAL NETWORKS WITH THE WORST-CASE BIT FLIPS ON COORDINATES

Kyungmi Lee, Anantha P. Chandrakasan

Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
Cambridge, USA

ABSTRACT

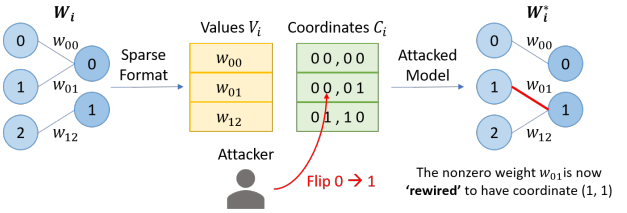
Deep neural networks (DNNs) are shown to be vulnerable to a few carefully chosen bit flips in their parameters, and bit flip attacks (BFAs) exploit such vulnerability to degrade the performance of DNNs. In this work, we show that DNNs with high sparsity that typically result from weight pruning have a unique source of vulnerability to bit flips when their coordinates of nonzero weights are attacked. We propose SparseBFA, an algorithm that searches for a small number of bits among the coordinates of nonzero weights when the parameters of DNNs are stored using sparse matrix formats. Using SparseBFA, we find that the performance of DNNs drops to the random-guess level by flipping less than 0.00005% (1 in 2 million) of the total bits.

***Index Terms*—** Bit flip attacks, deep neural networks, fault injection attacks, model compression, security

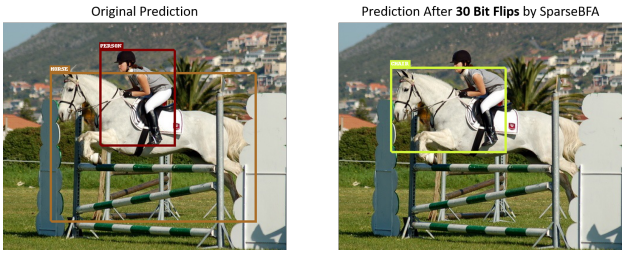
1. INTRODUCTION

Understanding potential sources of security threats for deep neural networks (DNNs) is becoming important, as they are increasingly deployed for high-stake applications and at edge devices. In particular, security threats that exploit hardware-level faults of memory elements and processors, such as fault injection attacks that can corrupt information and evade error-correcting mechanisms, can be concerning for edge devices that store and process sensitive information. Exploiting fault injection attacks, recent works [1, 2, 3] proposed *bit flip attacks (BFAs)* that can degrade the performance of DNNs by flipping a few selected weight bits.

While the risk posed by BFAs is analyzed for dense DNNs, where bit flips corrupt the values of weights, we note that sparse DNNs have a unique source of vulnerability when their weights are stored using sparse matrix formats, such as Coordinate List (COO) or Compressed Sparse Column (CSC). Those formats explicitly store the coordinates of nonzero weights, which can be corrupted by bit flips, and such bit flips will rewire the connections between neurons instead of changing the values of weights (Fig. 1).



(a) Visualizing how a bit flip in the coordinates affects a DNN



(b) Predictions of the object detection model are degraded by SparseBFA

Fig. 1. (a) When an attacker flips a bit in the coordinates representing the location of nonzero weights, the connection between neurons is rewired. (b) SparseBFA aims to find the minimal number of bit flips in the coordinates to degrade the performance of DNNs. In this example, the object detection model [4] that originally predicts both a person and a horse correctly loses its performance after 30 bit flips found by SparseBFA, and wrongly predicts that a chair is present in the image (from PASCAL-VOC [5]).

In this work, we examine how bit flips in the coordinates affect the performance of sparse DNNs. We propose SparseBFA, a novel BFA algorithm that iteratively finds the coordinate bits that maximally increase the loss of a DNN when they are flipped. While random bit flips in the coordinates do not significantly affect the performance of a DNN, we show that the performance of DNNs drops to the random-guess level for the classification tasks (CIFAR-10 [6], Tiny-ImageNet [7]) within $1 \sim 15$ bit flips, and below 0.10 mean average precision for the object detection task (PASCAL-

This work was funded in part by Facebook and Korea Foundation for Advanced Studies.

VOC [5]) within $30 \sim 40$ bit flips using SparseBFA. Overall, our work reveals the vulnerability of sparse DNNs, which are typically generated by model compression techniques [8, 9] and popularly used for edge devices due to their smaller memory footprint and better energy-efficiency [10], to security threats exploiting bit flips.

2. RELATED WORK

Fault injection attacks can change the values stored in memory elements by exploiting hardware-level vulnerabilities. For example, RowHammer [11, 12] exploits inter-row interference in DRAMs, and can precisely alter bit cells by repetitively accessing neighboring rows of those target bit cells. Furthermore, if an attacker has physical access to the hardwares, other sources of injection can be used, such as a precise laser [13]. Note that these attacks can flip multiple bits in the same row of memory elements. Thus, conventional error correction codes cannot guarantee protection [12].

Several works studied the implications of fault injection attacks on DNNs. [1] showed that a single bit flip, especially at the most significant bit of exponents, leads to $\sim 99\%$ accuracy drop for dense floating-point DNNs, and demonstrated the viability of this bit flip attack using RowHammer. While linearly-quantized DNNs represented with integer weights might appear to be more robust to bit flips due to their smaller range of possible weight values, [2, 3] showed that a gradient-based search algorithm can successfully find $2 \sim 24$ bit flips that cause accuracy drop to the random-guess level.

3. ALGORITHM

3.1. Problem Definition and Notations

Our goal is to find the smallest number of bits in the coordinates of nonzero elements of a sparse DNN $f(\cdot)$ such that flipping those bits results in large degradation of performance. Suppose $f(\cdot)$ is parameterized with $\Theta = \{W_i\}_{i=1}^{L}$ (ignoring biases for simplicity), and its weight matrices/tensors are stored using sparse matrix formats, such as COO or CSC. Although we will be using COO when explaining our algorithm, note that different sparse matrix formats are interchangeable and the analysis we provide is applicable to the other formats without much modification. We denote the list of nonzero values for the ith weight matrix W_i as V_i, and the list of the coordinates corresponding to those nonzero values as C_i as in Fig. 1. The jth element in C_i (denoted as $C_i[j]$) represents a coordinate as a d-tuple of n_c-bit unsigned integers, where d is the dimension of W_i (e.g., 4 for 2D convolutional layers). Also, we denote the loss of a DNN given input samples $\mathbf{x}$ and the corresponding ground truth targets $\mathbf{t}$ as $l(f(\mathbf{x}; \Theta), \mathbf{t})$. This loss can be cross-entropy loss for the classification tasks, or multi-box loss [4] for the object detection tasks. Furthermore, we assume that $f(\cdot)$ is generated by iteratively pruning a dense DNN to have the sparsity of $r \in [0, 1]$ for each layer, and that nonlinear quantization is applied for the value of weights [9].

Algorithm 1 SparseBFA for the COO format

$m \leftarrow 0$
while $m < M$ **do** ▷ M: the maximum number of iterations
  loss_dict $\leftarrow \emptyset$
  for $i \in S_L$ **do**
    $C_i, V_i \leftarrow$ CONVERT_SPARSE(W_i)
    $S_j \leftarrow$ GET_CANDIDATES_INDEX(C_i)
    for $j \in S_j$ **do**
      for $k \leftarrow 0, d$ **do**
        for $n \leftarrow 0, n_c$ **do**
          $C_i^\star \leftarrow C_i$
          $C_i^\star[j][k][n] \leftarrow 1 - C_i^\star[j][k][n]$
          if IS_VALID($C_i^\star$) **then**
            $W_i^\star \leftarrow$ RECONSTRUCT($C_i^\star, V_i$)
            $\Theta^\star \leftarrow \Theta \setminus \{W_i\} \cup \{W_i^\star\}$
            loss $\leftarrow l(f(\mathbf{x}; \Theta^\star), \mathbf{t})$
            loss_dict$[(i, j, k, n)] \leftarrow$ loss
  $(i_m, j_m, k_m, n_m) \leftarrow$ GET_MAX_KEY(loss_dict)
  $C_{i_m}, V_{i_m} \leftarrow$ CONVERT_SPARSE(W_{i_m})
  $C_{i_m}[j_m][k_m][n_m] \leftarrow 1 - C_{i_m}[j_m][k_m][n_m]$
  $W_{i_m} \leftarrow$ RECONSTRUCT(C_{i_m}, V_{i_m})
  $m \leftarrow m + 1$

3.2. SparseBFA

We propose the SparseBFA algorithm, which iteratively searches for a bit in the coordinate lists of $f(\cdot)$ that results in the largest increase of the loss using a small number of samples $(\mathbf{x}, \mathbf{t})$. In each iteration, SparseBFA checks every possible valid bit flip for the selected elements in the coordinate lists, and evaluates the loss for each bit flip by forward-propagating the samples.

This idea is manifested in Algorithm 1. First, SparseBFA visits each layer whose index is in S_L, a set of indices of layers that are to be attacked (e.g., if all layers are considered, $S_L = \{1, \ldots, L\}$). After converting the weight matrix W_i to the coordinate list C_i and the value list V_i, the `Get_Candidates_Index` function returns S_j, the indices of the selected elements in C_i. Then, for each selected element $C_i[j]$, SparseBFA checks all $d \times n_c$ possible bit flips for their validity (i.e., whether the flipped coordinate list $C_i^\star$ satisfies the properties of the sparse matrix format used, such as no overlapping coordinates in $C_i^\star$), and evaluates the loss for valid bit flips. Finally, SparseBFA will choose the bit that resulted in the maximum loss once the search is finished.

The complexity of SparseBFA is determined by the `Get_Candidates_Index` function: for example, if this function returns every indices of C_i, then the algorithm becomes exhaustive search. Alternatively, the complexity can be reduced if this function proposes only a few candi-

dates by approximating importance of each element in C_i, although the flipped bit might not maximally increase the loss. We present both the exhaustive search method and the approximation method, and analyze their effectiveness and computational complexity in the following section.

3.2.1. *Exhaustive Search Method*

An important characteristic of our problem that distinguishes SparseBFA from the previous BFAs is that it is a combinatorial optimization problem, since the bit flips in the coordinate lists 'rewire' the connections between neurons. Thus, we cannot directly adopt the previous BFA algorithms, such as [2] developed for linearly-quantized dense DNNs, that used gradient ascent to determine the candidate bits. Instead, we first start with exhaustive search to solve our problem, which can find the optimal bit flip in each iteration.

We apply exhaustive search for a relatively small-sized layer, such as the first convolutional layer of DNNs, and find that even a single bit flip in the coordinate list can be detrimental. For example, exhaustive search can find a single bit in the coordinate list of the first convolutional layer of a ResNet50 [14] model, which is trained to classify the Tiny-ImageNet dataset with top-1 accuracy of 60%, that results in accuracy drop to almost random-guess level ($< 1\%$) when it is flipped.

However, the computational complexity of exhaustive search can be prohibitive for applying SparseBFA to the entire model. If W_i has the size of $[N_1, \ldots, N_d]$, then exhaustive search requires total $r \cdot (\prod_{k=1}^{d} N_k)(\sum_{k=1}^{d} \log(N_k))$ forward-propagations, since there are $r \cdot \prod_{k=1}^{d} N_k$ nonzero elements and the minimum number of bits n_c to represent a coordinate is proportional to $\log(N_k)$ for each dimension k. This motivates us to develop an approximation method that chooses S_j to be a small subset of indices such that the complexity can be reduced to $|S_j| \cdot \sum_{k=1}^{d} \log(N_k)$.

3.2.2. *Approximation Methods*

A simple approximation method can randomly sample S_j, so that different subsets of nonzero coordinates can be checked at each iteration (SparseBFA-RandomSubset). Furthermore, we investigate other approximation methods that utilize the characteristics of bit flips in the coordinate lists. For example, the previously nonzero weights corresponding to the flipped coordinates will be set to zero, and choosing S_j can be interpreted as choosing which nonzero weights will be 'pruned' away (although the values of those weights will be reintroduced elsewhere). Therefore, one approach can be estimating the importance of each nonzero weight similar as in weight pruning and choosing S_j as indices of the coordinates with high importance. We examine two importance metrics from pruning literature: 1) the magnitude of values $|w|$ [8] (SparseBFA-Magnitude), and 2) the first-order term in Taylor's expansion $|w \cdot \frac{\partial l}{\partial w}|$ [15] for each nonzero weight w (SparseBFA-Taylor). Finally, we also investigate using the gradients $|\frac{\partial l}{\partial w}|$ to estimate importance as in [2] (SparseBFA-Gradient).

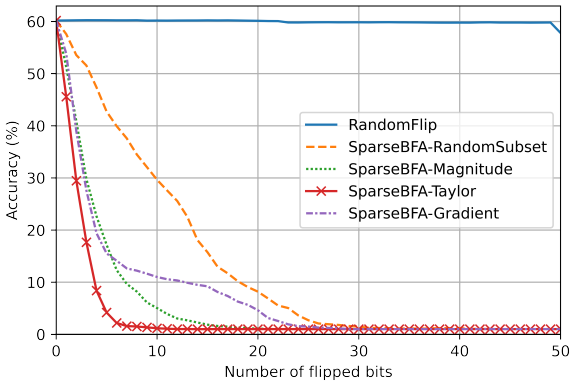


Fig. 2. Accuracy of the ResNet50 model as bits in the coordinate list are flipped using different approximation methods for SparseBFA. RandomFlip, which induces random bit flips similar to soft errors, is reported as a baseline. All experiments are repeated five times, and we report means.

Fig. 2 shows the effectiveness of four approximation methods. For the ResNet50 model we analyzed for the exhaustive search method, we measure how its accuracy changes as the bits in the coordinate lists are flipped using different approximation methods. For this experiment, we set $S_L = \{2, \ldots, L-1\}$, excluding the first and the last layer from the attack (otherwise a single bit flip at the first layer alone can degrade accuracy to the random-guess level), and set $|S_j| = 5$. We observe that accuracy of the ResNet50 model decreases to below 1% within $10 \sim 30$ bit flips for all four approximation methods, whereas completely random bit flips without SparseBFA do not largely affect accuracy. However, we find that SparseBFA-Taylor provides the best attack with the average of 11.8 bit flips to reach less than 1% accuracy.

4. RESULTS

4.1. Attacking DNNs with SparseBFA

We demonstrate that DNNs with different architectures and datasets are vulnerable to SparseBFA. Table 1 summarizes the properties of DNNs we examine (e.g., dataset, architecture, sparsity, and original accuracy) and the number of bit flips required to degrade the performance of each DNN to the target level when using SparseBFA-Taylor. We set $|S_j| = 5$, and use randomly sampled inputs $(\mathbf{x}, \mathbf{t})$ from the train dataset with a size of 128 for CIFAR10/TinyImageNet and 32 for PASCAL-VOC for SparseBFA.

First, we find that flipping only 0.00005% of total bits (1 in 2 million bits) can be detrimental to various DNNs. For

Table 1. Summary of the characteristics of DNNs and the number of bit flips found by SparseBFA-Taylor required to reach the target accuracy level. We report sparsity, total number of bits (when the model parameters are stored using COO, assuming that the values are nonlinearly-quantized using 4-bits for CIFAR-10/TinyImageNet and 6-bits for PASCAL-VOC), and the original accuracy of the models, along with the number of bit flips. All SparseBFA experiments are repeated 5 times with different random seeds, and we report means and standard deviations for the number of bit flips.

Dataset	Model	Sparsity	Total Bits	Original Accuracy	Target Accuracy	# Bits Flipped to Reach Target	
						All	Except First, Last
CIFAR-10 (Classification)	Conv4FC2 [1]	98.66%	14.26M	86.83%	$< 11\%$	1.4 ($\pm$0.55)	3.0 ($\pm$1.87)
	WideResNet [16]	94.37%	8.34M	93.14%		1.0 ($\pm$0)	12.8 ($\pm$2.17)
	MobileNetV2 [17]	94.37%	12.98M	90.06%		6.8 ($\pm$2.68)	8.8 ($\pm$2.17)
TinyImageNet (Classification)	ResNet50 [14]	94.37%	36.50M	60.18%	$< 1\%$ (Top-1)	3.2 ($\pm$4.38)	11.8 ($\pm$3.11)
	MobileNetV2 [17]	92.50%	17.73M	43.88%		1.4 ($\pm$0.89)	2.8 ($\pm$0.45)
PASCAL-VOC (Detection)	SSD300 [4]	82.20%	137.38M	0.70 mAP [2]	< 0.10mAP	34.2 ($\pm$3.42)	-

[1] 4 convolutional layers followed by 2 linear layers. Batch normalization is used for the convolutional layers.
[2] Note that the SSD300 model is not finetuned after quantization of the weight values. We report mean average precision (mAP) by randomly sampling 1000 images from PASCAL-VOC 2007 test dataset.

example, accuracy of the five DNNs for the image classification task decreases to the random-guess level with $1 \sim 15$ bit flips in all trials. Second, we observe that the first and the last layers are more sensitive to bit flips compared to intermediate layers. When the first and the last layers are not attacked, the number of bit flips to reach the target performance level increases by $1.3 \sim 12.8$ times compared to when all layers are attacked. Also, for the SSD300 model trained on PASCAL-VOC dataset, SparseBFA cannot reach the target performance level within 50 bit flips when excluding the first and the last layers from the attack. Thus, a simple scheme to improve robustness of sparse DNNs can be protecting at least the first and the last layers with security measures, such as more frequent refresh of DRAM rows to prevent RowHammer [12], or using a secure processor [18] during inference for those layers.

4.2. Effect of Sparsity

We examine the relationship between sparsity and the vulnerability of DNNs to SparseBFA. Fig. 3 shows how accuracy of the Conv4FC2 models with different sparsity changes as the number of bit flips increases. SparseBFA-Taylor is applied to all layers except the first and the last layers with $|S_j| = 5$. We find that the models with higher sparsity are more vulnerable to bit flips: accuracy of the model with 99% sparsity drops to below 11% with fewer than 5 bit flips, whereas the model with 76% sparsity still retains 55% accuracy even after 50 bit flips. One explanation for this trend is that there are fewer bit flips satisfying the `Is_Valid` function for denser models, thus reducing search space. Overall, this relationship manifests the trade-off between efficiency achieved with high sparsity and vulnerability to SparseBFA.

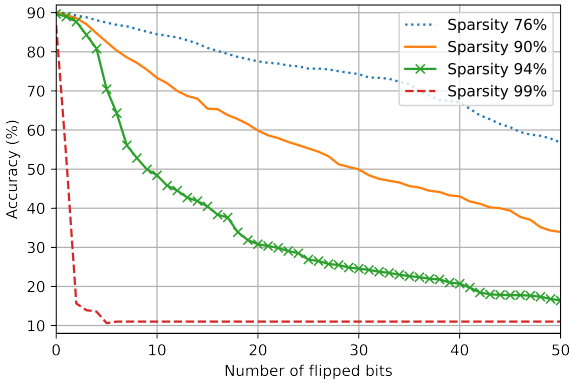


Fig. 3. Accuracy of the Conv4FC2 models trained on CIFAR-10 with different sparsity as the number of bit flips found by SparseBFA-Taylor increases. We report mean of five trials.

5. CONCLUSION

In summary, we propose SparseBFA that searches for a small number of bits in the coordinate lists to maximally increase the loss of sparse DNNs, and show the vulnerability of various sparse DNNs to this novel bit flip attack. This motivates future work on both hardware- and software-based protections against BFAs that will benefit the security of intelligent systems with DNNs.

Acknowledgement We thank Edith Beigne, Wojtek Powiertowski, Sudhir Satpathy, Tony Wu, and Huichu Liu for their helpful discussion.

6. REFERENCES

[1] Sanghyun Hong, Pietro Frigo, Yiğitcan Kaya, Cristiano Giuffrida, and Tudor Dumitraş, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *Proceedings of the 28th USENIX Conference on Security Symposium*, USA, 2019, SEC'19, p. 497–514, USENIX Association.

[2] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan, "Bit-flip attack: Crushing neural network with progressive bit search," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220.

[3] Fan Yao, Adnan Siraj Rakin, and Deliang Fan, "Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips," in *29th USENIX Security Symposium (USENIX Security 20)*. Aug. 2020, pp. 1463–1480, USENIX Association.

[4] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[5] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[6] Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[7] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[8] Song Han, Jeff Pool, John Tran, and William J. Dally, "Learning both weights and connections for efficient neural networks," 2015.

[9] Song Han, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2016.

[10] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2017, ISCA '17, p. 27–40, Association for Computing Machinery.

[11] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[12] Onur Mutlu and Jeremie S. Kim, "Rowhammer: A retrospective," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020.

[13] Xiaolu Hou, Jakub Breier, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu, "Physical security of deep learning on edge devices: Comprehensive evaluation of fault injection attack vectors," *Microelectronics Reliability*, vol. 120, pp. 114116, 2021.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[15] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[16] Sergey Zagoruyko and Nikos Komodakis, "Wide residual networks," 2016.

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[18] Shay Gueron, "A memory encryption engine suitable for general purpose processors," Cryptology ePrint Archive, Report 2016/204, 2016, https://ia.cr/2016/204.