Practical Fault Attack on Deep Neural Networks

Jakub Breier
Nanyang Technological University
jbreier@ntu.edu.sg

Xiaolu Hou
Nanyang Technological University
ho0001lu@e.ntu.edu.sg

Dirmanto Jap
Nanyang Technological University
djap@ntu.edu.sg

Lei Ma
Harbin Institute of Technology
malei@hit.edu.cn

Shivam Bhasin
Nanyang Technological University
sbhasin@ntu.edu.sg

Yang Liu
Nanyang Technological University
yangliu@ntu.edu.sg

ABSTRACT

As deep learning systems are widely adopted in safety- and security-critical applications, such as autonomous vehicles, banking systems, etc., malicious faults and attacks become a tremendous concern, which potentially could lead to catastrophic consequences. In this paper, we initiate the first study of leveraging physical fault injection attacks on Deep Neural Networks (DNNs), by using laser injection technique on embedded systems. In particular, our exploratory study targets four widely used activation functions in DNNs development, that are the general main building block of DNNs that creates non-linear behaviors – ReLu, softmax, sigmoid, and tanh. Our results show that by targeting these functions, it is possible to achieve a misclassification by injecting faults into the hidden layer of the network. Such result can have practical implications for real-world applications, where faults can be introduced by simpler means (such as altering the supply voltage).

KEYWORDS

deep learning security, fault attacks, adversarial attacks

1 INTRODUCTION

Internet of things (IoT) and artificial intelligence (AI) are the two integral components of modern paradigms like smart city, self-driving cars etc. The most efficient AI is known in the form of deep learning which has achieved great leaps of progress in many application domains.In parallel, IoT has pushed the computing elements (and sensors) outside traditional boundaries and motivates to place them everywhere. This has also enabled easy physical access to the computing elements which was not possible previously.

Deep learning is the family of neural networks composed of an input layer, three or more hidden layers and an output layer. Based on the internal structure, several candidates exist like multi-layer perceptron (MLP), convolutional neural networks (CNN), recurrent neural network (RNN) etc. These are popularly known as deep neural networks (DNN). While each of these architectures has unique functions, we focus on activation functions which remain common across architectures and are an important part of the algorithm to obtain non-linear behaviors [8]. These commonly used activation functions are: `softmax`, `ReLu`, `sigmoid` and `tanh`. Studying these functions under fault attacks allows to derive general conclusions on susceptibility of deep learning to fault attacks.

We implemented the most common activation functions used across DNNs on a low-cost microcontroller (often used in IoT). Next, we performed practical laser fault injection using a near-infrared diode pulse laser to inject faults during the processing of activation function. The use of laser facilitates a strong attacker model with extensive fault injection capabilities. With the models, derived from practical fault injection, we analyze the susceptibility of DNN against such attacks. The primary goal of the performed attacks is to achieve misclassification during the testing phase. In the hindsight, the achieved misclassification can jeopardize the functioning of DNN-based paradigms like smart city.

Extensive studies have been performed on adversarial attacks, that crafts the input data with little perturbation to fool deep learning systems [9]. To the best of our knowledge, our study is the first work to explore practical fault injection on deep neural network, where we focus on attacking the DNNs itself instead of creating input data to fool DNNs like adversarial attack does.

Fault injection attacks are a popular physical attack vector used against cryptographic circuits. By changing intermediate values during the cryptographic algorithm execution, they can efficiently provide information on secret values, helping to recover the secret key in just a few encryptions [4]. Normally, the secret key recovery would require infeasible amount of computing time. Similarly, these attacks can be used against verification circuits, such as `PIN` verification on a smartcard, where a comparison function can be skipped and grant access to a malicious user [7].

Up to date, to the best of our knowledge, only [10] describes fault injection attack on neural networks. In their paper, they only provide a white box attack on deep neural network through software simulation, while observing the changes in the output after introducing faults in the network's values. However, they do not provide insight on practicality of such attack. Whether such attacks could also be applied physically remained an open problem. Therefore, in our paper, we experimentally show what types of faults are achievable in practice and we further use this information to develop a realistic attack on DNNs.

2 PRACTICAL DNN ATTACK ANALYSIS

2.1 Attack Equipment Setup

The main component of the experimental laser fault injection station is the diode pulse laser. It has a wavelength of 1064 nm and pulse power of 20 W. This power is further reduced to 8 W by a 20× objective lens which reduces the spot size to 15×3.5 μm^2.

CCS '18, October 15–19, 2018, Toronto, ON, Canada

Table 1: Relation between correct output y and faulted output y' when a single fault is injected in target activation function

Target activation function	Relation between y and y'
ReLu: $y = max(0, x)$	$y' = 0$
sigmoid: $y = \frac{1}{1+e^{-x}}$	$y' = 1 - y$
tanh: $y = \frac{2}{1+e^{-2x}} - 1$	$y' = -y$

As the device under test (DUT), we used `ATmega328P` microcontroller, mounted on `Arduino UNO` development board. The package of this chip was opened so that there is a direct visibility on a back-side silicon die with a laser. The board was placed on an `XYZ` positioning table with the step precision of 0.05 μm in each direction. A trigger signal was sent from the device at the beginning of the computation so that the injection time could be precisely determined. After the trigger signal was captured by the trigger and control device, a specified delay was inserted before laser activation. Laser activation timing was also checked by a digital oscilloscope for a greater precision.

The chip area is 3×3 mm^2, while the area sensitive to laser is ≈ 50×70 μm^2. With a laser power of 4.5% we were able to disturb the algorithm execution, when tested with reference codes. By using some pre-tested benchmarking software, we determined that a laser power of 4.5% was enough to disturb the algorithm execution.

2.2 DNN Activation Function Fault Analysis

To evaluate different activation functions, we implemented three simple 3-layer neural networks with sigmoid, ReLu and tanh as the activation function for the second layer respectively. The activation function for the last layer was set to be softmax. The neural networks were implemented in C programming language, which were further compiled to AVR assembly and uploaded to the DUT.

As instruction skip/change are one of the most basic attacks on microcontrollers, with high repeatability rates [5], we aimed at this fault model in our experiments. The microcontroller clock is 16 MHz, one instruction takes 62.5 ns. Some of the activation functions took over 2000 instructions to execute. To check what are the vulnerabilities of the implementations, we have carefully varied the timing of the laser glitch from the beginning until the end of the function execution so that every instruction would be eventually targeted. We used a single fault adversarial model – exactly one fault was injected during one activation function execution.

After we observed a successful misclassification, we determined the vulnerable instructions by visual inspection of the compiled assembly code and by checking the timing of the laser in that particular fault injection instance.

In this exploratory study, we implemented a random neural network, consisting of 3 layers, with 19, 12, and 10 neurons in input layer, hidden layer, and output layer, respectively. Our fault attack was always targeting the computation of one of the activation functions in hidden layer. In the following, we will explain the experimental results on different activation functions in detail. A overview of the achieved results is shown in Table 1. It reports, for an input x, the relation between y (correct) and y' (faulted) output of different activation functions.

ReLu was implemented by a following code in C:

```
if (Accum > 0) {HiddenLayerOutput[i] = Accum;}
else {HiddenLayerOutput[i] = 0;}
```

where i loops from 1 to 12 so that each loop gives one output of the hidden layer. `Accum` is an intermediate variable that stores the input of activation function for each neuron.

The assembly code inspection showed that the result of successful attack was executing the statement after else such that the output would always be 0. The corresponding assembly code is:

```
1         ldi r1, 0       ;load 0 to r1
2         cp r1, r15      ;compare MSB of Accum to r1
3         brge else       ;jump to else if 0 >= Accum
4         movw r10, r15   ;HiddenLayerOutput[i] = Accum
5         movw r12, r17   ;HiddenLayerOutput[i] = Accum
6         jmp end         ;jump after the else statement
7   else: clr r10         ;HiddenLayerOutput[i]= 0
8         clr r11         ;HiddenLayerOutput[i]= 0
9         clr r12         ;HiddenLayerOutput[i]= 0
10        clr r13         ;HiddenLayerOutput[i]= 0
11  end:  ...             ;continue the execution
```

where each float number is stored in 4 registers. For example, `Accum` is stored in registers `r15,r16,r17,r18` and `HiddenLayerOutput[i]` is stored in `r10,r11,r12,r13`. Line 4,5 executes the equation `HiddenLayerOutput[i] = Accum`.

The attack was skipping the "`jmp end`" instruction that would normally avoid the part of code setting `HiddenLayerOutput[i]` to 0 in case `Accum` > 0. Therefore, such change in control flow renders the neuron inactive no matter what is the input value.

Sigmoid was implemented by a following C code:

```
HiddenLayerOutput[i] = 1.0/(1.0 + exp(-Accum));
```

After the assembly code inspection, we observed that the successful attack was taking advantage of skipping the negation in the exponent of `exp()` function, which compiles into one of the two following codes, depending on the compiler version:

```
A)  neg r16          ;compute negation r16
B)  ldi r15, 0x80    ;load 0x80 into r15
    eor r16, r15     ;xor r16 with r15
```

Laser experiments showed that both `neg` and `eor` could be skipped, and therefore, significant change to the function output was achieved.

Hyperbolic tangent was implemented by a following code in C:

```
HiddenLayerOutput[i] = 2.0/(1.0 + exp(-2*Accum)) - 1;
```

Similarly to sigmoid, the experiments showed that the successful attack was exploiting the negation in the exponential function, leading to an impact similar to sigmoid.

Softmax. In case of softmax function, we were unable to obtain any successful misclassification. There were only two different outcomes as a result of the fault injection: either there was no output at all, or the output contained invalid values. This lack of valid output prevented us to do further fault analysis to derive the actual fault model that happened in the device. Therefore, a thorough analysis of softmax behavior under faults would be an interesting topic for the future work. Another line of future work would be to analyze bit flip attacks on IEEE 754 floating point representation that is used for storing the weights. The representation follows 32-bit pattern ($b_{31}...b_0$): 1 sign bit (b_{31}), 8 exponent bits ($b_{30}...b_{23}$) and 23 mantissa (fractional) bits ($b_{22}...b_0$). The represented number is given by $(-1)^{b_{31}} \times 2^{(b_{30}...b_{23})_2 - 127} \times (1.b_{22}...b_0)_2$. A bit flip attack on the sign bit or on the exponent bits would make significant influence on the weight.

Table 2: Structure of DNNs used in fault evaluations.

Layer	No. of neurons	Activation function
Input layer	784	-
Hidden layer 1	500	ReLu
Hidden layer 2	500	ReLu
Hidden layer 3	500	ReLu
Hidden layer 4	n	target function
Output layer	10	Softmax

3 APPLICATION TO DNN

The results from previous section aiming at single functions can be directly used to alter the behavior of a neural network. In this section we extend the attack to a full network, while targeting several function computations at once with a multi-fault injection model. There are three possible places to introduce a fault in DNN:

- Input layer – such fault would be identical to introducing a change at the input data. Therefore, it is of little interest, since it would be normally easier for the attacker to directly alter the input data.
- Hidden layer(s) – since the structure of the hidden layer is normally unknown to the attacker, she cannot easily predict the outcome of the fault injection. However, she can still achieve the misclassification, although not necessarily to the class she decides. Therefore, such attack might be interesting in case the attacker does not care about the outcome class as long as it is missclassified.
- Output layer – normally, softmax is the function of choice for the output layer. According to our results, introducing a meaningful fault into softmax is harder compared to other functions.

One might be wondering how can the attacker develop a strategy if the network is unknown to her. As shown recently [3], it is possible to determine the activation function with a side-channel analysis, i.e. by measuring the power consumption or the electromagnetic emanation from the device during the computation. Getting the weights and deciding on attacking particular neurons might be trickier, so in this case, a random fault model would be the most reasonable assumption for the attacker. Deciding on what layer to attack, it makes sense to inject the fault as close to the output layer as possible to make the impact highest. Therefore, for our case, the attacker randomly injects faults into the 4^{th} hidden layer of the network, targeting multiple activation function computations. In the following we provide evaluation results for such case.

Evaluation of DNN. To test how our attack can influence a real-world DNN, we trained and evaluated different DNNs with the random fault model described above. The attack methods considered are described in Section 2.2. We have selected a popular MNIST dataset. The training of DNNs was accomplished using Keras (ver.2.1.6) [6] and Tensorflow libraries (ver.1.8.0) [1]. The structures of the DNNs are detailed in Table 2. For each target function (ReLu, sigmoid and tanh), 8 DNNs with different number of neurons (n) in hidden layer 4 were evaluated. We used a partially fixed structure of DNN in order to study the effects of fault attacks on different activation functions. The prediction accuracy (see Table 3) of the tested network is comparable to state of the art.

Figures 1 (a) and 1 (b) show a random fault model when attacking the last hidden layer of the network with 5 and 15 faults, respectively. Success rates are calculated for 800 random inputs. Naturally, with increasing number of neurons in the layer, the success rate

Table 3: Training/testing accuracy of DNNs used in evaluation.

Target function	ReLu							
n	15	20	30	40	50	60	70	80
Train. Acc.	97.8	98.9	99.5	99.1	99.2	99.4	99.2	99.1
Test. Acc.	96.6	98.0	98.3	97.6	97.4	98.1	98.1	97.6
Target function	sigmoid							
n	15	20	30	40	50	60	70	80
Train. Acc.	98.9	99.2	98.9	99.2	99.1	99.2	99.0	99.2
Test. Acc.	97.8	97.7	97.7	98.0	98.0	98.1	97.8	98.0
Target function	tanh							
n	15	20	30	40	50	60	70	80
Train. Acc.	98.6	99.2	99.1	99.0	99.0	99.1	99.2	99.0
Test. Acc.	97.4	97.9	97.8	97.8	98.0	97.8	97.9	97.8

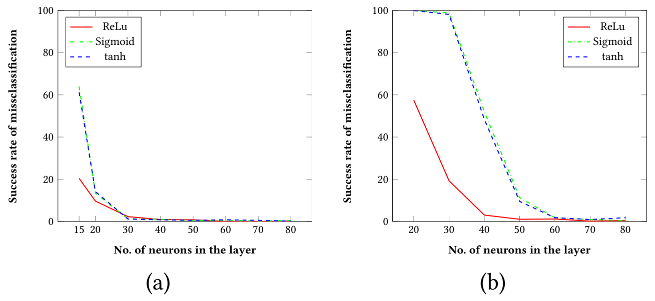

Figure 1: Injecting (a) 5 and (b) 15 random faults in hidden layer 4.

drops. The figures also show that sigmoid and tanh functions follow the same trend, which is caused by the same type of fault as explained in the previous section – skipping the negation in the exponentiation function.

Overall, it can be concluded that if the attacker wants to have a reasonable success rate (>50%), she should inject faults in at least half of the neurons in the chosen layer, in case of sigmoid and tanh. For ReLu, she should fault at least 3/4 of the neurons.

4 CONCLUSION AND FUTURE WORK

While the laser fault injection, used in our experiments, might not be possible in many scenarios, there are much simpler ways to disturb the circuits, such as voltage/clock glitching or electromagnetic fault injection [2]. In the future, we will also explore such applications.

It will also be interesting to look at possible countermeasures. While there are already techniques available that correct non-malicious alterations of the processed values in DNN (due to environmental conditions), the fault tolerance techniques against malicious entities have to be developed in the same way as in the area of applied cryptography.

REFERENCES

[1] M. Abadi et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.
[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. 2006. The sorcerer's apprentice guide to fault attacks. *Proc. IEEE* 94, 2 (2006), 370–382.
[3] L. Batina, S. Bhasin, D. Jap, and S. Picek. 2018. CSI Neural Network: Using Side-channels to Recover Your Artificial Neural Network Information. Cryptology ePrint Archive, Report 2018/477. (2018).
[4] E. Biham and A. Shamir. 1997. Differential fault analysis of secret key cryptosystems. In *Annual international cryptology conference*. Springer, 513–525.
[5] J. Breier, D. Jap, and C.-N. Chen. 2015. Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on AES. In *CPSS 2015*. ACM, 99–103.
[6] F. Chollet et al. 2015. Keras. (2015).
[7] C. Giraud and H. Thiebeauld. 2004. A survey on fault attacks. In *Smart Card Research and Advanced Applications VI*. Springer, 159–176.

[8] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press.

[9] I. J Goodfellow, J. Shlens, and C. Szegedy. 2015. Explaining and harnessing adversarial examples. *ICLR* (2015).

[10] Y. Liu, L. Wei, B. Luo, and Q. Xu. 2017. Fault injection attack on deep neural network. In *ICCD 2017*. IEEE Press, 131–138.