Security Evaluation of Deep Neural Network Resistance Against Laser Fault Injection

Xiaolu Hou[1], Jakub Breier[1], Dirmanto Jap[1], Lei Ma[2], Shivam Bhasin[1] and Yang Liu[1]

[1]Nanyang Technological University, Singapore
[2] Kyushu University, Japan
Email: ho0001lu@e.ntu.edu.sg, jbreier@jbreier.com, djap@ntu.edu.sg,
malei@ait.kyushu-u.ac.jp, sbhasin@ntu.edu.sg, yangliu@ntu.edu.sg

***Abstract*—Deep learning is becoming a basis of decision making systems in many application domains, such as autonomous vehicles, health systems, etc., where the risk of misclassification can lead to serious consequences. It is necessary to know to which extent are Deep Neural Networks (DNNs) robust against various types of adversarial conditions.**

In this paper, we experimentally evaluate DNNs implemented in embedded device by using laser fault injection, a physical attack technique that is mostly used in security and reliability communities to test robustness of various systems. We show practical results on four activation functions, ReLu, softmax, sigmoid, and tanh. Our results point out the misclassification possibilities for DNNs achieved by injecting faults into the hidden layers of the network. We evaluate DNNs by using several different attack strategies to show which are the most efficient in terms of misclassification success rates. Outcomes of this work should be taken into account when deploying devices running DNNs in environments where malicious attacker could tamper with the environmental parameters that would bring the device into unstable conditions, resulting into faults.

***Index Terms*—fault attack, neural networks, deep learning**

I. Introduction

Deep learning is a 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 (CNNs), recurrent neural networks (RNNs) etc. These are popularly known as deep neural networks (DNN). While each of these architectures has unique applications, activation functions remain common across architectures and are an important part of the algorithm to obtain non-linear behaviors [1]. 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.

In this work, we focus on a class of physical attacks known as fault attacks, which have become a reality owing to decreasing price and expertise required to mount such attack. Fault attacks are active attacks on a given implementation which try to perturb the internal software/hardware computations by external means. The adversary uses methods like voltage glitches, electromagnetic pulses, or laser injection to introduce perturbations for various purposes, ranging from erroneous computation, denial of service etc. Such attacks are commonly used for mounting secret key recovery attacks in cryptography or for violating/bypassing security checks [2].

In this paper, we extend the scope of the work in [3]. We first study practical laser fault injection on critical components of DNN running on an embedded device. Later, we experimentally evaluate the impact of identified fault models on big networks.

To study the impact of faults on DNN, we implemented the most common activation functions used across DNNs on a low-cost microcontroller. 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/deployment phase. Our results indicate that in some cases, 30% of faulty neurons in the last hidden layer can already present a high risk of misclassification ($\approx 82\%$). In the hindsight, the achieved misclassification can jeopardize the functioning of DNN-based paradigms like autonomous driving, smart city etc..

A. Related Works

Fault injection attacks are popular physical attacks, used against cryptographic circuits [4]. Recently, fault injection attacks have been applied to create misclassification in neural networks, leading to accuracy degradation of the trained network. For example, Liu et al. [5] proposed fault injection attack on DNN in a white box model through software simulation, while observing the changes in the output after introducing faults in the network's values. Hong et al. [6] explore the impact of (simulated) single bit flip model on accuracy loss of DNN. Zhao et al. [7] study method to inject stealthy faults for selected inputs while keeping overall accuracy unchanged. These works have been focusing only on simulation. In contrast, our previous work, [3] present the first practical fault injection on DNN using high power laser pulses, targeting the activation function. Another practical attack was proposed by Alam et al. [8], which exploit FPGA features to create remote faults in a DNN, running on an FPGA fabric. The difference is that [3] uses laser based fault

injection on embedded systems with precise control on timing and operation while [8] proposes remote but rather difficult to control timing and fault location.

II. Background

In this section, we recall basic concepts of deep neural networks and activation functions.

A. Deep Neural Networks

Artificial neural networks (ANNs) are computing units designed on basis of biological neural networks. ANN is a network of interconnected nodes or neurons where a signal is transmitted from input neurons towards output neurons. Arranged in layers, each neuron computes an output based on sum of (weighted) inputs from other neurons, followed by a non-linear function. The weights are determined during the training process. The non-linear layer function, also known as the activation function, is what gives an ANN its power to learn and classify difficult problems. A simple ANN can be composed of an input layer, one hidden layer and an output layer. To train the network, the backpropagation algorithm is used, which is a generalization of the least mean squares algorithm in the linear perceptron. Backpropagation is used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function [9].

Deep neural networks (DNNs) are fairly new variants of ANNs with three or more hidden layers. DNNs have become realistic with the latest advances in computing power, thanks to high performance graphical processing units (GPU). Several variants of DNN exist, including multi-layer perceptron (MLP), convolutional neural networks (CNNs), recurrent neural networks (RNNs), etc. Owing to the deep architecture, they have shown great success across domains – the most prominent being image classification, with the biggest ones composed of as many as 152 layers (Resnet [10]).

As it was pointed out in [11], in case of large neural networks, there are many nodes that do not contribute to the neural network function. However, there are some nodes which are crucial for correct functionality and if these are faulted, it can result in a failure.

B. Activation Functions

The activation functions we consider are the following: `softmax`, `ReLu`, `sigmoid` and `Tanh` [1].

Softmax is normally used as the activation function for output layer. It takes a vector $\boldsymbol{\alpha}$ as input, ith entry of the output gives the probability of a given input belonging to class i:

$$\text{softmax}(\boldsymbol{\alpha})_i = \frac{\exp(\alpha_i)}{\sum_j \exp(\alpha_j)}, \tag{1}$$

where exp is the exponentiation function with base e.

In modern neural networks, the default recommendation for activation function is the rectified linear unit or ReLu defined as follows:

$$\text{ReLu}(\alpha) = \max\{0, \alpha\}. \tag{2}$$

It is a piecewise linear function which preserves properties that make the optimization of linear model easy.

Before the introduction of ReLu, commonly used activation functions are logistic sigmoid activation function

$$\text{sigmoid}(\alpha) = \frac{1}{1 + \exp(-\alpha)}, \tag{3}$$

and hyperbolic tangent function

$$\tanh(\alpha) = \frac{2}{1 + \exp(-2\alpha)} - 1. \tag{4}$$

The sigmoid function is normally used to introduce non-linearity in the model. A reason for its popularity comes from the simple equation between its derivative and itself

$$\text{sigmoid}'(\alpha) = \text{sigmoid}(\alpha)(1 - \text{sigmoid}(\alpha)).$$

However, sigmoid functions becomes insensitive to inputs with large absolute values. In such cases, the hyperbolic tangent activation function is used as an alternative.

C. Difference from Adversarial Learning

A huge amount of research is undergoing towards adversarial learning [12], [13]. It basically involves constructing special inputs which are capable of confusing the machine learning models, often leading to output misclassification. In this work, we explore an alternate avenue to arrive at the same. The proposed fault attacks target the implementation of the DNN, particularly the critical activation function to achieve misclassification without any perturbation of the input. Depending on the application scenario and adversary model, one attack might be more suited than the other.

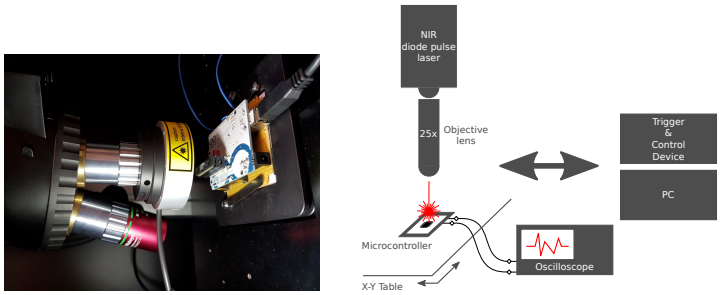
III. Practical DNN attack feasibility analysis

In this part we first show the practical laser fault attack setup in Section III-A. In Section III-B we show the possible fault attacks on activation functions that we have discovered with practical experiments. In Section 4, those attacks will be used for simulating missclassification attacks on MNIST DNNs.

A. 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.

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. Our setup is depicted in Figure 1. We assume the



(a) (b)

Fig. 1: Experimental laser fault injection setup – (a) device under test, (b) setup components.

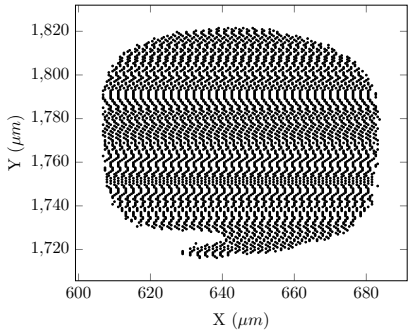


Fig. 2: Area plot depicting successful instruction skip experiments.

attacker has access to the target implementation of neural network as well as to the device where the implementation resides. We would like to point out that to carry out the attack without the usage of trigger, it is possible to determine the timing of the target operation by side-channel analysis [14].

B. 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 fuction 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.

We surrounded the activation functions in the second layer with a trigger signal that raised a voltage on a selected Arduino board pin to 5 V, helping us to determine the laser timing.

As instruction skip/change are one of the most basic attacks on microcontrollers, with high repeatability rates [15], 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.

Please note that we used a single fault adversarial model, meaning that exactly one fault was injected during one activation function execution. We consider an attack is successful for a given input data if the output classification is different from the classification obtained by the original network. And we refer to such a successful attack as misclassification.

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. Area of the chip vulnerable to these disturbances is depicted in Figure 2. The chip area is 3×3 mm^2, while the area sensitive to laser is $\approx$ 70×100 μm^2. With a laser power of 4.5% we were able to disturb the algorithm execution, when tested with reference codes. More details on the behavior on this particular microcontroller under laser fault injection can be found in [15].

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.

ReLu. This function is implemented in C as follows:

```
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 as follows:

```
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. This function is implemented by a following code in C:

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

Target activation function	Relation between x and x'
ReLu	$x' = 0$
sigmoid	$x' = 1 - x$
tanh	$x' = -x$

TABLE I: Relation between correct output x and faulted output x' when a single fault is injected in target activation function

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. This function is 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 outputs 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 [16]. The first application of such attack would be to target IEEE754 floating point representation for 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. Another application of bit flip attack would be to fault interconnecting weights, resulting in incorrect input to the next layer. We leave both directions for future investigation as they are out of scope for the current work.

If we let x and x' denote the correct and faulted output of the target activation function, the relation between x and x' is summarized in Table I.

IV. 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. When it comes to deep neural networks, there are three possible places to introduce a fault: input layer, hidden layer(s), and output layer. 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 injects faults into the last hidden layer of the network, targeting multiple activation function computations.

In the following we consider DNNs served for classification purposes; the activation function of the output layer is sofmax. We further assume the output layer is dense and the goal of the attacker is to misclassify an input. In Section IV-A we discuss the possible strategies of an attacker. In Section IV-B we present the evaluation results using the strategies on a sample DNN.

A. Algorithms for attacking the last hidden layer

We model the last two layers of a DNN as follows: let $\mathbf{x}$ denote the output of the last hidden layer and let W and B denote the matrix of weights and the vector of bias weights for output layer. Let $\mathbf{z}$ denote the input of softmax function. Suppose there are m neurons in the last hidden layer and n neurons in the output layer. Let W_k, $k = 1, 2, \ldots, n$ be the columns of W. Then the output is given by

$$\text{output}_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)} = \frac{\exp(\mathbf{x}W_i + B_i)}{\sum_{j=1}^{n} \mathbf{x}W_j + B_j}, \quad i = 1, 2, \ldots, n.$$

The final classification is given by ℓ such that $\max_i \text{output}_i = \text{output}_\ell$. For any sequence of z_j, $j = 1, 2, \ldots, n$, we have

$$\max_i \text{output}_i = \max_i \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)} = \frac{\max_i \exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)} = \frac{\exp\left(\max_i z_i\right)}{\sum_{j=1}^{n} \exp(z_j)}.$$

And the output classification is equal to ℓ s.t. $\max_i z_i = z_\ell$.

The attacker injects faults in the computation of the activation functions for neurons in the last hidden layer and gets a faulted $\mathbf{x}'$. Correspondingly we have a faulted vector $\mathbf{z}'$. Thus, for a given input with correct classification ℓ, the goal of misclassification is equivalent to: achieve $\mathbf{z}'$ such that there exists j with $z'_j > z'_\ell$ or $z'_j - z'_\ell > 0$. Consequently, an input can be misclassified if and only if

$$\begin{aligned}
(\mathbf{x}'W_j + B_j) - (\mathbf{x}'W_\ell + B_\ell) &> 0 \\
(\mathbf{x}W_j + B_j + (\mathbf{x}' - \mathbf{x})W_{jk}) - (\mathbf{x}W_\ell + B_\ell + (\mathbf{x}' - \mathbf{x})W_{\ell k}) &> 0 \\
\mathbf{x}W_j + B_j - \mathbf{x}W_\ell - B_\ell + (\mathbf{x}' - \mathbf{x})(W_{jk} - W_{\ell k}) &> 0 \\
z_j - z_\ell + (\mathbf{x}' - \mathbf{x})(W_{jk} - W_{\ell k}) &> 0 \\
z_j - z_\ell + \sum_{x'_k \neq x_k} (x'_k - x_k)(W_{jk} - W_{\ell k}) &> 0 \quad (5)
\end{aligned}$$

Algorithm 1 gives matrix A such that $A[k][j] = (x'_k - x_k)(W_{jk} - W_{\ell k})$ and diagonal matrix D whose diagonal is given by $\mathbf{x}' - \mathbf{x}$. Line 2 calculates the matrix A with column i given by $W_i - W_\ell$. Depending on the activation function, $\mathbf{x}'$ is related to $\mathbf{x}$ as described in Table I. At line 13, the $(k, j)-$entry of matrix A is given by $(x'_k - x_k)(W_{jk} - W_{\ell k})$.

Single fault strategy. When a single fault model is considered, $\mathbf{x}$ and $\mathbf{x}'$ only differs in one entry, say x_k. Equation (5) becomes

$$z_j - z_\ell + (x'_k - x_k)(W_{jk} - W_{\ell k}) > 0 \quad (6)$$

Algorithm 1: Calculation of matrix A

Input : W: matrix of weights for the last layer with columns $W_1, W_2, \ldots, W_n$; B vector of bias weights for the last layer; ℓ: the correct class of target input; $\mathbf{x}$: output of the last hidden layer for target input; activation function: ReLu, sigmoid or Tanh.
Output: Matrices A, D.

1 **for** $i = 1, 2, \ldots, n$ **do**
2 $A[i] = W_i - W_\ell$;
3 **if** *activation function is ReLu* **then**
4 **for** $k = 1, 2, \ldots, m$ **do**
5 $\mathbf{x}'[i] = 0$;
6 **if** *activation function is sigmoid* **then**
7 **for** $k = 1, 2, \ldots, m$ **do**
8 $\mathbf{x}'[i] = 1 - \mathbf{x}[i]$;
9 **if** *activation function is Tanh* **then**
10 **for** $k = 1, 2, \ldots, m$ **do**
11 $\mathbf{x}'[i] = -\mathbf{x}[i]$;
12 D = diagonal matrix with diagonal $\mathbf{x}' - \mathbf{x}$;
13 $A = DA$;
14 **return** A, D;

Algorithm 2: Single fault strategy

Input : A:obtained from Algorithm 1; $\mathbf{z}$: input of softmax function.
Output: `True/False` indicating if an attack exists or not; k s.t. the input can be misclassified with fault attack on neuron k.

1 **for** $k = 1, 2, \ldots, m$ **do**
2 **for** $j = 1, 2, \ldots, n,\ j \neq \ell$ **do**
3 **if** $z_j - z_\ell + A[k][j] > 0$ **then**
4 output k;
5 **return** `True`;
6 **return** `False`;

For given DNN and a target input, Algorithm 2 outputs k, the neuron to attack so that a misclassification can be achieved. In particular, line 3 checks if Equation (6) is satisfied for any j, k. If it can be satisfied for some k, j, the target input can be misclassfied with a fault attack on neuron k.

For multiple fault model, a natural strategy is **random faults**, i.e. random number of neurons in the last hidden layers are faulted. Here we provide another strategy which utilizes the information of weights and bias of the last layer.

Multiple faults strategy. For a target input with correct class ℓ, we aim to find neurons such that when attacked the probability of class ℓ in the output will be reduced. More precisely, we look for indices k such that $(x'_k - x_k)W_\ell < 0$. This strategy was implemented using Algorithm 3.

B. Evaluation of a sample DNN

To test how our attack can influence a real-world DNN, we trained and evaluated different DNNs with the attack strategies described above. The attack vectors considered are as described in Section III-B. We have selected a popular

Algorithm 3: Multiple faults strategy

Input : D: obtained from Algorithm 1; W_ℓ: the ℓth column of W; M: number of faults.
Output: `indices`: a list of neurons to attack.

1 `indices`= [];
2 $B = DW_\ell$;
3 **for** $k = 1, 2, \ldots, m$ **do**
4 **if** $B[k][j] < 0$ **then**
5 add k to `indices`;
6 **if** *length of* `indices`$== M$ **then**
7 **return** `indices`;
8 **return** `indices`;

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 activation function
Output layer	10	Softmax

TABLE II: Structure of the DNN used in evaluations.

MNIST dataset [17]. The training of DNNs was accomplished using Keras (ver.2.1.6) and Tensorflow libraries (ver.1.8.0). The structures of the DNNs are detailed in Table II. For each target function (ReLu, sigmoid and tanh), 10 DNNs with different number of neurons ($n = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500$) 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 training and test accuracy obtained are summarized in Table III. The accuracy shows that although the DNNs used are relatively simple, their accuracy is comparable with the state of the art.

For multiple fault model, we evaluated the DNNs with number of faults equal to $10, 20, 30, 40, 50$ percent of the number of neurons in hidden layer 4. The simulation results for targeting activation function being ReLu, Sigmoid and tanh are presented in Figures 3, 4 and 5 respectively.

Overall, it can be concluded that in case of sigmoid and tanh, if the attacker wants to have a reasonable success rate (>50%), she should inject faults in at least 40% of the neurons using multiple faults strategy in the chosen layer. But for ReLu, when the number of neurons is big, the DNN becomes more resistant to fault attacks.

The results 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.

Target	ReLu	sigmoid	tanh
Train. Acc.	98.4 – 99.2	99.0 – 99.4	98.2 – 99.3
Test. Acc.	97.3 – 98.0	97.6 – 98.1	97.4 – 98.1

TABLE III: Training/testing accuracy of DNNs used in evaluation. Number of neurons in Hidden layer 4 ranges from 50 to 500.

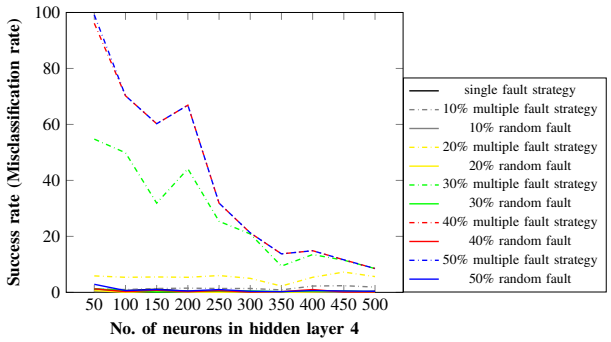


Fig. 3: Target activation function – ReLu.

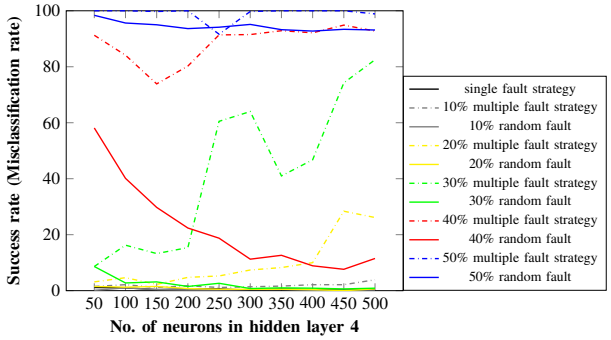


Fig. 4: Target activation function – Sigmoid.

V. Conclusion and Future Work

In this paper, we have studied laser fault injection attack technique on the major activation functions of deep neural networks. We stated implications how such attack can alter the behavior of targeted network, together with simulations. Our results demonstrate practicality of the attack on ReLu, sigmoid, and tanh.

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) [18], the fault tolerance techniques against malicious entities have to be developed in the same way as in the area of applied cryptography [19], [20].

Acknowledgment

National Research Foundation (NRF) Singapore, Prime Ministers Office under its National Cyber-security R&D Program (Award No. NRF2014NCR-NCR001-30 and No. NRF2018NCR-NCR005-0001), National Research Foundation (NRF) Singapore, National Satellite of Excellence in Trustworthy Software Systems under its Cybersecurity R&D Program (Award No. NRF2018NCR-NSOE003-0001), and National Research Foundation Investigatorship Singapore (Award No. NRF-NRFI06-2020-0001). The authors acknowledge the support from the 'National Integrated Centre of Evaluation' (NICE); a facility of Cyber Security Agency, Singapore (CSA).

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] M. Joye and M. Tunstall, *Fault analysis in cryptography*. Springer, 2012, vol. 40.

[3] J. Breier, X. Hou, D. Jap, L. Ma, S. Bhasin, and Y. Liu, "Practical fault attack on deep neural networks," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2204–2206.

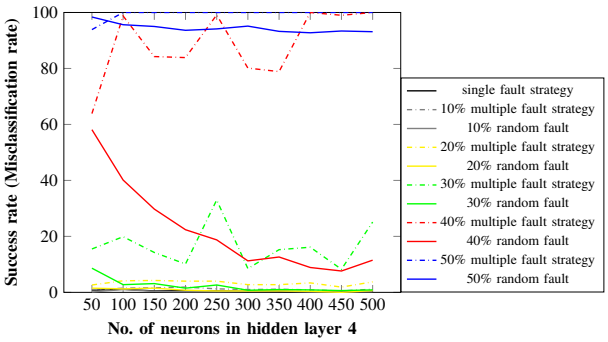


Fig. 5: Target activation function – tanh.

[4] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *CRYPTO'97*. Springer, 1997, pp. 513–525.

[5] Y. Liu, L. Wei, B. Luo, and Q. Xu, "Fault injection attack on deep neural network," in *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press, 2017, pp. 131–138.

[6] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraș, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 497–514.

[7] P. Zhao, S. Wang, C. Gongye, Y. Wang, Y. Fei, and X. Lin, "Fault sneaking attack: A stealthy framework for misleading deep neural networks," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[8] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "Ram-jam: Remote temperature and voltage fault attack on fpgas using memory collisions," in *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2019, pp. 48–55.

[9] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[10] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," *arXiv preprint arXiv:1611.05431*, 2016.

[11] A. M. Nia and K. Mohammadi, "A generalized abft technique using a fault tolerant neural network," *Journal of Circuits, Systems, and Computers*, vol. 16, no. 03, pp. 337–356, 2007.

[12] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 641–647.

[13] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[14] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium*, N. Heninger and P. Traynor, Eds. USENIX Association, 2019, pp. 515–532.

[15] J. Breier, D. Jap, and C.-N. Chen, "Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on aes," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. ACM, 2015, pp. 99–103.

[16] M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria, "How to flip a bit?" in *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*. IEEE, 2010, pp. 235–239.

[17] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[18] M. Lee, K. Hwang, and W. Sung, "Fault tolerance analysis of digital feed-forward deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 5031–5035.

[19] J. Breier and X. Hou, "Feeding two cats with one bowl: On designing a fault and side-channel resistant software encoding scheme," in *Cryptographers' Track at the RSA Conference*. Springer, 2017, pp. 77–94.

[20] V. Servant, N. Debande, H. Maghrebi, and J. Bringer, "Study of a novel software constant weight implementation," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2014, pp. 35–48.