Performance Analysis of Multilayer Perceptron in Profiling Side-Channel Analysis

Léo Weissbart[1,2](✉)

[1] Delft University of Technology, Delft, The Netherlands
weissbart@cs.ru.nl
[2] Digital Security Group, Radboud University, Nijmegen, The Netherlands

Abstract. In profiling side-channel analysis, machine learning-based analysis nowadays offers the most powerful performance. This holds especially for techniques stemming from the neural network family: multilayer perceptron and convolutional neural networks. Convolutional neural networks are often favored as results suggest better performance, especially in scenarios where targets are protected with countermeasures. Multilayer perceptron receives significantly less attention, and researchers seem less interested in this method, narrowing the results in the literature to comparisons with convolutional neural networks. On the other hand, a multilayer perceptron has a much simpler structure, enabling easier hyperparameter tuning and, hopefully, contributing to the explainability of this neural network inner working.

We investigate the behavior of a multilayer perceptron in the context of the side-channel analysis of AES. By exploring the sensitivity of multilayer perceptron hyperparameters over the attack's performance, we aim to provide a better understanding of successful hyperparameters tuning and, ultimately, this algorithm's performance. Our results show that MLP (with a proper hyperparameter tuning) can easily break implementations with a random delay or masking countermeasures. This work aims to reiterate the power of simpler neural network techniques in the profiled SCA.

1 Introduction

Side-channel analysis (SCA) exploits weaknesses in cryptographic algorithms' physical implementations rather than the algorithms' mathematical properties [16]. There, SCA correlates secret information with unintentional leakages like timing [13], power dissipation [14], and electromagnetic (EM) radiation [25]. One standard division of SCA is into non-profiling (direct) attacks and profiling (two-stage) attacks. Profiling SCA is the worst-case security analysis as it considers the most powerful side-channel attacker with access to a clone device (where keys can be chosen and known by the attacker). During the past few

J. Zhou et al. (Eds.): ACNS 2020 Workshops, LNCS 12418, pp. 198–216, 2020.
https://doi.org/10.1007/978-3-030-61638-0_12

years, numerous works showed the potential and strength of machine learning in profiling side-channel analysis. Across various targets and scenarios, researchers were able to show that machine learning can outperform other techniques considered state-of-the-art in the SCA community [2,15]. More interestingly, some machine learning techniques are successful, even on implementations protected with countermeasures [2,12]. There, in the spotlight are techniques from the neural network family, most notably, multilayer perceptron (MLP) and convolutional neural networks (CNNs).

When considering the attack success, we commonly take into account only the performance as measured by the number of traces needed to obtain the key. While this is an important criterion, it should not be the only one. For instance, attack complexity (complexity of tuning and training a model) and interpretability of the attack are also essential but much less researched. For instance, CNNs are often showed to perform better than MLPs in SCA's context [2,15,22], as they make the training of a model more versatile and alleviate the feature engineering process. On the other hand, MLP has a more straightforward structure and is probably easier to understand than CNNs, but still, the performance of MLP for SCA raises less attention. Consequently, this raises an interesting dilemma: do we consider profiling SCA as a single-objective problem where the attack performance is the only criterion or should it be a multi-objective problem where one considers several aspects of "success"? We believe the proper approach is the second one as, without a better understanding of attacks, we cannot make better countermeasures, which is an integral part of the profiling SCA research.

In this paper, we experimentally investigate the performance of MLP when applied to real-world implementations protected with countermeasures and explore the sensitivity of the hyperparameter tuning of a successful MLP architecture. We emphasize that this work does not aim to compare the performance of different techniques, but rather to explore the multilayer perceptron's capabilities. To achieve this, we use two datasets containing different AES implementations protected with random delay countermeasure and masking countermeasure. Our results show that we require larger architectures only if we have enough high-quality data. Hence, one can (to a certain degree) overcome the limitation in the number of hidden layers by providing more perceptrons per layer or vice versa. Finally, while our experiments clearly show the difference in the performance concerning the choice of hyperparameters, we do not notice that MLP is overly sensitive to that choice. This MLP "stability" means it is possible to conduct a relatively short tuning phase and still expect not to miss a hyperparameter combination yielding high performance.

2 Background

2.1 Profiling Side-Channel Analysis

Profiling side-channel analysis is an efficient set of methods where one works under the assumption that the attacker is in full control of an exact copy of the targeted device. By estimating leakage profiles for each target value during the profiling step (also known as the training phase), the adversary can classify new traces obtained from the targeted device by computing the probabilities of each target value to match the profile. There are multiple approaches to compute these probabilities, such as template attack [3], stochastic attack [26], multivariate regression model [28], and machine learning models [12,15]. When profiling the leakage, one must choose the appropriate leakage model, which will result in a certain number of classes (i.e., possible outputs). The first common model is the intermediate value leakage model, which results in 256 classes if we consider the AES cipher with an 8-bit S-box:

$$Y(k) = \texttt{Sbox}[P_i \oplus k].$$

The second common leakage model is the Hamming weight (HW) leakage model:

$$Y(k) = \texttt{HW}(\texttt{Sbox}[P_i \oplus k]).$$

The Hamming weight leakage model results in nine classes for AES. Note that the distribution of classes is imbalanced, which can lead to problems in the classification process [22].

2.2 Multilayer Perceptron

The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP has multiple layers of nodes in a directed graph, where each layer is fully connected to the next layer. The output of a neuron is a weighted sum of m inputs x_i evaluated through a (nonlinear) activation function A:

$$Output = A(\sum_{i=0}^{m} w_i \cdot x_i). \tag{1}$$

An MLP consists of three types of layers: an input layer, an output layer, and one or more hidden layers [5]. If there is more than one hidden layer, the architecture can be already considered as deep. A common approach when training a neural network is to use the backpropagation algorithm, which is a generalization of the least mean squares algorithm in the linear perceptron [9].

The multilayer perceptron has many hyperparameters one can tune, but we concentrate on the following ones:

1. The number of hidden layers. The number of hidden layers will define the depth of the algorithm and, consequently, the complexity of relations the MLP model can process.
2. The number of neurons (perceptrons) per layer. The number of neurons per layer tells us the width of the network and what is the latent space. Interestingly, there exists a well-known result in the machine learning community called the Universal Approximation Theorem that states (very informally) that a feed-forward neural network with a single hidden layer, under

some assumptions, can approximate a wide set of continuous functions to any desired non-zero level of error [7]. Naturally, for this to hold, there need to be many neurons in that single hidden layer, and knowing how many neurons are needed is not straightforward.
3. Activation functions. Activation functions are used to convert an input signal to an output signal. If complex functional mappings are needed, one needs to use nonlinear activation functions.

When discussing machine learning algorithms, it is common to differentiate between parameters and hyperparameters. Hyperparameters are all those configuration variables that are external to the model, e.g., the number of hidden layers in a neural network. The parameters are the configuration variables internal to the model and whose values can be estimated from data. One example of parameters is the weights in a neural network. Consequently, when we talk about tuning a machine learning algorithm, we mean tuning its hyperparameters.

2.3 Datasets

We consider two datasets presented in previous researches and that we denote as ASCAD and AES_RD. Both datasets are protected with countermeasures: the first one with masking and the second one with the random delay interrupts.

The ASCAD dataset, introduced in the work of Prouff et al. [24], consists of electromagnetic emanations (EM) measurements from a software implementation of AES-128 protected with first-order Boolean masking running on an 8-bit AVR microcontroller (ATMega8515). This dataset counts 60 000 traces of 700 samples each and targets the third byte of the key. The SNR for this dataset is around 0.8 if the mask is known and 0 if it is unknown. The trace set is publicly available at https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key.

The AES_RD dataset, introduced in the work of Coron and Kizhvatov [6], consists of power traces from a software implementation of AES-128 protected with random delayed interruptions running on an 8-bit AVR microcontroller (ATmega16). This dataset has 50 000 traces with 3 500 samples each, and targets the first byte of the key. The SNR has a maximum value of 0.0556. The trace set is publicly available at https://github.com/ikizhvatov/randomdelays-traces.

3 Related Work

The corpus of works on machine learning and SCA so far is substantial, so we concentrate only on works considering multilayer perceptron. Yang et al. considered neural networks and backpropagation as a setting for profiling SCA [32]. They indicated that "...neural network based power leakage characterization attack can largely improve the effectiveness of the attacks, regardless of the impact of noise and the limited number of power traces". Zeman and Martinasek investigated MLP for profiling SCA where they mentioned the machine learning algorithm simply as "neural network" [17]. They considered an architecture with only a single hidden layer and experimented with several possible numbers of neurons in that layer. Finally, they only considered a sigmoid for the activation function. After those, there have been several papers using MLP with good results, but usually comparable with other machine learning techniques [8,11,18]. Still, the hyperparameter tuning was often not sufficiently explored. Despite our attempts, we could not confirm the first paper using MLP in a deep learning paradigm, i.e., with more than a single hidden layer. Interestingly, first papers with MLP were often not clear on the number of layers, as the tuning phase played an even smaller role than today.

In 2016, Maghrebi et al. conducted the first experiments with convolutional neural networks for SCA, and they compared their performance with several other techniques (including MLP) [15]. Their results indicated that, while MLP is powerful, CNNs can perform significantly better. From that moment on, we observe a number of papers where various deep learning techniques have been considered in comparison with MLP, see, e.g., [10,20,22,23].

Pfeifer and Haddad considered how to make additional types of layers for MLP to improve the performance of profiling SCA [19]. B. Timon investigated the "non-profiled" deep learning paradigm, where he first obtained the measurements in a non-profiled way, which are then fed into MLP or CNN [30]. Interestingly, the author reported better results with MLP than CNNs. Finally, Picek et al. connected the Universal Approximation Theorem and performance of the side-channel attack, where they stated that if the attacker has unlimited power (as it is usually considered), most of the MLP-based attacks could (in theory) succeed in breaking implementation with only a single measurement in the attack phase [21].

4 Experimental Setup

In this section, we present our strategy to evaluate and compare the performance of the different MLP attacks on different datasets. We want to observe the influence of the choice of leakage model, information reduction, and major hyperparameters defining an MLP (i.e., number of layers, number of perceptrons per layers, and activation function).

We provide results with power leakage models of both the S-box output (intermediate value model) and the Hamming Weight (HW) representation of the S-box output.

Besides considering the raw traces (i.e., no pre-processing and feature engineering), we apply the Difference-of-Means (DoM) feature selection method [16]. DoM method selects the samples of a dataset that have the highest variance for a given leakage model. Even though selecting features with high variance is likely to preserve the information about the leakage, it is better to select a number of features with different variance since the features containing the leakage are not always the features with the highest or the lowest variance.

To compare the hyperparameters' influence, we conduct a grid search for hyperparameter optimization and consider each resulting model as a profiling model for an attack. Considering the MLP hyperparameters, we fix some parameters (i.e., number of training epochs and learning rate) and explore the influence of the three following hyperparameters:

- The number of perceptrons, with a fixed number of layers.
- The number of layers, with a fixed number of perceptrons.
- The activation function used for the perceptrons in the hidden layers.

In Table 1, we list all the explored hyperparameters. The total number of models trained per experiment is of $n_{act} * n_l * n_p = 2 * 6 * 10 = 120$, where n_{act}, n_l, n_p, represent the number of activation functions, layers and perceptrons per layers explored respectively. We run our experiments with Keras [4], and we use 200 epochs for the training phase, with a learning rate of 0.001. To assess the performance of a profiling model for an attack, we use the guessing entropy (GE) metric [27]. GE defines the average rank position of the correct key candidate in the guessing vector. In other words, when considering N attack traces, each of which results in a guessing vector $\mathbf{g} = [g_0, g_1, \ldots, g_{|K-1|}]$ containing the probabilities of each key candidates in the keyspace K, ordered by decreasing probability. For all experiments, when computing GE, we use the generalized guessing entropy introduced in [31]. GE equal to 0 means that the first key guess is correct, while GE of 128 indicates a random behavior. GE can also show stability or consistent increase above 200 for the correct key candidate when the computation method for GE don't consider averaging several attacks on different traces. Such behavior indicates that the trained model failed to learn how to classify data.

The metric used during a neural network training phase is training accuracy. Note, this metric can be deceiving for assessing the quality of a side-channel attack because it evaluates the attack one trace at a time, while SCA metrics take several traces into account, giving a more accurate estimation for a real attack scenario [22].

Table 1. List of evaluated hyperparameters.

Hyperparameter	Range
Activation function	$ReLU, Tanh$
Number of layers	$1, 2, 3, 4, 5, 6$
Number of perceptrons per layer	$10, 20, 30, 40, 50, 100, 150, 200, 250, 300$

5 Experimental Results

The results for all experiments on both datasets (ASCAD, AES_RD) and leakage models are given in four figures: the final key ranking from the guessing entropy of each model is represented for the activation functions explored in the first two figures. Next, we depict guessing entropy of the attack for all trained

MLP architectures. The last figure presents the integrated gradient of the best-obtained model and the median model with the corresponding color value of its final guessing entropy. By doing so, we depict the differences in important features when comparing the best attack model and average model. The integrated gradient is a method introduced in [29], which attributes the prediction of a deep neural network to its inputs. The integrated gradient can be used in the side-channel analysis to visualize the part of the traces that influence the most a network prediction and understand what trace samples the network evaluates as the leakage.

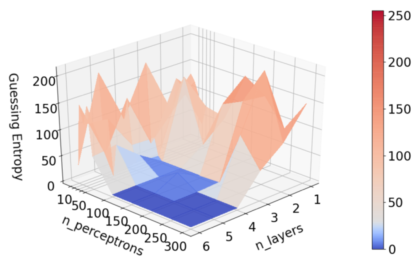
5.1 ASCAD Results

Intermediate Leakage Model: In Fig. 1, we depict the influence of all combinations of hyperparameter choices for the *ReLU* and *Tanh* activation functions when considering the intermediate value leakage model. For both choices of activation functions, some models reach guessing entropy of 0 within 1 000 attack traces. More models achieve a low guessing entropy with the *ReLU* activation function than with *Tanh*. On the other hand, *Tanh* seems to behave more stable as the resulting GE is more uniform across many explored hyperparameters settings. Several models with *ReLU* activation function and a low number of perceptrons (down to 50) can reach GE near zero.

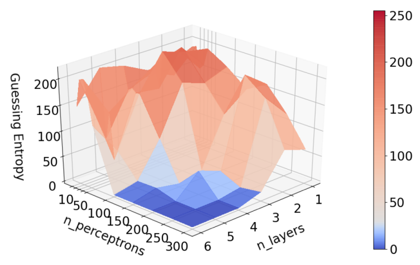
The authors of the ASCAD dataset report the best performance using an MLP with six layers containing 200 units and *ReLU* activation function trained over 200 epochs. The same hyperparameters are also evaluated and show similarly good results. However, This hyperparameters choice is not unique, and other models show equivalent performances with fewer layers and perceptrons per layers. As represented in Fig. 1a, for settings with 200 perceptrons per layer, all MLPs with more than two hidden layers converge approximately equally fast to GE of 0. In Figu. 1c, we see that many settings reach GE of 0 and that some have poor performance even after 2 500 attack traces with GE around 200. We interpret this as expected sensitivity to the hyperparameter tuning. Models with too few layers and perceptron per layers failed to properly fit the data because of their poor learnability. Finally, Fig. 1d shows that the model that reaches the smaller GE in the attack (in blue) is more sensitive to the various samples of the input than other models that fail to learn the leakage. The leakage seems entirely spread over all samples, which indicates reducing the number of features will reduce the attack performance. The model that reaches a median GE considering all experiments (in orange) has smaller integrated gradients on every data sample, which explains why this model shows poor performance for the attack.

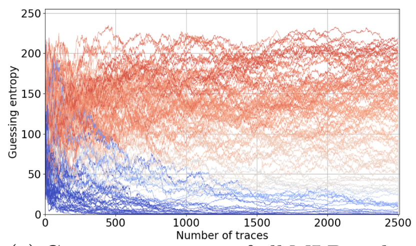
Reduced Number of Features: We now reduce the number of features to 50 with the Difference-of-Mean method. We train different MLPs with the traces that have a reduced number of features. We apply the same reduction for the attack dataset and compute guessing entropy, and we show the results in Fig. 2.

(a) ReLU.

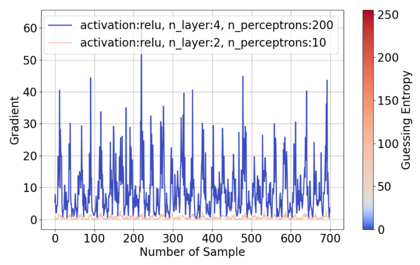


(b) Tanh.





(c) Guessing entropy of all MLP architectures versus the increasing number of attack traces.



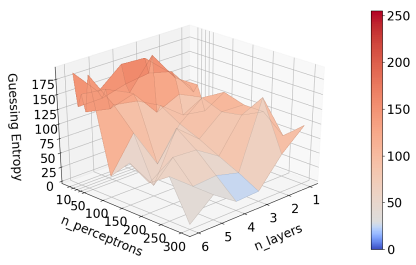
(d) Integrated Gradient.

Fig. 1. ASCAD guessing entropy for the intermediate leakage model. (Color figure online)

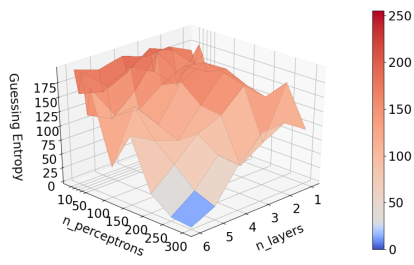
The area where GE converges toward zero is now smaller. For the *ReLU* activation function, this area is located around three and four layers with 250 and 300 perceptrons per layer. For the *Tanh* activation function, it is located above five layers and 250 perceptrons per layer. Interestingly, the highest score in Fig. 2a is not obtained for the highest number of layers. For both activation functions, the hyperparameters leading to a good attack performance are shifted toward larger hyperparameter values. This indicates that when considering features selected with the DoM method (i.e., using less information), we require deeper MLP to reach the same performance level, as the information is still present but more difficult to fit for the model. Fig. 2c shows sensitivity to hyperparameter tuning similar to the case with no feature selection. From Fig. 2d, the best fitting model has higher gradient values than the median model. Consequently, for the best model, we use most of the available features, while the average models do not manage to combine available features in any way that would indicate influence in the classification process.

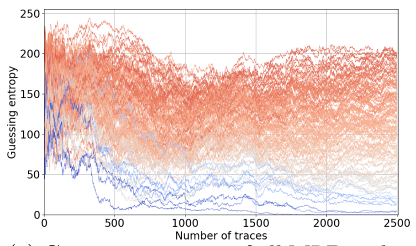
HW Leakage Model: Next, we consider the Hamming Weight (HW) leakage model. From Fig. 3, we see similar results when compared to the intermediate value leakage model. Still, in Fig. 3b, the number of perceptrons per layer has a more substantial influence on the guessing entropy than the number of layers. We can notice a better behavior for MLP with a small number of layers compared

(a) ReLU.

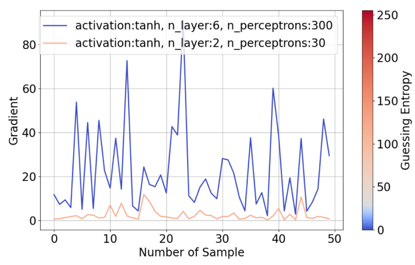


(b) Tanh.





(c) Guessing entropy of all MLP architectures versus the increasing number of attack traces.



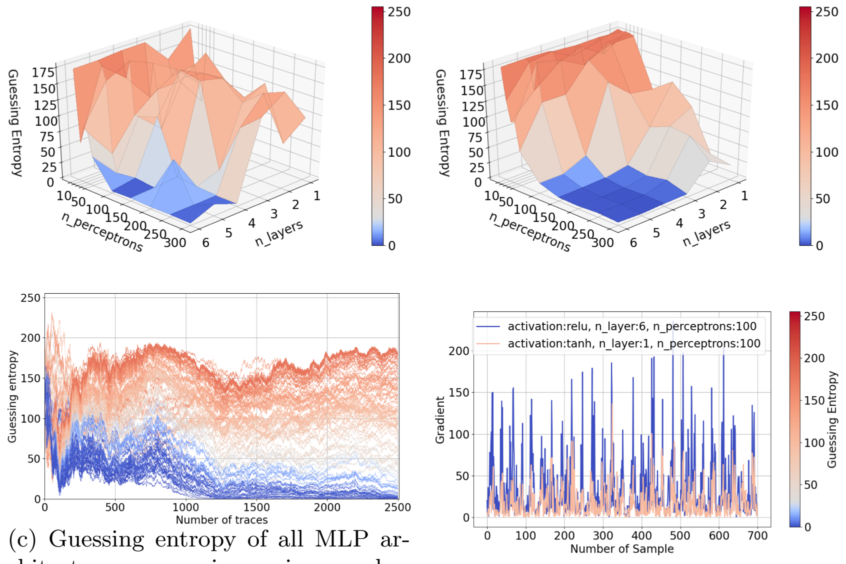
(d) Integrated Gradient.

Fig. 2. ASCAD guessing entropy with a reduced number of features and the intermediate leakage model.

to the intermediate value leakage model scenario. We believe this happens as more perceptrons per layer give more options on how to combine features, while deeper networks would contribute to more complex mappings between input and output, which is not needed for the HW leakage model as the classification task is simpler than when using the intermediate value leakage model. We can also see a stable area for several models with a number of perceptrons above 150 and a number of layers above three. In this area, the hyperparameters choice does not influence the performance of the MLP anymore. Like the intermediate value leakage model, the sensitivity to the hyperparameter tuning (Fig. 3c) is as expected, with many settings reaching top performance, but also many performing poorly. Interestingly, again we observe a more stable behavior from *Tanh* than the *ReLU* activation function. From Fig. 3d, the best fitting model and the median model have similar integrated gradient values. However, the highest peaks are different, showing that the leakage learned by the two models is different, which also accounts for the differences in GE results.

HW Leakage Model and Reduced Number of Features: We use the reduced number of feature representation of the dataset and apply the Hamming weight leakage model. We can see in Fig. 4c that many MLP architectures differ significantly with a GE spread between 0 and 175. In Fig. 4b, no MLP with the *Tanh* activation function succeeds in the attack. Finally, in Fig. 4a, MLP with

(a) ReLU. (b) Tanh.



(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

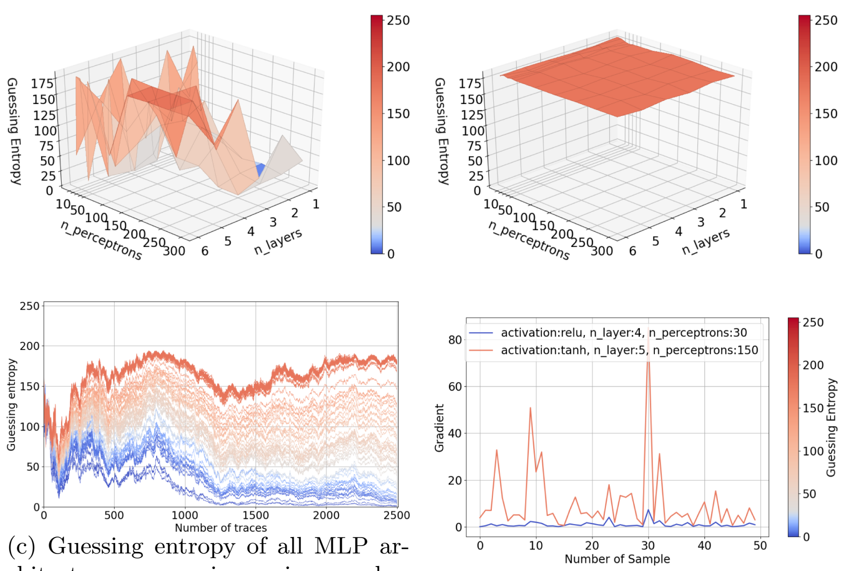
(d) Integrated Gradient.

Fig. 3. ASCAD guessing entropy in the Hamming weight leakage model.

ReLU reaching GE of 0 has only one hidden layer, and when the number of layers increases, the performance decreases. Based on the ruggedness of the landscape for *ReLU*, it is clear that the choice of the number of layers/perceptrons plays a significant role. In Fig. 4c, slightly differing from previous cases (cf. Fig. 3c), we see more groupings in the GE performance. This indicates that a reduced number of features in the HW leakage model is less expressive, so more architectures reach the same performance. From Fig. 4d, the median model presents a higher integrated gradient than the best fitting model. This behavior differs from the previous experiments and shows that a wrong fitting model has high sensitivity on samples that do not correlate with the correct leakage. This also explains the spread of GE results, as there are many subsets of features combinations that result in high GE.

5.2 AES_RD Results

Intermediate Leakage Model: Given the intermediate value leakage model (Fig. 5), all MLP architectures, including the smallest ones (one hidden layer with ten perceptrons), are capable of reaching GE below 30 within 2 500 attack traces. Increasing the number of layers does not have an impact on the *ReLu* activation function. For the *Tanh* activation function, it even seems to increase GE (thus, decreasing the attack performance). For both activation functions,



(a) ReLU.

(b) Tanh.

(c) Guessing entropy of all MLP architectures versus increasing number of attack traces.

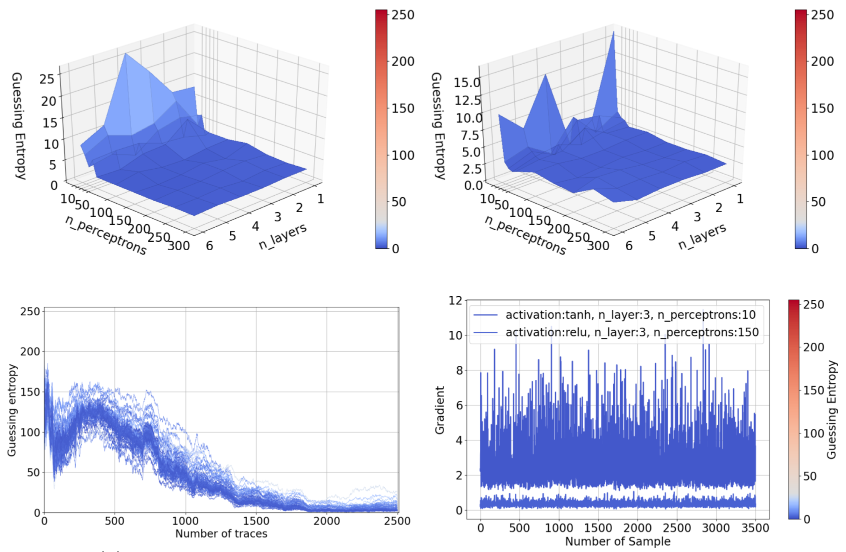
(d) Integrated Gradient.

Fig. 4. ASCAD guessing entropy with a reduced number of features and the Hamming weight leakage model.

increasing the number of perceptrons per layer decreases GE. Still, from Fig. 5c, regardless of the architecture chosen, all MLP settings converge within the same amount of attack traces. This indicates that there is not enough useful information that larger networks can use, and as such, using them brings no performance gain (consequently, there is not much benefit from detailed hyperparameter tuning). The best-fitting model and the median model are both models that fit the dataset correctly. However, from Fig. 5d, the integrated gradient method reveals that the two models have very different sensitivity on the input. Such a result could have been expected as the AES_RD dataset deals with randomly delayed traces, meaning that the leakage is not located in a precise area of the input.

Reduced Number of Features: In Fig. 6, we observe a similar performance when training MLPs with a reduced number of features for the AES_RD dataset and the intermediate leakage model (containing only 50 selected features). Again, this implies there is no useful information in additional features, and that is why MLP cannot perform better even if we use larger/deeper architectures. This is following the expected behavior for the random delay countermeasure as the features are not aligned. Finally, the landscape is smoother for *Tanh* than for *ReLU* (similar to ASCAD but also different from AES_RD with all features). The outcome from Fig. 6d is quite similar to the integrated gradient obtained on the raw traces. While the gradient values for the two models have the same

(a) ReLU. (b) Tanh.



(c) Guessing entropy. (d) Integrated Gradient.

Fig. 5. AES_RD guessing entropy for the intermediate leakage model.

levels, no maximum or minimum values are the same, meaning that no samples contribute significantly to network prediction.

Hamming Weight Leakage Model: When considering the HW leakage model for the AES_RD dataset, even after 2 500 traces, the attack is still unsuccessful. More precisely, in Fig. 7, no hyperparameter setting results in a model that can reach a GE below 60, which is not even close to a successful attack. Note we do not depict results for the reduced number of features as the attack was not successful even with the full number of features. With the intermediate value leakage model, we required around 1 500 traces to succeed in the attack. Now, we use a leakage model with a simpler classification problem and fail with more measurements. This result shows that the HW leakage is either not present or that the trained models are too simple to fit the leakage. Interestingly, all architectures behave relatively similarly, as visible in Fig. 7c. The integrated gradient on Fig. 7d shows similar results as obtained for the intermediate value leakage model, but in this case, both models do not fit the dataset correctly, which means it is difficult to talk about features that contribute more to the classification result. No trace samples show a higher sensitivity for the network prediction because of the random delay nature of the dataset.

As no MLP architecture can succeed in the HW leakage model's attack on the AES_RD dataset, we cannot conclude whether more layers or perceptrons would improve the attack performance. The phenomenon preventing MLPs from obtaining good attack performance might be linked to the class imbalance,

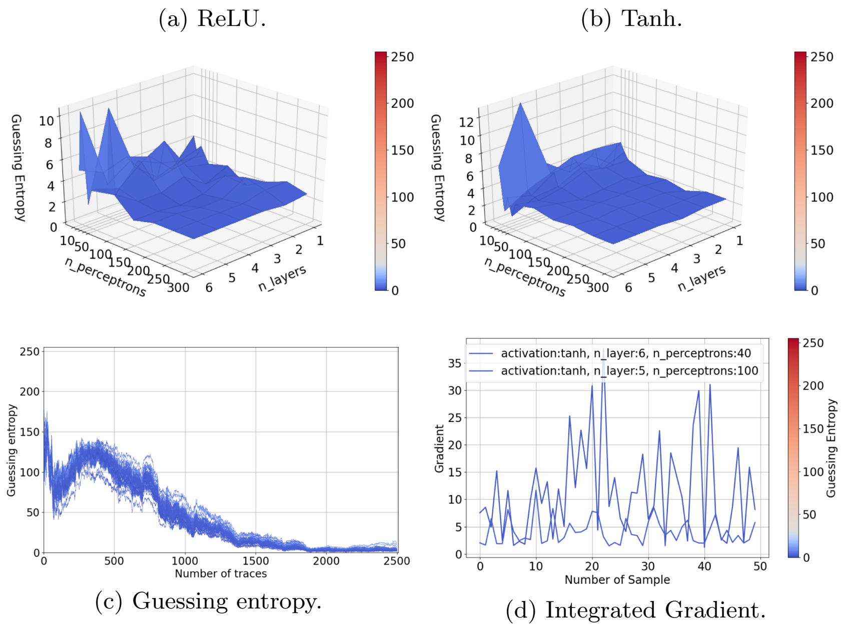


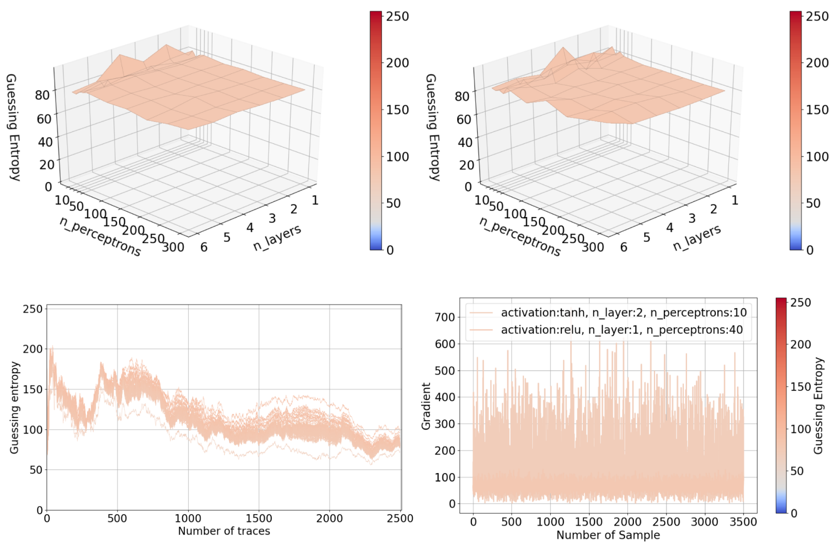
Fig. 6. AES_RD guessing entropy with a reduced number of features and the intermediate leakage model.

pointed out by Picek et al. [22], where they obtain similar results for different architectures of MLP using the HW leakage model. Additionally, they observe increasing performance when balancing the training data among the classes.

6 Discussion

MLP can break the masking countermeasure of the ASCAD dataset and the random delay countermeasure of the AES_RD implementation even when training a rather small model. For AES_RD, the smallest models (one layer, 200 perceptrons, and six layers, ten perceptrons) share the best outcome of all the models in the comparison. The same results are observed when using only the most important features. An important leakage of the secret could explain these results if the countermeasure were turned off. Although the random delays shift the first round S-box operation from the start of the encryption execution, a strong leakage of the operation handling the secret information is still present. Consequently, using an MLP is enough to overcome this countermeasure. This result indicates that the current consensus in the SCA community on MLP performance should change. Indeed, CNNs are considered especially good for random delay countermeasure and MLP for masking countermeasure [15,22]. Our results indicate there is no reason not to consider MLP successful against the random delay countermeasure given the satisfying results obtained on AES_RD with

(a) ReLU. (b) Tanh.



(c) Guessing entropy. (d) Integrated Gradient.

Fig. 7. AES_RD guessing entropy for the Hamming weight leakage model.

intermediate value. When selecting 50 POIs with a Difference-of-Mean method, the selected points represent only $50/3\,500 \simeq 1\%$ of the original traces in the dataset, and the information about the leakage is reduced. Still, the attack succeeds in the same way, which can be explained because the leakage only comes from the selected POIs. Finally, the integrated gradient is more difficult to interpret as the dataset has randomness in the time domain, which means it becomes difficult to pinpoint a few features with a significant contribution toward the classification result.

For the ASCAD dataset, we observe that the best score obtained for MLP has the following hyperparameters: six layers and 200 perceptrons. Still, we see in Figs. 1a and 1b that MLP with similar hyperparameters can perform equally good (where the red point represents the result obtain with the architecture of the best MLP MLP_{best} from the ASCAD paper). When selecting POIs with the Difference-of-Mean method, we can observe that the performance decreases, meaning that the useful information is decreased. This, in turn, results in attacks not able to recover the full secret key. Still, some MLPs can obtain the secret key in the given number of traces, and we observe that both the number of layers and the number of perceptrons influence their performance. Finally, the performance of MLPs with the Hamming weight leakage model gives better performance than for the intermediate value. The range of hyperparameters that can achieve the best results is smaller than for the intermediate value leakage model. From the integrated gradient perspective, we see that many features contribute to a successful attack, but MLP makes slightly different feature selection than DoM, as obviously not all 50 selected features contribute significantly. For the HW leakage model, the integrated gradient is somewhat more aligned, which means that more features in this leakage model contribute similarly. Such behavior is again expected as the HW leakage model forms larger clusters with S-box output values, where the importance of features is more spread within clusters.

To answer the question of how challenging is the tuning of MLP hyperparameters, we observe that there is nearly no influence using a (relatively) big or small MLP for the AES_RD dataset. When considering the ASCAD dataset with the masking countermeasure, depending on the leakage model considered, the size of the MLP can play a significant role. There, either by increasing the number of perceptrons per layer or the number of layers with a fixed number of perceptrons, we can decrease the guessing entropy.

From the activation function perspective, *ReLU* behaves somewhat better for the intermediate leakage model when compared to *Tanh*, i.e., it can reach the top performance with a smaller number of layers/perceptrons. For the Hamming weight leakage model, *Tanh* seems to work better on average, but *ReLU* reaches top performance with smaller architectures than *Tanh*. Finally, *Tanh* gives more stable behavior when averaged over all settings, i.e., with the *Tanh* activation function, the hyperparameter tuning seems to be less sensitive. To conclude, *ReLU* appears to be the preferred option if going for top performance or using smaller architectures. In contrast, *Tanh* should be preferred if stability over a more scenarios is required.

MLP is (or, at least, can be) a deep learning algorithm that has a simple architecture and a few hyperparameters but can show good performance in the side-channel analysis. What is more, our results show it can break implementations protected with both masking or hiding countermeasures. If there is no sufficient useful input information (as one would expect when dealing with the random delay countermeasure), a reasonable choice is to go with a relatively small architecture. For masked datasets, the number of perceptrons or the number of layers must be large, but the activation function's choice also plays an important role. Finally, we observe that in all considered scenarios, the MLP architectures are not overly sensitive to the hyperparameter choice, i.e., there does not seem to be a strong motivation to run very fine-grained hyperparameter tuning.

Based on those observations, we list general recommendations for MLP in the profiled SCA context[1]:

1. Many hyperparameter settings can lead to good performance, which makes the benefit of an exhaustive search very limited.

[1] The recommendations are based on the tested configurations. There is no guarantee that different results could not be achieved with significantly different settings, e.g., having a different number of perceptrons per layer. Still, following our recommendations should provide good performance in most of the scenarios commonly encountered in profiling SCA.

2. *ReLU* is better for top performance, while *Tanh* is more stable over different hyperparameter combinations.
3. Smaller depth of an MLP can be compensated with wider layers.
4. Integrated gradient is an efficient method for evaluating the influence of features if MLP manages to reach good performance.
5. Simpler leakage models require fewer layers.

7 Conclusions and Future Work

In this paper, we considered the behavior of a multilayer perceptron for profiling side-channel analysis. We investigated two datasets protected with countermeasures and a number of different MLP architectures concerning three hyperparameters. Our results clearly show that the input information to the MLP plays a crucial role, and if such information is limited, larger/deeper architectures are not needed. On the other hand, if we can provide high-quality input information to the MLP, we should also use larger architectures. At the same time, our experiments revealed no need for very fine-grained hyperparameter tuning. While the results for MLP maybe cannot compare with state-of-the-art results for CNNs, we note that they are not far apart in many cases. If we additionally factor in that MLP is simpler and faster to train, the choice between those two techniques becomes even more difficult to make and should depend on additional goals and constraints. For example, reaching the top performance is the argument for the usage of CNNs, but if one requires small yet powerful architecture, a more natural choice seems to be MLP.

In this work, we concentrated on scenarios where each hidden layer has the same number of perceptrons. It would be interesting to investigate the performance of MLP when each layer could have a different number of perceptrons. Naturally, this opens a question of what combinations of neurons/layers to consider as one could quickly come to thousands of possible settings to explore. Similarly, for activation functions, we consider only the two most popular ones where all hidden layers use the same function. It would be interesting to allow different layers to have different activation functions. Recent experiments showed that MLP could outperform CNNs when considering different devices for training and testing (i.e., the portability case) [1]. We plan to explore the influence of the hyperparameter choice in those scenarios. Finally, as we already mentioned, MLP architectures are usually simpler than CNNs, which should mean they are easier to understand. We aim to explore whether we can design stronger countermeasures against machine learning based-attacks based on MLP inner working.

References

1. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: a warriors guide through realistic profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/661 (2019). https://eprint.iacr.org/2019/661

2. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_3
3. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
4. Chollet, F., et al.: Keras (2015). https://github.com/fchollet/keras
5. Collobert, R., Bengio, S.: Links between perceptrons, MLPs and SVMs. In: Proceedings of the Twenty-First International Conference on Machine Learning, ICML 2004, p. 23. ACM, New York (2004). https://doi.org/10.1145/1015330.1015415
6. Coron, J.-S., Kizhvatov, I.: An efficient method for random delay generation in embedded software. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 156–170. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04138-9_12
7. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Sig. Syst. **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274
8. 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), pp. 106–111, May 2015. https://doi.org/10.1109/HST.2015.7140247
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
10. Hettwer, B., Gehrer, S., Güneysu, T.: Profiled power analysis attacks using convolutional neural networks with domain knowledge. In: Cid, C., Jacobson Jr., M. (eds.) Selected Areas in Cryptography - SAC 2018–25th International Conference, Calgary, AB, Canada, 15–17 August 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11349, pp. 479–498. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-10970-7_22
11. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Side-channel analysis of lightweight ciphers: does lightweight equal easy? In: Hancke, G.P., Markantonakis, K. (eds.) RFIDSec 2016. LNCS, vol. 10155, pp. 91–104. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62024-4_7
12. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. Unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Trans. Cryptographic Hardware Embed. Syst. **2019**(3), 148–179 (2019). https://doi.org/10.13154/tches.v2019.i3.148-179. https://tches.iacr.org/index.php/TCHES/article/view/8292
13. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25. http://dl.acm.org/citation.cfm?id=646764.703989
15. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 3–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_1

16. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Boston (2006). http://www.springer.com/. ISBN 0-387-30857-1. http://www.dpabook.org/
17. Martinasek, Z., Zeman, V.: Innovative method of the power analysis. Radioengineering **22**(2) (2013)
18. Martinasek, Z., Hajny, J., Malina, L.: Optimization of power analysis using neural network. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications, pp. 94–107. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_7
19. Pfeifer, C., Haddad, P.: Spread: a new layer for profiled deep-learning side-channel attacks. Cryptology ePrint Archive, Report 2018/880 (2018). https://eprint.iacr.org/2018/880
20. Picek, S., Heuser, A., Alippi, C., Regazzoni, F.: When theory meets practice: A framework for robust profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1123 (2018). https://eprint.iacr.org/2018/1123
21. Picek, S., Heuser, A., Guilley, S.: Profiling side-channel analysis in the restricted attacker framework. Cryptology ePrint Archive, Report 2019/168 (2019). https://eprint.iacr.org/2019/168
22. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 209–237 (2019). https://doi.org/10.13154/tches.v2019.i1.209-237
23. Picek, S., Samiotis, I.P., Kim, J., Heuser, A., Bhasin, S., Legay, A.: On the performance of convolutional neural networks for side-channel analysis. In: Chattopadhyay, A., Rebeiro, C., Yarom, Y. (eds.) Security, Privacy, and Applied Cryptography Engineering, pp. 157–176. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-05072-6_10
24. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. Cryptology ePrint Archive, Report 2018/053 (2018). https://eprint.iacr.org/2018/053
25. Quisquater, J.-J., Samyde, D.: ElectroMagnetic analysis (EMA): measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45418-7_17
26. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_3
27. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01001-9_26
28. Sugawara, T., Homma, N., Aoki, T., Satoh, A.: Profiling attack using multivariate regression analysis. IEICE Electron. Express **7**(15), 1139–1144 (2010)
29. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. arXiv preprint arXiv:1703.01365 (2017)
30. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptographic Hardware Embed. Syst. **2019**(2), 107–131 (2019). https://doi.org/10.13154/tches.v2019.i2.107-131. https://tches.iacr.org/index.php/TCHES/article/view/7387

31. Wu, L., et al.: Everything is connected: From model learnability to guessing entropy. Cryptology ePrint Archive, Report 2020/899 (2020). https://eprint.iacr.org/2020/899
32. Yang, S., Zhou, Y., Liu, J., Chen, D.: Back propagation neural network based leakage characterization for practical security analysis of cryptographic implementations. In: Kim, H. (ed.) ICISC 2011. LNCS, vol. 7259, pp. 169–185. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31912-9_12