Smart Redundancy Schemes for ANNs against Fault Attacks

Köylü, Troya Çağıl ; Hamdioui, Said; Taouil, Mottaqiallah

DOI
10.1109/ETS54262.2022.9810380

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 2022 IEEE European Test Symposium (ETS)

Smart Redundancy Schemes for ANNs Against Fault Attacks

Troya Çağıl Köylü, Said Hamdioui, Mottaqiallah Taouil
Computer Engineering Group
Delft University of Technology
Delft, the Netherlands
{T.C.Koylu, S.Hamdioui, M.Taouil}@tudelft.nl

2022 IEEE European Test Symposium (ETS) | 978-1-6654-6706-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ETS54262.2022.9810380

***Abstract*—Artificial neural networks (ANNs) are used to accomplish a variety of tasks, including safety critical ones. Hence, it is important to protect them against faults that can influence decisions during operation. In this paper, we propose smart and low-cost redundancy schemes that protect the most vulnerable ANN parts against fault attacks. Experimental results show that the two proposed smart schemes perform similarly to dual modular redundancy (DMR) at a much lower cost, generally improve on the state of the art, and reach protection levels in the range of 93% to 99%.**

***Index Terms*—artificial neural network, redundancy, fault injection, countermeasure, machine learning**

I. Introduction

After the presentation of deep architectures, artificial neural networks (ANNs) made groundbreaking achievements in visual tasks such as object detection [1]. This ability to distinguish between objects is key in several safety-critical applications such as autonomous driving. One major threat against ANNs are fault injection attacks, as they can alter ANN operation and hence affect their outcome. Many fault injection techniques have been proposed such as voltage underfeeding, EM glitching, and Rowhammer [2]. It is important to protect ANNs against such fault attacks.

A number of solutions have been presented for the protection of ANNs against faults. A group of countermeasures use inherent mechanisms of ANNs to protect against faults [3]: some networks are more fault resilient [4] and some are trained in a fault-aware method [5]. As it is hard to protect vulnerable parts of an ANN with these methods, many other solutions relied on applying redundancy instead. These include replicating last hidden [6] and output layers [7], and when these fail, applying DMR or triple modular redundancy (TMR) on the whole network. The latter two, while effective, are very costly. Thus, it is important to look for effective redundancy schemes that limit the cost. Such a method was introduced in [8], but it is based on a naïve method, i.e., the weights of neurons.

In this paper, we propose two redundancy schemes for neural networks, which use the more informative gradient descent algorithm to determine the vulnerable parts (layer, neuron, or weight). This is a more efficient way of protecting the ANN than DMR.

The remainder of this paper is organized as follows. Section II describes the threat model, concept, methodology, and implementation. Section III presents the performed experiments and results. Finally, Section IV concludes the paper.

II. Methodology

We assume an attacker that wants to achieve misclassifications by injecting faults into the weights and biases [9]. Hence, our aim is to identify and protect the most vulnerable (or equally, impactful) elements of the ANN. Our method, *gradient descent-based impact analysis*, depends on the second additive operand of $\hat{w} = w + \alpha \frac{\partial(t-o)}{\partial w}$, i.e., the weight update term. Here w represents a weight value, α the learning rate, and $t - o$ the difference between expected and obtained output.

We suggest to process the updates per weight over a subset of images after training is completed. To elaborate, we assign the impact of a weight based on (1) the absolute summation of the updates (i.e., summation scheme), or (2) the variance of the updates (i.e., variance scheme). The first scheme assumes that the most impactful weight is the one that requires the largest update, while the second scheme assumes that it is the one with the most variation (i.e., has conflicting update values for different inputs). Furthermore, when these updates are calculated for each weight, it is also possible to assign the impact/vulnerability of each neuron (summation of the impact of all weights and biases connected to this neuron) or layer (summation of the impact of all neurons in this layer). This enables the selective protection of an ANN at the desired granularity (layer, neuron, or weight).

Finally, our method can both be deployed for software-based ANN applications and hardware accelerators. In software, the selected elements for protection (i.e., weights, neurons, or layers) should be repeated in time. In a hardware ANN accelerator, the determined weights, neurons, or entire layers should be duplicated. Consequently, the overhead of our method equals the ratio of applied redundancy rat_{red}.

III. Experimental Results

In this section, we present the experimentation to illustrate the comparative effectiveness and efficiency of our two proposed schemes.

ANNs and dataset: In this work, we use three very widely used convolutional neural networks for visual recognition tasks: AlexNet [1], VGG (CNN S) [10], and GoogleNet [11]; all are trained on the ImageNet 2012 challenge [12]. As dataset, we created two sets from the validation repository of this challenge. The first is called the *calibration set* that consists of images 1 to 1000 from the repository and are used to determine which elements of the ANN to protect. The second *verification set* consists of images 1001 to 2000 and is used to verify the effectiveness and efficiency of the two schemes. We conduct all our experiments using the Pycaffe toolbox [13].

Fault injection framework: We inject bit and byte faults to the weights and biases in a network. In each run, we inject 1, 5, or 10 faults to random weights or biases.

Fault detection and coverage: When a fault affects a protected element of the ANN, it is detected. To evaluate how effective a redundancy scheme is, we also evaluate the top5 and top1 undetected misclassifications arising from faults, i.e., when a fault makes a correct top5 or top1 decision faulty.

Comparison: We make comparisons with two state-of-the-art redundancy-based approaches. The first approach in [6] duplicates the neurons in the last hidden layer and divides the values of weights that leave these neurons by half. The second approach determines the importance of a neuron based on the absolute summation of the weights that leave this neuron [8]. On top of this variant (denoted as [8]-1), it further proposes a second variant where the weight values are adjusted based on the neuron importance they are connected to (denoted as [8]-2).

Table I presents the undetected top5/top1 misclassifications using redundancy at neuron level for $rat_{red} = 0.3$ when 1, 5, and 10 faults are injected; each cell presents the results for 1000 images. The red cells indicate the cases that state of the art outperforms both our schemes. The method in [6] has a 37 - 39% overhead. To further understand the effect of not protecting the output layer in [8], we also include cases where faults only target the last layer (denoted by l.l.).

From Table I, it is clear that both our summation and variance schemes outperform [6] significantly. The increase in performance becomes more apparent as the number of injected faults increases. Note also that we use less redundant neurons to obtain a better performance. The improvement is less significant when compared to [8]-1 and [8]-2, but very significant for the variance scheme when faults are injected to the last layer. Lastly, not shown here for brevity, but for many cases, our schemes perform similarly to DMR (with 0 undetected misclassifications) and significantly outperform randomly applied redundancy at the same cost.

TABLE I: Undetected top5/top1 Misclassifications

	rat_{red}	0.37 - 0.39	0.3			
network	**#faults**	[6]	[8]-1	[8]-2	sum.	var.
AlexNet	1	23/19	13/13	8/5	7/5	3/3
	1 (l.l.)	26/18	16/10	15/14	20/18	16/15
	5	113/88	0/1	0/0	0/0	0/0
	5 (l.l.)	44/36	33/26	33/25	36/26	8/5
	10	213/155	0/0	0/0	0/0	0/0
	10 (l.l.)	36/34	32/28	30/24	39/29	2/1
VGG	1	37/31	4/3	7/7	11/11	7/7
	1 (l.l.)	19/17	16/14	26/27	17/16	9/5
	5	121/88	0/0	0/0	0/0	0/0
	5 (l.l.)	36/34	29/22	33/25	36/35	2/2
	10	220/163	0/0	0/0	0/0	0/0
	10 (l.l.)	43/35	35/34	33/30	31/30	0/0
GoogleNet	1		24/21	17/15	21/20	25/20
	1 (l.l.)		22/20	24/25	35/30	25/24
	5		16/15	7/6	22/17	22/21
	5 (l.l.)		55/51	57/50	41/37	8/6
	10		6/4	0/0	13/10	4/2
	10 (l.l.)		47/38	50/49	52/51	2/2

IV. Conclusion

In this paper we presented two novel schemes based on the gradient descent algorithm for protecting ANNs against fault attacks. The method can be applied to all ANNs, without costly retraining. The experimental results on three different networks show that our method is effective and efficient.

V. Acknowledgment

This work was labelled by the EUREKA cluster PENTA and funded by Dutch authorities under grant agreement PENTA-2018e-17004-SunRISE.

References

[1] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *CACM*, 2017.
[2] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *SIGARCH*, 2014.
[3] T. Ç. Köylü *et al.*, "Deterministic and statistical strategies to protect anns against fault injection attacks," in *PST*. IEEE, 2021.
[4] G. Bolt, "Investigating fault tolerance in artificial neural networks," 1991.
[5] T. Ito *et al.*, "On fault injection approaches for fault tolerance of feedforward neural networks," in *ATS*. IEEE, 1997.
[6] M. D. Emmerson *et al.*, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE transactions on neural networks*, 1993.
[7] D. A. Medler *et al.*, "Training redundant artificial neural networks: Imposing biology on technology," *Psychological Research*, 1994.
[8] Y. Li *et al.*, "D2nn: a fine-grained dual modular redundancy framework for deep neural networks," in *ACSAC*, 2019.
[9] Y. Liu *et al.*, "Fault injection attack on deep neural network," in *ICCAD*. IEEE, 2017.
[10] K. Chatfield *et al.*, "Return of the devil in the details: Delving deep into convolutional nets," *arXiv*, 2014.
[11] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, 2015.
[12] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, 2015.
[13] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *arXiv*, 2014.