Not a Free Lunch but a Cheap Lunch: Experimental Results for Training Many Neural Nets Efficiently

Joey Green[1], Tilo Burghardt[1], Elisabeth Oswald[1,2]

[1] Department of Computer Science, University of Bristol,
firstname.lastname@bristol.ac.uk
[2] Department of Applied Informatics, University of Klagenfurt,
firstname.lastname@aau.at

Abstract. Neural Networks have become a much studied approach in the recent literature on profiled side channel attacks: many articles examine their use and performance in profiled single-target DPA style attacks. In this setting a single neural net is tweaked and tuned based on a training data set. The effort for this is considerable, as there a many hyper-parameters that need to be adjusted. A straightforward, but impractical, extension of such an approach to multi-target DPA style attacks requires deriving and tuning a network architecture for each individual target.
Our contribution is to provide the first practical and efficient strategy for training many neural nets in the context of a multi target attack. We show how to configure a network with a set of hyper-parameters for a specific intermediate (SubBytes) that generalises well to capture the leakage of other intermediates as well. This is interesting because although we can't beat the no free lunch theorem (i.e. we find that different profiling methods excel on different intermediates), we can still get "good value for money" (i.e. good classification results across many intermediates with reasonable profiling effort).

1 Introduction

Profiled side channel attacks are the canonical methods to determine the level of side channel security from a worst case perspective. Classical profiling methods are based on building (Gaussian) templates [4] and linear regression models [9]. Profiling methods build explicit statistical representations of the average leakage for specific intermediate values. They can therefore be used as predictors with comparison-based distinguishers[3].

Many side channel attacks run in a "single target" setting: a single intermediate value (typically the output of a non-linear component in a round function) is selected and a statistical characterisation is derived. Either this characterisation is defined for all possible values of the intermediate, or by pre-selecting representative classes (i.e. a functional model is selected a priori).

[3] Machine learning methods have also been extensively studied (predominantly as classifiers), and many have found their use together with partition-based distinguishers.

Over the recent years, a new class of side channel attacks has emerged, so called "multi-target attacks". In a profiled setting, an adversary combines the leakage of multiple intermediate values using belief propagation [23,6]. Multi target attacks are substantially more trace efficient than single target attacks and are the most natural use case for profiling.

Recently, deep learning has become a focus of attention in the side channel community. Restricting our discussion on papers based on AES implementations, a number of recent studies provide first results for protected and unprotected AES implementations [3,10,18]. All these papers have in common that they train networks for the "best" intermediate (i.e. SubBytes outputs are used as labels, alongside a "window" in which the corresponding leaks are to be found in traces). A common theme for these works is that they exclusively operate in a single target setting.

The deep learning approach (as any profiling method) to side channel analysis proceeds in two stages in a typical DPA setting. In a training phase one acquires leakage information from (a copy of) the target device with known inputs and key(s), which becomes the training data set. A deep network model is then selected, informed by previous work [3] and carefully tuned based on the available training data [18]. Then, during the attack phase, one is able to query the trained model using new unseen data to recover information about the (now) unknown key.

In contrast to classical profiling attacks which require the adversary to find the most leaky time points manually (or via some other method, e.g. LDA [9] or PCA [2]), deep learning offers the tantalising promise of automatically identifying the most leaky time points (even in misaligned traces [3]). The latter point is of particular value for profiled multi-target attacks.

1.1 Contributions and Outline of this Paper

All published work in the context of deep learning applied to side channels focuses on methods of selecting a network architecture, and tuning its parameters to attacking a single intermediate value (typically the Sbox output in the case of AES). Whilst this makes sense in the context of a single target attack, it opens the question if deep learning can at all be used in the context of *multi target attacks*. Tuning a deep network is very expensive (in terms of computing time), and a multi target attack on AES such as [23] utilises hundreds of intermediates values. Even if we employ optimisation techniques for constructing belief propagation graphs such as described in [6], we still need to profile over one hundred intermediates. Training a deep network from scratch for each of these intermediate values would clearly not scale.

Our work is the first to tackle this challenging problem. We demonstrate how to efficiently train MLPs for many intermediate values from an `AES FURIOUS` [17] implementation on an ARM Cortex M0 processor. We achieve this by building on two observations: we utilise a batch size of one during validation of models whilst we search for the best architectural parameters; we utilise the median probability during the search for the best architectural parameters. These two choices enable

us to define a network architecture which generalises to all intermediates for our AES implementation. **Thus we provide a first working solution to a training regime that results in a deep network architecture that generalises to many intermediate values for a given implementation.**

Outline. We briefly review the working principle of profiled attacks in Section 2. Alongside this we provide information about our attack setup, including the relevant implementation details of the targeted AES implementation. We also discuss the salient background, our training assumptions, and hyperparameters for the MLP type network that we mainly consider.

We report on two approaches that we investigated. First, in Section 3, we briefly describe an unsuccessful approach that was based on *transfer learning*. We hope that reporting this negative result helps save time of other researchers.

Secondly, in Section 4, we discuss our successful approach in detail, and compare this approach with two classical types of profiling: classical univariate (Gaussian) templates and LDA based multivariate templates. We find, in contrast to other work, that it is efficient to concentrate the tuning of our deep networks to maximise the **per trace classification performance** (instead of using a batch of test traces). Also significant is our finding regarding the choice of metric for the classification performance when determining the best hyperparameters for a network. Initially we utilised the "median rank" metric (as per [18]), but using "median rank" led to networks that behaved in a rather arbitrary and poor manner when trained for different intermediates. This behaviour has never been noticed before simply because no previous research has attempted to train so many networks for a range of intermediates (which relate to different target functions and different leakage functions). We found that **using the median probability as a measure to determine the best hyperparameters** enabled us to develop a network configuration that we could retrain/adapt for all other intermediate values.

In Section 5 we relate our findings to the open question if the *no free lunch (NFL) theorem* (for supervised machine learning, CITE) may apply to more challenging targets for profiling, such as the ARM M0 core that we use. Our findings confirm that **the NFL may have a role to play:** across all intermediates there is **no clear winner in terms of profiling method**. Even for the same type of intermediate (e.g. the leakage corresponding to a byte undergoing SubBytes) different algorithms perform differently across the 16 state bytes. This is significant in the context of leakage evaluation regimes (such as Common Criteria) where there is now a requirement to at least attempt a deep learning based evaluation: firstly there is no reason to belief that deep learning would be better than classical profiling in terms of classification performance, and any form of optimal choice can only be made if the leakage of the device under evaluation *is alrealy understood*. This leaves as only valid use case for deep learning scenarios where randomness is "locked" in the device (and cannot be given to the evaluator to aid classical profiling) or where the noise/delay characteristics are so obscure that an evaluator cannot make any meaningful choices.

Finally, we compare all classifiers in the context of two attacks (a DPA attack on SubBytes and a belief propagation attack on the first two AES rounds) in Section 6.

1.2 Related work

Song et al. [20] provides an overview on the various deep learning models that have been employed in recent literature, ranging from Multi-Layer Perceptrons to Convolutional Neural Networks. They provide insight into the current research situation, highlighting the success of published attacks against classical power analysis methods.

Benadjila et al. [18] proposed an independent study of deep learning algorithms when applied to side channel analysis. They give an example of how one would choose the architecture and training parameters of Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs) in order to optimise the network for their dataset: a masked implementation of the AES algorithm, with a degree of clock jitter. They conclude, after comparing a trained MLP to a trained CNN, that the MLP outperforms the CNN in a jitter-free scenario, whereas the CNN is more resilient against clock jitter. This is expected given the shift invariance of the convolutional operation.

Other work in this area [3] showed that using data augmentation on available training data through a combination of methods (shifting and add-remove) one can get even better performance with CNNs on jittery traces. This outperforms an implementation of a Gaussian template attack with trace realignment.

Kim et al. [10] compare a number of CNN network structures against four separate data sets: the DPAcontest v4 (masked AES), an unprotected AES-128 on an FPGA, an AES implementation with a random delay software countermeasure, and the ASCAD dataset as previously described. They conclude that CNNs even with slight changes to their network structure have varying success rates on different data sets; therefore, network parameters and structures must be tuned according to their required target data. They propose the need of a suite of CNN instances, in which the user can switch their CNN structure to find the best network for their use case. The paper additionally proposes adding Gaussian noise to the training set, as they show this allows the network to be more resilient to noise in the attack phase, as they use the ASCAD model as an example.

Martinasek et al. [14] proposes preprocessing the training data set by calculating the average trace and finding the difference between this average and all other traces. By using an MLP in an example, they show that they increase the success rate of the classification by using this preprocessing method. However, they include that this method has drawbacks: it suppresses the alternative probabilities, meaning that an attacker would not be able to try a second key guess if the most probable value is incorrect. In our work, we will show that this suppression is severely detrimental to inference based attacks.

The same MLP was then used to target the DPAcontest v4.2 [15]. Although the authors mention this method is not fully explored, as they encounter some

problems on the seventh dataset, but they are successfully able to recover the key on the other datasets. The MLP was compared to a template attack in [13]. Their metric for success was the guessing entropy. Unfortunately, perhaps due to the paper length, there was no mention of how the hyperparameters were chosen for this MLP. Instead, they conclude that the template attack performs three times better than the MLP approach when the training data has not been preprocessed. After preprocessing (using the methods described in [14]) they conclude the success of the MLP matches that of the template attack.

2 Preliminaries

We provide a brief description of our setup and the necessary background for the learning methods that we use.

2.1 Attack Setup and Implementation Details

The `AES FURIOUS` algorithm has been used widely in both commercial devices and for attacking purposes; we use an adaptation of this algorithm written in the ARM Thumb assembly language. It is a native 8 bit implementation which performs the `SubBytes` step as a table lookup.

The physical device we target is based on an ARM Cortex-M0 of the LPC series. The ARM Cortex M0 is a well characterised and understood processor: previous work has shown that its leakage function has linear terms as well as statistically significant second order terms (Hamming distance leaks, as well as bit interactions) [16]. The noise is not significantly different for different instructions. Thus by training networks on this platform we are able to examine the case where the leakages from different intermediates are share similarities and they are all complex enough to be interesting (i.e. they are not pure Hamming weight).

The processor is mounted on a small purpose built development board, which contains a signal amplifier and filter. To ensure we do not capture any clock jitter, we use a stable external 8MHz clock. We record the power leakage with a PicoScope 2000 Series oscilloscope, with a sampling rate of 125MS/s.

Using this setup, we took 210000 traces from the target device, split into three groups:

- 190000 traces with random keys and random plaintexts, to be used for *training*
- 10000 traces with random keys and random plaintexts, to be used for *validation*
- 10000 traces with a fixed key and random plaintexts, to be used for *testing*

Our traces have 51250 time points, ranging from the start of AES to the end of the second round.

We implemented the (Gaussian) template building in Python following [11]. To implement multivariate templates, we use Linear Discriminant Analysis (LDA)

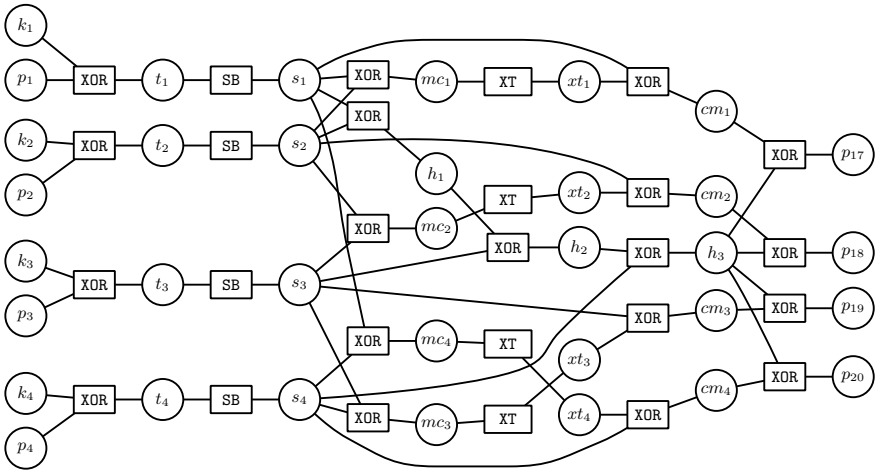


Fig. 1: Factor Graph representation of the first column of the first round of the `AES FURIOUS` implementation

in the `scikit-learn` module, which comes packaged with an LDA classifier. To implement the neural networks, we used the Keras API, using TensorFlow as a backend (identical to [18]).

In this work, we aim to use the output of our distinguishers as the input to an inference based attack. The key idea behind inference based attacks is to combine information from multiple intermediate values, via a method called belief propagation. This type of attack requires the construction of a so-called factor graph, and we use the best factor graph for `AES FURIOUS` as provided in [6] (see Figure 1 for a graphical representation of part of that factor graph). This particular cyclic graph contains leakage information from the first two rounds of AES. Using the classification results as initial distributions in this graph, we run the belief propagation algorithm and extract a ranking of the key upon termination.

2.2 Profiling

(Gaussian) Templates A template is defined as a pair of a mean vector and a standard deviation vector (μ, σ) (like in many existing works we set covariances to zero in order to avoid numerical issues wit the matrix inversion [11]). These pairs are created during the template building phase (often referred to as the *offline phase*) by finding the point of interest within the traces where the target leakage occurs.

The template matching phase (or *online phase*) involves matching the templates to newly obtained power leakage, and (via Bayes theorem) deriving a distinguishing vector that represents the likelihoods of subkey values.

We can normalise and combine these probabilities over multiple traces to form a ranking of the possible key. In our implementation we specialise these

types of templates to be univariate; i.e. we select only the most representative point of interest from the traces.

Linear Discriminant Analysis Templates Linear Discriminant Analysis (LDA) is a generalisation of Fisher's linear discriminant, which is a statistical method used to find a linear combination of features that characterises multiple classes of objects. LDA is closely related to regression analysis, and is a widely used tool as a linear classifier. For our multivariate (multiclass) LDA templates we now consider data over a large window surrounding the point of interest chosen for the univariate templates. Hence the LDA approach is our multivariate extension for the univariate templating from before.

Neural Networks Neural networks are frameworks for machine learning algorithms to process complex data inputs. In the application area of side channels, the most common use case is to use them in a supervised fashion as a classifier (i.e. we use them in the same way as templates and LDA templates).

In contrast to the relatively straightforward configuration of (LDA) templates, where the biggest challenge is in the selection of points of interest, there are many parameters that need to be selected and fine tuned in the case of deep networks.

There are multiple ways of structuring a network, and hence there are many models from which to choose; some excelling in specific scenarios but failing in others.

For our leakage classification use case, we will be using a Multi-Layer Perceptron. This is because the results presented in [18] show the MLP outperforms other models (a CNN and VGG16) in a jitter-free scenario.

We want to stress that our most significant findings apply to all the considered learning algorithms in our paper: univariate templates, LDA templates and the MLP. Therefore we argue that our findings are not specific to just the MLP.

2.3 Multi-Layer Perceptron

A Multi-Layer Perceptron is an example of a feedforward artificial neural network - it does not contain any cycles. Typically, an MLP has at least three layers: an input layer, a hidden layer (or multiple hidden layers), and an output layer. All layers are dense (also called fully connected): every input is connected to every output by a weight. Each node (bar the input nodes) is a neuron that uses a nonlinear activation function.

When constructing and training an MLP, there are several hyperparameters that must be considered. All of these can vastly change the effectiveness of a trained model, so appropriate values must be selected accordingly. These include the following: number of hidden layers, number of nodes per hidden layer, activation function for hidden layers, number of training epochs, batch size, learning rate, and the optimiser. We now briefly touch on these hyperparameters in turn.

The universal approximation theorem states that networks with two hidden layers and a suitable activation function can approximate any continuous function on a compact domain to any desired accuracy [5,8]. However, the size of such a shallow neural network would be prohibitive: the number of neurons per layer would be exponential in the input size. Deep neural nets trade the number of layers for the the number of neurons per layer, and are suspected to learn "natural functions" fast with fewer neurons [21].

The activation function of a node defines the output of that node given a set of inputs. Common activation functions consist of ReLU (a linear unit employing a 'rectifier', which takes the positive part of its argument) and Softmax (a function that normalises an input vector into a probability distribution).

The batch size is the number of samples processed before the model is updated, and the number of epochs is the number of complete passes through the training dataset. The learning rate is how quickly the model learns; that is, a network with a larger learning rate will abandon old beliefs quicker, whereas a network with a low learning rate will be less susceptible to outliers.

Optimisation algorithms are used to minimise the objective function; in our case, categorical cross entropy. RMSProp [22] is a commonly used optimiser in classification networks.

Literature in Deep Learning does not provide much insight into how one chooses these hyperparameters. One either employs a manual search (guess parameters and compare results, selecting the one that gives the best results), or uses some automatic parameter selection method, such as grid search or random search.

Training We use the same setup as for classical profiling. The most important difference to the trace sets for classical profiling is that we provide a large window of "potentially interesting points" rather than preselecting interesting points. Along with the training data we provide our training labels, also in the same fashion as in the classical profiling.

Training labels are one-hot encoded; they are vectors of size 256 where all values are zeros, bar a single 1 at the index of the correct value k^*, as shown in Eq. 1.

$$\text{oneHot}(k^*) = (\underset{0}{0}, \ldots, \underset{k^*}{1}, \ldots, \underset{255}{0}) \tag{1}$$

Validating The validation data is also based on random keys and random plaintexts. This data is used *during training* to validate the current effectiveness of the model against unseen data, using the specified loss function. Tools such as TensorBoard provide us with a visual graph depicting the accuracy and loss of the training, and allows us to pinpoint the exact time in which the model starts to overfit. The loss function used throughout this work (and in previous works) is categorical cross entropy.

Testing The testing data was produced in an identical fashion to the training data, but this time we fixed the key in order to emulate the scenario of an actual attack. In an actual attack scenario (targeting a real device) the key will remain constant among all traces, so by using a fixed key in the testing data, we can get an accurate analysis of how the model would perform in an attack scenario. Testing on a single fixed key is sufficient because of the EIS property [19].

Because we are interested in the per trace classification performance, our testing essentially uses testing sets of size one. Consequently cross validation is not necessary (or possible).

2.4 Success metrics

We denote a subkey value with k and the correct subkey value with k^*. A classifier produces a vector d that represents an estimated probability distribution for the unknown subkey values $k \in \mathcal{K}$. The classification outcome for the correct subkey value k^* is given by $d[k^*]$.

There are a number of metrics used in the side channel community to measure the performance of a classifier. We briefly review the salient ones for our work.

We define the rank of k^* given a new trace T in Eq.2 as an integer between 1 (the best and largest probability relative to others) and $|\mathcal{K}|$ (the worst).

$$\text{rank}(k^*, T) = |\{k \in \mathcal{K} | d[k] > d[k^*]\}| \tag{2}$$

The rank is a random variable and thus measuring it based on a single experiment with a single test trace is not meaningful. Consequently we consider the rank over many experiments (each with a single test trace). To simplify notation, we collect the single-trace experiments in a set $\mathbf{T}$, based on which we will be studying the behaviour of the median rank, alongside the median probability, defined in Eq.3 and Eq.4 respectively.

$$\text{medianRank}(k^*, \mathbf{T}) = \underset{T \in \mathbf{T}}{\text{median}}(\text{rank}(k^*, T)) \tag{3}$$

$$\text{medianProbability}(k^*, \mathbf{T}) = \underset{T \in \mathbf{T}}{\text{median}}(d[k^*]) \tag{4}$$

The loss function, which is the cross entropy, can be defined in Eq. 5.

$$\text{crossEntropy}(d, k^*) = -\sum_{i=0}^{255} (i == k^*) \log d[i] = -\log(d[k^*]) \tag{5}$$

3 Initial study: transfer learning based on the ASCAD MLP and CNN

The idea of "re-using" a successful network architecture is referred to as *transfer learning* in the machine learning context and it is known to be a successful

Table 1: Classification results using different learning algorithms, attacking the first `SubBytes` output byte

Classifier	Median Rank	Median Probability	Mean Probability
Uniform	128	0.00390625	0.00390625
CNN, Pretrained	127	0.001150969	0.003813843
MLP, Pretrained	128	0.003908087	0.003909754
CNN	126	2.69421e-21	0.006996953
MLP	73	0.004286217	0.008182878
Univariate	98	0.004243865	0.004606495
LDA	64	0.005063529	0.007880825

strategy. In transfer learning we use a model that was trained on one data set and re-purpose it on another, related data set.

The ASCAD data was trained based on an 8-bit processor with simple Hamming weight leakage executing a masked AES. The M0 leakage functions (i.e. our setup) all include a strong Hamming weight component (like the ASCAD leakages), and are also based on 8-bit intermediate values. However, the M0 processor features a wider range of leakage functions [16]. The ASCAD implementation was based on masking thus we conjectured that the resulting networks must have captured more than just simple 8 bit Hamming weights, and thus their complexity could potentially be sufficient to capture the components of the more complex M0 leakages. Therefore, it was interesting to test if we could utilise the ASCAD MLP (and CNN) as a reasonable initial base architecture to *efficiently train many networks*.

3.1 Results

Table 1 shows the classification results for the SubBytes output for the M0 data. The first column refers to the type of network (CNN/MLP Pretrained are the actual ASCAD networks, CNN/MLP are based on the ASCAD hyperparameters but newly trained on the M0 data, Univariate refers to univariate templates, LDA refers to multivariate templates). We provide the expected values for random guessing as "Uniform" classifier in the first row of the table.

It should be evident that the pretrained models are unsuccessful at classifying the new data; this can be seen by the median classification probability being equal to or less than guessing randomly (out of 256 possible values).

The CNN trained solely on our data sports a confident mean probability, but an exceptionally low median probability. This is due to the neural network being confident but wrong more often than not. The MLP model is able to improve upon the uniform distribution, and is just able to outperform the standard univariate templating method. However, the MLP is outperformed by the multivariate (LDA based) templates. Thus the ASCAD MLP and CNN architectures (for SubByates) do not generalise well to the M0 SubBytes data and do not appear to be a promising starting point to train many networks.

Nevertheless we did utilise the somewhat successful MLP architecture and attempted a further transfer learning experiment where we retrained it for the classification of the intermediate that corresponds to loading of the key byte. In this experiment, the network performs as bad as guessing randomly.

4 Determining the Best Network Architecture: Mind your Metric

Abandoning the unsuccessful attempt of transfer learning, we then followed another established approach of determining the (architectural) hyperparameters recursively from scratch.

We set out to determine a "best" MLP for the SubBytes leakages first. Concretely this implied that for a number of hyperparameter choices we tested the performance of the resulting network via a custom "testing" phase. We initially followed the established practice to utilise the **mean subkey rank** to judge the performance of a network/classifier.

The mean rank is estimated from the test dataset via cross validation, which is known to result in a reliable estimation. However, having a test dataset with multiple traces may already reduce the ability of a network architecture to generalise. Furthermore the assumption that an attack always uses a "batch of traces" somewhat relates to DPA style attacks. This is per se not a problem, but our attack scenario is that of belief propagation which does not proceed in "batches of traces". Our goal is to select a network architecture that generalises to many intermediate values leaking exhibiting different leakage functions.

Therefore we tested two different metrics for hyperparameter selection: the median rank (as replacement for the mean rank to better account for outliers), and the median probability.

4.1 Median Rank as Metric

Figure 2a shows the performance figures for the 16 SubBytes leakages, where the MLP was optimised for the first SubBytes leakage (lower ranks indicate better performance). The performance is a marked improvement over the results from (re)using the ASCAD networks, but the most striking feature of the results is their variability. Intuitively we would expect the performance for the first SubBytes operation to be slightly better because this should correspond best to the training data, but the performance of the other SubBytes leakage should be nearly identical; after all, it is the same Assembly instruction sequence and there are no other processes running on the device that could influence the leakage (or noise). It is also striking that the performance of the classical profiling methods is extremely variable (for no apparent reason).

Figure 2b shows the performance figures for the 16 AddRoundKey leakages and Figure 2b shows the performance figures for the loading of the key bytes. We certainly wouldn't expect it to perform much better here, but it is again

Table 2: Table comparing the locally optimal parameter values between various networks

Parameter	ASCAD Network	Rank Network	Probability Network
Number of Hidden Layers	4	2	3
Number of Nodes in Hidden Layers	200	200	100
Activation Function[a]	ReLU	ReLU	ReLU
Number of Epochs	200	6000	100
Window Size	700	700	2000
Batch Size	100	200	50
Learning Rate	10^{-5}	10^{-5}	10^{-5}
Optimiser	RMSProp	RMSProp	RMSProp

[a] The output layer uses the Softmax activation function to ensure the network outputs a normalised probability distribution

striking that the performance figures for all classifiers are extremely variable for no apparent reason.

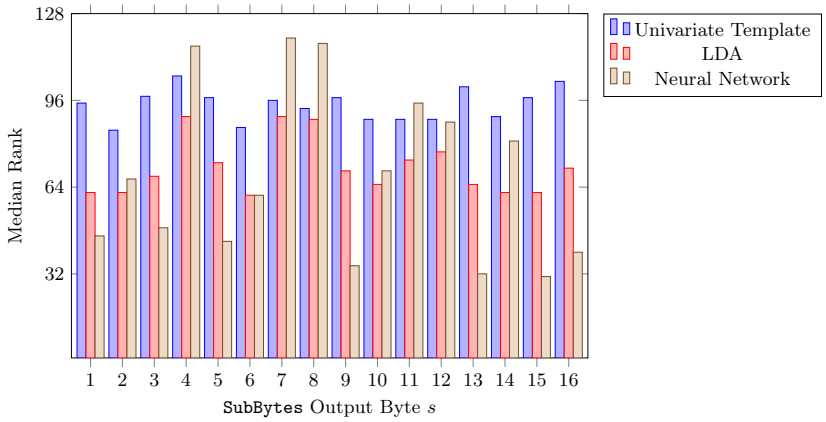
Given that the performance figures are variable for all classifiers and different intermediate leakages, *our hypothesis for the explanation of this phenomenon is that it must be related to how we measure their performance.* This is because the classifiers are are all very different, the only commonality is how we measure their performance—via the median rank.

The subkey rank is a useful measure (in principle) because it directly relates to how we evaluate attack outcomes (see Eq.2). However, as explained before, this metric is typically used in conjunction with DPA style attacks. In this use case we use many (enough) leakages to produce stable ranks. In the classification experiments we judge the classification per trace and thus it is possible that the classifier produces rather erroneous ranks. By only utilising the rank of the classification results, rather than the resulting distribution, we throw away information that may tell us that the network is in fact not very confident about the resulting classification result. Thus maybe a better strategy for judging the classification performance could be to utilise the median probability as a metric.

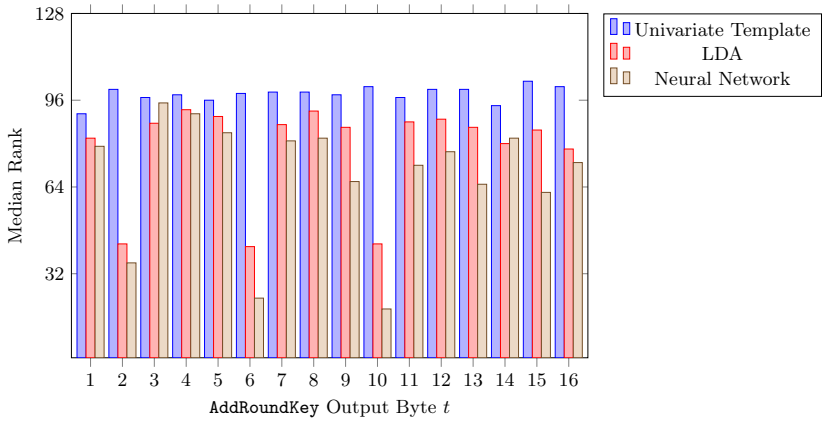
4.2 Probability as metric

Our next experiment consisted of optimising the MLP by judging its performance via the median probability, as defined in Eq. 4.

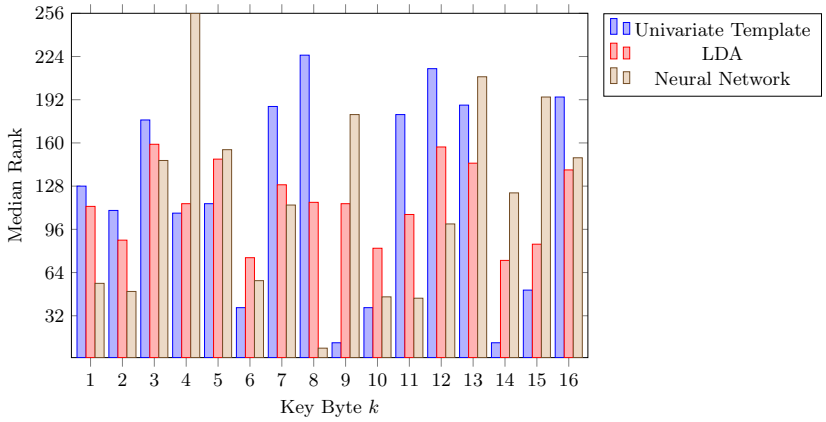
Utilising the median probability as metric dramatically changed the configuration of the resulting best network, see Table 2 for an overview of the hyperparameters of the ASCAD MLP, the rank based MLP, and the probability based MLP. The network based on rank has the fewest hidden layers but requires a large number of epochs, whereas the number of hidden layers for the probability based network is between the ASCAD and the rank based network. Noticeably it utilises the largest window size (i.e. it asks for the most trace points of all networks). This may indicate that it best utilises the available information: the



(a) Plot showing the classification results of the `SubBytes` output using Median Rank as a performance metric



(b) Plot showing the classification results of the `AddRoundKey` output using Median Rank as a performance metric



(c) Plot showing the classification results of the key bytes using Median Rank as a performance metric

Fig. 2: Plot showing the classification results of various intermediates using Median Rank as a performance metric

SubBytes output is fetched from memory, and we know that on this particular M0 implementation there is a buffer between the registers and the external memory which causes values to "hang around' in the architecture for several cycles; in addition we know that the SubBytes values utilised again as part of MixColumns.

There is also a dramatic difference in the number of epochs between the rank based network and both the ASCAD and the probability based network. We investigated the behaviour on the training and validation data sets and noticed that the model overfits after around 2000 epochs, see Fig. 3a. Before this point, the model continues to 'learn' more about the task.

Interestingly, upon comparing the median probability results on different numbers of epochs, we found a local optimum when using 100 epochs. This seems interesting; Fig. 3b shows the network after 100 epochs, and it appears as though it is still learning. However, past this point, we find a lower median probability classification result.

We conjecture that this is most likely due to the discrepancy between the cross entropy loss function (used to calculate the training and validation accuracy) and the median probability metric we use to find the best model.

5 An Architecture that Generalises, but No Free Lunch

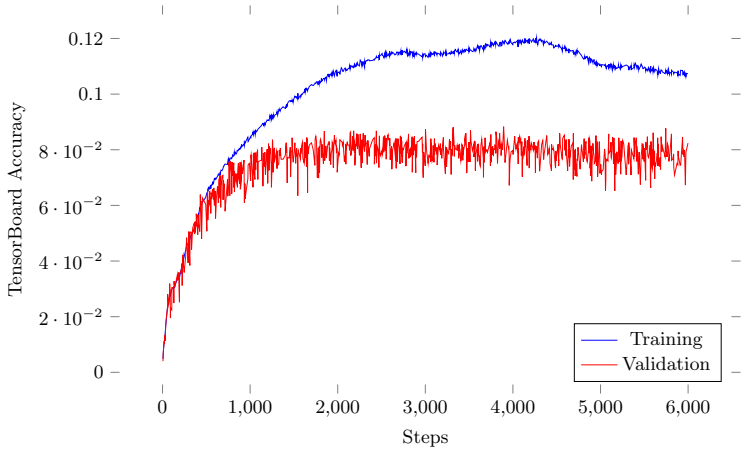
Now that we have a network architecture that generalise well across several intermediate values, we can train and therefore specialise this architecture for various intermediate values. For the sake of conciseness, we compare resulting networks to univariate templates and to multivariate LDA based templates (derived with the same data but preselected points of interest).

Figure 4a shows the comparison (based on the median probability metric), when targeting the outputs of the 16 `SubBytes` intermediates. In comparison to Figures 2a, 2b, and 2c, the classification performance is much more stable, which gives us high confidence in the quality of the deep networks. It is evident that for some targets (e.g. s_2, s_9), the neural networks perform much better than the other classification methods. However, this is not always the case; sometimes (e.g. s_8, s_{11}), the neural networks are outperformed by the univariate templating method and/or the LDA classifier.

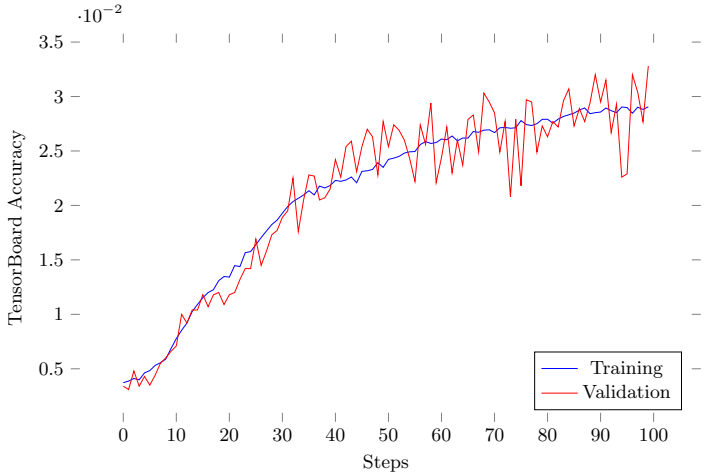
These observations are echoed when targeting the `AddRoundKey` outputs and the key bytes directly, as shown in Figures 4b and 4c respectively. Although the neural networks outperform the other templating methods most of the time, there exist intermediate variables that are better classified by either univariate or LDA classifiers (e.g. t_3, k_{11}).

5.1 Theory in Practice? Evidence for the NFL at Play

Should we expect that any single optimiser (learning approach) could be the best across a range of different leakage functions? The "no free lunch" (NFL) theorems for supervised learning suggest that a truly best learning algorithm

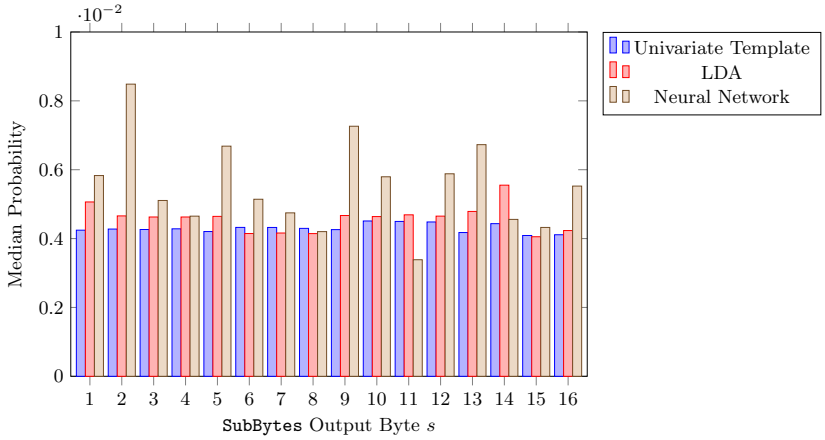


(a) TensorBoard Training Plots training for s_1 using 6000 epochs

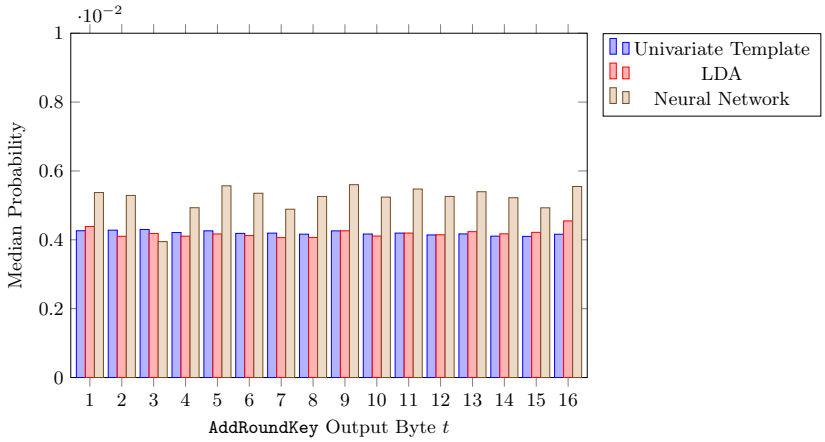


(b) TensorBoard Training Plots training for s_1 using 100 epochs

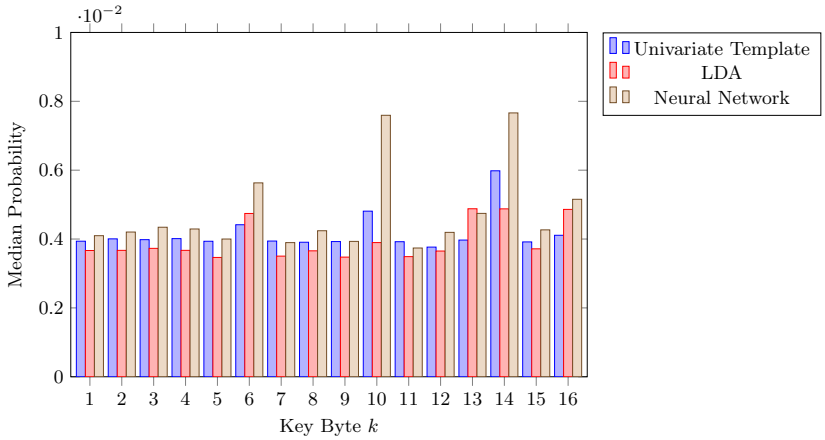
Fig. 3: TensorBoard Training Plots training for s_1 using different numbers of epochs; network parameters maximising the Median Probability metric



(a) Plot showing the classification results of the `SubBytes` output using Median Probability as a performance metric



(b) Plot showing the classification results of the `AddRoundKey` output using Median Probability as a performance metric



(c) Plot showing the classification results of the key bytes using Median Probability as a performance metric

Fig. 4: Plot showing the classification results of various intermediates using Median Probability as a performance metric

cannot exist in general [24]. The reasoning behind the NFL is that we cannot know what we have not seen: in other words, any learning algorithm only knows the "structure" of the training data, but, assuming that the training data does not represent the full input space, a classifier cannot be expected to generalise to the unseen test data.

Specifically theorem 2 from [24] says that for a number of probabilistic measures of "error" (on a data set distinct from the training set), the average performance of any (pair of) algorithms is the same. Putting this more concretely, any algorithm that turns out to generalise well for some particular dataset will perform badly for some other dataset. Only if we have some prior knowledge, and the training and test data are known to have the same distribution, we can choose an a priori best algorithm. For instance, if we know that data is linearly separable in both the training and the test data set then an optimal classifier can be configured accordingly; but this also means that we need some prior step or information to learn this fact. Thus we cannot hope to achieve generalisation in a black box setting.

In the case of typical power or EM leakage attacks we tend to encounter a variety of leakage functions: "tame" 8-bit devices (such as those used in [11]) often just leak the Hamming weight, but slightly more complex devices already might leak the Hamming distance between consecutive data values; typical 32-bit ARM processors of the CORTEX M family have confirmed second-order leakage [16], and more complex devices (whether they feature cryptographic hardware or not) have leakage functions with statistically significant higher order terms.

In the case of simple linear leakage functions (noiseless or very little noise), there is the possibility of an optimal classifier (given very large sets of sample data): previous work has shown that a deep linear net will converge to a true global minimum for the training [1]; but such a simple function can be very effectively characterised with standard statistics and does not benefit from any more sophisticated approach.

In the case of more complex leakage functions where the training dataset cannot adequately reflect the nature of the test data, the NFL naturally applies: this means that we cannot expect to find any learning algorithm (may this be some form of deep neural net, a classical machine learning approach, or Bayesian classification) that is optimal across all possible leakage functions. Thus for any intermediate value on an "interesting device" a different learning approach could beat the others (i.e. it generalises best for some test data, but badly for other cases). It is practically infeasible to try and find the optimal learning algorithm for each intermediate — in particular if neural nets are of interest because there are many different types, each of which requires finding the best set of hyperparameters.

Thus the behaviour that we observed for the classifiers on the M0 data (which we know features a range of challenging leakage models), is, from our point of view, a manifestation of the no free lunch theorem.

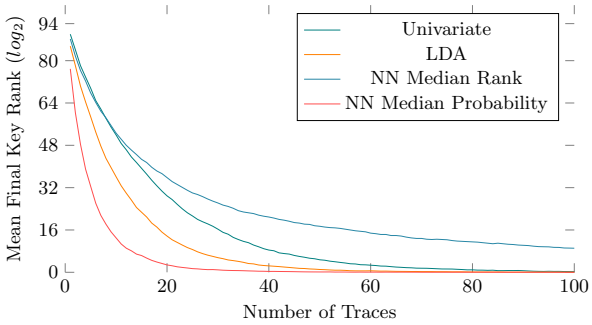


Fig. 5: Multi target attack scenario

No other published work in the side channel context has made this observation before, most likely because other work hasn't attempted to derive an architecture that generalises.

6 Performance in Attacks

Our main attack scenario is that of a multi target attack using belief propagation. We therefore use all developed classifiers in this context to determine which one gives the overall best performance in some actual attacks. Thereafter we also investigate the performance of classifiers in the context of a single target attack.

6.1 Multi target attack scenario

A concrete attack in the belief propagation setting samples a new leakage trace (for an unknown key), utilises some classifiers to extract information about the intermediate values, and feeds their classification results (in the form of probability distributions) into an implementation of belief propagation ([7] in our case). The result of the belief propagation algorithm is then a set of probability distributions (one for each subkey). Using a canonical key rank algorithm (e.g. [12]) we can derive the rank of the (known) key in our (certification) attacks. Successively adding one trace at a time, we create a performance graph for the three classifiers (univariate templates, multivariate LDA, deep networks).

Figure 5 shows the results of using different classification methods. Following on from the results shown in Section 4, the rank based network performs badly and seems to show no significant improvements from about 80 traces onwards.

The best attack performance comes from using the probability based network, which achieves first order success after around 30 traces, and outperforms both the univariate and the LDA classification methods. It brings the key space down to an easily enumerable 2^{40} with just a couple of traces and thus is extremely efficient at utilising the available information.

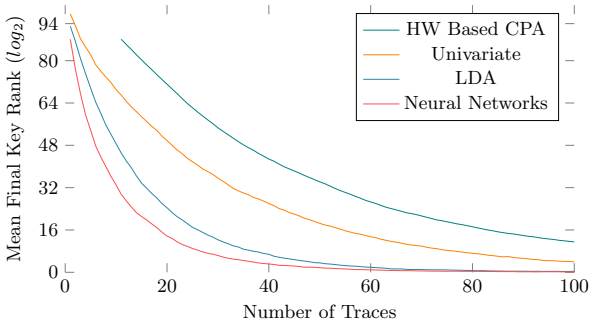


Fig. 6: Single target attack (targetting the `SubBytes` output, aka DPA style attack).

6.2 Single target attack scenario

A question that naturally arises is whether our tuning for single trace classification impacts on the use of our trained deep networks in DPA style template attacks. Figure 6 compares the attack performance when targeting the `SubBytes` output using the different profiling/learning methods. We also include the performance of a non-profiled DPA style attack as a reference (using correlation as a distinguisher and the Hamming weight of intermediates as a power model). The profiled attacks outperform the non-profiling DPA, with the deep network based classifier being again the most efficient. It successfully recovers the key with first-order success in 60 traces, and brings the key space down to an easily enumerable 2^{40} with just under 10 traces.

7 Conclusions

This paper tackles the challenge of developing a deep network architecture that generalises to many intermediate values (in the context of a multi target attack scenario). The task of determining the network architecture is about finding suitable hyperparameters for a chosen type of neural net (an MLP in our specific use case). If many neural nets have to be trained, then, in principle, a new architecture would first need to be developed for each of them. We, however, develop a technique, which enables us to develop an MLP architecture that successfully generalises to many intermediate values for a given AES implementation.

One might argue that this should be impossible, because of the "no free lunch theorem" of supervised machine learning. We do not dispute that this theorem applies: on the contrary we believe that our experiments generate (for the first time in our community) evidence that the theorem actually applies to our practice. However, our network architecture, whilst not being the best classifier for each and every intermediate (as to be expected due to the no free lunch theorem), outperforms the other classifiers for many intermediates.

Thus we are the first to develop and demonstrate an architecture that can generalise, and in doing so, we put forward a number of interesting observations.

References

1. S. Arora, N. Cohen, N. Golowich, and W. Hu. A convergence analysis of gradient descent for deep linear neural networks. *ArXiv*, abs/1810.02281, 2018.
2. L. Batina, J. Hogenboom, and J. G. J. van Woudenberg. Getting more from pca: First results of using principal component analysis for extensive power analysis. In O. Dunkelman, editor, *Topics in Cryptology – CT-RSA 2012*, pages 383–397, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
3. E. Cagli, C. Dumas, and E. Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In W. Fischer and N. Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.
4. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski, ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
5. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.
6. J. Green, A. Roy, and E. Oswald. A systematic study of the impact of graphical models on inference-based attacks on aes. CARDIS, 2018. `https://eprint.iacr.org/2018/671`.
7. J. Green, A. Roy, and E. Oswald. A systematic study of the impact of graphical models on inference-based attacks on aes. In B. Bilgin and J.-B. Fischer, editors, *Smart Card Research and Advanced Applications*, pages 18–34, Cham, 2019. Springer International Publishing.
8. K. Hornik, M. B. Stinchcombe, and H. White. Multi-layer feedforward networks are uni-versal approximators. 1988.
9. P. Karsmakers, B. Gierlichs, K. Pelckmans, K. D. Cock, J. A. K. Suykens, B. Preneel, B. D. Moor, K. H. Kempen, and I. Kleinhoefstraat. Side channel attacks on cryptographic devices as a classification problem. 2007.
10. J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):148–179, May 2019.
11. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
12. D. P. Martin, L. Mather, E. Oswald, and M. Stam. Characterisation and Estimation of the Key Rank Distribution in the Context of Side Channel Evaluations. In *ASIACRYPT 2016, Proceedings, Part I*, pages 548–572, 2016.
13. Z. Martinasek, P. Dzurenda, and L. Malina. Profiling power analysis attack based on mlp in dpa contest v4.2. pages 223–226, 06 2016.
14. Z. Martinasek, J. Hajny, and L. Malina. Optimization of power analysis using neural network. In A. Francillon and P. Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 94–107, Cham, 2014. Springer International Publishing.
15. Z. Martinasek, L. Malina, and K. Trasy. *Profiling Power Analysis Attack Based on Multi-layer Perceptron Network*, pages 317–339. Springer International Publishing, Cham, 2015.

16. D. McCann, E. Oswald, and C. Whitnall. Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.*, pages 199–216, 2017.
17. B. Poettering. AVRAES: The aes block cipher on avr controllers, 2003.
18. E. Prouff, R. Strullu, R. Benadjila, E. Cagli, and C. Dumas. 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`.
19. W. Schindler, K. Lemke, and C. Paar. A stochastic model for differential side channel cryptanalysis. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 30–46, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
20. S. Song, K. Chen, and Y. Zhang. Overview of side channel cipher analysis based on deep learning. *Journal of Physics: Conference Series*, 1213:022013, jun 2019.
21. M. Telgarsky. Benefits of depth in neural networks. *CoRR*, abs/1602.04485, 2016.
22. T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.
23. N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert. Soft analytical side-channel attacks. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 282–296. Springer, Heidelberg, Dec. 2014.
24. D. Wolpert. The supervised learning no-free-lunch theorems. 01 2001.