# Simple Electromagnetic Analysis Against Activation Functions of Deep Neural Networks

Go Takatoi, Takeshi Sugawara, Kazuo Sakiyama and Yang Li

August 20, 2020

# Simple Electromagnetic Analysis Against Activation Functions of Deep Neural Networks

Go Takatoi, Takeshi Sugawara, Kazuo Sakiyama, and Yang Li

Graduate School of Informatics and Engineering,
Department of Informatics, The University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
{g.takatoi,liyang}@uec.ac.jp

**Abstract.** From cloud computing to edge computing, the deployment of artificial intelligence (AI) has been evolving to fit a wide range of applications. However, the security over edge AI is not sufficient. Edge AI is computed close to the device and user, therefore allowing physical attacks such as side-channel attack (SCA). Reverse engineering the neural network architecture using SCA is an active area of research. In this work, we investigate how to retrieve an activation function in a neural network implemented to an edge device by using side-channel information. To this end, we consider multilayer perceptron as the machine learning architecture of choice. We assume an attacker capable of measuring side channel leakages, in this case electromagnetic (EM) emanations. The results are shown on an Arduino Uno microcontroller to achieve high quality measurements. Our experiments show that the activation functions used in the architecture can be obtained by a side-channel attacker using one or a few EM measurements independent of inputs. We replicate the timing attack in previous research by Batina et al., and analyzed it to explain how the timing behavior acts on different implementations of the activation function operations. We also prove that our attack method has the potential to overcome constant time mitigations.

**Keywords:** Machine learning · Deep learning · Side-channel · Activation function · SEMA

## 1 Introduction

Machine learning has been researched in many areas due to its practicality and effectiveness. Deep learning especially is rapidly becoming a popular machine learning method. Image recognition [9, 15], robotics [13], natural language processing [24], security [1, 16, 28], and medical science [6, 7] are all areas in which deep learning are being used.

Neural networks are trained with high costs, and has a possibility of including confidential information from the training phase. Machine learning models are stored with valuable intellectual property, which are quickly becoming a target. Therefore, security over artificial intelligence (AI) is a growing concern. There are already a variety of attacks against AI [2, 12, 20]. For example, model extraction attacks [26], membership inference attacks [22], and model inversion attacks [5] are all attacks that target valuable information from AI. Model extraction attack presented by Tramer et al. can reverse engineer a machine learning model with high efficiency, as it only requires less than 10,000 online queries to the target machine learning model to replicate their attack. [26]. Shokri et al. proved that by using membership inference attack, prediction application programming interface (API) leaks information if an input was used as the training data [22]. Fredrikson et al. discussed that the model inversion attack could reverse engineer the training data just from the label and access to the prediction API [5].

In recent years, communication, privacy, and latency issues have caused deep neural networks to be calculated on the edge instead of the cloud servers. Edge devices are existent close by, therefore allowing physical attacks such as side-channel attacks. There are side-channel attacks to recover the architectures and parameters of neural networks. Leaked side-channel information include information from the operation. Recovering neural network architectures with cache side-channel attack is one way to attack edge AI [11,29]. Fault attacks are also used to recover neural network parameters [4]. There are side-channel attacks against specific neural network accelerators [27,30].

One recent work by Batina et al. has shown that a black box multilayer perceptron (MLP) and convolutional neural network (CNN) implemented on a 8bit and 32bit CPU can be reverse engineered using merely side-channel information [3]. They had separated the recovery of the network architecture into 4 key parameters: the activation function, pre-trained weights, the number of hidden layers, and the number of neurons in each layer. The activation function was discerned by timing attack from its distinctive computation time. Timing patterns or average timing can be compared with the profile of each function to determine the activation function. They recovered weight parameters with correlation electromagnetic analysis (CEMA), looking at the leakage in the Hamming weight of the input and weight multiplications. The layer boundaries and the number of nodes can be distinguished from the electromagnetic (EM) trace using the leakage signatures.

In this work, we have focused on the problem in the recovery of the activation function. The previously proposed recovery on the activation function has a limitation that it depends on non-constant timing behavior. By implementing constant time activation function, their attack can be easily mitigated. In this work, we have the following contributions.

- We propose a new type of attack to identify activation functions based on simple electromagnetic analysis (SEMA) [14]. Our proposed attack is implemented and demonstrated on a Arduino Uno, we were able to identify the activation functions used in the network.
- We have replicated the timing attack on the activation functions by Batina et al., and analyzed their attack and how the timing behavior acts on different implementations of the activation function operations. This has shown the significance of our proposed method as it is versatile to different activation function implementations and independent of inputs to the network.

- We have compared our new attack to the attack proposed in previous work, and we have discussed the potential of our attack to overcome constant time mitigation.

The rest of this paper is organized as follows. In Sect. 2, we briefly review the background. Sect. 3 outlines the methodology and the benefits of the SEMA attack. Subsequently, our experiment setup is shown in 4.1, and the proposed signal processing method is described in Sect. 4.2. The proposed signal processing is applied in Sect. 4.3. The analysis of the experiment results are shown in Sect. 5. Sect. 5.3 also discusses the analysis of previous work and constant time activation function implementations. Finally, the conclusions are presented in Sect. 6.

## 2 Background

In this section, we describe the details and architectures of the artificial neural network (ANN) used in this work.

### 2.1 Multilayer Perceptron

A MLP is a very simple type of neural network, and is made of fully connected layers. Fully connected layers means that all of the nodes in a layer is connected to all of the nodes in the next layer. A model of a node is depicted in Fig. 1. The circle surrounding $a$ and $y$ is called a node (or neuron). The output $y$ of a node is calculated in Eq. (1) as follows.

$$y = h\left(\sum_{i=1}^{n} x_i \times w_i + b\right) \tag{1}$$

Here, $(x_1, x_2, \dots, x_n)$ represents the inputs, $(w_1, w_2, \dots, w_n)$ represents the weights, $b$ represents the bias, and $h(a)$ represents the activation function. The bias is often programmed as the weight of an input value 1.

The model of a 4 layer MLP that is used in this work is depicted in Fig. 2. A MLP must have at least 3 layers, composed of at least one input layer, hidden layer, and output layer. The MLP in Fig. 2 consists of an input layer, two hidden layers, and an output layer.

### 2.2 Activation Functions

Here we describe the activation functions used in this works, which are sigmoid function, tanh function, softmax function, and Rectified Linear Unit (ReLU) function.

The sigmoid (logistic) function is a nonlinear function as shown in Eq. (2). This function will be most effective when used in a neural network trained with back propagation. The sigmoid function plots inputs ranged $(-\infty, \infty)$ to outputs ranged $(0, 1)$.

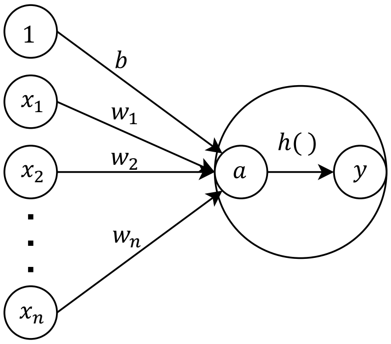


Fig. 1: A Model of a Node

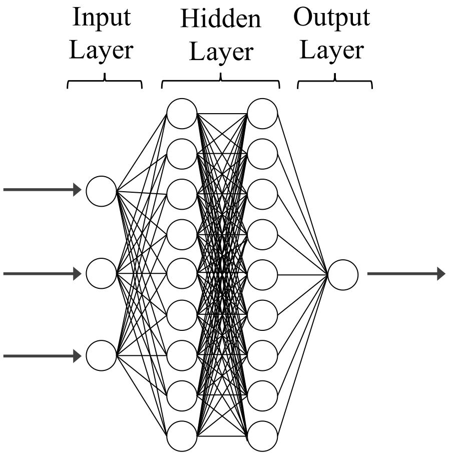


Fig. 2: A Model of a Multilayer Perceptron

$$h(a) = \frac{1}{1 + e^{-a}} \tag{2}$$

The tanh function is a rescaling of the sigmoid function, and the main difference is that it is symmetric by the origin. The tanh function maps inputs ranged $(-\infty, \infty)$ to outputs ranged $(-1, 1)$ as shown in Eq. (3).

$$h(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} = \frac{2}{1 + e^{-2a}} - 1 \tag{3}$$

The softmax function is able to map values into several outputs (or classes) which sum becomes 1. The output range is $(0, 1)$. It is able to be seen as probability, and is used for classification problems. Eq. (4) below is the softmax function where the vector is shown in bold.

$$h(\mathbf{a})_j = \frac{e^{a_j}}{\sum_{k=1}^{K} e^{a_k}}, \text{ for } j = 1, \ldots, K \text{ and } \mathbf{a} = (a_1, \ldots, a_K) \in \mathbb{R}^K \tag{4}$$

As shown in Eq. (5), the ReLU function is an extremely simple function, therefore mainly used as an activation function for ANNs. For networks with many nodes, this type of simple function can reduce the time of training and computing.

$$h(a) = \begin{cases} 0, & \text{for } a \leq 0 \\ a, & \text{for } a > 0 \end{cases} \tag{5}$$

## 3 Problem and Methodology

### 3.1 Identification of Activation Functions

Activation functions are used in many neural networks, and they play a very important role in the network. Non-linear functions are used as activation functions to output a result from the sum of the inputs and to solve non-linear problems. Designing and choosing activation functions that enable fast training of accurate deep neural networks is an active area of research. From this, it can be said that it is important to conceal the information of activation functions used in a neural network architecture.

### 3.2 Previous Work: Identification Based on Timing Behavior

In the previous work by Batina et al., they have used timing attacks to identify activation functions. The activation function was discerned by timing attack from its distinctive computation time. They showed that the timing behavior of the activation function can be directly observed on the EM trace. They collected EM traces and measured the timing of the activation function computation. The measurements were taken when the network were processing random inputs in the range they had chosen beforehand. A total of 2000 EM measurements were captured for each activation function. By plotting the processing time of each activation function by inputs, distinct signatures can be seen from each timing behavior. By making a profile, timing patterns or averaged timing can be compared with the profile of each function to determine the activation function.

The method proposed by Batina et al. has a few limitations as listed as follows.

1. Multiple measurements are required to use the distribution of the calculation time to identify the activation function.
2. Multiple inputs following a uniform distribution are required.
3. The distinct signatures of each activation function from the timing behavior has a possibility to differ depending on the implementation or processor.
4. The timing difference could be mitigated with constant time implementation of the activation functions.

### 3.3 New Method: Identification Using SEMA

The timing attack proposed by Batina et al. had several limitations. Therefore we took a different approach. While they collected EM traces and measured the timing of the activation function computation, we observe the leakage patterns of the EM trace directly and try to discern what operation is being computed in the trace from the different leakage signatures. We call this SEMA attack. SEMA attack requires only one or a few EM traces, compared to the timing attack which required multiple EM traces. We were successful in recovering the activation function by applying signal processing to the measurement. By reducing the noise in the EM trace, we were able to extract peak patterns from the measurement. We have found out that there are unique peak patterns for each operation in the activation function. Our attack is also independent of the inputs, as the patterns generally are invariant for any input. We observe the activation function operations from the peaks of the EM trace, thus this method has potential to overcome constant time mitigation.

## 4 Experiments

### 4.1 Experimental Setup

Here we describe the experimental setup to measure the EM emanations from an MLP trained for 3-input XOR.

**Target Device** The MLP is implemented on the microcontroller Atmel ATmega328P. The reasons for using Atmel ATmega328P as a target platform is motivated as follows.

- CPU and GPU are frequently used platforms for DNN computation, and use optimized libraries for its operations [23]. By using Atmel ATmega328P, we can implement the operations in a similar way.

**Software Setup** There are several ways to implement activation functions into a neural network. In our work, we have used activation functions that operates mathematically as shown in Eqs. (2)-(5). The exponential function and tanh function are implemented using standard library functions in C++ language. The MLP used in this work has the same architecture as Fig. 2, and has 2 hidden layers with 9 nodes in both layers. However for the MLP with the softmax function, the dimensions for the 2 hidden layers are 3 nodes for the first hidden layer, and 9 nodes for the second hidden layer.

**Hardware Setup** Tektronix MS064 oscilloscope was used to capture EM measurements, and used an RF-U 5-2 near-field EM probe from Langer to collect EM measurements. All measurements were $500M$Samples$/s$. We also used a low-pass filter BLP-50+ from Mini-Circuits with cutoff frequency of 48MHz to get a clear signal. To improve the quality of measurement of the microcontroller, we scraped the outer package, and decapsulated the microcontroller [21] as shown in Fig. 3. The EM probe is placed above the decapsulated chip and is chosen by hand. The full measurement setup is depicted in Fig. 4.

### 4.2 Attack Scenario

The goal of this work is to show an alternative method to recover the activation function instead of observing the pattern in the process time distribution that was presented in previous work. The proposed method has a few advantages over the previous method.

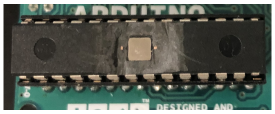


Fig. 3: Target Microcontroller Decapsulated

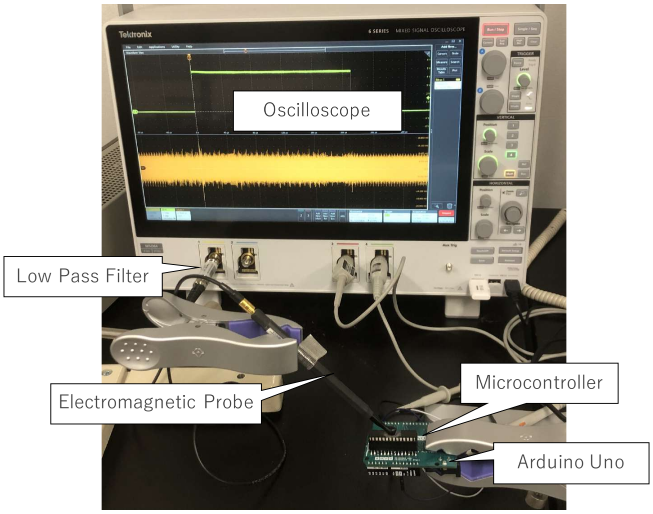


Fig. 4: Measurement Setup

- No information and access to the inputs required.
- Less executions required.
- Less implementation dependency, as we can show that the timing behavior used to identify sigmoid function and tanh function in previous work is implementation dependent.

Here we specify the considered attack scenario. Several pre-trained networks are implemented in C++ language and then compiled on to the edge device. Pre-trained networks are intellectual property, and accordingly the activation functions in those networks are confidential. The attacker's motive is to identify the activation functions used in the network. The attacker's capability is as follows.

- The attacker does not know the architecture of the network, but can access the network predictions.
- The attacker knows what set of activation functions could be implemented on the architecture, in this work, the sigmoid function, tanh function, softmax function, and ReLU function.
- The attacker is capable of measuring electromagnetic emanations from the target device.

Batina et al. has used the MLP to validate their attack methodology [3]. In this work, we also use the MLP as the DNN of choice. The motives are as follows.

- MLP is a widely used neural network architecture [8,10,19,25].
- Every node from a MLP is fully connected. Fully connected layer is a feature that can be seen in convolutional neural networks, recurrent neural networks, and other neural network architectures.

- All layers are identical, making side-channel analysis difficult than other neural network architectures.

Thereby, a generic attack is possible in developing our methodology. In other words, our methodology can be applied to many other DNN models.

### 4.3 Signal processing

We apply several methods to retrieve distinctive patterns from the EM trace. In this section, we propose a 4-step methodology to obtain the desired EM trace. First, a measurement (or trace) is taken from the target device. Next, averaging is applied to the measured trace. Then, to extract the peaks, we compute the upper half of the EM trace's envelope. For the last step, by smoothing the trace, the desired trace is acquired. The detailed actions of each step are as follows.

**Step 1: Measuring the EM trace** Here, an EM measurement of a MLP predicting the output class probabilities is taken from a microcontroller. MLPs including each of the 4 activation functions discussed in Sect. 2.2 are being computed. Then a EM trace is collected by an electromagnetic probe from the predicting MLP.

In our experiment, the measurements were taken from the device processing the input to the outputs of the first hidden layer's first node. In other words, the measurement is from the device computing Eq. (1). 4 measurements are obtained, each with 4 different activation functions mentioned in Sect. 2.2.

**Step 2: Averaging the EM trace** For this step, averaging is applied to the trace collected in step 1. By averaging the trace, it can improve the signal-to-noise ration (SNR). We used the oscilloscope in-built feature for averaging.

Each measurement from step 1 was averaged with 400 traces. However even with noise reduction, it is difficult to separate the boundaries of the nodes, multiplication operations and activation functions. The peaks are still very hard to distinguish, therefore signal processing is applied.

**Step 3: Extracting the upper half of the EM trace's envelope** To make the peak stand out from the averaged trace, we apply signal processing using Matlab. The upper half of the EM trace's envelope is calculated to check the peak of the trace. However, if only the envelope is calculated, the pulse component still remains.

The averaged traces in step 2 were processed using Matlab. The peaks of the traces were extracted by calculating the upper half of the envelope. The peaks can be seen, however there is still noise in the trace, making it hard to characterize each pattern.

**Step 4: Smoothing the EM trace** Last step is signal smoothing, to extract the noise from the envelope. In this work, we used the Gaussian-weighted moving average filter, calculated using Matlab. The smaller the window size, the higher frequency components stand out, and larger the window size, the lower frequency components are extracted. We want to extract the high frequency noise, therefore we use a large window size.

For the last step by using the Gaussian-weighted moving average filter, smoothing was applied to the noisy traces keeping important patterns in our measurements while leaving out noise. Table 1 presents the window size used for smoothing each of the measurements. The smoothed trace is shown in Fig. 5. The patterns can be compared easier with smoothing. The multiplication and activation function can be easily distinguished with their different patterns. The red lines in Fig. 5 represents the boundary of the multiplication and activation function. By observing the patterns of the multiplication operation, the weight multiplication, the addition of the outputs, and addition of the bias can be distinguished from the trace.

Table 1: Window size (in sample points) for different activation functions

| Activation Function | Window Size |
|---|---|
| Sigmoid | 4000 |
| Tanh | 4000 |
| Softmax | 1000 |
| ReLU | 2000 |

## 5 Analysis of the Results

### 5.1 Analysis of the Activation Function Operations

Here we analyze the processed measurements to discern activation functions. First, we start by examining the computation time of the activation functions. The computation time can be observed from Fig. 5. Table 2 presents the computation time of the activation functions.

Table 2: Computation time (in $\mu s$) for different activation functions

| Activation Function | Computation Time |
|---|---|
| Sigmoid | 190.79 |
| Tanh | 204.35 |
| Softmax | 695.52 |
| ReLU | 13.76 |

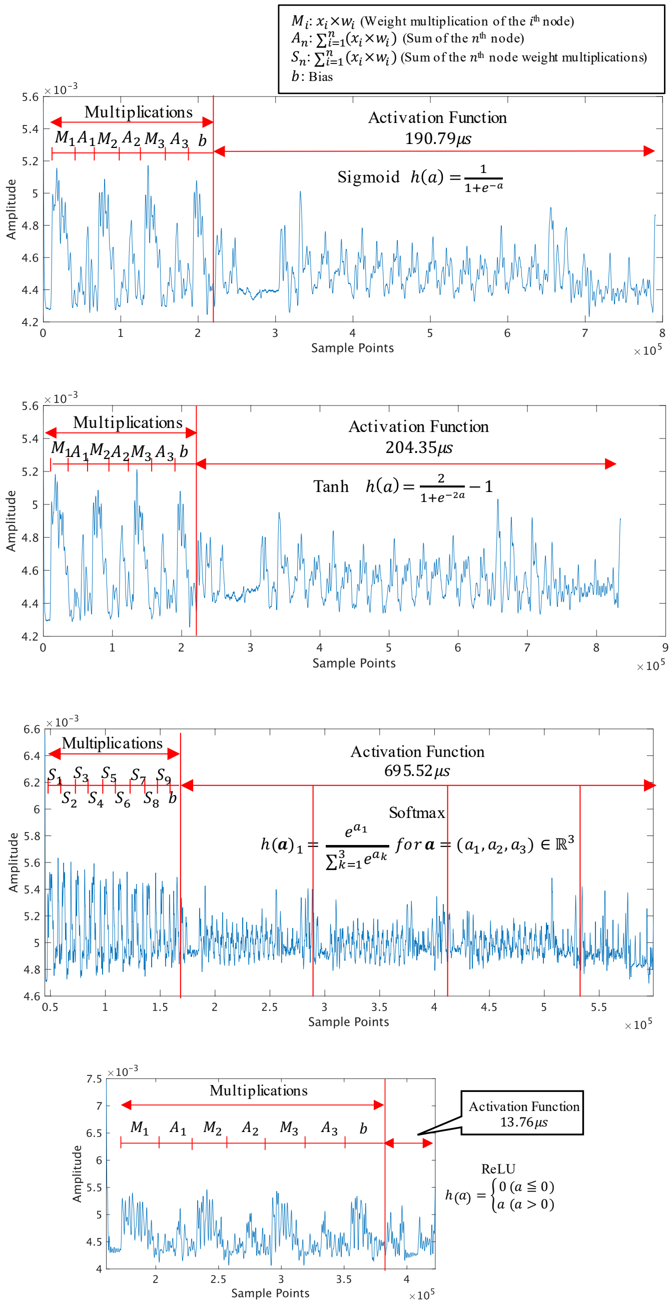


Fig. 5: Extracted Pattern Measurements of Activation Functions as Sigmoid, Tanh, Softmax and Relu.

As the activation functions differ in operations, so does the computation time. It can be observed from Table 2 that the ReLU function has the least computation time at $13.76\mu s$, and that the softmax function has the most computation time at $695.52\mu s$. Due to the simplicity of the ReLU function, it can be computed in a short time. The ReLU function does not have an exponentiation operation, unlike the other 3 activation functions. The softmax function computes the exponentiation operation several times, depending on the number of nodes in the output layer as shown in Eq. (4). Due to the complexity of the function, it takes the longest time to process. These two activation functions can be easily distinguished. However the sigmoid function and tanh function have similar computation times, sigmoid at $190.79\mu s$ and tanh at $204.35\mu s$.

Next, we observe the processed leakage patterns through SEMA. The softmax function computes the exponentiation operation several times, therefore the pattern of the exponentiation operation will repeat itself for the number of nodes in the output layer. The number of nodes in the output layer is 3 in this work, thus the multiplication pattern will repeat 9 times, and the exponentiation pattern 3 times. The vertical red lines separate the exponentiation operations and the division operation in the activation function.

The ReLU function does not include the exponentiation operation, therefore cannot observe the same patterns in the other activation functions.

Figure 6 compares the leakage patterns of sigmoid and tanh function. It can be observed that although there is no obvious gap in the processing time, the peak patterns differ. Extra peaks can be seen in 2 sections from the tanh function when comparing with the sigmoid function. The extra peaks observed are surrounded in a red box. The first peak can be seen right after the multiplication. The second peak can be seen in the latter half of the activation function. The differences in peaks comes from the difference in the functions. Tanh function has an additional multiplication and subtraction compared with the sigmoid function. We have observed that these operations causes the difference in peak patterns. The sigmoid function and tanh function can be distinguished with the different peak patterns, with tanh function having more peaks patterns.

### 5.2 Distinctive Features of Activation Functions from SEMA

Table 3 presents the features of each activation function when we used SEMA. To conclude, softmax function and ReLU function can be distinguished out of the 4 activation functions with their computation time. Also, by examining at the processed measurement patterns, all 4 activation functions can be discerned. Our experiments has shown that the EM trace leaks information on the operations in the activation function. Our method can be applied to an attack to identify the activation functions by making a template and pre-characterizing the operations in the EM trace.

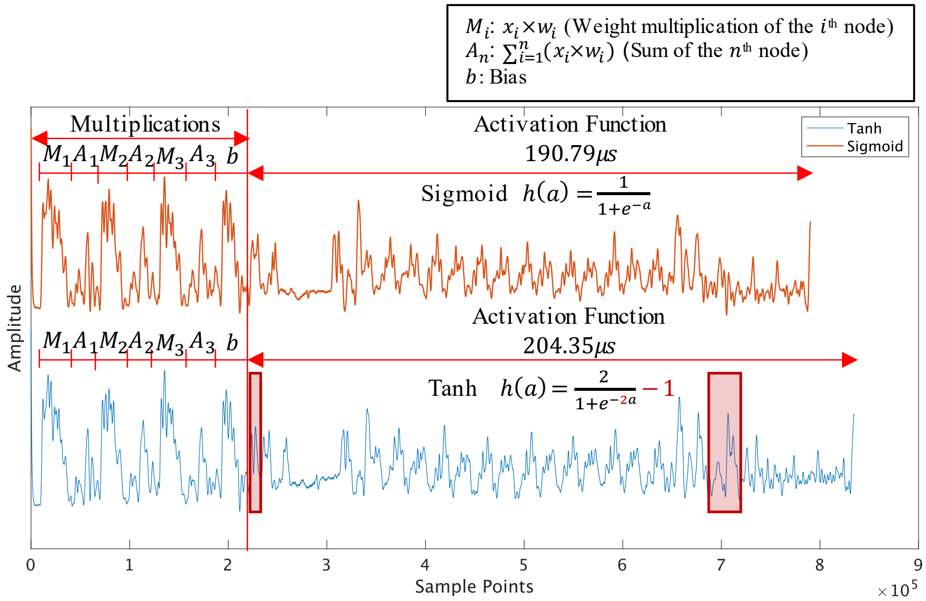


Fig. 6: Comparison of the patterns of sigmoid and tanh function

Table 3: Features of activation functions from SEMA

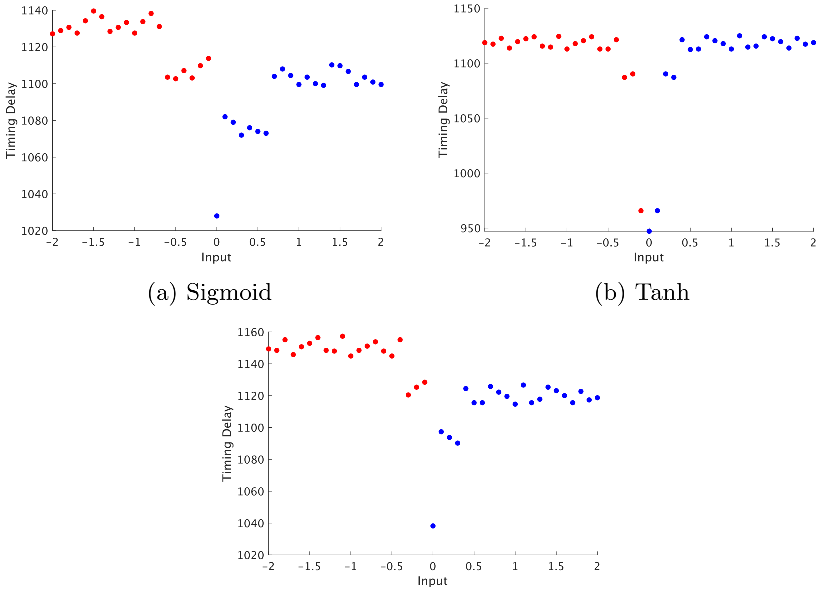
| Activation Function | Computation Time | Trace Pattern |
|---|---|---|
| Sigmoid | - | 2 less peaks than tanh |
| Tanh | - | 2 more peaks than sigmoid |
| Softmax | Long | Repeated exponentiation pattern |
| ReLU | Short | Not have exponentiation pattern |

### 5.3 Discussions

We were able to extract features from the EM trace, and identify each activation function, sigmoid function, tanh function, softmax function, and ReLU function. The results could be expected as the information of the operation leaks itself from EM emanations. Our hypothesis was that since the operations leaks information into the measurements, we could recover the operations from the measurements itself if we can reduce the noise in the trace. The signal processing allowed the peaks to have distinctive features for each operation. We were able to match the features to the activation functions for identification. We believe this attack could be applied to different neural network models as long as the activation functions operate directly. We have used Arduino Uno as the target platform. We believe our approach and methodology could be applied to different platforms. such as a similar microcontroller platform, a GPU platform or a FPGA platform. This is because side-channel analysis are demonstrated on these platforms on several previous works [3, 17, 27].

**Analyzing Previous Work** We have also analyzed the work by Batina et al. and their timing attack against activation functions. They plot the processing time of each activation function by inputs, and stated that distinct signatures could be seen from each timing behavior. However, the problem lies on why the timing behavior acts in such a way, which they have not explained in their work.

We were able to recreate the timing behavior of sigmoid function and the tanh function as shown in Fig. 7. The timing delay is displayed in $\mu$s. The experiments were done on an Arduino Uno simulator Tinkercad. The timing delay were measured with micros function, which returns the number of microseconds since the Arduino board began running the program. The processing time were averaged 10 times per input.



(a) Sigmoid

(b) Tanh

(c) Mathematically Calculated Tanh

Fig. 7: Timing Behavior for Different Activation Functions

Batina et al. have stated that tanh is more symmetric in pattern compared to sigmoid, for both positive and negative inputs which has been completely replicated in Figs. 7a and 7b. Based on our analysis of the results, we believe that the standard library functions in C++ language cause the distinct signatures. The exponentiation function causes the symmetric pattern in timing delay, with positive inputs having slightly longer computation time. The tanh function is an optimized operation, having symmetric patterns for both positive and negative inputs. However, if we did not use the standard library tanh function, and used the exponentiation function to mathematically calculate the tanh function, the timing behavior acts the same way as the sigmoid function as depicted in Fig. 7c. The patterns are both symmetric with positive inputs having slightly shorter computation time for Figs. 7a and 7c. The minimum, maximum, and mean values of the timing delay do not differ as significantly as Figs. 7a and 7b. The similarity in pattern and timing behavior makes the two activation functions very difficult to distinguish depending on the implementation method.

They have also stated that they take measurements when the network is processing random inputs in the range, i.e., $x \in \{-2, 2\}$. This input refers to the inputs to the activation function. To plot the timing behavior for different activation functions based on the inputs, there would be a need to calculate Eq. (6) below.

$$a = \left( \sum_{i=1}^{n} x_i \times w_i + b \right) \tag{6}$$

where $a$ is the input to the activation functions. Here, $(x_1, x_2, \ldots, x_n)$ represents the inputs, $(w_1, w_2, \ldots, w_n)$ represents the weights, $b$ represents the bias. Without the knowledge of the weights, bias, and nodes in a layer, this timing attack will not be possible. This proves the significance of our attack, as we do not need any information on the inputs to the activation function nor the input to the network. This means that there is no need to calculate the input to the activation function with our method, therefore shortening the step to identify the activation function.

**Implementation of Constant-time Activation Functions** To mitigate the timing attack on the activation function, dummy operations can be included to the computation of the activation functions. By making the computation time of the activation function constant with no operation instruction, it will become difficult to guess the activation function from the time computed. ReLU function will be the easiest to implement the constant-time operation, as the computation time only differs when the input is before 0 and after 0 [18]. However, with SEMA attack the operation itself could be seen through the EM trace, therefore it will become easy to see what operation is being done in the processor.

## 6 Conclusion

The need for implementations of neural networks on to the edge device is increasing. These edge devices, however, tend to be vulnerable to side-channel attacks. In this paper, we introduce an attack methodology that can recover activation functions from a black box network using side-channel information. We conducted the experiment on a MLP processing 3-input XOR implemented on a Arduino Uno microcontroller. Using SEMA and signal processing, the activation functions were successfully identified. Compared with previous work by that identify the activation functions using timing behavior, the SEMA attack proposed in this work does not rely on the inputs, requires fewer measurement and is less dependant on non-constant timing behavior of the activation function. Our experiment showed the vulnerabilities of edge AI to side-channel attacks. Neural networks on edge devices need to implement effective countermeasures against side-channel attacks to strengthen their security.

In future works, we will further research how SEMA could potentially break the constant-time implementation. We have also left out relatively new activation functions such as Exponential Linear Unit (ELU) function, Leaky Rectified Linear Unit (Leaky ReLU) function, and Scaled Exponential Linear Unit (SELU) function. We will further look into these activation functions and see how to distinguish them from ReLU function as they all have the same outputs for positive inputs, making them hard to identify with our current method.

**Acknowledgements** This work was supported by JST AIP Acceleration Research Grant Number JPMJCR20U2, Japan.

## References

1. Riscure. `https://www.riscure.com/blog/automatedneural-network-construction-genetic-algorithm/` (Last accessed 10 June 2020)
2. Ateniese, G., Mancini, L.V., Spognardi, A., Villani, A., Vitali, D., Felici, G.: Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. Int. J. Secur. Networks **10**(3), 137–150 (2015)
3. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: Heninger, N., Traynor, P. (eds.) 28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019. pp. 515–532. USENIX Association (2019)
4. Breier, J., Jap, D., Hou, X., Bhasin, S., Liu, Y.: SNIFF: reverse engineering of neural networks with fault attacks. CoRR **abs/2002.11021** (2020)
5. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 1322–1333. ACM (2015)
6. Fredrikson, M., Lantz, E., Jha, S., Lin, S.M., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. pp. 17–32. USENIX Association (2014)
7. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016)

8. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015. pp. 106–111. IEEE Computer Society (2015)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 770–778. IEEE Computer Society (2016)
10. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers (2017)
11. Hong, S., Davinroy, M., Kaya, Y., Dachman-Soled, D., Dumitras, T.: How to 0wn the NAS in your spare time. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net (2020)
12. Ilyas, A., Engstrom, L., Athalye, A., Lin, J.: Black-box adversarial attacks with limited queries and information. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 2142–2151. PMLR (2018)
13. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. I. J. Robotics Res. **32**(11), 1238–1274 (2013)
14. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999)
15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States. pp. 1106–1114 (2012)
16. Kucera, M., Tsankov, P., Gehr, T., Guarnieri, M., Vechev, M.T.: Synthesis of probabilistic privacy enforcement. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 391–408. ACM (2017)
17. Luo, C., Fei, Y., Luo, P., Mukherjee, S., Kaeli, D.R.: Side-channel power analysis of a GPU AES implementation. In: 33rd IEEE International Conference on Computer Design, ICCD 2015, New York City, NY, USA, October 18-21, 2015. pp. 281–288. IEEE Computer Society (2015)
18. Nakai, T., Suzuki, D., Omatsu, F., Fujino, T.: Evaluation of timing attacks against deep learningon a microcontroller and countermeasures. In: 2020 Symposium on Cryptography and Information Security - SCIS 2020, Kochi, Japan, January 28-31, 2020. vol. 3E4-4. The Institute of Electronics, Information and Communication Engineers (2020)
19. Naraei, P., Abhari, A., Sadeghian, A.: Application of multilayer perceptron neural networks and support vector machines in classification of healthcare data. In: 2016 Future Technologies Conference (FTC). pp. 848–852. IEEE (2016)
20. Papernot, N., McDaniel, P.D., Goodfellow, I.J., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Karri, R., Sinanoglu, O.,

Sadeghi, A., Yi, X. (eds.) Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017. pp. 506–519. ACM (2017)

21. Patranabis, S., Mukhopadhyay, D.: Fault Tolerant Architectures for Cryptography and Hardware Security. Springer (2018)
22. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 3–18. IEEE Computer Society (2017)
23. Sze, V., Chen, Y., Yang, T., Emer, J.S.: Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE **105**(12), 2295–2329 (2017)
24. Teufl, P., Payer, U., Lackner, G.: From NLP (natural language processing) to MLP (machine language processing). In: Kotenko, I.V., Skormin, V.A. (eds.) Computer Network Security, 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security, MMM-ACNS 2010, St. Petersburg, Russia, September 8-10, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6258, pp. 256–269. Springer (2010)
25. Thomas, P., Suhner, M.: A new multilayer perceptron pruning algorithm for classification and regression applications. Neural Processing Letters **42**(2), 437–458 (2015)
26. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016. pp. 601–618. USENIX Association (2016)
27. Wei, L., Luo, B., Li, Y., Liu, Y., Xu, Q.: I know what you see: Power side-channel attack on convolutional neural network accelerators. In: Proceedings of the 34th Annual Computer Security Applications Conference, ACSAC 2018, San Juan, PR, USA, December 03-07, 2018. pp. 393–406. ACM (2018)
28. Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., Song, D.: Neural network-based graph embedding for cross-platform binary code similarity detection. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 363–376. ACM (2017)
29. Yan, M., Fletcher, C.W., Torrellas, J.: Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. CoRR **abs/1808.04761** (2018)
30. Yu, H., Ma, H., Yang, K., Zhao, Y., Jin, Y.: DeepEM: Deep neural networks model recovery through em side-channel information leakage (2020)