Counteract Side-Channel Analysis of Neural Networks by Shuffling

Manuel Brosch*, Matthias Probst*, Georg Sigl*†
*Department of Electrical and Computer Engineering, Technical University of Munich, Munich, Germany
†Fraunhofer Institute for Applied and Integrated Security (AISEC), Munich, Germany
{manuel.brosch, matthias.probst, sigl}@tum.de

***Abstract*—Machine learning is becoming an essential part in almost every electronic device. Implementations of neural networks are mostly targeted towards computational performance or memory footprint. Nevertheless, security is also an important part in order to keep the network secret and protect the intellectual property associated to the network. Especially, since neural network implementations are demonstrated to be vulnerable to side-channel analysis, powerful and computational cheap countermeasures are in demand.**

In this work, we apply a shuffling countermeasure to a microcontroller implementation of a neural network to prevent side-channel analysis. The countermeasure is effective while the computational overhead is low. We investigate the extensions necessary for our countermeasure, and how shuffling increases the effort for an attack in theory. In addition, we demonstrate the increase in effort for an attacker through experiments on real side-channel measurements. Based on the mechanism of shuffling and our experimental results, we conclude that an attack on a commonly used neural network with shuffling is no longer feasible in a reasonable amount of time.

***Index Terms*—neural networks, side-channel analysis, countermeasure, shuffling**

I. Introduction

Artificial Intelligence (AI) experiences an increasing demand in the domain of edge devices. There are two scenarios of how to deploy AI applications on resource constrained devices, namely cloud-AI and edge-AI. In the cloud-AI scenario the edge device forwards its data to a server, where the classification takes place. Therefore, a network connection is required, increasing latency and making the system operation non-deterministic. However, compression techniques [1], [2], as well as optimizations in software and hardware, enable an AI-based classification directly on small devices [3]. When classification is performed on a resource constrained device, it is referred to as edge-AI. Benefits over cloud-AI are the improved power efficiency and real-time capability. [4]

While edge-AI brings also a lot of advantages from a security perspective, e.g. privacy of the data, the protection of know-how stored in the trained network is critical. Any information leak of network architecture or neuron weights should be avoided. One way to extract this information are power or Electro-Magnetic emanations (EM) side-channel attacks, if an adversary has physical access or remote power attack possibilities. Consequently, in the edge-AI scenario, Side-Channel Analysis (SCA) is a serious threat because it offers the opportunity to circumvent the training-phase and extract the learned network parameters directly from a competitior device.

A. Related Work

Recent publications show that with SCA it is possible to reveal information about Neural Networks (NNs). Prominent examples to extract the architecture of a NN are utilizing the timing behavior [5], [6] or the power side-channel [7]. In addition to the network structure, the parameters of a NN, such as the weights, represent valuable assets. Batina et al. recover a complete NN implemented on a microcontroller from EM measurements [8]. They utilize several side-channels, as well as Simple Power Analysis (SPA) and Differential Power Analysis (DPA), to extract the network.

Takatoi et al. extend the work of Batina et al. by extracting the type of activation function from EM traces as well [9]. As a result, there is no longer the need to profile the timing of several activation functions.

Considering parallel hardware implementations of NNs, Hua et al. reverse engineer a Convolutional Neural Network (CNN), executed on a hardware accelerator, by analyzing the memory access patterns during the execution of the NN [10].

Existing side-channel attacks on NNs demonstrate the need for effective countermeasures. Liu et al. use shuffling in order to reduce the information leakage about the structure of a neural network [11]. In contrast to our countermeasure, they are focusing on memory access patterns and are using shuffling to randomize the memory accesses. Moreover, they are protecting the structure of the network and not the weights and bias values.

Dubey et al. also address the need for countermeasures against SCA and propose a boolean masking scheme [12], [13]. However, their countermeasure is limited to hardware implementations and Binarized Neural Networks (BNNs), which have binary weight and bias values. Our countermeasure is not limited to a certain type of network and any feed-forward network can be protected with comparably low overhead. Moreover, software and hardware implementations can be secured by our countermeasure, if they perform computations at least partially sequentially.

B. Contribution and Organization

While shuffling is a well known countermeasure agains SCA, its application to NNs has never been investigated. We close this gap and show the effectiveness and possible pitfalls in this work. The proposed countermeasure focuses on protecting the trained parameters of a NN, i.e., the weights and bias values, since these parameters contain information about the training

978-3-9819263-6-1/DATE22/©2022 EDAA

data and their determination is time consuming. Hyperparameters, such as the number of neurons per layer, are not taken into account by the introduced shuffling countermeasure, similar to earlier work [12]–[14].

Shuffling is applicable to any sequential or partially sequential implementation of a NN, and is not restricted to any specific activation function or quantization method. Due to the size of even small networks, shuffling makes SCA practically impossible by increasing the required number of traces.

The remainder of the paper is structured as follows. Section II provides the fundamentals to NNs and shuffling to counteract SCA. In Section III we explain the threat model and how to apply shuffling to NNs. Experimental results based on a microcontroller implementation are discussed in Section IV. Lastly, we conclude the paper in Section V.

II. Background

A. Neural Networks

The basic building blocks of a NN are its neurons. A neuron reacts on the input vector by performing a dot product with its weight vector. The result of the dot product is then evaluated by some non-linear activation function, defining the output of the neuron. Feed-forward NNs implement multiple of such neurons and can be categorized into Multi-Layer Perceptrons (MLPs) and CNNs.

In order to give a NN a specific function, e.g., image classification, the network parameters need to be found, i.e., the NN must be trained. For training, special algorithms like back propagation [15] are used, which iteratively update the weights and bias values with respect to a loss-function. Overall, the training is time-consuming and computationally expensive because the parameters must be repeatedly updated in several epochs. [16]

To execute a trained NN with a reasonable throughput on a resource constrained device, compression techniques are required. In this work, we utilize a particular compression method called quantization. Quantization reduces the required memory bandwidth, energy and area by simplifying the calculations, i.e., use integer arithmetic instead of floating-points [17]. Jacob et al. propose an integer quantization scheme that finds application in *TensorFlow* and the *CMSIS-NN* library [1]. The only requirement for the mentioned quantization scheme is a 32-bit platform. According to the scheme, weights are quantized to a bitwidth of 8-bit. The bias is quantized to a 32-bit signed integer and is added to the 32-bit result of the Multiply and Accumulate (MAC) operation.

B. Shuffling as Countermeasure against Side-Channel Analysis

SCA is a powerful tool to reveal secret information from electronic devices. Shuffling reduces the Signal-to-Noise Ratio (SNR) within the measurements, thus, it increases the effort for an attack, but cannot completely counter a first-order side-channel attack. The SNR gets decreased by randomizing the execution sequence of operations. Hence, the intermediate results, an attacker uses, are present at different points in time within the measurements. When l operations can be shuffled, the correlation coefficient in a Correlation Power Analysis (CPA) is reduced by the same factor. As shown by Mangard et al., the squared correlation coefficient of the correct hypothesis is proportional to the SNR, for small SNRs [18]. Consequently, the SNR decreases quadratically, meaning that the amount of traces increases by l^2 to mount a successful attack [18]. As a result, shuffling becomes a very effective countermeasure with a high number of mixable operations, since this directly influences the effort for an attack.

Nevertheless, the effect of shuffling can be reduced, if an attacker gets knowledge about the points of interest, where the critical operations are performed. Windowing sums up sections, where the targeted operation is performed, in order to diminish the reduction in the SNR. One can show that if windowing is applied correctly, the number of required traces increases only linear and not quadratically, reducing the effort for an attack [18].

III. Shuffling for Neural Networks

The goal of this work is to protect the parameters of a NN in such a way that they are practically impossible to be retrieved by first-order SCA because the attack effort is too excessive. The NN architecture is not protected by the proposed countermeasure.

During theoretical analysis of the countermeasure we will focus on MLPs. However, the convolution performed by CNNs is also realized with MAC operations and, therefore, shuffling is an effective countermeasure applicable for CNNs as well.

A. Threat Model

For our analysis we assume a passive attacker, who has physical access to the device. As a result, the attacker is able to observe the device during its computations. Moreover, the attacker has full control over the input and can read the output. But, the attacker is not able to interfere with the operation of the device by introducing glitches, for example. Consequently, active attacks, such as fault injection or control flow manipulation, are out of scope of this work. Additionally, we assume that the attacker has no access to the memory or the bus system. Thus, the attacker is not able to monitor memory accesses or read data from the memory.

Since the attacker is passive and has physical access, he/she can observe side-channel information by measuring the power consumption or EM emanations, for instance. Further, we assume that the attacker knows the structure of the NN. This assumption is similar to the cryptography scenario, where an attacker knows the implementation of a cipher, but not the secret key. However, without the parameters of a NN, an attacker will not be able to reverse engineer the complete NN, if he/she has no access to the training dataset or the required compute performance.

B. Shuffling for Neural Networks

When executing a NN, the outputs of the neurons are calculated layerwise[1]. As a consequence, the evaluation of

[1]This is also the case for the result of a convolution, where the result is calculated step-by-step depending on the stride.

a layer starts as soon as the outputs of all neurons of the preceding layer are calculated. Within a layer, consisting of j neurons and n inputs, the output y_k of a neuron k is calculated by

$$y_k = \alpha \left(\sum_{i=0}^{n-1} (w_{k,i} \cdot x_i) + b_k \right), \tag{1}$$

where $w_{k,i}$ represents the weight associated to the input x_i, b_k is the bias of the neuron and α defines the non-linear activation function.

In a pure sequential implementation of a network, the neuron outputs of a layer are calculated one after the other according to Eq. (1). Since neurons are not evaluated in parallel, an attack is easier. However, the sequential execution allows the implementation of shuffling in a very effective manner, preventing SCA of the parameters in reasonable time. Shuffling randomizes for every new input to the NN the execution sequence of the neurons $(y_0, y_1, y_2, \ldots, y_{j-1})$, as well as the calculation sequence of the multiplications $(w_{k,0} \cdot x_0, w_{k,1} \cdot x_1, \ldots, w_{k,n-1} \cdot x_{n-1})$ within a neuron.

Our threat model assumes that an attacker knows the input data x_i of a network. In order to retrieve information about $w_{k,i}$, an attacker will concentrate on the multiplication involving x_i and $w_{k,i}$. As soon as all weights $w_{k,i}$ of a neuron k are known, an attacker is able to recover the bias b_k, targeting the addition of the bias and the MAC-result[2].

Since the preliminaries for a DPA are fulfilled, we perform a correlation-based DPA or CPA in the following. To mount a CPA targeting the weights, an attacker must assume hypothetical values $h_{k,i}$ for the corresponding weight $w_{k,i}$. By multiplying the assumed values $h_{k,i}$ with the used input x_i, possible intermediate results are generated. An attacker will calculate the intermediate results for every applied value of input x_i, for which also a measured trace is existing. As a result, a matrix $\mathbf{H}_k$ is generated of dimension (number of traces $\times$ number of hypotheses), which includes the possible intermediate results. With the help of a power model the content of $\mathbf{H}_k$ is transformed into theoretical power values. Now every column of $\mathbf{H}_k$, i.e., each hypothesis, is correlated with the measured traces. The correlation coefficients for a neuron k are calculated as follows

$$\rho_{d,c} = \frac{\sum_{m=0}^{N-1}((h_{m,d} - \overline{h}_d) \cdot (t_{m,c} - \overline{t}_c))}{\sqrt{\sum_{m=0}^{N-1}(h_{m,d} - \overline{h}_d)^2 \cdot \sum_{m=0}^{N-1}(t_{m,c} - \overline{t}_c)^2}}, \tag{2}$$

where N corresponds to the number of measurements, $h_{m,d}$ represents a theoretical power value for measurement m and the hypothesis d, and $t_{m,c}$ is the m-th power trace at sample point c.

At some point in time t_{proc} the attacked operation is performed and the intermediate results of the correct hypothesis most likely correlates significantly higher with the measured traces, than all other hypotheses. In the following, we will denote $\rho_{ckt,proc}$ as the correlation coefficient of the correct hypothesis at the mentioned point in time, and $\rho_{oth,proc}$ represents the correlation of all other hypotheses at this time.

To ensure that $\rho_{ckt,proc}$ is significantly larger than $\rho_{oth,proc}$ a certain amount of measurements n_{traces} is required. For small values of the SNR and if $|\rho_{ckt,proc}| \leq 0.2$ holds, Mangard et al. [18] give the amount of necessary traces n_{traces} by

$$n_{traces} \approx \frac{28 \cdot l^s}{\rho^2_{ckt,proc}}, \tag{3}$$

where $s = 0$ for an unprotected implementation and l is only relevant for the shuffling case, which will be discussed later. Equation (3) holds for most devices, power models and measurement setups.

1) Shuffling: A sequential execution of the neurons and multiplications within a neuron, allows a powerful implementation of shuffling. Not only the execution order of the neurons within the same layer can be altered, but also the multiplication order inside a neuron can be randomized. As a result, it is no longer possible for an attacker to identify at which point in time which neuron of a layer is executed, and when which input is multiplied with the corresponding weight. Thus, the correlation coefficient of the correct hypothesis is reduced by $l = j \cdot n$, where j and n represent the number of neurons and the number of inputs of the layer, respectively. Note, that $s = 2$ in Eq. (3) for the shuffling case, i.e., the number of traces to perform a successful DPA increases quadratically by l^2. Since NNs have typically a large number of inputs and neurons, l is also large. Thus, the effort for an attacker increases dramatically.

Moreover, shuffling causes an additional problem an attacker has to solve, since the execution order of the critical operations is randomized. As a result, an attacker will retrieve weights, but he/she cannot identify the neuron the respective weight belongs to. Therefore, an attacker must consider several hypotheses as true, that achieve significantly higher correlations[3] than all other hypotheses. Furthermore, the attacker needs to find a way to determine which of the recovered weights belongs to which neuron of the layer.

Please note, that the random execution sequences during the shuffling procedure must be statistically independent and uniformly distributed, to ensure the quadratic increase in measurements.

2) Windowing Attack on Shuffling: Shuffling can be significantly weakened, if an attacker applies windowing. To reduce the number of required measurements for a successful attack, an attacker can adapt the attack, if he/she knows when the targeted operation is executed. In our case, the attacker needs to know when a multiplication of a weight with an input is performed, in order to preprocess the traces by means of windowing. By combining windows, where the targeted multiplications take place, the SNR can be increased compared to the normal CPA on shuffling. Windowing ensures that the targeted part of the power consumption or EM emanations is included in

[2]When all weights of a neuron are known, an attacker would assume hypothetical values for the bias b_k of the neuron. However, only the hypotheses and the targeted operation change, the principle of the attack remains the same.

[3]In the provided figures, we suppressed the correlations of the weight values of other neurons processing the same input. Since these hypotheses have a similarly high correlation coefficient, they would affect the clarity of the figures.

the preprocessed trace. Thus, one can show that the correlation coefficient $\rho_{ckt,proc}$ decreases only by $\sqrt{l}$ instead of l, if the summed up samples of a single measurement are statistical independent [18]. As a result, the number of traces, required to perform a successful DPA, increases only linear by l and not quadratically as without windowing. Hence, $s = 1$ in Eq. (3).

Nevertheless, NNs have typically a high number of inputs and neurons within the first few layers, therefore, the linear increase is still sufficient to make shuffling effective for most NNs.

3) Optimal Selection of Inputs: In our threat model special care must be taken, as the attacker controls the input given to the NN he/she can weaken the shuffling countermeasure, at least for the very first layer. Lets assume we want to recover the weights of all neurons of the first layer that are sensitive to input x_i. Therefore, we could set input x_i to some value unequal to zero and all other inputs $x_m \neq x_i$ to zero. As a consequence, Eq. (1) simplifies to

$$y_k = \alpha \left(w_{k,i} \cdot x_i + b_k \right). \tag{4}$$

Since the intermediate result $w_{k,i} \cdot x_i$ is present latest after the last multiplication, an attacker can use only these sample points. Thus, the correlation of the correct hypothesis is decreasing only by $l = j$, with j representing the number of neurons of the layer.

To prevent the reduction of $l = j$, one has to ensure that multiple weights are involved in the MAC-operations within a neuron in any case. We propose a limitation of the value range of the inputs, to exclude $x_m = 0$. Before the execution of the NN on our device, the device adds the smallest possible value to every input, except those where the input is already set to the maximum value[4]. As an example, if we have a NN that uses 8-bit unsigned integer values as inputs, we limit our range to $x_i \in \{1, 2, \ldots, 255\}$.

Given that the multiplication between a weight and an input is constant in time, the proposed adaption of the inputs does not lead to recognizable differences in the measurements. Consequently, the attack stated above is no longer possible, and an extraction of the necessary parts from the traces with SPA is also not applicable for our microcontroller.

IV. Results

In order to verify the effectiveness of our countermeasure, within a reasonable amount of time, we implement two neurons with six inputs each. The weights and bias values are part of a MLP, trained for the MNIST dataset of handwritten numbers. The network follows the compression scheme by Jacob et al. [1]. Unfortunately, we cannot provide a comparison with other countermeasures because the only other available countermeasure is a masked hardware implementation of a BNN [12], [13]. As a result, a fair comparison is not feasible, since the network types are different and a collation between hardware and software implementations is not very meaningful in terms of different design parameters, such as the execution time.

[4]In order to maintain the accuracy of the network, we also need to train the network with an adapted version of the training data, where we replace the data in the same manner.

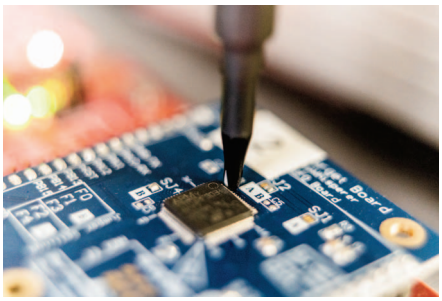

Figure 1. Position of our near field EM-probe above the V_{ss}-pin of the STM-microcontroller.

A. Experimental Setup

As Device Under Test (DUT), we utilize the NAE-CW308T-STM32F3 UFO Board, which features a STM32F303 microprocessor, implementing an ARM Cortex M4 softcore. The microprocessor is clocked at $66.33\,\mathrm{MHz}$. Since a quantized NN is used, pure integer arithmetic is utilized to perform the multiplications of the weights and the inputs. Therefore, the execution time of a multiplication is constant, c.f. [19], and leaks no information through the timing.

The traces are acquired by a Picoscope 6402D, running at a sampling frequency of $312.5\,\mathrm{MHz}$, using a near field Langer EM-probe (RF U 2,5-2). The signal of the probe is amplified by $30\,\mathrm{dB}$ by a preamplifier (Langer PA 303). The probe is placed on the digital ground-pin (V_{ss}) of the microcontroller, as one can see in Fig. 1.

In our experiments, we are using the Hamming Weight (HW) power model. The HW model assumes that the power consumption P of the device is proportional to the number of bits set to 1 in a data value $\mathbf{v}$ of bitwidth n, i.e.,

$$P \propto \mathrm{HW}(\mathbf{v}) \tag{5}$$

$$\mathrm{HW}(\mathbf{v}) = \sum_{i=0}^{n-1} (v_i). \tag{6}$$

For the attack, we perform a CPA that targets the multiplication between the inputs and weights. Our methodology is to apply random inputs and measure the EM emanations during the operation of the device.

B. Attack on an Unprotected Neural Network

At first, we determine the minimum number of traces required by an attacker, to extract the parameters from an unprotected implementation. From Fig. 2 we can derive that about $4,000$ traces are required, to clearly distinguish the correct hypothesis from all other hypotheses. In the following, we use $4,000$ traces as a baseline to compare the results of the shuffling countermeasure.

However, please note at this point, that due to using 8-bit integer weights, an attacker is not able to recover single

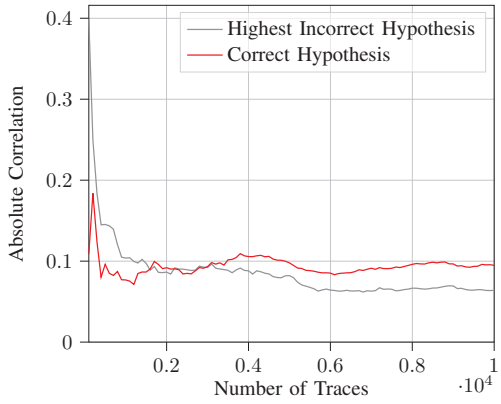

Figure 2. Absolute correlation when attacking weight four of the first neuron of the unprotected implementation. With about 4,000 traces the correct hypothesis is significantly larger than all other hypotheses.

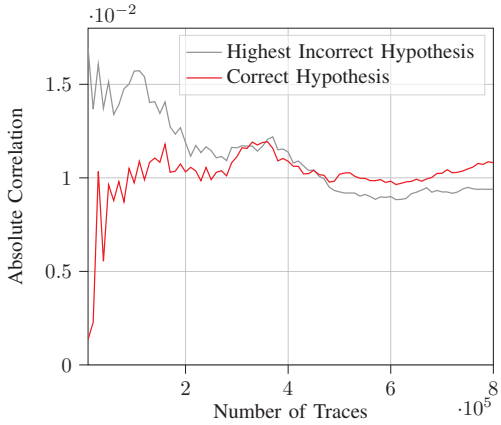

Figure 3. Resulting absolute correlation for the shuffled implementation at same point in time as for the unprotected implementation. With approximately 550,000 traces the correct hypothesis is significantly larger than all other hypotheses.

weights, but rather gets a set of possible values for each weight. Nevertheless, the possible values within this set are connected with each other, as they are only binary-left or binary-right shifted versions of the true weight. The reason, why an attacker retrieves a set of possible values is related to the used correlation coefficient to perform the CPA. As an example, if the attacked weight has the value 17, then the hypotheses for the values 17, 34 and 68 have the same HW. Thus, the results of the multiplications with the used inputs have also the same HWs, as they are only binary shifted versions of the correct hypothesis. Consequently, the correlations for these hypotheses are the same. For negative weights, the HW decreases (left shift) or increases (right shift) depending on the shift direction. This affects the HW of the targeted multiplication result. However, the differences in the HWs for shifted versions of the correct weight hypothesis are proportional to the number of shifts. This proportional difference in the HWs does not matter, since a CPA evaluates the course of the HW across the traces to find similarities. As a result, the correlation coefficients for shifted versions of a negativ hypothesis are similar. Hence, an attacker has to find a way to determine the correct weight hypothesis from the set. Though, the determination is out of scope of this paper, thus, we simply assume that an attacker is able to select the correct weight from the set of values.

C. Attack on a Neural Network Protected with Shuffling

When shuffling is applied[5], we need to collect $576,000$ traces to ensure that $\rho_{ckt,proc}$ is significantly larger than the correlation for all other hypotheses, according to Eq. (3) with $l = 12$ and $s = 2$. To verify this, Fig. 3 depicts the correlation along the number of traces. We can see that, with approximately $550,000$ measurements the correct hypothesis can be clearly distinguished from all other hypotheses, which matches about the theoretical results of $576,000$ traces.

[5]The randomness required to generate the shuffled sequence is 23 bits in our case, when using the Fisher-Yates algorithm. However, it can be reduced to 12 bits if the sequence for the weights is reused for both neurons.

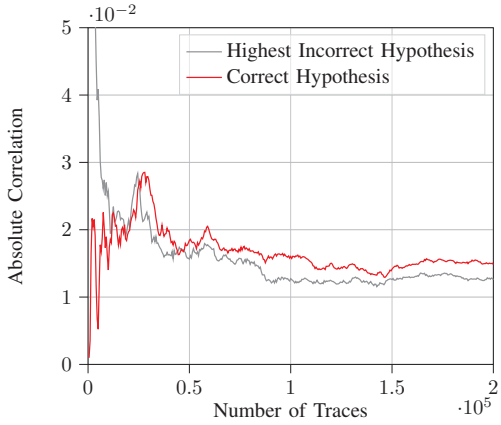

Figure 4. Result of the CPA of the protected implementation, if the traces are preprocessed by windowing. Again the fourth weight of the first neuron is attacked. With approximately 60,000 traces the correct hypothesis is significantly larger than all other hypotheses.

We also perform the windowing approach, by summing up the sample points where the multiplications are performed. The results for a CPA with windowing are depicted in Fig. 4. As expected, the amount of required traces increases only linear by a factor of approximately $l = 12$, due to the preprocessing. Therefore, with about $60,000$ traces, the weight can be recovered, which fits around the theoretical amount of $48,000$ traces. The theoretical and practical results do not match exactly, due to some small timing delays in the measurements. These delays cause a partially wrong sample selection for windowing, which increases the amount of required traces.

D. Attack on a Complete MLP

Additionally, to our experimental attack of the two neurons, we attack a full MLP trained for the MNIST dataset of handwritten digits. Our network is a very small three layer MLP with

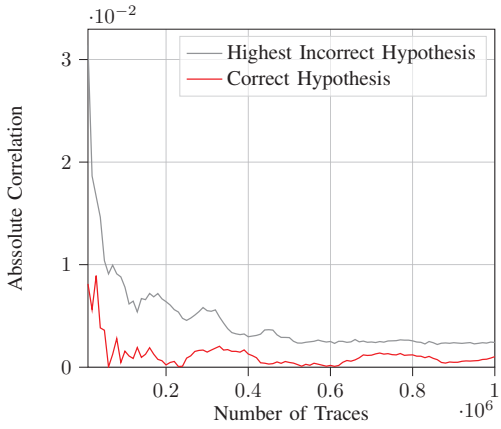

Figure 5. Result of a CPA of one weight of the first neuron of the complete MLP protected by shuffling. One million traces are not enough to extract the correct weight from the measurements. In theory, 553 billion traces would be required to perform a successfull attack, in our case.

15, 10 and 10 neurons in the layers and a classification accuracy of about 95 %. The execution time of the network increases due to shuffling from 3.276 ms to 3.85 ms, i.e., we have a penalty of about 18 %. Figure 5 shows the absolute correlation along the number of traces for a weight of a neuron of the first layer of the shuffled network. As we can see, even with one million traces we are not able to perform a successful attack. In order to extract the weights correctly, approximately 553 billion measurements for a standard DPA would be required, or 47 million if we perform windowing. However, we would need about 245 days[6] to acquire the necessary amount of measurements, for the windowing case.

It is important to note, that as soon as an attacker can recover the weights of the largest layer, also the other layers can be attacked. Consequently, the layer with the most shuffling opportunities l determines the effort for an attack.

V. Conclusion

Implementations of neural networks are shifted from the cloud to edge devices that can easily fall into the hands of an attacker. Therefore, side-channel attacks must be taken into account. We have investigated shuffling as countermeasure against side-channel attacks for software implementations of NNs, to avoid extraction of the weights and bias values. We have demonstrated on a small example that the theoretical calculated complexity increase for the attack matches the measurements. From this we concluded that an attack on a realistic NN, which implements shuffling, is not possible in reasonable time. Furthermore, we introduced a small tweak, which avoids that an attacker sets all but one inputs to zero and, thus, can extract the weights easier. Our experiments have shown, that this tweak eliminates this attack path, but does not influence the performance of the NN. Even a small NN with three layers and 15, 10, 10 neurons in each layer, could not be attacked anymore with reasonable effort, i.e., we would need 47 million measurements. Overall, our investigations show that the simple countermeasure shuffling is effectively protecting the weights and bias values, if applied carefully.

[6] Assuming the 4,000 traces from the unprotected implementation, a factor $l = 784 \cdot 15$ and a time per measurement of 450 ms (including data transfer to the DUT, as well as data transfer from oscilloscope to computer and storing the data).

Acknowledgment

This research was partly funded by the Bavarian funding program for research and development "Information and Communication Technology" through the project MITHRIL, grant number IUK623/002.

References

[1] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[2] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, 2020.

[3] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[4] W. Li and M. Liewig, "A survey of ai accelerators for edge environment," in *World Conference on Information Systems and Technologies*, pp. 35–44, Springer, 2020.

[5] M. Yan, C. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn dnn architectures," *arXiv preprint arXiv:1808.04761*, 2018.

[6] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[7] Y. Xiang, Z. Chen, Z. Chen, Z. Fang, H. Hao, J. Chen, Y. Liu, Z. Wu, Q. Xuan, and X. Yang, "Open dnn box by power side-channel attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[8] 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 (USENIX Security 19)*, pp. 515–532, 2019.

[9] G. Takatoi, T. Sugawara, K. Sakiyama, and Y. Li, "Simple electromagnetic analysis against activation functions of deep neural networks," tech. rep., EasyChair, 2020.

[10] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.

[11] Y. Liu, D. Dachman-Soled, and A. Srivastava, "Mitigating reverse engineering attacks on deep neural networks," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 657–662, IEEE, 2019.

[12] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: A pathway for secure inference against power side-channel attacks," *arXiv preprint arXiv:1910.13063*, 2019.

[13] A. Dubey, R. Cammarota, and A. Aysu, "Bomanet: Boolean masking of an entire neural network," *arXiv preprint arXiv:2006.09532*, 2020.

[14] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "Prada: Protecting against dnn model stealing attacks,"

[15] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.

[16] I. N. Da Silva, D. H. Spatti, R. A. Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, "Artificial neural network architectures and training processes," in *Artificial neural networks*, pp. 21–28, Springer, 2017.

[17] T. Liang, J. Glossner, L. Wang, and S. Shi, "Pruning and quantization for deep neural network acceleration: A survey,"

[18] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*, vol. 31. Springer Science & Business Media, 2008.

[19] Arm Limited, https://developer.arm.com/documentation/100166/0001, *Arm Cortex-M4 Processor - Technical Reference Manual*, r0p1 ed., May 2020.