CSI Neural Network: Using Side-channels to Recover Your Artificial Neural Network Information

Lejla Batina[1], Shivam Bhasin[2], Dirmanto Jap[2], and Stjepan Picek[3]

[1] Radboud University, Digital Security, Nijmegen, The Netherland
[2] Physical Analysis and Cryptographic Engineering, Temasek Laboratories at Nanyang Technological University, Singapore
[3] Delft University of Technology, Delft, The Netherlands

Abstract. Machine learning has become mainstream across industries. In this work we pose the following question: Is it possible to reverse engineer a neural network by using only side-channel information? We answer the question affirmatively. To this end, we consider a multi layer perceptron as the machine learning architecture of choice and assume a passive attacker capable of measuring only passive side-channels like power, electromagnetic radiation, and timing.

We conduct all experiments on real data and common neural net architectures in order to properly assess the applicability and extendability of such attacks. Our experiments show that the side-channel attacker is able to obtain information about the activation functions, the number of layers and neurons in layers, the number of output classes, and weights in the neural network. Thus, the attacker can efficiently reverse engineer the network using side-channel information.

Next, we show that if the attacker has the knowledge about the neural network architecture, he/she could also recover the inputs to the network with only a single measurement. Finally, we discuss several mitigations one could use to thwart such attacks.

Keywords: Side-channel Analysis, Artificial Neural Networks, Power, Reverse Engineering, Countermeasures

1 Introduction

Machine learning, and more recently deep learning, has become hard to avoid for research in distinct areas, such as image recognition [1], robotics [2], natural language processing [3], and security [4,5] mainly due to its unquestionable practicality and effectiveness. Ever increasing computational capabilities and huge amounts of data are resulting in more complex machine learning architectures. As an example, AlexNet architecture consisting of 8 layers was the best performing algorithm in image classification ILSVRC2012 classification task. In 2015, the best performing architecture for the same task was ResNet consisting of 152

layers [6]. This trend is not expected to stagnate any time soon, so it is prime time to consider machine learning from a novel perspective and in new use cases.

In this work, we focus on the widely used machine learning family of algorithms: the neural networks family. With the increasing number of design strategies and elements to use, fine tuning of hyperparameters of these algorithms is emerging as one of the main challenges. When considering industry, we are evident of increasing popularity of IP model. Basically, for those cases when optimized networks are of commercial interest their details are kept undisclosed. For example, EMVCo (formed by MasterCard and Visa to manage specifications for payment systems and to facilitate worldwide interoperability) nowadays requires deep learning techniques for security evaluations [7]. This has an obvious consequence in security labs creating (and using) neural networks for evaluation of products for their customers.

There are also other reasons for keeping the neural network architectures secret. Often, these pretrained models might provide additional information regarding the training data, which is very sensitive. For example, if the model is trained based on medical record of a patient or social security number, confidential information could be encoded into the network during the training phase. Also, machine learning models that are used for guiding medical treatments are often based on a patient's genotype making this extremely sensitive from the privacy perspective [8]. Hence, determining the layout of the network with trained weights is a desirable target for the attacker. One could ask why would attacker want to reverse engineer the neural network architecture instead of just training the same network on its own. There are several reasons making such approach difficult. First, maybe the attacker does not have access to the same training set in order to train his neural network. Second, as the architectures become more complex, there are more parameters to tune and it could be extremely difficult for the attacker to find the same values.

Our main question relates to the feasibility of reverse engineering such architectures. Although binary analysis can already give useful information about the network, in practical cases, binary readback could be disabled by blocking JTAG access [9]. Still, side-channel analysis remains a viable option. Side-channel analysis has been widely studied in the community of information security and cryptanalysis, due to its potentially devastating impact on otherwise (theoretically) secure algorithms. Concretely, it has been observed that different physical leakages from devices on which cryptography is implemented, such as timing delay, power consumption, and electromagnetic emanation (EM) during the computation of the data will be dependent on the processed internal state and thus data. By statistically combining this physical leakage of the specific internal state and hypothesis on the data being manipulated, it is possible to recover the intermediate state being processed by the device.

In this study, our aim is to highlight the potential vulnerabilities of standard (naive from the security perspective) implementations of neural networks. Here, we consider some of the basic building blocks of neural networks: the number of hidden layers, the basic multiplication operation, and the activation functions.

Assuming that the multiplications are performed on one known and one unknown operand and by observing e.g., the power (or other side-channels, such as electromagnetic radiation) leakage, additional information about the output of the multiplication becomes available. In this case, different hypotheses of the possible values can be correlated with the leakage to recover the unknown input up to a certain precision. We show that for our target implementation, the value of unknown input to the multiplication could be estimated with up to 0.01 precision. This was possible by choosing the suitable leakage model to emulate floating-point operations. The approach can be further strengthened if the attacker is allowed profiling a clone device and characterize all the leakages beforehand.

The complex structure of activation function often leads to conditional branching due to required exponentiation and division operations. Conditional branching introduces input dependent timing differences resulting in different timing behavior for different activation function, allowing function identification. Moreover, simply by observing the side-channel signature, it is possible to deduce number of nodes, and also number of layers in the networks. By using the divide-and-conquer approach for side-channel analysis, the information at each layer could be recovered, and the recovered information can be used as input for recovering the subsequent layers. Consequently, in this work, we show it is possible to recover the layout of unknown networks by exploiting the side-channel information.

The motivation for our work comes from ever more pervasive usage of neural networks in security-critical applications and the fact that the architectures are becoming proprietary knowledge for the security evaluation industry. Thus, reverse engineering a neural net has become a new target for the adversaries and we need better understanding of the vulnerability to side-channel leakages in this case to be able to protect the users' rights and data.

1.1 Related Work

There are many papers considering machine learning and more recently, deep learning for improving the effectiveness of side-channel attacks. For instance, few works have focused to demonstrate the effectiveness of classical profiled side-channel attacks against machine learning [10–12]. Lately, few works explored the power of deep learning in the context of side-channel analysis [13].

On the other hand, using side-channel analysis in order to attack machine learning architectures is much less investigated and to our best knowledge there exist only two papers taking this turn. Hua et al. showed how to reverse engineer two convolutional neural networks, namely AlexNet and SqueezeNet through memory and timing side-channel leaks [14]. Authors measure side-channel through an artificially introduced hardware Trojan. They also need access to original training data for part of the attack, which might not always be available. Lastly, in order to obtain the weight of the neural networks, they attack very specific operation i.e., zero pruning [15], which to an extent is more common for ReLU.

Wei et al. also performed an attack on an FPGA-based convolutional neural network accelerator [16]. They recovered the input image from the collected power consumption traces. The proposed attack exploits a specific design choice i.e., the line buffer in a convolution layer of a CNN. In a nutshell, both previous attacks are performed on specific design choices for neural networks which allow optimized implementations. Conversely, we explore the problem of reverse engineering of neural networks from a more generic perspective and in a grey to black box setting.

1.2 Contribution and Organization

The main contributions of this paper are:

1. We propose a full reverse engineering of neural network parameters based on side-channel attacks. A combination of side-channel attacks are proposed to recover key parameters i.e., activation function, pre-trained weights, number of hidden layers and neurons in each layer. The proposed attack does not need any information of the sensitive training data, which might not be available to the attacker.
2. All the proposed attacks are practically implemented and demonstrated on an embedded microcontroller, allowing full reverse engineering of network architecture.
3. Further, a single trace input recovery attack has been proposed, which recovers dataset when applied on the initial layers. This implies that the attacker can recover all the inputs tested with a known neural network, recovering each input from a single measurement only. Such attacks put user sensitive data at risk.
4. We highlight some interesting aspects of side-channel attacks when dealing with real numbers, unlike cryptography. We show that even a failed side-channel attack due to precision error can provide sensitive information about the target.
5. Finally, we propose a number of mitigation techniques that will render side-channel attacks more difficult.

The rest of this paper is organized as follows. In Section 2, we give details about specific machine learning algorithms we consider and side-channel analysis techniques we use. Section 3 gives results on reverse engineering of various elements of neural networks and Section 4 on input recovery attack. In Section 5, we briefly discuss possible countermeasures one could apply to make our attacks more difficult. Finally, in Section 6, we conclude the paper and discuss potential future research directions.

2 Background

In this section, we give details about artificial neural networks and their building blocks. Next, we discuss side-channel analysis and several types of attacks we use in this paper.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) is an umbrella notion for all computer systems loosely inspired by biological neural networks. Such systems are able to "learn" from examples, which makes them a strong (and very popular) paradigm in the machine learning domain. Any ANN is built from a number of nodes called artificial neurons. The nodes are connected in order to transmit a signal. Usually, in an ANN, the signal at the connection between artificial neurons is a real number and the output of each neuron is calculated as a nonlinear function of the sum of its inputs. Neurons and connections have weights that are adjusted as the learning progresses. Those weights are used to increase or decrease the strength of a signal at a connection. In the rest of this paper, we use the notions artificial neural network, neural network, and network interchangeably.

A very simple type of a neural network is called perceptron. A perceptron is a linear binary classifier applied to the feature vector. Each vector component has an associated weight w_i and each perceptron has a threshold value θ. The output of a perceptron equals "1" if the direct sum between the feature vector and the weight vector is larger than zero and "-1" otherwise. A perceptron classifier works only for data that are linearly separable, i.e., if there is some hyperplane that separates all the positive points from all the negative points [17]. We depict a model of a artificial neuron in Figure 1. In the case of perceptron, the activation function is the step function.

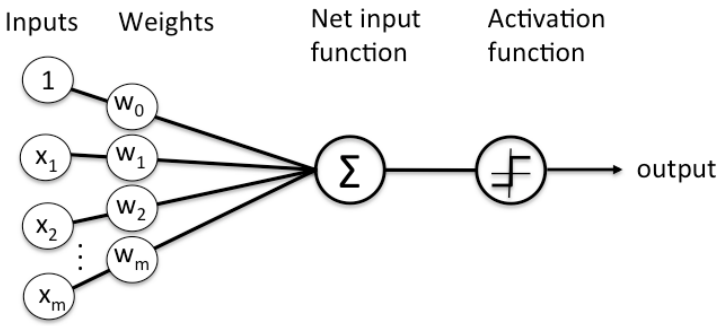


Fig. 1: Depiction of an artificial neuron.

By adding more layers to perceptron, we arrive to the multi layer perceptron algorithm. Multi layer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. It consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one. Consequently, each node in one layer connects with a certain weight w to every node in the following layer. Multi layer perceptron algorithm consists of at least three layers: one input layer, one output layer, and one hidden layer. Those layers must consist of nonlinearly activating nodes [18].

We depict a model of a multi layer perceptron in Figure 2. Note, if there is more than one hidden layer, then it can be considered a deep learning architecture. At the same time, if the activation function for a neuron is the step function, it is easy to show that any number of layers can be reduced to two layers (one input and one output layer). Differing from linear perceptron, MLP can distinguish data that are not linearly separable. 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 [17].

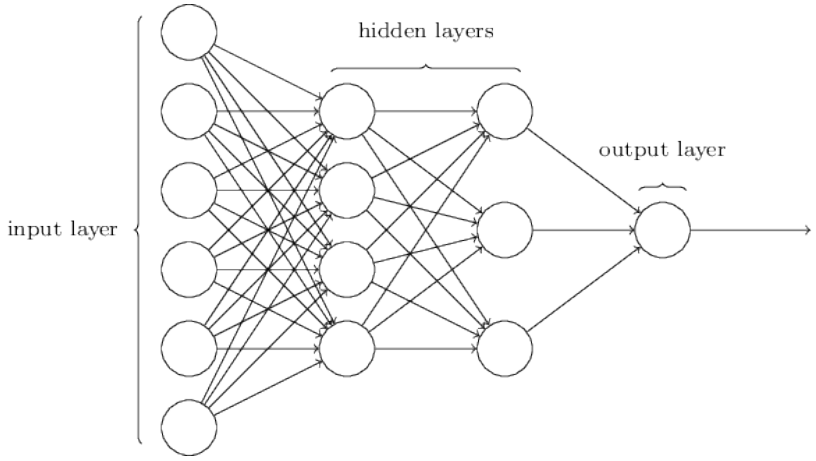


Fig. 2: Multi layer perceptron.

An activation function of a node is a function f defining the output of a node given an input or set of inputs, see Eq. (1). In order for ANN to be able to calculate nontrivial functions using a small number of nodes, we need to use nonlinear activation functions.

$$y = Activation(\sum(weight \cdot input) + bias). \tag{1}$$

In this paper, we consider logistic (sigmoid) function, tanh function, softmax function, and Rectified Linear Unit function. The logistic function is a nonlinear function giving smooth and continuously differentiable results [19]. The range of a sigmoid function is $[0, 1]$, which means that all the values going to the next neuron will have the same sign.

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2}$$

The tanh function is a scaled version of logistic function where the main difference is that it is symmetric over the origin. The tanh function ranges in

$[-1, 1]$.

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{3}$$

The softmax function is a type of sigmoid function able to map values into multiple outputs (e.g., classes). The softmax function is ideally used in the output layer of the classifier in order to obtain the probabilities defining a class for each input [20].

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, for\ j = 1, \dots\ K. \tag{4}$$

The Rectified Linear Unit (ReLU) is a nonlinear function that differing from the previous two activation functions, does not activate all the neurons at the same time [21]. By activating only a subset of neurons at any time, we make the network sparse and easier to compute. Consequently, such properties make ReLU probably the most widely used activation function in ANNs today.

$$f(x) = max(0, x). \tag{5}$$

2.2 Side-channel Analysis

Side-channel Analysis (SCA) exploits weaknesses on the implementation level [22]. More specifically, all computations running on a certain platform result in unintentional physical leakages. Those leakages are a sort of physical signatures from the reaction time, power consumption, and EM emanations released while the device was manipulating data. SCA exploits those physical signatures aiming at the key (secret data) recovery. In its basic form, SCA was proposed to perform key recovery attacks on implementation of cryptography [23, 24]. One advantage of SCA over traditional cryptanalysis is that SCA can apply a divide-and-conquer approach. Thus, instead of testing and recovering the full key at once, SCA can be used to recover small parts of the key independently, exponentially reducing the attack complexity.

However, the scope of SCA is much wider. For example, SCA was recently used to demonstrate IP theft from 3D printers [25]. Based on the analysis technique, several variants exist. In the following, we recall few analysis techniques used later in the paper. Although the following terms suggest power analysis, these techniques apply to other side-channels as well.

Simple Power Analysis (SPA) Simple power analysis, as the name suggest, is the most basic form of SCA [24]. It targets information from a sensitive computation which can be recovered from a single or a few traces.

As a common example, SPA can be used against a straightforward implementation of the RSA algorithm. Namely, the RSA exponentiation is composed of a sequence of square and multiply operations which depends on secret key bit (multiply follows square only when the secret bit is 1, else only square is executed). If square and multiply have distinct physical signatures, by observing

it on an oscilloscope, the adversary can directly read out the key bits. Similar attacks have been applied to secret-key algorithm like AES [26] but then targeting key schedule. In this work, we apply SPA to reverse engineer the architecture of the neural network.

Differential Power Analysis (DPA) DPA is an advanced form of SCA, which applies statistical techniques to recover secret from physical signatures, when SPA is not possible. The attack normally tests for dependencies between actual physical signature (or measurements) and hypothetical physical signature i.e., predictions on intermediate data. The hypothetical signature is based on a leakage model and key hypothesis. With divide-and-conquer approach, parts of the secret key (e.g., one byte) can be tested independently, allowing exhaustive search on key hypothesis. The knowledge of leakage model comes from the adversary's intuition and expertise. Some commonly used leakage models for representative devices are: the Hamming weight in microcontroller and the Hamming distance in FPGA, ASIC, and GPU [27, 28]. Apart from the leakage model, the analysis principle remains unchanged.

As the measurements can be noisy, the adversary often needs many measurements. Next, statistical tests like correlation [29] are applied to identify correct key hypothesis from other wrong hypothesis. As we show later in the paper, DPA is used to recover secret weights from a pre-trained network.

Horizontal Power Analysis (HPA) HPA is another sort of side-channel attack using power as the source of leakage [30]. While DPA recovers the secret key statistically over multiple measurements, HPA is a single trace attack exploiting several elementary operations in a single computation. The idea behind it is that identical data being manipulated even in different computation steps will have the same power signatures and can be recovered by e.g., pattern recognition techniques. HPA can be used against protected implementation, for example with exponent blinding, where adversary is limited to only one measurement. In this paper, we use HPA to perform input recovery attack for a known network where we show that it works very well for medium to large sized networks.

3 Side-Channel Based Reverse Engineering of Neural Networks

As already discussed, side-channel leakages have been frequently used for cryptanalysis, in particular for key recovery attacks in cryptography and for reverse engineering of cryptographic algorithms. In this work, we demonstrate the first application of SCA for reverse engineering of neural networks, with practical measurements.

3.1 Threat Model

The two main goals of this paper are to recover the neural network architecture and its inputs using only side-channel information.

Scenario. We select to work with MLP since 1) it is a commonly used machine learning algorithm in modern applications, see e.g., [31–34]; 2) it consists of fully connected layers which are also occurring in other architectures like convolutional neural networks or recurrent neural networks; and 3) the layers are all identical, which makes it more difficult for SCA and could be consequently considered as the worst-case scenario. We choose our attack to be as generic as possible while discarding common assumptions, which would make the attack easier but also more limited in scope. For instance, we have no assumption on type of inputs or its source, as we work with real numbers. If the inputs are in form of integers (like the MNIST database), the attack becomes easier, since we would not need to recover mantissa bytes and deal with precision. We also assume that the implementation of the machine learning algorithm does not include any side-channel countermeasures, which is a standard setting.

Attacker's capability. We consider a passive attacker who is only capable of acquiring measurements of the device while operating "normally" and not interfering with its operations. We consider two settings:

1. Attacker does not know the architecture of the used network but can feed random (or known) inputs to the architecture

   An adequate use case would be when the attacker legally acquires a copy of the network in a black box setting and aims at recovering its internal details, for IP theft. The attacker can query the device with random/chosen inputs and perform side-channel measurements of the processing. The goals for this setting are to reverse engineer information about neural network architecture: number of layers, number of outputs, activation functions, weights in the network.
2. Attacker knows the architecture but does not know the inputs to it

   A suitable use case is where a secret dataset is tested with a public MLP network. The input can correspond to sensitive data such as medical records of patients. The goal for this setting is to obtain the inputs (the data to be classified) to the network and we achieve this with a single measurement only.

3.2 Experimental Setup

Here we describe our measurement setup. The target platform is an Atmel ATmega328P, and the side-channel measurements are collected during the execution of the classification and they are captured using the Lecroy WaveRunner 610zi oscilloscope. The oscilloscope measurements are synchronized with the operations by common hand shaking signals like start and stop of computation. To

further improve the quality of measurements, we opened the chip package mechanically (see Figure 3a). An RF-U 5-2 near field electromagnetic (EM) probe from Langer is used to collect the measurements (see Figure 3b). Note that EM measurements also allow to observe the timing of all the operations and thus the setup allows for timing side-channels based analysis as well. The setup is depicted in Figure 3c.

Our choice of the target platform is motivated by the quality of available measurements. With proper adjustments of our setup we are able to achieve a high signal-to-noise ratio (SNR), allowing us to focus on the analysis techniques. For a different platform, the leakage model could change, but this would not limit our approach and methodology. In fact, those leakage models are well known for other common platforms like FPGA [27] and GPU [28]. Moreover, low SNR of the measurement might force the adversary to increase the number of measurements and apply signal pre-processing techniques, but the principles of the analysis remain valid.

As already stated above, the exploited leakage model of the target device is the Hamming weight (HW) model. A microcontroller loads sensitive data to a data bus to perform indicated instructions. This data bus is pre-charged to all '0's' before every instruction. Note that data bus being pre-charged is a natural behavior of the microcontroller and not a vulnerability introduced by the attacker. Thus, the new power consumption (or EM radiation) is modeled as the number of bits equal to '1' in the loaded data. In other words, the power consumption of loading data x is:

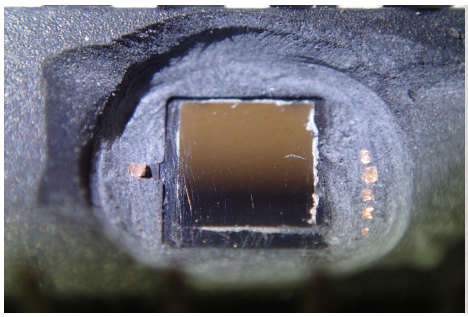
$$HW(x) = \sum_{i=1}^{n} x_i \ , \tag{6}$$

where x_i represents the i^{th} bit of x. In our case, it is the secret pre-trained weight which is regularly loaded from memory for processing and results in the HW leakage. To conduct the side-channel analysis, we perform divide-and-conquer approach, where we target each operation separately. The full recovery process is described in Section 3.6.

Several pre-trained networks are implemented on the board. The training phase is conducted offline, and the trained network is then implemented in C language and compiled on the microcontroller. In our experiments, we consider multi layer perceptron architectures consisting of different number of layers and nodes in those layers. Note, with our approach there is no limit in the number of layers or nodes we can attack. The methodology is developed to demonstrate that the key parameters of the network, namely the weights and activation functions can be reverse engineered. Further experiments are conducted on deep neural networks with three hidden layers. We emphasize that the method we use can be applied to larger networks as well.

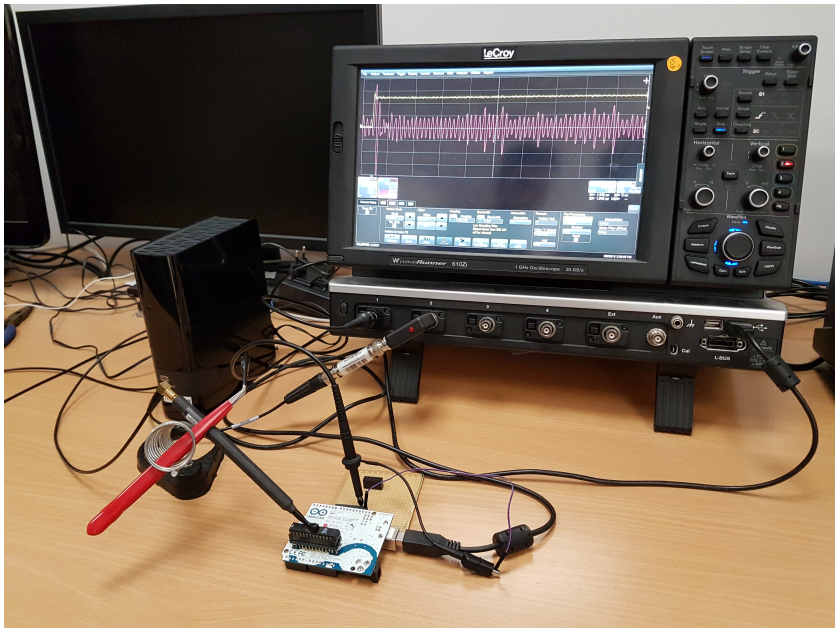
3.3 Reverse Engineering the Activation Function

We remind the reader that nonlinear activation functions are necessary in order represent nonlinear functions with a small number of nodes in a network. As such,



(a) Target 8-bit microcontroller mechanically decapsulated

(b) Langer RF-U 5-2 Near Field Electromagnetic passive Probe



(c) The complete measurement setup

Fig. 3: Experimental Setup

they are elements used in virtually any neural network architecture today [1,6]. If the attacker is able to deduce the information on the type of used activation functions, he/she can use that knowledge together with information about input values to deduce the behavior of the whole network.

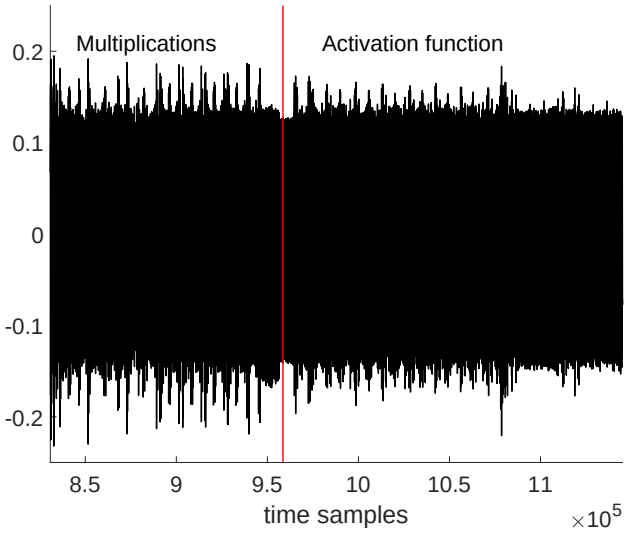
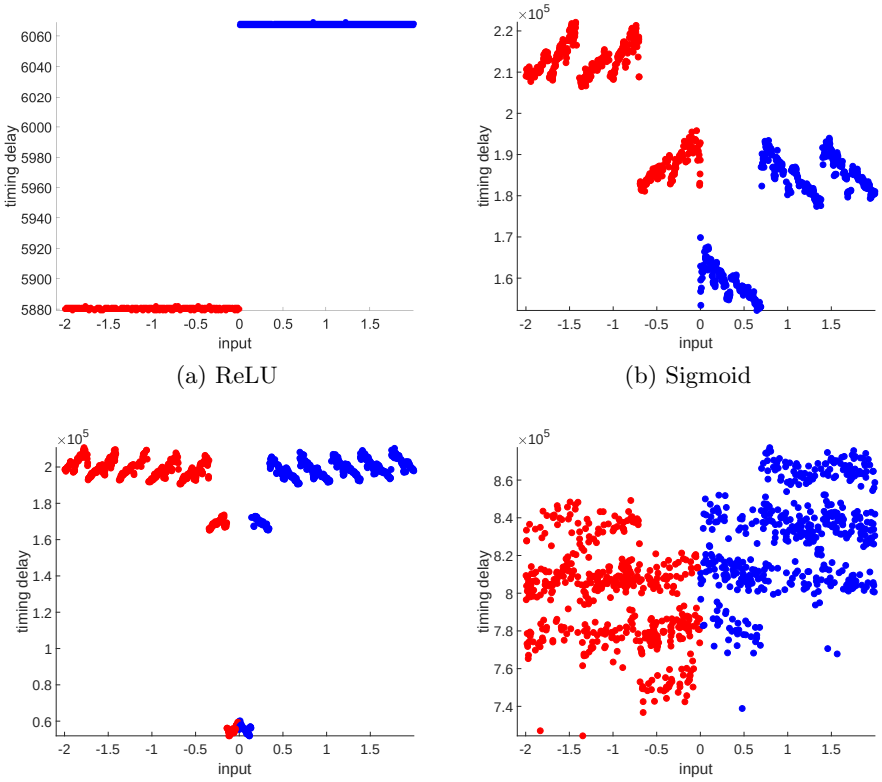


Fig. 4: Observing pattern and timing of multiplication and activation function

We analyze the side-channel leakage from different activation functions. We consider the most commonly used activation functions, namely ReLU, sigmoid, tanh, and softmax [19,21]. The timing behavior can be observed directly on the EM trace. For instance, as shown later in Figure 9a, a multiplication is followed by activation with individual signatures. For a similar architecture, we test different variants with each activation function. We collect EM traces and measure the timing of the activation function computation from the measurements. The measurements are taken when the network is processing random inputs in the range, i.e., $x \in \{-2, 2\}$. A total of 2 000 EM measurements are captured for each activation function. As shown in Figure 4, the timing behavior of the four tested activation functions have distinct signatures allowing easy characterization.

Different inputs result in different processing times. Moreover, the timing behavior for the same inputs largely vary depending on the activation function. For example, we can observe that ReLU will require the shortest amount of time, due to its simplicity (see Figure 5a). On the other hand, tanh and sigmoid might have similar timing delay, but with different pattern considering the input (see Figure 5b and Figure 5b), where tanh is more symmetrical in pattern compared to sigmoid, for both positive and negative inputs. We can observe that softmax



(a) ReLU (b) Sigmoid

(c) Tanh (d) Softmax

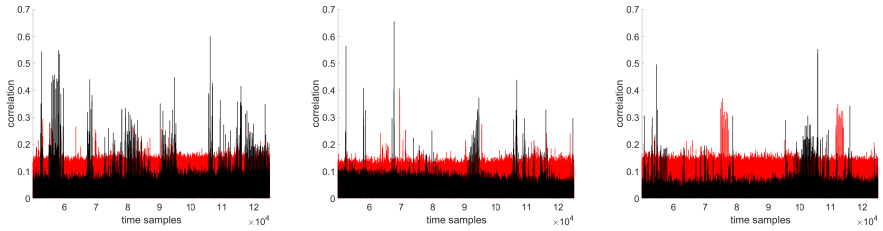
Fig. 5: Timing behavior for different activation functions

function will require most of the processing time, since it requires the exponentiation operation which also depends on the number of neurons in the output layer. As neural network algorithms are often optimized for performance, presence of such timing side-channels is often ignored. Function such as tanh or sigmoid requires computation of e^x and division and it is known that such functions are difficult to implement in constant time. In addition, constant time implementations might lead to a substantial performance degradation. Other activation functions can be characterized similarly. Finally, Table 1 presents the minimum, maximum, and mean computation time for each activation function over captured 2 000 measurements. While ReLU is fastest, the timing difference of each function stands out individually, thus allowing a straightforward recovery.

Table 1: Minimum, Maximum, and Mean computation time (in ns) for different activation functions

Activation Function	Minimum	Maximum	Mean
ReLU	5 879	6 069	5 975
Sigmoid	152 155	222 102	189 144
Tanh	51 909	210 663	184 864
Softmax	724 366	877 194	813 712

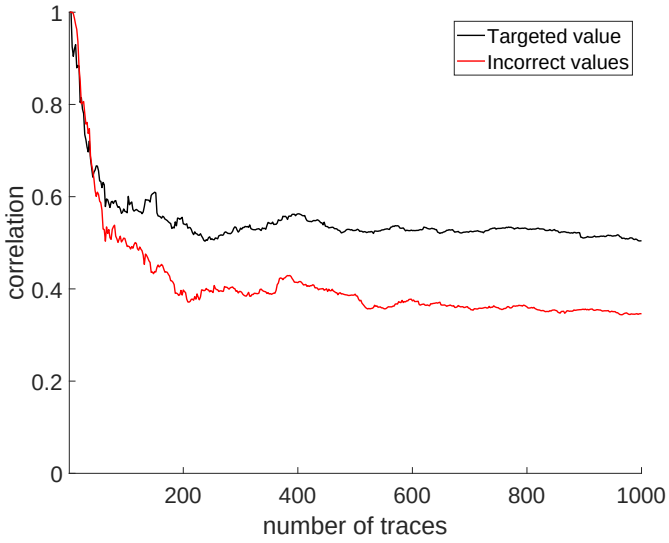
3.4 Reverse Engineering of the Multiplication Operation



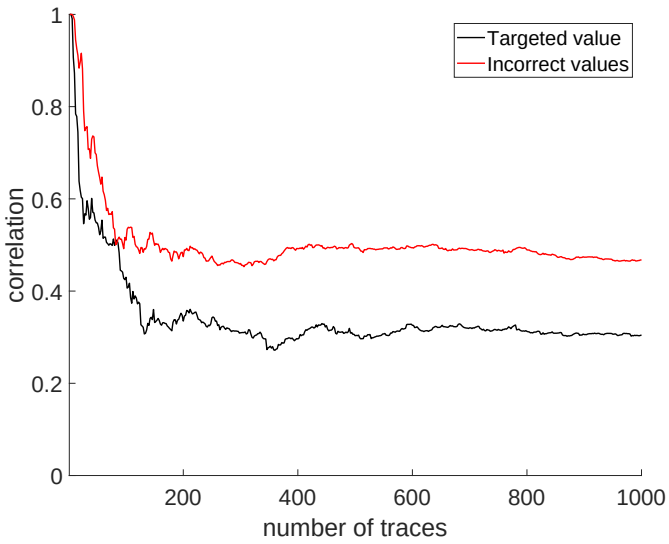
(a) First byte mantissa for weight = 2.43 (b) Second byte mantissa for weight = 2.43 (c) Third byte mantissa for weight = 2.43

Fig. 6: Correlation of different weights candidate on multiplication operation

A well-trained network can be of a significant value. What distinguishes a good versus poorly trained network for a given architecture are the weights. With fine-tuned weights, we can improve the accuracy of the network, which has both commercial and academic interest. In the following, we demonstrate a way to recover those weights by using SCA.



(a) weight = 1.635



(b) weight = 0.890

Fig. 7: Correlation comparison between correct and incorrect mantissa of the weights

For the recovery of the weights, we use the Correlation Power Analysis (CPA) i.e., a variant of DPA using the Pearson's correlation as a statistical test. CPA targets the multiplication $m = x \cdot w$ of a known input x with a secret weight w. Using the HW model, the adversary correlates the activity of the predicted output m for all hypothesis of the weight. Thus, the attack computes $\rho(t, w)$, for all hypothesis of the weight w, where ρ is the Pearson correlation coefficient and t is the side-channel measurement. The correct value of the weight w will result in a higher correlation standing in this way out from all other wrong hypotheses w^*, given enough measurements. Although the attack concept remains the same as in the case of attack on cryptographic ciphers, the actual attack used here is quite different. While cryptographic operations are always performed on fixed length integers, in ANN we are dealing with real numbers.

We start by analyzing the way that the compiler is handling floating-point operations for our target. The generated assembly is shown in Table 2, which confirms the usage of IEEE 754 compatible representation as stated above. The knowledge of the representation allows one to better estimate the leakage behavior. Since the target device is an 8-bit microcontroller, the representation follows 32-bit pattern $(b_{31}...b_0)$, which is stored in 4 registers. The 32-bit consist of: 1 sign bit (b_{31}), 8 biased exponent bits $(b_{30}...b_{23})$ and 23 mantissa (fractional) bits $(b_{22}...b_0)$. It can be formulated as:

$$(-1)^{b_{31}} \times 2^{(b_{30}...b_{23})_2 - 127} \times (1.b_{22}...b_0)_2.$$

For example, the value 2.43 can be expressed as
$(-1)^0 \times 2^{(1000000)_2 - 127} \times (1.00110111000010100011111)_2$. The measurement t is considered when the computed result m is stored back to the memory, leaking in the HW model i.e., $HW(m)$. Since 32-bit m is split into individual 8-bits, each byte of m is recovered individually. Hence, by recovering this representation, it is enough to recover the estimation of the real number value.

To implement the attack two different approaches can be considered. The first approach is to build the hypothesis on the weight directly. For this experiment, we target the result of the multiplication m of known input values x and unknown weight w. For every input, we assume different possibilities for weight values. We then perform the multiplication, and estimate the IEEE 754 binary representation of the output. To deal with the growing number of possible candidates for the unknown weight w, we assume that the weight will be bounded in a range $[-N, N]$, where N is a parameter chosen by the adversary, and the size of possible candidates is denoted as $s = 2N/p$, where p is the precision when dealing with floating-point numbers.

Then, we perform the recovery of the 23-bit mantissa of the weight. The sign and exponent could be recovered separately. Thus, we are observing the leakage of 3 registers, and based on the best CPA results for each register, we can reconstruct the mantissa. Note that the recovered mantissa does not directly relate to the weight, but with the recovery of the sign and exponent, we could obtain the unique weight value. The traces are measured when the microcontroller performs secret weight multiplication with uniformly random

Table 2: Code snippet of the returned assembly for multiplication: $x = x \cdot w (=$ 2.36 or 0x3D0A1740 in IEEE 754 representation). The multiplication itself is not shown here, but from the registers assignment, our leakage model assumption holds.

#	Instruction	Comment
11a	ldd r22, Y+1	0x01
11c	ldd r23, Y+2	0x02
11e	ldd r24, Y+3	0x03
120	ldd r25, Y+4	0x04
122	ldi r18, 0x3D	61
124	ldi r19, 0x0A	10
126	ldi r20, 0x17	23
128	ldi r21, 0x40	64
12a	call 0xa0a	multiplication
12e	std Y+1, r22	0x01
130	std Y+2, r23	0x02
132	std Y+3, r24	0x03
134	std Y+4, r25	0x04

values between -1 and 1 ($x \in \{-1, 1\}$) to emulate normalized input values. We set $N = 5$ and to reduce the number of possible candidates, we assume that each floating-point value will have precision of 2 decimal points, $p = 0.01$. Since we are dealing with mantissa only, we can then only check the weight candidates in range $[0, N]$, thus reducing the number of possible candidates.

In Figure 6, we show the result of the correlation for each byte with the measured traces. Horizontal axis shows time of execution and vertical axis correlation. The experiments were conducted on 1 000 traces for each case. In the figure, the black plot denotes the correlation of the "correct" mantissa weight ($|m(\hat{w}) - m(w)| < 0.01$), whereas the red plots are from all other weight candidates in the range described earlier. Since we are only attacking mantissa in this phase, several weight candidates might have similar correlation peaks. After the recovery of the mantissa, the sign bit and exponent can be recovered similarly, which narrows down the list candidate to a unique weight. Another observation is that the correlation value is not very high and scattered across different clock cycles. This is due to the reason that the measurements are noisy and since the operation is not in constant time, the interesting time samples are distributed across multiple clock cycles. Nevertheless, it is shown that the side-channel leakage can be exploited to recover the weight up to certain precision. Multivariate side channel analysis [35] can be considered if distributed samples hinder recovery.

We emphasize that attacking real numbers as in the case of weights of ANN can be simpler than attacking cryptography. This is because cryptography works on fixed length integers and exact values must be recovered. When attacking real numbers, small precision errors due to rounding off the intermediate values still results in useful information.

To deal with more precise values, we can target the mantissa multiplication operation directly. In this case, the search space can either be $[0, 2^{23}-1]$ to cover all possible values for the mantissa (hence, more computational resources will be required) or we can focus only on the most significant bits of the mantissa (lesser candidates but also with lesser precision). Since the 7 most significant bits of the mantissa are processed in the same register, we can aim to target only those bits, assigning the rest to 0. Thus, our search space is now $[0, 2^7-1]$. The mantissa multiplication can be performed as $1.mantissa_x \times 1.mantissa_w$, then taking the 23 most significant bits after the leading 1, and normalization (updating the exponent if the result overflow) if necessary.

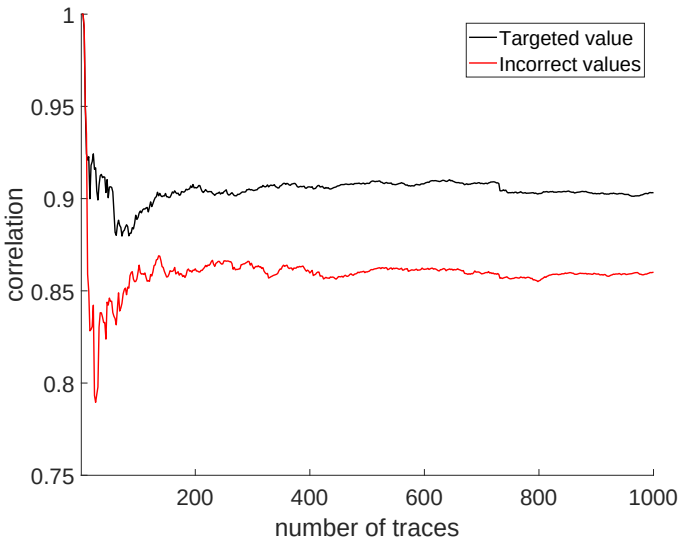
In Figure 7, we show the result of the correlation between the HW of the first 7-bit mantissa of the weight with the traces. Except Figure 7b, the other results show that the correct mantissa can be recovered. The most interesting result is shown in Figure 7b, which at the first glance looks as a failure of the attack. Here, the target value of the mantissa is **1100011**10...10, while the attack recovers **1100100**000..00. Considering the sign and exponents, the attack recovers *0.890625* instead of *0.89*, i.e., a precision error at 4^{th} place after decimal point. Thus, in both cases, we have shown that we can recover the weights from the SCA leakage.

Lastly, in Figure 8, we show the composite recovery of 2 bytes of the weight representation i.e., a low precision setting where we recover sign, exponent and most significant part of mantissa. Again, the targeted (correct) weight can be easily distinguished from the other candidates. Hence, once all the necessary information has been recovered, the weight can be reconstructed accordingly.

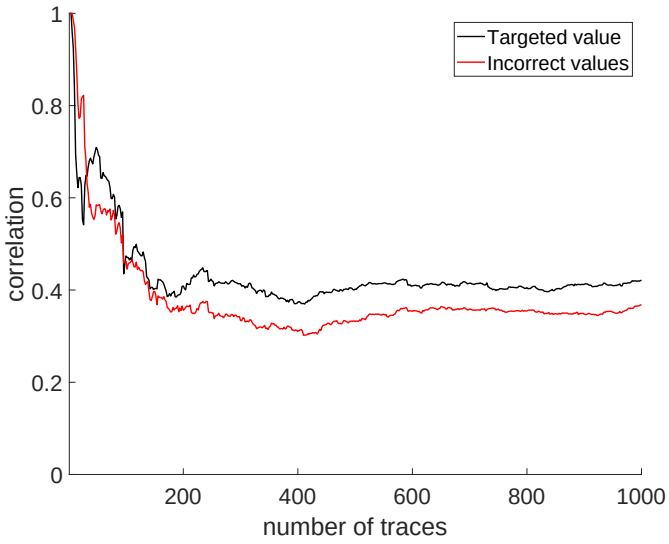
3.5 Reverse Engineering the Number of Neurons and Layers

After the recovery of the weights and the activation functions, in this step, we use SCA to determine the structure of the network. Mainly, we are interested to see if we can recover the number of hidden layers and number of neurons for each layer. To perform the reverse engineering, we first use SPA. SPA is the simplest form of SCA which allows information recovery in a single (or a few) traces with methods as simple as visual inspection. The analysis is performed on three networks with different layouts.

The first analyzed network is an MLP with one hidden layer with 6 neurons. The EM trace corresponding to processing of a randomly chosen input is shown in Figure 9a. By looking at the EM trace, the number of neurons can be easily counted. The observability arises from the fact that multiplication operation and the activation function (in this case, it is the Sigmoid function) have completely different leakage signatures. Similarly, the structures of deeper networks are also shown in Figure 9b and Figure 9c. The recovery of output layer then provides information on the number of output classes. However, distinguishing different layers might be difficult, since the leakage pattern is only dependent on multiplication and activation function, which are usually present in most of the layers. We observe minor features allowing identification of layer boundaries but only

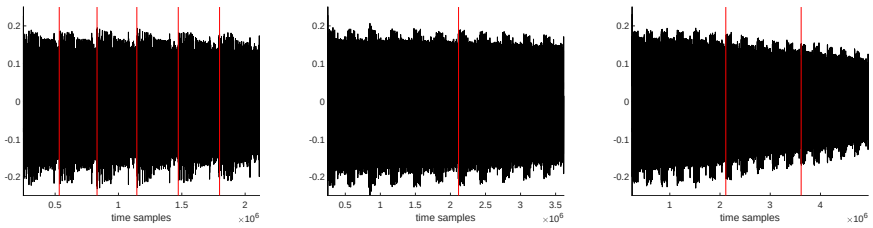


(a) First byte recovery (sign and 7-bit exponent)



(b) Second byte recovery (lsb exponent and mantissa)

Fig. 8: Recovery of the weight



(a) One hidden layer with 6 neurons (b) 2 hidden layers (6 and 5 neurons each) (c) 3 hidden layers (6,5,5 neurons each)

Fig. 9: SPA on hidden layers

with low confidence. Hence, we develop a different approach based on CPA to identify layer boundaries.

The experiments follow similar methodology as in the previous experiments. To determine if the targeted neuron is in the same layer as previously attacked neurons, or in the next layer, we perform a weight recovery using two sets of data.

Let us assume that we are targeting the first hidden layer (the same approach can be done on different layers as well). Assume that the input is a vector of length N_0, so the input x can be represented $x = \{x_1, ..., x_{N_0}\}$. For the targeted neuron y_n in the hidden layer, perform the weight recovery on 2 different hypotheses. For the first hypothesis, assume that the y_n is in the first hidden layer. Perform weight recovery individually using x_i, for $1 \leq i \leq N_0$. For the second hypothesis, assume that y_n is in the next hidden layer (the second hidden layer). Perform weight recovery individually using y_i, for $1 \leq i \leq (n-i)$. For each hypothesis, record the maximum (absolute) correlation value, and compare both. Since the correlation depends on both inputs to the multiplication operation, incorrect hypothesis will result in lower correlation value. Thus, this can be used to identify layer boundaries.

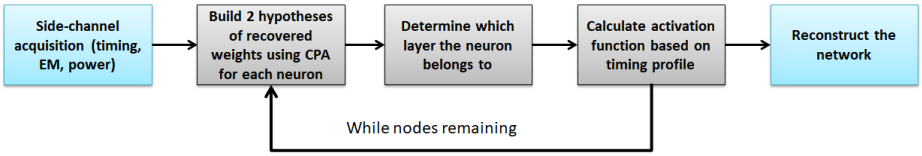


Fig. 10: Methodology to reverse engineer the target neural network

3.6 Recovery of the Full Network Layout

The combination of previously developed individual techniques can thereafter result in a full reverse engineering of the network. The full network recovery is performed layer by layer, and for each layer, the weights for each neuron have to be recovered one at a time. The reverse engineering is performed with the following steps:

1. The first step is to recover the weight of each connection starting from the input layer and the first hidden layer. After the weights are recovered, the output of the sum of multiplication can be retrieved. This information provides us with the input to the activation function.
2. In order to determine output of sum of the multiplications, the number of neurons in the layer must be known. This can be recovered by the combination of SPA and DPA technique described in the previous subsection, in parallel with the weight recovery. When all the weights of the first hidden layer are recovered, the following steps are executed.
3. Using the same set of traces, timing patterns for different inputs to the activation function can be build, similar to Figure 5. Timing patterns or average timing can then be compared with the profile of each function to determine the activation function. Afterwards, the output of the activation function can be computed, which provides the input to the next layer.
4. The same steps are repeated in the subsequent layers until the structure of the full network is recovered.

The whole procedure is depicted in Figure 10.

4 Single Trace Input Recovery Attack on MLP

In previous section, the methodology to reverse engineer a neural network has been described and practically demonstrated. In this section, we consider an alternate scenario, where unknown or secret input is fed to a known network. By known network, we mean that the architecture and weights are either public or known to the adversary (e.g., recovered by reverse engineering). Generally, it can be extremely complex to recover the input by observing outputs from a known network. It involves several classifications in order to solve a system of equations, while some of the functions might not be invertible, i.e., ReLU. When considering theoretical attacks, the system of equations can soon become unmanageable as the architecture of the network becomes complex.

The proposed attack targets the multiplication operation in the first hidden layer. It is exactly the opposite of the previous weight recovery attack, as the weights w are known while input x is unknown. However, there is strong limitation with this attack. As x changes from one measurement to another, information learned from one measurement cannot be used with another measurement, preventing any statistical analysis. In this case, the adversary is forced to exploit all the measurements from a single measurement. Thus, to perform

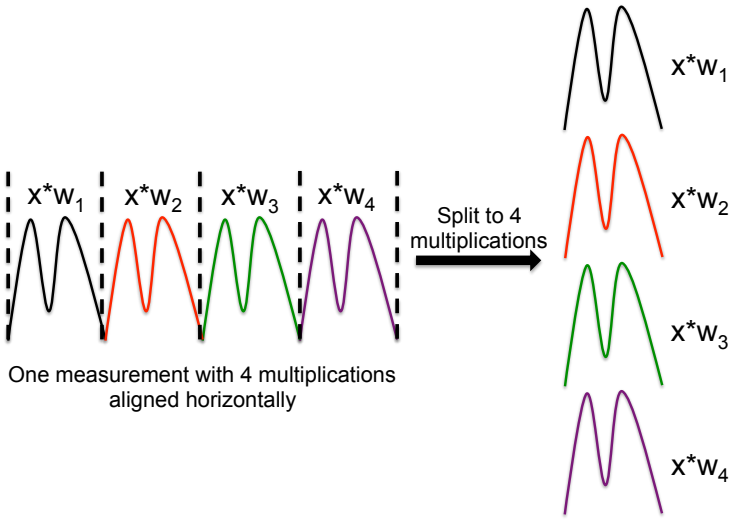
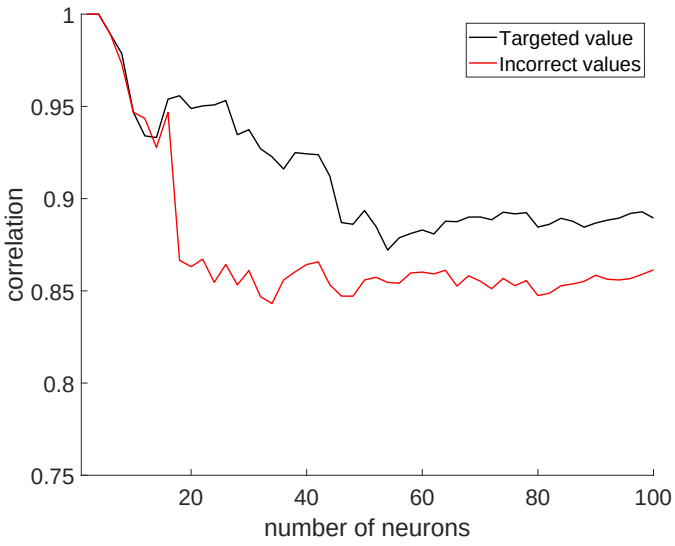


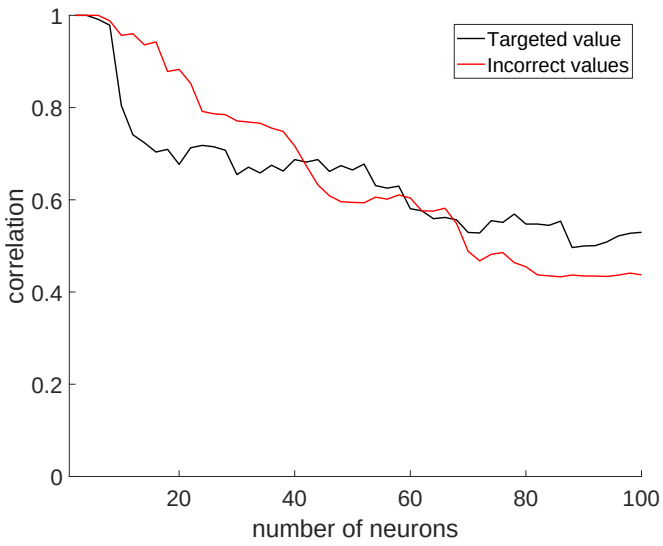
Fig. 11: Illustration of a recovery of multiple measurements from a single measurement processing several elementary operations sequentially

information exploitation over a single measurement, we use HPA. The weights in the first hidden layer are all multiplied with the same input x, one after the other. Drawing analogy with SCA on cryptography, several known plaintexts (weights in case of MLP) are processed for a single unknown key (input x here). The only difference is that all the processing is done in different parts of a single trace. An input recovery attack was proposed in [16], which requires multiple traces targeting a line buffer, which is an optimization oriented design choice. Contrary, our proposed attack targets the generic multiplication in a single trace setting.

We measured the EM trace to perform an input recovery attack. M multiplications, corresponding to M different weights (or neurons), in the first hidden layer were isolated. An illustrative example is shown in Figure 11 where $M = 4$ traces corresponding to 4 weights are recovered from a single trace. Thus, a single trace is cut into M smaller traces, each one corresponding to one multiplication with an associated weight. Next, the value of input is statistically inferred by applying a standard DPA on the N smaller traces. The results are shown in Figure 12 for different bytes of the same input. The black curve shows the correlation of the correct input while all wrong inputs are represented in red. The attack needs 20 or more multiplications to reliably recover the input. This means that in the current setting, the proposed attack works very well on medium to large sized networks, with at least 40 neurons in the first hidden layer (which is no issue in modern architectures used today).



(a) First byte recovery (sign and 7-bit exponent)



(b) Second byte recovery (lsb exponent and mantissa)

Fig. 12: Input recovery attack on the initial layer

5 Mitigations

As demonstrated above, various side-channel attacks can be applied to reverse engineer certain components of a pre-trained network. To mitigate such a recovery, several countermeasures can be deployed:

1. Hidden layers of an MLP must be executed in sequence but the multiplication operation in individual neurons within a layer can be executed independently. An example is shuffling [36] as a well-studied side-channel countermeasure. It involves shuffling/permuting the order of execution of independent sub-operations. For example, given N sub-operations $(1, \ldots, N)$ and a random permutation σ, the order of execution becomes $(\sigma(1), \ldots, \sigma(N))$ instead. In this case, we propose to shuffle the order of multiplications of individual neurons within a hidden layer during every classification step. Shuffling modifies the time window of operations from one execution to another, mitigating a classical DPA attack.
2. Weight recovery as well as the single trace input recovery can benefit from application of masking countermeasures [35, 37]. Masking is another widely studied side-channel countermeasure that is even accompanied by a formal proof of security. It involves mixing of sensitive computations with random numbers to remove the dependencies between actual data and side-channel signature, thus preventing the attack. For every operation $f(x, w)$, it is transformed into $f_m(x \oplus m_1, w \oplus m_2) = f(x, w) \oplus m$, where m, m_1, m_2 are uniformly drawn random mask, and f_m is the masked function which apply mask m at the output of f, given masked inputs $x \oplus m_1$ and $w \oplus m_2$. If each neuron is individually masked with an independently drawn uniformly random mask for every iteration and every neuron, the proposed attacks can be prevented. However, this might result in a substantial performance penalty.
3. The proposed attack on activation functions is possible due to the non-constant timing behavior. Mostly considered activation functions perform exponentiation operation. Implementation of constant time exponentiation has been widely studied in the domain of public key cryptography [38]. These well-studied ideas can be adjusted to implement constant time activation function processing.

Clearly, all those countermeasures come with an area and performance cost. In particular, shuffling and masking require a true random number generator that is typically very expensive in terms of area and performance. Similarly, constant time implementations of exponentiation [39] also come at performance efficiency degradation. Thus, the optimal choice of protection mechanism should be done after a systematic resource and performance evaluation study.

6 Further Discussions and Conclusions

Neural networks are widely used machine learning family of algorithms due to its versatility across domains. Their effectiveness depends on the chosen architecture and fine-tuned parameters along with the trained weights, which can

be a proprietary information. In this work, we practically demonstrate reverse engineering of a neural network using side-channel analysis techniques. Practical attacks are performed on measured data corresponding to chosen networks. To make our setting more general, we do not assume any specific form of the input data (except that inputs are real values).

We conclude that using an appropriate combination of SPA and DPA techniques, all sensitive parameters of the network can be recovered. Moreover, a powerful HPA method is used to recover secret inputs from a known network in a single shot side-channel analysis.

Multi layer perceptron architectures are widely used but arguably not the most common choice in state-of-the-art applications. Modern deep learning techniques like convolutional neural networks or recurrent neural networks recently took over and judging on the results they will remain as preferred methods of choice in coming years. Yet, even those networks use the same activation functions we consider here as well as the fully connected layers (the difference is that they also have other types of layers). Since we are able to differentiate between the same type of layers in architectures, we expect the difference to be even more profound when comparing with other layer types.

When considering the weight vectors, here we consider the case where each node has a separate weight. Convolutional neural networks can actually also share those weights to lower the degree of the problem. The same technique we use here to obtain the independent weights can be used to obtain the shared weights (with in the worst case scenario, multiple unnecessary calculations for those shared weights).

The proposed attacks are both generic in nature and more powerful than the two previous works in this direction. Finally, suggestions on countermeasures are provided to help designer mitigate such threats. However, the proposed countermeasures are borrowed mainly from side-channel literature and can incur huge overheads. Nevertheless, we believe that they could motivate further research on optimized and effective countermeasures for neural networks. Besides continuing working on countermeasures, as the main future research goal we envision the need to explore other types of layers, like convolution layers or max pooling layers.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. NIPS'12, USA, Curran Associates Inc. (2012) 1097–1105
2. Kober, J., Peters, J. In: Reinforcement Learning in Robotics: A Survey. Volume 12. Springer, Berlin, Germany (2012) 579–610
3. Teufl, P., Payer, U., Lackner, G.: From nlp (natural language processing) to mlp (machine language processing). In Kotenko, I., Skormin, V., eds.: Computer Network Security, Berlin, Heidelberg, Springer Berlin Heidelberg (2010) 256–269

4. 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: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17, New York, NY, USA, ACM (2017) 363–376
5. Kučera, M., Tsankov, P., Gehr, T., Guarnieri, M., Vechev, M.: Synthesis of probabilistic privacy enforcement. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17, New York, NY, USA, ACM (2017) 391–408
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015)
7. Riscure: https://www.riscure.com/blog/automated-neural-network-construction-genetic-algorithm/ (2018)
8. Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., Ristenpart., T.: Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In: USENIX Security. (2014) 17–32
9. Khan, A., Goodhue, G., Shrivastava, P., Van Der Veer, B., Varney, R., Nagaraj, P.: Embedded memory protection (November 22 2011) US Patent 8,065,512.
10. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: International Workshop on Constructive Side-Channel Analysis and Secure Design, Springer (2015) 20–33
11. Jap, D., Stöttinger, M., Bhasin, S.: Support vector regression: exploiting machine learning techniques for leakage modeling. In: Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy, ACM (2015) 2
12. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: Neural Networks (IJCNN), 2017 International Joint Conference on, IEEE (2017) 4095–4102
13. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, Springer (2016) 3–26
14. Hua, W., Zhang, Z., , Suh, G.E.: Reverse engineering convolutional neural networks through side-channel information leaks) (2018) preprint.
15. Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S.W., Dally, W.J.: Scnn: An accelerator for compressed-sparse convolutional neural networks. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). (June 2017) 27–40
16. Wei, L., Liu, Y., Luo, B., Li, Y., Xu, Q.: I know what you see: Power side-channel attack on convolutional neural network accelerators. CoRR **abs/1803.05847** (2018)
17. Mitchell, T.M.: Machine Learning. 1 edn. McGraw-Hill, Inc., New York, NY, USA (1997)
18. Collobert, R., Bengio, S.: Links Between Perceptrons, MLPs and SVMs. In: Proceedings of the Twenty-first International Conference on Machine Learning. ICML '04, New York, NY, USA, ACM (2004) 23–
19. Haykin, S.: Neural Networks: A Comprehensive Foundation. 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (1998)
20. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006)

21. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10, USA, Omnipress (2010) 807–814
22. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (December 2006) ISBN 0-387-30857-1, `http://www.dpabook.org/`.
23. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Annual International Cryptology Conference, Springer (1996) 104–113
24. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference, Springer (1999) 388–397
25. Faruque, A., Abdullah, M., Chhetri, S.R., Canedo, A., Wan, J.: Acoustic side-channel attacks on additive manufacturing systems. In: Proceedings of the 7th International Conference on Cyber-Physical Systems, IEEE Press (2016) 19
26. Mangard, S.: A simple power-analysis (spa) attack on implementations of the aes key expansion. In: International Conference on Information Security and Cryptology, Springer (2002) 343–358
27. Bhasin, S., Guilley, S., Heuser, A., Danger, J.L.: From cryptography to hardware: analyzing and protecting embedded xilinx bram for cryptographic applications. Journal of Cryptographic Engineering **3**(4) (2013) 213–225
28. Luo, C., Fei, Y., Luo, P., Mukherjee, S., Kaeli, D.: Side-channel power analysis of a gpu aes implementation. In: Computer Design (ICCD), 2015 33rd IEEE International Conference on, IEEE (2015) 281–288
29. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2004) 16–29
30. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: International Conference on Information and Communications Security, Springer (2010) 46–61
31. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers (2017) 1–1
32. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). (May 2015) 106–111
33. 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). (Dec 2016) 848–852
34. Thomas, P., Suhner, M.C.: A new multilayer perceptron pruning algorithm for classification and regression applications. Neural Processing Letters **42**(2) (Oct 2015) 437–458
35. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2013) 142–159
36. Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., Standaert, F.X.: Shuffling against side-channel attacks: A comprehensive study with cautionary note. In: International Conference on the Theory and Application of Cryptology and Information Security, Springer (2012) 740–757
37. Coron, J.S., Goubin, L.: On boolean and arithmetic masking against differential power analysis. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2000) 231–237

38. Hachez, G., Quisquater, J.J.: Montgomery exponentiation with no final subtractions: Improved results. In: International Workshop on Cryptographic Hardware and Embedded Systems, Springer (2000) 293–301
39. Al Hasib, A., Haque, A.A.M.M.: A comparative study of the performance and security issues of aes and rsa cryptography. In: Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on. Volume 2., IEEE (2008) 505–510