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

Joey Green, Tilo Burghardt, Elisabeth Oswald

Department of Computer Science, University of Bristol, Merchant Venturers Building, Woodland Road, Bristol, BS8 1UB, United Kingdom.
firstname.lastname@bristol.ac.uk

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. This paper contributes to this ongoing discourse by taking a slightly different angle: we train networks for many intermediates of a typical AES implementation on an ARM Cortex-M0 processor, and compare their performance with classical profiling methods. Because the cost of finding good hyperparameters for networks is high, we demonstrate how to configure a network with a set of hyperparameters for a specific intermediate (SubBytes) that can also be used for learning the leakage of other intermediates. 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. reasonable classification results across many intermediates with reasonable profiling effort). To put the trained classifiers into side channel practice we integrate them not only into a standard profiled single-target attack on SubBytes, but we use them as part of a (multi-target) belief-propagation attack.

Keywords: AES, Inference Based Attacks, Side Channel Attacks, Template Attacks, Deep Learning, Neural Network

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 [8]. Such methods build an explicit statistical representation of the average leakages for specific intermediate values and can therefore be used as predictors with comparison-based distinguishers. Machine learning methods have also been extensively studied (predominantly as classifiers), and many have found their use together with partition-based distinguishers.

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 [3,9,18]. 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), and papers utilise leaks from fairly simple processors.

The deep learning approach to side channel analysis proceeds in two stages, similar to template attacks 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 network model is then selected, informed by previous work [3] and carefully tuned based on the available training data [18], noting that adopting models built using separate data in a different scenario is challenging. The network is trained using the training data against appropriate labels, usually the Hamming Weight or identity value of the targeted intermediate value. 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 template attacks which require the adversary to find the most leaky time points manually (or via some other method, e.g. LDA [8] or PCA [2]), deep learning offers the tantalising promise of learning to automatically identify the most leaky time points even in misaligned traces [3].

Previous work utilising deep learning in side channel analysis considers batches of traces for testing to resemble the use of deep networks as part of DPA style attacks [10]. We, however, are interested in the use of deep networks as part of profiled attacks that combine the information that leaks from multiple intermediate values as part of an inference based attack, as described in [23,**?**]. Consequently we are interested in training networks that excel in classifying single traces.

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 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 to full input space, it 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 priori knowledge, and the training and test data are known to have the same distribution, we can choose an a priori best algorithm (e.g. 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 that means we didn't need to learn this fact).

In the case of typical power or EM leakage 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 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). 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.

1.1 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. [9] 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 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.

We can see from the rather different findings reported in the previous studies that there is no clear best learning or profiling approach: we suggest that this is not in any way unexpected or that papers contradict each other. It seems more likely to be a manifestation of the 'no free lunch' theory. Given that there is no optimal learning algorithm, and the cost of training and tuning deep nets in particular, is there any practical way forward in contexts where we want to take advantage of the many leaking intermediates that typical cryptographic implementations offer?

1.2 Contributions and Outline of this Paper

Contributions Our approach is based on training MLPs for many intermediate values from an `AES FURIOUS` [17] implementation on an ARM Cortex M0 processor. Alongside training the networks we also build classical (Gaussian) templates and LDA based templates. 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). 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 very similar and they are all complex enough to be interesting (i.e. they are not pure Hamming weight).

In contrast to other work, we concentrate the tuning of our deep networks to maximise the per trace classification performance (instead of using a batch of test traces). Because our training sets have size one, cross validation does not make sense in estimating the networks performance.

Our findings confirm that the NFL may have a role to play: across all intermediates (trained networks, and classical profiling methods) there is **no** clear winner: 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. Another interesting aspect of our work is related to the choice of metric to judge the classification performance of a network. Initially we utilised the "median rank" metric (as per [18]) to judge the classification performance, and by doing so selecting the best network configuration (hyperparameters). Using "median rank" only led to networks that behaved in a rather arbitrary and poor manner when trained for different intermediates. We switched to using the "median probability" as a measure and determined the best hyperparameters for a network learning the leakage related to SubBytes: this particular network configuration was able to learn very efficiently the leakage for all other intermediate values. Thus our conclusion that even though a free lunch isn't possible, we can still get a relatively cheap lunch.

Outline We briefly review the working principle of profiled attacks, focusing on standard univariate templating methods and Linear Discriminant Analysis in Section 2. Alongside this we provide information about our attack setup, as well the relevant implementation details of the AES implementation on which we base our experiments. In the same section we discuss the details of multi-layer perceptrons in the context of side channel analysis, with a focus on the results of [18].

In Section 3 we investigate whether reusing the MLP architecture from [18] could work and conclude that the performance is too weak, measured by three different metrics: median rank, median probability and mean probability. The rank metric shows that the MLPs are nearly as bad as a random guess. There is an interesting difference between the median probability and the mean probability outcomes, in particular for the ASCAD CNN which has very low median probability but a high mean probability indicating that it is confident but wrong.

Thus we require a different network structure, which we initially attempt to determine by using a rank based metric. In Section 4 we show that, in our use case, using the 'rank' of the classified value is an inappropriate metric that leads to poor results. We propose to use the median probability to fine tune the network hyperparameters, which results in a large improvement in the results.

Section 4.3 compares the results of the newly configured MLP with classical univariate templates and LDA based multivariate templates. We still find

evidence of the no free lunch theorem: different methods "win" for different intermediates.

Finally we compare the learning algorithms in the context of two attacks (a DPA attack on SubBytes and a belief propagation attack on the first two AES rounds) in Section 5.

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

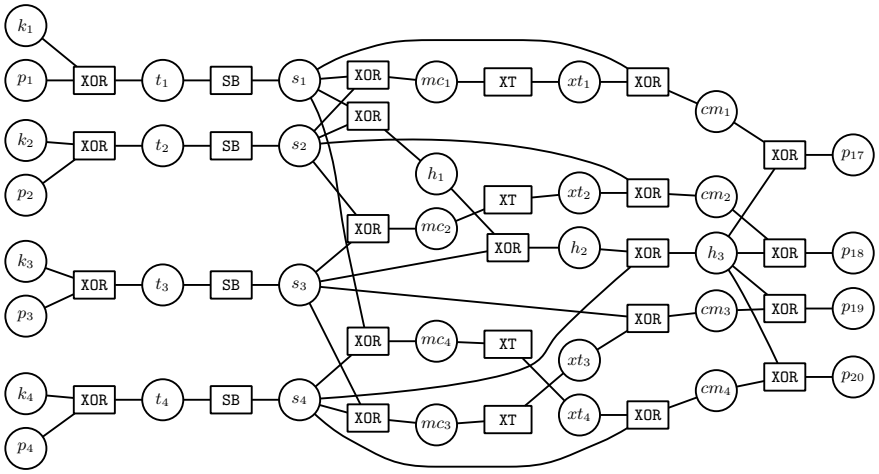


Fig. 1: Factor Graph representation of the first column of the first round of the `AES FURIOUS` implementation

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. Figure 1 is provided as a visual aid, showing the first column of the first round, providing insight into how the `MixColumns` step is performed.

The physical device we target is based on an ARM Cortex-M0 of the LPC series, and contains an on board 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 Linear Discriminant Analysis (LDA), we use 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 Classical Profiling

(Gaussian) Templates A template is defined as a mean vector and a standard deviation vector pair (μ, σ). These 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. Suppose we are targeting the first output byte of the `SubBytes` operation in AES. We acquire a large number of traces from the copy of our target device, using random keys and plaintexts. By partitioning the traces according to the value of the first output byte of the `SubBytes` operation, then only selecting the power value at our point of interest, we end up with 256 sets of power values. By taking the mean and standard deviation of each set, we end up with 256 templates.

The template matching phase (or *online phase*) involves matching the templates to unseen power leakage. We use Bayes' rule to obtain predictions:

$$P(y = c|X) = \frac{P(X|y = c)P(y = c)}{P(X)} = \frac{P(X|y = c)P(y = c)}{\sum_i P(y = i|X) \cdot P(y = i)}$$

By calculating the Gaussian probability density function of a leakage value with all our templates, we get a vector of probabilities that correspond to how

likely that value was at being the correct identity of the target. We can normalise and combine these probabilities over multiple traces to form a ranking of the possible key.

Linear Discriminant Analysis 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. By considering data over a large window surrounding a point of interest, the LDA approach is an example of a multivariate templating method.

We consider multiclass LDA, as we wish to classify each intermediate byte (256 values). In the training phase, LDA models the conditional distribution of $P(X|y = c)$ for each class c (and a vector of power values X). In the classification phase, LDA uses Bayes' rule to obtain predictions, in an identical manner to the Gaussian templating method.

2.3 Neural Networks

Neural networks are frameworks for machine learning algorithms to process complex data inputs. These networks have the ability to "learn" how to perform tasks when provided with input data, even without having any knowledge on how to perform the task in advance. Neural networks are used extensively in the field of image recognition, where they are able to identify images that contain specific objects, by analysing some training data where the images have been marked whether the specific object is present (or not). Our aim was to use a Neural Network to classify an intermediate variable. That is, if we provide the network with the power consumption of the target device over the period of time when an intermediate variable is computed (and stored in a register), then the network will provide us with a likelihood vector of the possible intermediate values.

A neural network has an input layer, an output layer, and a number of 'hidden' layers. Each layer is made up of a number of neurons (often called nodes) which mimic the function of a neuron in the brain. The neurons perform an evaluation of its input data, and pass the result of these evaluations onto the next layer in the network. There are multiple ways of structuring this network, and hence many models from which to choose; some excelling in different scenarios. 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.

Multi-Layer Perceptron A Multi-Layer Perceptron is an example of a feed-forward 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.

MLPs use backpropagation for training; this technique is commonly used in networks with multiple hidden layers. The advantage of an MLP over a standard linear perceptron is that it is able to distinguish data that is not linearly separable. In practice, they are a very quick network to train and test, compared to some of the more complex models.

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.

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 [5,7]. 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 In order to train our network to classify a variable node x, we require training data and training labels. We provide a large window of power values within which we are certain the targeted intermediate is loaded / stored in memory. This is to ensure we capture the leaking value without providing the entire trace, which would be providing redundant information. Along with the training data we provide our training labels in the form of the correct value of variable x for each trace. We are able to know this value as we use the copy of the target device to generate the training information - by using a known plaintext and a known key, we can compute the values of all intermediate variables.

To train our networks, we use 190000 traces. All of these traces use a random key and a random plaintext. This number was chosen as we found it to be sufficient to generate accurate models to classify the data.

Validating The validation data also uses random keys and random plaintexts. This data is used during training to validate the current effectiveness of the model in training 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 the key was fixed. In our 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. Due to the work presented in [19], as the AES SubBytes operation has the EIS property (*equal images under different subkeys*), we test based on a single key. 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).

Success metrics The result of using a classifier on some new test data is a vector d that represents an estimated probability distribution for the unknown subkey $k \in \mathcal{K}$. Several metrics exist in the side channel context that measure the performance of a classifier based on d. The classification outcome for the correct subkey value k^* is given by $d[k^*]$.

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

$$\text{rank}(k^*, T) = |\{k \in \mathcal{K} | d[k] > d[k^*]\}| \tag{1}$$

The rank is a random variable and thus measuring it based on a single new test trace is not useful. A better metric will take multiple new test traces $\mathbf{T}$ into account, and we will be studying the behaviour of the median rank, alongside the median probability, defined in Eq. 2 and Eq. 3 respectively.

$$\text{medianRank}(k^*, \mathbf{T}) = \underset{T \in \mathbf{T}}{\text{median}}(\text{rank}(k^*, T)) \tag{2}$$

$$\text{medianProbability}(k^*, \mathbf{T}) = \underset{T \in \mathbf{T}}{\text{median}}(d[k^*]) \tag{3}$$

Training labels are one-hot encoded; they are vectors of size 256 where all values are zeros, bar a single 1 at index k^*, as shown in Eq. 4.

$$\text{oneHot}(k^*) = (\underset{0}{0}, \ldots, \underset{k^*}{1}, \ldots, \underset{255}{0}) \tag{4}$$

Therefore, the loss function cross entropy can be defined in Eq. 5.

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

2.4 The ASCAD MLP and results

The paper titled *Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database* [18] was a significant step towards studying the effects of Deep Learning when applied in a Side Channel Analysis context. They work with an AES implementation (similar to `AES FURIOUS`) on a simple 8-bit device, which is known to predominantly leak the Hamming weight of intermediate values.

The paper featured a number of important contributions regarding the use and performance of two different types of deep neural nets. For our work the following aspects are of particular importance:

- they provide a detailed discussion of choosing hyperparameters in the context of learning the leakage from an AES SubBytes intermediate;
- they compare an MLP to a CNN (the CNN is best suited to learn misaligned traces, but the MLP is marginally better for aligned traces; template attacks outperformed both deep learning approaches).

Because the interest of our study is not to defeat hiding or masking countermeasures, but to explore if it was possible to build many neural nets (for many intermediates) efficiently, we opted to use an MLP as our basic choice of neural net.

In the ASCAD paper they split their choice of hyperparameters into two parts: training parameters and architecture parameters. The authors select hyper parameters either one by one or two at a time, and manually test their network using 10-fold cross validation with different sizes of dataset. Upon finding a locally optimal value for a hyperparameter, this parameters is saved and used when determining the next hyperparameter. In order to determine the hyper parameters, the training set is partitioned into 50,000 profiling traces and 10,000 validation traces.

In a first step the training parameters are determined. Table 1 shows the training parameters (parameter, set from which to choose from, actual choice) in the order in which they were chosen (i.e. firstly the size of the training set (along with epochs, to keep a constant computational time for fair comparison), followed by the Batch Size and the Number of Epochs, and last the Learning Rate and Optimiser).

In a second step the architecture parameters are chosen. Table 2 lists the parameters again in the order by which they were determined. The output layer of the network consists of the Softmax function, which "converts" the output to a probability distribution. The loss function is always chosen to be categorical cross entropy, and no other loss functions are considered. There are a number

Table 1: The tested values and best values for chosen training parameters for the MLP in the ASCAD paper.

Parameter	Tested Values	Best Value
Size of Training Set	10000, 20000, 30000, 50000, 70000, 90000	50000
Number of Epochs	100, 200, 400	200
Batch Size	50, 100, 200, 500	100
Learning Rate	10^{-7}, 10^{-6}, ..., 10^{-3}	10^{-5}
Optimiser	Adadeita, Adagrad, Adam, RMSProp, SGD	RMSProp

of performance figures shown in the ASCAD paper, but it is not entirely clear if they are related to attacking always the same SubBytes leakage (we assumed this to be the case). With this assumption we can summarise their findings for the aligned traces as follows: the MLP performs marginally better than the CNN, whereby the MLP reaches (stable) first order success after around 300 traces.

3 Initial study: (re)using the ASCAD MLP and CNN on the M0 data

The starting point for our investigation was the ASCAD MLP (and CNN) because although we use a more complex processor, the intermediate values produced by `AES FURIOUS` are all 8 bit values (thus architecturally the implementations are similar). However, as elaborated in the introduction, the M0 processor features a range of (interesting enough) leakage functions. We were thus interested if the ASCAD networks (i.e. an MLP configured with the ASCAD hyperparameters, a CNN with the ASCAD parameters, as well as the actual best ASCAD MLP and CNN) could be the basis for training also for the M0 (we fix a single SubBytes for this experiment). Given that the M0 leakage functions all include a strong Hamming weight component (like the ASCAD leakages), and are also based on 8-bit intermediate values, we hypothesised that the ASCAD MLP (and CNN) would be a reasonable initial choice (reusing existing network configurations is an accepted strategy in computer vision and was also the starting point in the ASCAD paper).

Table 3 shows the classification results. The first column refers to the type of learning (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 Gaussian 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

Table 2: The tested values and best values for chosen architecture parameters for the MLP in the ASCAD paper.

Parameter	Tested Values	Best Value
Number of Hidden Layers	3, 4, 5, 6, 7, 8, 11	4
Number of Nodes per Hidden Layer	20, 50, 100, 150, 200, 250, 300, 500	200
Activation Functions	Hard Sigmoid, Linear, ReLU, Sigmoid, Softmax, Softplus, Softsign, Tanh	ReLU

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 standard machine learning LDA approach. The ASCAD MLP and CNN architectures do not generalise at all to the M0 data. [1]

Table 3: 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

4 Mind your Metric

Reusing the ASCAD MLP or CNN leads to entirely unsatisfactory results on the M0 data. This could be because the ASCAD networks were trained on masked data and thus their architecture is specific to learning the combination of mask

[1] We are aware the the ASCAD architectures train on masked data, such that the network learns the bivariate distribution of the mask and the real value. We understand this may factor into the lack of generalisation of the ASCAD networks.

and masked SubBytes leakages rather than predominantly the SubBytes leakages. Therefore we have to reconsider the choice for the hyperparameters and essentially determine them from scratch.

Following the established approach of determining the hyperparameters recursively, we set out to determine a "best" MLP for the SubBytes leakages first. Concretely this implied that for a number of choices for a hyperparameter we tested the performance of the resulting network via a custom "testing" phase. Initially we followed the recommendations of the ASCAD paper; in particular the recommendation to utilise the **mean rank** (as defined over a single SubBytes output, which we chose to be the one on the first state byte) to judge the performance of a network. The mean rank is estimated from the test dataset via cross validation, which is known to result in a reliable estimation. Our test datasets consist of single traces however, and the subkey ranks after a single observation are too variable to be useful. Thus the mean rank is not a suitable metric for our use case. The median rank may be a much more robust choice, which we utilised instead.

4.1 Rank as metric

Figure 2a shows the performance figures for the 16 SubBytes leakages, where the MLP was optimised for the first SubBytes leakage. 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 (we reused the MLP that we trained for the first SubBytes). We certainly wouldn't expect it to perform much better here, but it is again 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 may not be related to the classifiers (as they are all very different) but how we measure their performance—via the median rank.

The subkey rank is a useful measure because it directly relates to how we evaluate attack outcomes, and is defined in Eq.1. However, typically this metric is used in conjunction with DPA style attacks, where due to the fact that we use many (enough) leakages, we 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

Table 4: 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

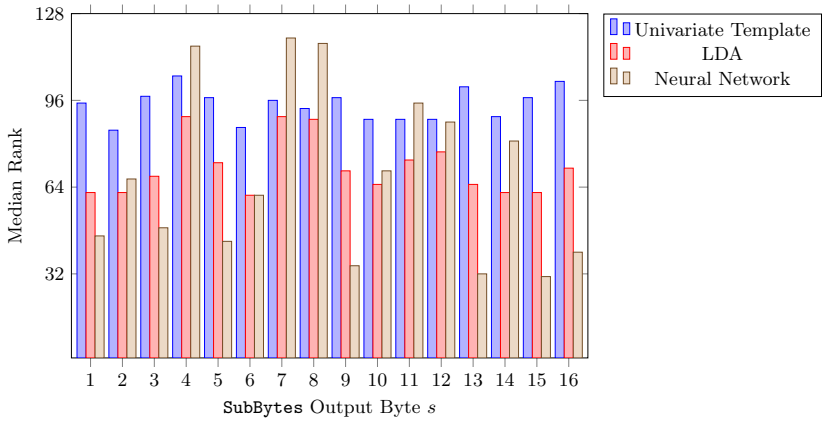
[a] The output layer uses the Softmax activation function to ensure the network outputs a normalised probability distribution

the resulting classification result. Thus maybe a better strategy for judging the classification performance could be to utilise the **median probability** as a metric.

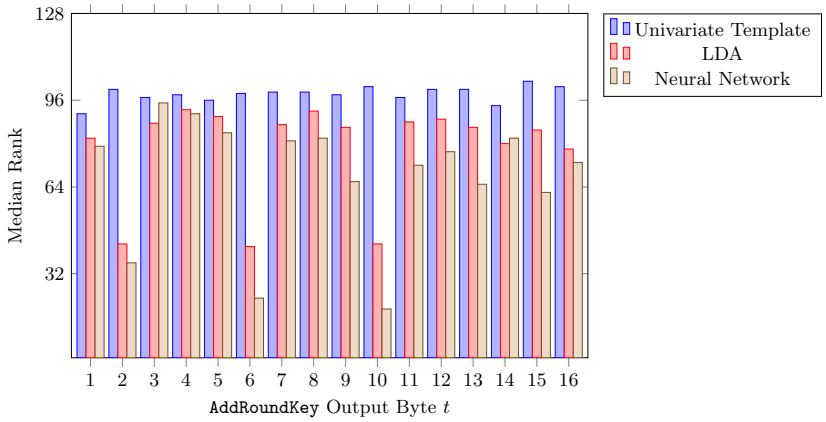
4.2 Probability as metric

Our next experiment consisted of optimising the MLP by judging its performance via the median probability, as defined in Eq. 3. Utilising the median probability as metric dramatically changed to configuration of the network, see Table 4 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 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.

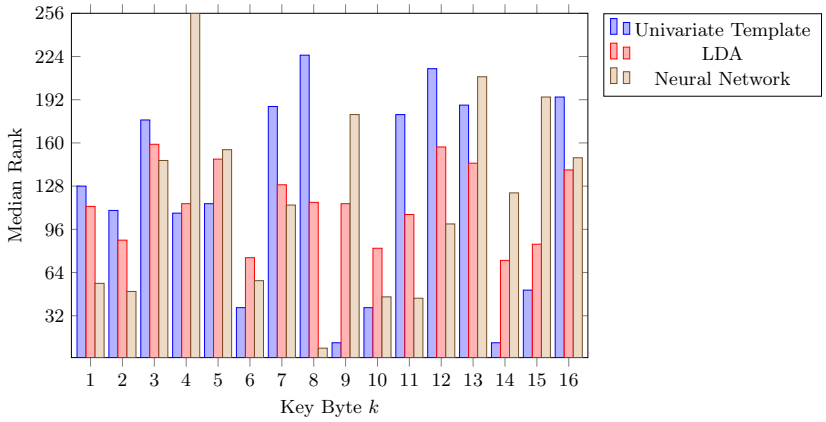
There is also a dramatic difference in the number of epochs between the rank based network and both the ASCAD and the probability based network. We investigated the behaviour on the training and validation data sets and noticed that the model overfits after around 2000 epochs, see Fig. 3a. Before this point, the model continues to 'learn' more about the task. However, upon comparing the median probability results on different numbers of epochs, we found a local optimum when using 100 epochs. This seems unusual; Fig. 3b shows the network after 100 epochs, and it appears it is still learning. However, past this point, we find a lower median probability classification result. This is most likely due to the discrepancy between the cross entropy loss function (used to calculate the



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



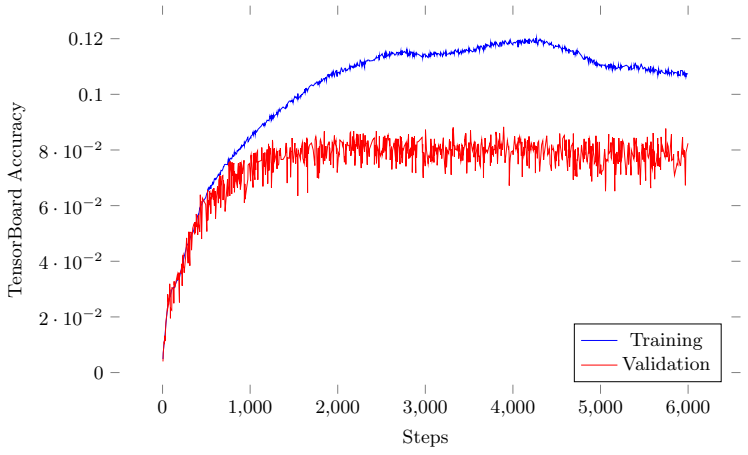
(b) Plot showing the classification results of the `AddRoundKey` output using Median Rank as a performance metric



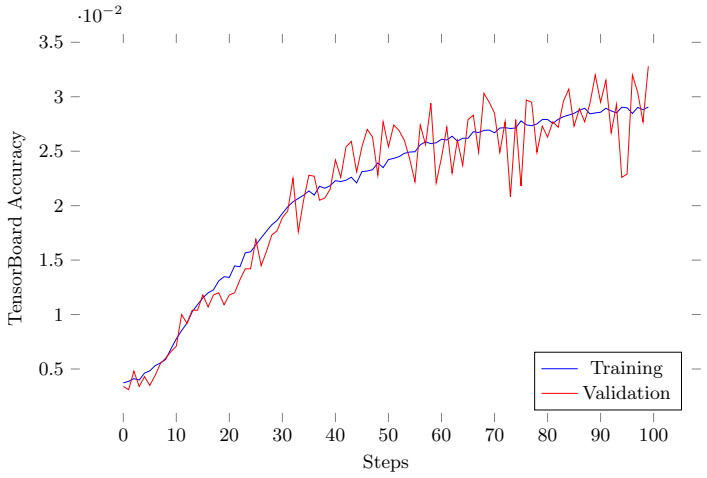
(c) Plot showing the classification results of the key bytes using Median Rank as a performance metric

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

training and validation accuracy) and the median probability metric we use to find the best model.



(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

4.3 No Free Lunch

We now compare the probability based deep networks to the univariate templating method, along with Linear Discriminant Analysis.

Figure 4a compares the median probability classification of the probability networks to the standard univariate templating method along with linear discriminant analysis, when targeting the output of the `SubBytes` step. In comparison to Figures 2a, 2b, and 2c, the classification performance is much more stable, which gives us more 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}).

This behaviour is, from our point of view, a manifestation of the no free lunch theorem, i.e. even restricted to a type of intermediate it is very likely that there is no single learning approach that always outperforms the other learning approaches. In our study though, the deep networks delivered the best classification performance for a majority of the intermediate values, which indicates that they should have the edge also when it comes to using them in concrete attacks.

5 Performance in Attacks

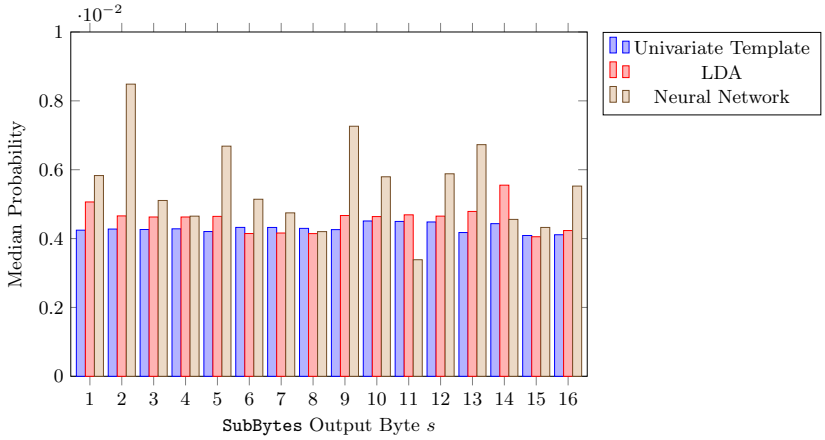
Our main attack scenario is that of a multi target attack that represents the worst case attacker. In this setting, an attack strategy that combines the information derived from the leakage of multiple intermediate values via belief propagation is currently regarded as the most effective.

5.1 Multi target using belief propagation

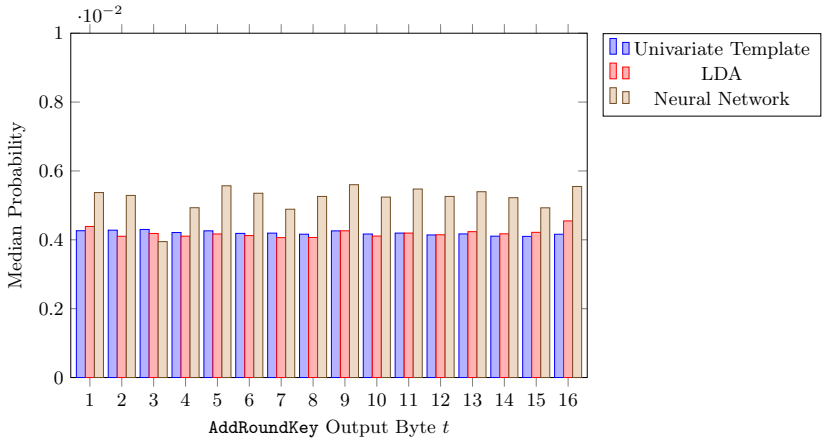
A concrete attack in the belief propagation setting samples a new leakage trace (for an unknown key), utilises some classifiers to extract information about the intermediate values, and feeds their classification results (in the form of probability distributions) into an implementation of belief propagation ([6] 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 5a 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.

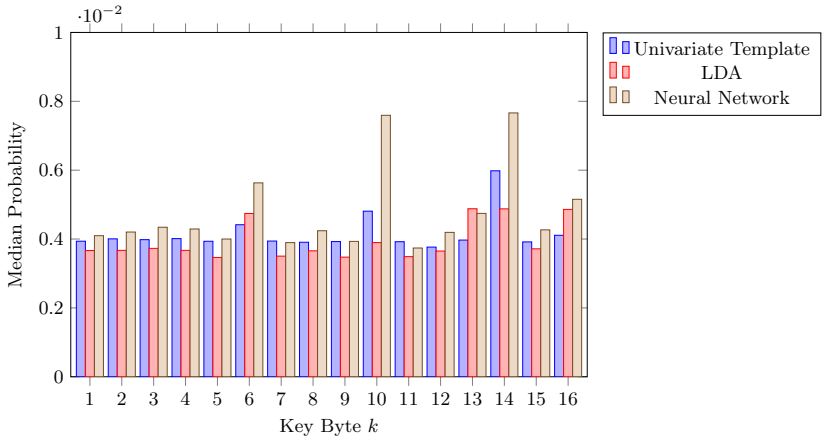
The best attack performance comes from using the probability based network, which achieves first order success after around 30 traces, and outperforms both the univariate and the LDA classification methods. It brings the key space down



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

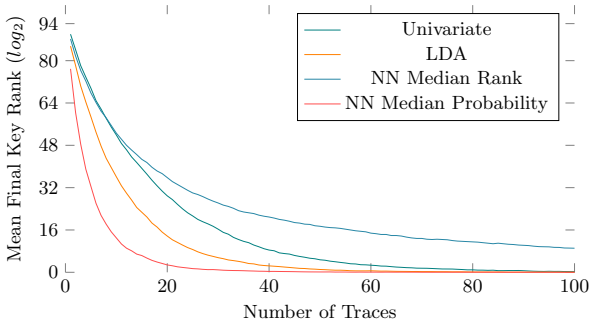


(b) Plot showing the classification results of the `AddRoundKey` output using Median Probability as a performance metric

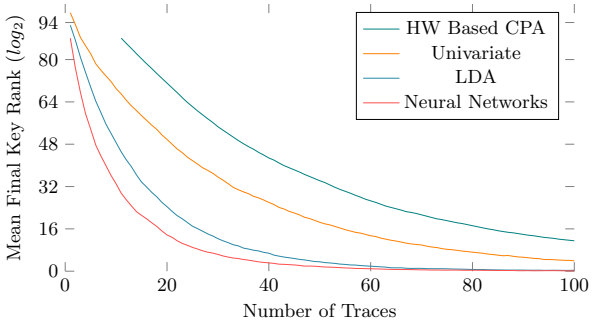


(c) Plot showing the classification results of the key bytes using Median Probability as a performance metric

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



(a) Running a multi target attack using belief propagation.



(b) Running attacks using different templating methods, solely targeting the `SubBytes` output (DPA style attack).

Fig. 5: Plots showing attack results

to an easily enumerable 2^{32} with less than 10 traces and thus is extremely efficient at utilising the available information.

5.2 Single target using correlation

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 5b compares the attack performance when targeting the `SubBytes` output using different profiling/learning methods. We also include the performance of a non-profiled DPA style attack 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.

6 Conclusions

This paper examines the challenge to determine a neural network architecture that works well as a basis to train Neural Nets for many (i.e. one hundred) intermediate values of an AES implementation, to be used in a worst-case belief propagation attack. Determining the network architecture is about finding suitable hyperparameters for a chosen type of neural net (an MLP in our specific use case), which is an onerous task because it involves training nets for many different hyperparameter configurations and validating them.

In our attempt to tackle the challenge of determining a good base architecture we initially utilise a recommended validation metric (median rank) and determine the hyperparameters that are best for one particular intermediate value (one instance of a SubBytes output). When we utilise the resulting best base architecture to train neural nets for other intermediate values our experimental results show that the best base architecture generalises badly (the validation results indicate poor performance for other intermediate values). We also observe poor generalisation when utilising other classical profiling methods. This enables us to make a novel observation: whilst the learning approach that we use for profiling differ, we utilised the same validation metric (median rank). We hypothesise that it is the choice of metric which leads to our unsatisfactory results. Median rank as a performance metric is intuitive when determining batch classification performance, but perhaps in our use case we should choose a metric that more accurately represent the per trace classification performance.

Consequently we reproduce our experimental results utilising this proposed novel metric (the median probability) during the validation phase to determine the best hyperparameters (aka network architecture). The resulting network shows a very different configuration to the previously determined network and generalises well across all the intermediate values.

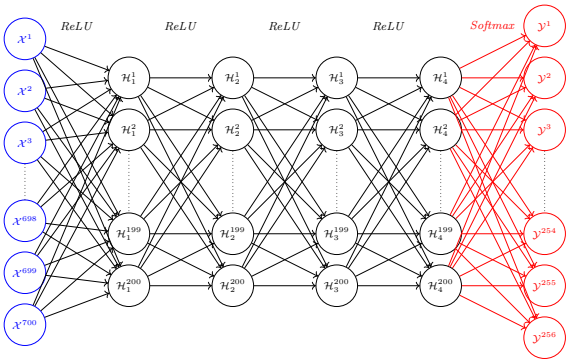
Thus we are the first to demonstrate an architecture that can generalise to more than one target intermediate in the context of side channel attacks, and we put forward an interesting observation regarding the choice of validation metric that should be of interest for further study.

Following our experimental results, we nevertheless conclude that the 'no free lunch' theorem holds in our context; our results show that there was no clear winner in classification method (comparing a Gaussian univariate templating method, a multivariate Linear Discriminant Analysis classifier, and the Neural Networks) over all intermediates. However, we are the first to demonstrate that it is possible to find a Neural Network architecture that generalises across different intermediate values, and thus provides the best overall classification results.

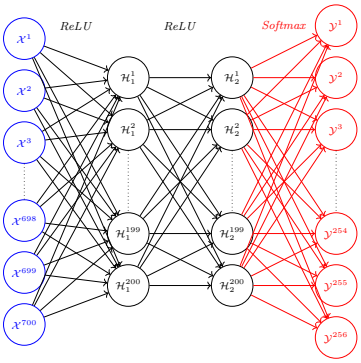
7 Acknowledgements

Joey Green has been funded by an NCSC studentship. Elisabeth Oswald was funded in part by EPSRC under grant agreement EP/N011635/1 (LADA) and the ERC via the grant SEAL (Project Reference 725042).

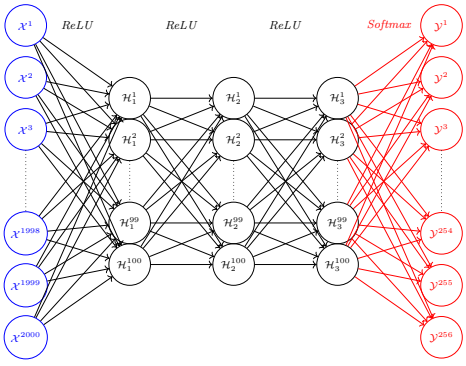
A Graphical Representation of MLPs



(a) ASCAD MLP



(b) Rank based MLP



(c) Probability based MLP

Fig. 6: MLP architectures

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. In B. Bilgin and J.-B. Fischer, editors, *Smart Card Research and Advanced Applications*, pages 18–34, Cham, 2019. Springer International Publishing.
7. K. Hornik, M. B. Stinchcombe, and H. White. Multi-layer feedforward networks are uni-versal approximators. 1988.
8. 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.
9. 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.
10. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, Aug. 1999.
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.