Scoop: An Optimization Algorithm for Profiling Attacks against Higher-Order Masking

Nathan Rousselot[1,2], Karine Heydemann[1], Loïc Masure[2] and Vincent Migairou[1]

[1] Thales, France, {firstname}.{lastname}@thalesgroup.com
[2] LIRMM, Univ. Montpellier, CNRS, France, {firstname}.{lastname}@lirmm.fr

Abstract. In this paper we provide new theoretical and empirical evidences that gradient-based deep learning profiling attacks (DL-SCA) suffer from masking schemes. This occurs through an initial stall of the learning process: the so-called *plateau effect*. To understand why, we derive an analytical expression of a DL-SCA model targeting simulated traces which enables us to study an analytical expression of the loss. By studying the loss landscape of this model, we show that not only do the magnitudes of the gradients decrease as the order of masking increases, but the loss landscape also exhibits a prominent saddle point interfering with the optimization process. From these observations, we (1) propose the usage of a second-order optimization algorithm mitigating the impact of low-gradient areas. In addition, we show how to leverage the intrinsic sparsity of valuable information in SCA traces to better pose the DL-SCA problem. To do so, we (2) propose to use the implicit regularization properties of the *sparse mirror descent*. These propositions are gathered in a new publicly available optimization algorithm, Scoop. Scoop combines second-order derivative of the loss function in the optimization process, with a sparse stochastic mirror descent. We experimentally show that Scoop pushes further the current limitations of DL-SCA against simulated traces, and outperforms the state-of-the-art on the ASCADv1 dataset in terms of number of traces required to retrieve the key, perceived information and plateau length. Scoop also performs the first non-worst-case attack on the ASCADv2 dataset. On simulated traces, we show that using Scoop reduces the DL-SCA time complexity by the equivalent of one masking order.

Keywords: Side-channel Analysis · Profiling Attacks · Deep learning · Masking · Optimization

1 Introduction

1.1 Context

To protect against side-channel attacks (SCA), the cryptographic community has developed two families of countermeasures: the masking countermeasures and the hiding countermeasures. One of them, d-th order masking, consists in splitting the sensitive values into $d+1$ random values, called *shares*, which when recombined, allow to recover the original sensitive variables. This allows to never manipulate sensitive data explicitly and hence theoretically protects against d-*th* order attack, *i.e.,* it protects against attacks recombining at most d time samples of the trace, such as attacks based on any statistical moment up to d.

A specific type of SCA, known as *profiling attacks*, consists in learning a statistical model of the leakage on a clone of the targeted device, and then using this model to recover

[0]This is a preprint of an article and is currently being revised.

the secret key on the target. Profiling attacks can be formulated as a machine learning problem, hence techniques such that neural-network based profiling attacks are found in the literature [MPP16]. Thanks to the universal approximation theorem [HSW89], neural networks, and more specifically multi-layer perceptrons, are theoretically able to break any higher-order masking schemes. Deep neural networks (DNNs) are constrained by a specific family of functions, the so-called architecture of the model. In some specific scenarios, such as against desynchronization, the choice of the architecture can be crucial, relaxing the need of pre-processing of the traces [CDP17]. However, despite many explorative works on architectures, DNNs still struggle in presence of higher-order masking schemes.

The problem we are interested in is targeting masked implementations with no prior knowledge on the masks on the clone device, the so-called *non-worst-case* setting [MCLS23].[1]

1.2 The Issue of Profiling in a Non-Worst-Case Setting

Deep learning models are trained by minimizing the empirical risk, the average of the loss function[2] over the data measured on the clone device (or *training data*). Hence, training a DNN is an instance of an optimization problem. In the general deep-learning case, this problem is usually solved by stochastic gradient descent (SGD) or some of its variants such as Adam [KB15], RMSprop, *etc.*, and the user is typically interested in the generalization properties of the model, *i.e.,* how well the model performs on unseen data.[3] Recent advances in applied optimization for DNN trainings are more focused on diminishing training costs, due to the large-scale problems posed by language model [LLH+23]. In the context of profiling attacks, the datasets are comparatively small (few thousands data samples versus billions) and appear to fall outside the scope of these advances, SGD is thus expected to be a good choice. Yet multiple works exhibit optimization artifacts [Tim19, PP20, CLM20, LZC+21, CLM23, MCLS23], in particular an initial stall of the optimization process, the so-called *plateau effect* [MCLS23]. This effect is particularly visible in the context of deep learning based profiling attacks against masking schemes. This plateau effect is conjectured to increase exponentially with the masking order [MCLS23]. This observation raises questions about the efficiency of deep-learning based side-channel attacks (DL-SCA) against higher-order masked implementations. The current DL-SCA literature is directly inspired by the general deep learning corpus, which implicitly relies on training routines (architecture, optimization algorithm, regularization, *etc.*) designed for optimal generalization on different tasks, being computer vision or natural language processing. Moreover, it assumes that the optimization process easily finds a local minimum. There is no guarantee that these routines and goals are adapted for DL-SCA.

Having an inappropriate optimization algorithm to train a neural network can pose critical issues. The plateau effect, as one can see in DL-SCA, leads to exponentially long waiting time before the model gains theoretical information about the targeted secret. Actually, even some second-order masking schemes datasets are still out of reach for the current state-of-the-art deep learning models [MS23]. Surprisingly, for some popular datasets that have been long-time considered as benchmarks such as ASCADv1 [BPS+20] (a first-order-masked dataset with some light hiding), years-old models are still competitive to the current state of the art. This lack of progress has led researchers to explore new architectures aimed at alternative tasks, such as multitask learning that targets the entire key at once [MO24], or transformer networks for their shift-invariant properties [HSAM22]. Unfortunately, none of them significantly improves the performance (*i.e.,* profiling phase) of DL-SCA against masked implementations.

[1] It is also referred to as a *black-box* attack in the literature.

[2] The loss function measures the difference between the model's prediction and the true value.

[3] In SCA: traces measured on the actual target device.

Yet, a long plateau does not imply a high-security device. Instead, all it implies is that the security level can only be assessed after this plateau phase is passed. Consequently, in case it requires weeks or even months to pass this phase, a security evaluation could give a false sense of security.

1.3 Contributions and Outline of the Paper

In this paper, we introduce Scoop, a novel optimization algorithm aimed at pushing the boundaries of deep-learning-based profiling attacks against masking. It addresses the aforementioned challenges by leveraging second-order optimization algorithms (Section 3) and mirror descent methods (Section 4). Scoop (Section 5) reduces the length of the plateau effect, and sets new state-of-the-art results on public datasets.

This algorithm is the result of a thorough theoretical and empirical analysis of the optimization process involved in the training of deep learning models for profiling attacks (Section 2). This analysis, based upon a visualization of the *loss landscape* of DL-SCA, leads to two observations. First, as the masking order increases, the magnitudes of the gradients decrease, slowing down the optimization process. Second, we observe the apparition of a saddle point at the initialization region of the loss landscape, which interferes with the optimization process.

These findings come from the theoretical study of a simplified model (Section 2.3.2). We believe this model to be a good starting point for further fundamental research in the field.

We show on an exhaustive simulated dataset that Scoop reduces the plateau effect (Section 6.1). Our experiments show that, despite the plateau effect still being present, its length is reduced by a factor up to 5 for a masking order of 3 and higher when training a CNN with Scoop, while increasing the profiling time cost by less than 5% on the same hardware.

We perform experimental evaluations on two public datasets: ASCADv1 [BPS⁺20] and ASCADv2 [MS23]. On ASCADv1 (Section 6.2), we verify the relevance of both the second-order optimization and the stochastic mirror descent, as well as the combination of both: Scoop. We show that DNNs trained with Scoop outperform the state of the art of ASCADv1 on the number of traces required to retrieve the key, perceived information, training cost and plateau length. In addition, we show that the hyperpameters optimization of a DNN trained with Scoop is computationally more efficient than with Adam. Finally, on the ASCADv2 dataset (Section 6.3), using Scoop, we perform a successful non-worst-case attack, which, to the best of our knowledge, has not been reached before[4].

2 The *Plateau Effect*: the Impact of Masking on DL-SCA

In this section, we revisit some aspects of the so-called *plateau effect* observed by Masure *et al.* [MCLS23], appearing in profiling attacks on higher-order masking. To this end, we first set some notations and recall some background knowledge in Section 2.1. We then recall the current knowledge on the origins of the plateau effect in DL-SCA (Section 2.2). Then, we present an analytical model to study DL-SCA (Section 2.3). From there, we conduct an analysis of the loss associated to this model, and find interesting geometrical and analytical properties at the root cause of the plateau effect (Section 2.4).

2.1 Notations and Background

In this paper, calligraphic letters such as $\mathcal{X}$ denote sets, random variables are written in uppercase letters, such as X, and their realizations in lowercase letters, such as x. Bold

[4]A gray box attack has been achieved [WPP23], although in a non-comparable setting, this is discussed in Section 6.3.

uppercase letters such as $\mathbf{X}$ denote random vectors. Matrices are noted in uppercase letters A and a_{ij} represents an element of this matrix. The eigenvalues of A are noted λ_i.

A probability is noted $\mathbb{P}[\cdot]$, expected value $\mathbb{E}[\cdot]$, variance $\mathbb{V}[\cdot]$ and $\mathbb{H}[\cdot]$ is the entropy.

Deep Neural Networks (DNNs) models are noted $\mathbf{F}$ and are constrained by a family of functions (the architecture) noted $\mathcal{H}$ a.k.a. *hypothesis class*. $\mathbf{F}$ is parametrized by $\theta \in \Theta$, where Θ is a convex subset of $\mathbb{R}^D$,[5] and the output of the model is noted $\mathbf{F}(\mathbf{x}|\theta)$ where $\mathbf{x}$ is the input data vector, *i.e.* a trace in SCA. The loss function is noted L. The gradient of the loss function is noted $\nabla L(\theta)$ and the Hessian matrix is noted $\nabla^2 L(\theta)$. Training the model $\mathbf{F}$ is done by minimizing the loss function L.

In DL-SCA, the targeted sensitive value is s and has a certain entropy $\mathbb{H}[s]$. A model $\mathbf{F}$ is said to be correct if $L(\theta) < \mathbb{H}[s]$ whenever $L(\theta)$ corresponds to a negative log-likelihood loss. The quantity $\mathbb{H}[s] - L(\theta)$ is called the *Perceived Information* (PI) of the model [MDP19a]. The number of traces required to nullify the Guessing Entropy (GE) is noted Na.

2.2 The Theoretical Origins of the Plateau Effect

The plateau effect is a phenomenon appearing in the beginning of the optimization process of DL-SCA. As already mentioned, it has been observed in multiple works [Tim19, PP20, CLM20, LZC+21, CLM23, MCLS23]. Masure *et al.* [MCLS23] aim at bringing an explanation to this artifact and link the plateau effect to the following theorem:

Theorem 1 ([SSSS17, Thm. 3])**.** *Let* $\mathbf{X}$ *be a dataset* $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$*, and assume each* $\mathbf{x}_i$ *is i.i.d.standard Gaussian in* $\mathbb{R}^p$*. Let's define the target function* $h_{\mathbf{u}}(\mathbf{X}) = \prod_{i=1}^{n} sign\left(\mathbf{u}^T \mathbf{x}_i\right)$*, for some normalized hyperplane* $\mathbf{u} \in \mathbb{R}^p$*. Let* $\mathbf{F}(\mathbf{x}|\theta)$ *be a predictor differentiable to its parameters* θ*, such that* $\mathbb{E}_{\mathbf{X}}[\|\nabla_\theta \mathbf{F}(\mathbf{x}|\theta)\|] \leq G(\theta)^2$ *for some scalar function* $G(\theta)$*. And let* $L(\theta)$ *be the loss function to minimize. Then,*

$$\mathbb{E}_{\mathbf{u}}\left[\|\nabla_\theta L(\theta) - \mathbb{E}_{\mathbf{u}}[\nabla_\theta L(\theta)]\|^2\right] \leq G(\theta)^2 . \mathcal{O}\left(\sqrt{\frac{n \log p}{p}}\right)^n .$$

This theorem, when adapted to DL-SCA context, informally tells us that for sufficiently high value of n (analogous to the masking order), the gradient takes essentially the same direction (or remains stuck at the same point) regardless of the actual leakage model (modelized here by the vector $\mathbf{u}$). In other words, the gradient descent cannot convey any information about what the model is trying to learn. While this theorem gives interesting insights, Masure *et al.* do not investigate it further. In particular, they do not explain *why* one could observe such a phenomenon. In this section, we aim at complementing the observations of Theorem 1 with further investigations.

2.3 Study of a Simplified Model

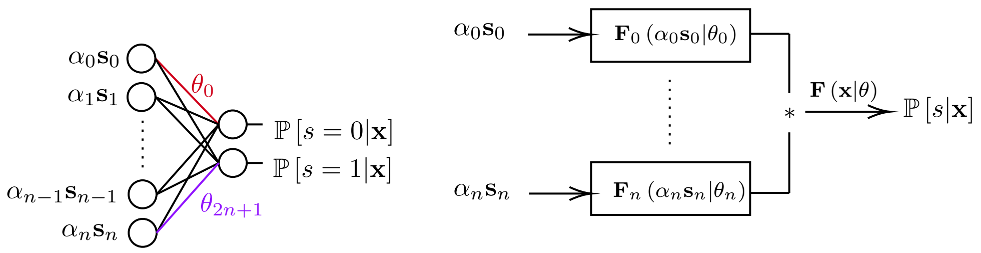
To reach analytical expressions of DL-SCA, we propose to study a simplified model on synthetic traces.

2.3.1 Analytical Framework

We consider an n-th-order masked implementation of a cryptographic algorithm. The synthetic traces are generated: $\mathbf{x} = [\alpha_0 s_0, \ldots, \alpha_n s_n]$, where s_i are the shares of a secret bit such that $s = s_0 \oplus \ldots \oplus s_{n-1} \oplus s_n$, where $\oplus$ is the operation[6] used to mask the secret s. All the

[5] *i.e.,* $\forall\left(\theta, \theta'\right) \in \Theta^2, \forall \alpha \in [0,1], \alpha\theta + (1-\alpha)\theta' \in \Theta$.

[6] In this paper, we consider boolean masking, but this is trivially reproducible with any masking scheme.



$\alpha_0 \mathbf{s}_0$
$\alpha_1 \mathbf{s}_1$
$\alpha_{n-1} \mathbf{s}_{n-1}$
$\alpha_n \mathbf{s}_n$
θ_0
θ_{2n+1}
$\mathbb{P}[s=0|\mathbf{x}]$
$\mathbb{P}[s=1|\mathbf{x}]$
$\mathbf{F}_0(\alpha_0 \mathbf{s}_0|\theta_0)$
$\mathbf{F}_n(\alpha_n \mathbf{s}_n|\theta_n)$
$*$
$\mathbf{F}(\mathbf{x}|\theta)$
$\mathbb{P}[s|\mathbf{x}]$

Figure 2.3.1: Classic model with $2(n+1)$ parameters (left) and *scheme-aware* model with $n+1$ parameters (right).

shares and the secret values are uniformly distributed in the $\mathbb{F}_2$ field. The α_i each constitute the leakage model such that α_i follows a Rademacher distribution[7]. Let us assume an arbitrary neural network model $\mathbf{F}$. The negative log-likelihood loss can be written as:

$$L(\theta) = -\sum_{\mathbf{x}\in\mathcal{X}} \log\left(\left\{\phi\left(\mathbf{F}\left(\mathbf{x}|\theta\right)\right)\right\}_s\right). \tag{2.3.1}$$

Where $\phi(\cdot)$ is the soft-argmax function and converts the output of the model into a probability distribution. Minimizing Equation 2.3.1 is equivalent to maximizing the perceived information [MDP19a]. By sampling a grid over Θ, we can visualize a discretized version of the *loss landscape*, *i.e.,* a D-dimensional tensor of the loss values w.r.t. the weights of the model. This visualization helps to understand the optimization process, identifying potential issues like local minima or saddle points, and gaining insights into the model's behavior. However, to make this representation visualizable, we need to reduce $D = \dim(\Theta)$ to be lower or equal to 2. While one can use principal component analysis to reduce dimensionality, it may not preserve the geometric properties that we try to investigate as it projects the data onto a new basis. A natural choice for a small model is to use a single-layer MLP model as illustrated in Figure 2.3.1 (left). However, this model has $2(n+1)$ parameters: *e.g.* for a first-order masking scheme, such a model has 4 parameters, consequently the loss landscape is a 4D tensor and is not visualizable.

2.3.2 Scheme-Aware Model

To go further in the simplification of the model we propose to use a *scheme-aware* model [MCLS23]. We consider a set of models $\{\mathbf{F}_0, \ldots, \mathbf{F}_n\}$, where:

$$\mathbf{F}_i\left(\alpha_i s_i|\theta_i\right) = \phi\left(\begin{pmatrix} \theta_i \alpha_i s_i \\ 1-\theta_i\alpha_i s_i \end{pmatrix}\right).$$

It returns that:

$$\mathbf{F}\left(\mathbf{x}|\theta\right) = \mathbf{F}_0\left(\alpha_0 s_0|\theta_0\right) * \cdots * \mathbf{F}_n\left(\alpha_n s_n|\theta_n\right). \tag{2.3.2}$$

Where $*$ is the convolutional product. A graphical representation of the scheme-aware model is depicted in Figure 2.3.1 (right). This time, for an n-share $\mathbf{x}$, the model has n parameters, hence the loss landscape is n-dimensional. An analytical expression of $\mathbf{F}$ can be derived from this model, either by hand, or by using a computer algebra system such as Sympy [MSP+17]. It has been observed that the scheme-aware models scale worse than the classic DL-SCA models [MCLS23]. Hence, there might be a bias while transferring our analysis to the general DL-SCA case. Nevertheless, results in Section 6 show that our findings using this model remain relevant in broader contexts.

[7]A Rademacher distribution is a distribution that takes the value -1 and 1 with probability $1/2$.

2.4 The Failures of the Optimization Process

The previous section sets an analytical formulation of DL-SCA, in this section we use it to theoretically study DL-SCA.

2.4.1 Understanding the Loss in DL-SCA

From the analytical formulation of $\mathbf{F}$ in Equation 2.3.2, we derive using Sympy the analytical expression of $\mathbf{F}$, of the loss and its gradient. We give hereafter the result for $n = 1$, *i.e.,* a two shares leakage:

$$\left\{\mathbf{F}\left(\mathbf{x}|\theta_0, \theta_1\right)\right\}_0 = \frac{e^{2\alpha_0 s_0 \theta_0 + 2\alpha_1 s_1 \theta_1} + e^2}{\left(e^{2\alpha_0 s_0 \theta_0} + e\right)\left(e^{2\alpha_1 s_1 \theta_1} + e\right)},$$

and the expression for its second component $\left\{\mathbf{F}\left(\mathbf{x}|\theta_0, \theta_1\right)\right\}_1$ is similar. The loss function can then be derived by taking the negative logarithm of the outputs: $L(\theta) = -\log_2\left(\left\{\mathbf{F}\left(\mathbf{x}|\theta_0, \theta_1\right)\right\}_s\right)$ for the secret value s. From there, we can also compute the expression of the gradient of the loss function. $\nabla_\theta L(\theta)$ belongs to $\mathbb{R}^2$ and its first component is given by:

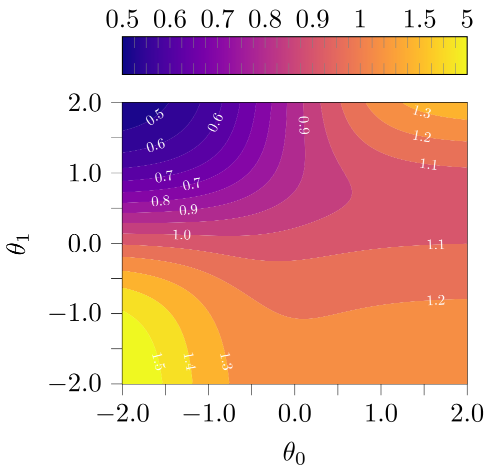
$$\left\{\nabla_\theta L(\theta)\right\}_0 = \frac{2\log(2)\alpha_0 s_0\left(e - e^{2\alpha_1 s_1 \theta_1}\right) e^{2\alpha_0 s_0 \theta_0 + 1}}{e^{2\alpha_0 s_0 \theta_0 + 2} + e^{4\alpha_0 s_0 \theta_0 + 2\alpha_1 s_1 \theta_1} + e^{2\alpha_0 s_0 \theta_0 + 2\alpha_1 s_1 \theta_1 + 1} + e^3}.$$

Hence, in this case, $L(\theta)$ is an $\mathbb{R}^2 \mapsto \mathbb{R}^2$ function, and by discretizing the Θ space, we can visualize the landscape of the loss and its gradient (Figure 2.4.1). We use a discretizing step $h = 0.04$ for computational purposes. Recalling that the secret belongs to $\mathbb{F}_2$, areas where the model correctly predicts the secret information are reached when $L(\theta) < 1$. It can be theoretically verified that the loss landscape's global minimum is unique and corresponds to $\theta_i = \alpha_i$ as shown in Appendix A. In Figure 2.4.1a, we see that the upper-left quadrant (meaning $\theta_0 < 0$ and $\theta_1 > 0$) corresponds to the area where correct models are reached, which is coherent with the leakage model which is $\alpha_0 = -1$ and $\alpha_1 = 1$. Within the bounds of observation set on θ_i, the loss reaches values below 0.5. Figure 2.4.1b shows the gradient of the loss landscape, blue areas are low-gradient zones, and hot-colors correspond to high-gradient zones. The red circle corresponds to the area of initialization, given a method that is discussed in Section 2.4.2. In addition, the white arrows illustrate the flow of the gradients, *i.e.,* the opposite direction of the gradient. We see that the white arrows tend to converge uniformly toward the correct quadrant.

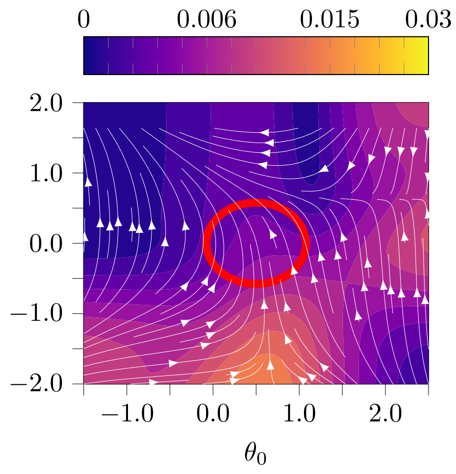
When adding a third share, the loss landscape significantly changes as illustrated in Figure 2.4.1c. First, it becomes a 3D tensor, making it harder to visualize. To overcome this, we decide to take a slice of the loss landscape by fixing $\theta_2 = \theta_2^\star$ where $\theta_2^\star$ is an optimal value of θ_2 that minimizes the loss within the observation frame. Hence, the resulting slice of the loss landscape contains a global minimum. Looking at Figure 2.4.1c, we observe that the loss does not reach values as low as in the two-share case, the paths to local minima are less steep. In other words, we say the *magnitudes* of the gradients decrease. Second, we observe an interesting geometry of the landscape of the loss in the neighborhood of the origin. To understand it, taking a look at the gradient of the loss landscape in Figure 2.4.1d is crucial. We see that around the origin, the gradient is very low (blue area) and that it attracts the gradients flow. Yet, looking back at the loss landscape (Figure 2.4.1c), we see that this region is neither a minimum nor a maximum. What we observe is then a saddle point. And as a consequence, we see that few gradients flow directly to minimal areas, and that paths often require a detour close to this saddle point before heading towards a local minimum.

2.4.2 Consequences on the Optimization Algorithm

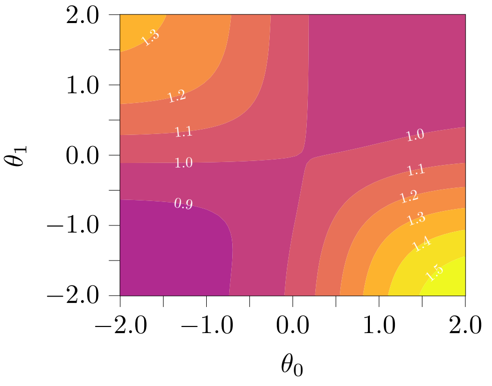
To understand the consequences of both the saddle points and overall loss of gradients' intensities on the optimization algorithm, we need to mimic the behavior of Adam [KB15]



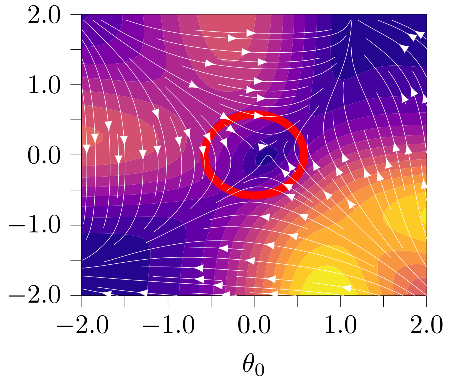
(a) d=2, $\alpha_0 = -1$ and $\alpha_1 = 1$



(b) d=2, $\alpha_0 = -1$ and $\alpha_1 = 1$



(c) d=3, $\alpha_0 = -1$, $\alpha_1 = -1$, $\alpha_2 = 1$



(d) d=3, $\alpha_0 = -1$, $\alpha_1 = -1$, $\alpha_2 = 1$

Figure 2.4.1: (a) and (c): Loss landscapes of the model $\mathbf{F}$; (b) and (d): Their gradients for first and second order boolean masking schemes.

and other SGD variants, as they are predominantly used in deep learning models training [Lan20], and are the only ones used in DL-SCA context [PP20]. Hence, we decide to study the behavior of regular gradient descent applied to our analytical analysis. The weights at iteration t are noted $\theta^{(t)}$, and are initialized following a *Kaiming uniform* distribution [HZRS15] (Equation 2.4.1), the implicit one used in PyTorch [PGM+19]. In Figures 2.4.1b and 2.4.1d, the red circle corresponds to the area of the weights' initialization. From there, $\theta^{(t+1)}$ is computed with the gradient descent update rule (Equation 2.4.2):

$$\theta^{(0)} \sim \mathcal{U}\left(-\sqrt{1/\text{in\_dim}}, \sqrt{1/\text{in\_dim}}\right), \tag{2.4.1}$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla L\left(\theta^{(t)}\right). \tag{2.4.2}$$

Here, η_t is the learning rate at step t, and in_dim the dimension of the traces. Section 2.4.1 shows that by increasing the masking order by one, the overall gradients' intensities is reduced. Noting that the step-size is directly proportional to the gradients' norms, it follows that the expected step size also decreases, which hinders the profiling performance for DL-SCA and increases the plateau length.

A natural reaction to this observation is to increase η_t to compensate the gradients' decaying. The limitation of this idea lies in analytical properties of L. The learning rate η_t is constantly bounded by a certain constant related to the so-called *Lipschitz* constant

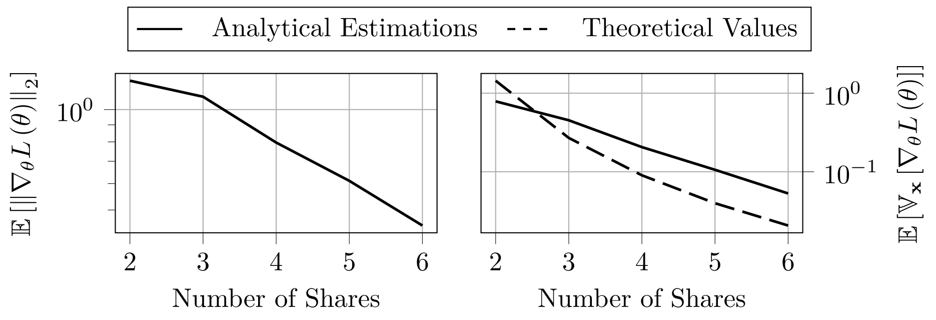


Figure 2.4.2: Impact of the masking order on the expected norm (a) and variance of the gradient (b) and comparison with Theorem 1.

of ∇L [N+18]. This constant is related to the local curvature of the loss landscape. As we do not know the exact value of this constant, we need to be cautious and set η_t to a small value to avoid loss divergence [KB15, LLH+23]. If one increases η_t to fight the plateau effect, loss divergence is then a likely outcome.

Another solution may be to use a learning-rate scheduler as it is a common practice in deep learning. Moreover, recent advances in optimization theory show that periodic learning rate schedules that are much larger than the Lipschitz bound lead to a better convergence [GSW23]. Despite schedulers have been used in DL-SCA [ZBHV19, HCM24], no formal study has been conducted to understand how to use them in DL-SCA context, and current attempts still exhibit the plateau effect [ZBHV19].

The second challenge posed by the loss landscape is the presence of a saddle point at the origin. Recalling the distribution of $\theta^{(0)}$ (Equation 2.4.1), we see that the higher the input dimension, the closer to the saddle point we initialize. This means that regardless of the locations of the local minima (*i.e.,* regardless of the countermeasures or weaknesses of the implementation), the optimization process inevitably detours around this saddle point before heading towards a local minimum.

To mitigate the effect of the saddle point, one might think to modify the initialization rule (Equation 2.4.1) to a larger area, diminishing the probability of falling around the saddle point. However, if normalization on the data is done, then the expected value of the weights should be zero, and with enough small variance to avoid learning saturation [BN06]. Hence, we are forced to use a distribution similar to Equation 2.4.1.

This finding is coherent with Masure *et al.* conjecture that the plateau length exponentially increases with respect to the masking order [MCLS23]. To validate our observations with analytical evidences, we scaled the analysis up to six shares and tracked down $\mathbb{E}[\|\nabla_\theta L(\theta)\|_2]$ and $\mathbb{E}[\mathbb{V}_{\mathbf{x}}[\nabla_\theta L(\theta)]]$ analytically in function of the masking order. The first quantity is related to the overall loss of gradients' intensity, and the second to the saddle point and the Shalev-Shwartz bound (Theorem 1). Figure 2.4.2 depicts the results of this analysis. We see in Figure 2.4.2 (left) that indeed the expected norm of the gradient decreases exponentially as the masking order increases. Figure 2.4.2 (right) shows that the variance of the gradients is also decreasing exponentially at the same rate as the Shalev-Shwartz bound.[8]

These two effects add up together and explain where the plateau effect is coming from. To the best of our knowledge, only Marquet and Oswald attempt to reduce the impact of the plateau effect through multitask learning [MO24]. While indeed showing slight improvements, their approach increases the complexity of the attack as all bytes

[8]Note that in Theorem 1, $G(\cdot)$ is an unknown scalar function, it was chosen to easily compare the rate of decrease of the variance of the gradient with the Shalev-Shwartz bound.

are targeted simultaneously. Moreover, plateau reduction seems limited. We see from this section that overcoming the plateau effect requires more thoughtful solutions.

3 Beyond Gradient Descent: Second-Order Algorithms

So far, we have only considered the gradient descent as a baseline for optimization algorithms, and only looked into its hyperparameters (learning rate, schedulers and weights initialization) as a solution to answer the challenges posed by DL-SCA profiling. In this section, we propose to reconsider the choice of the optimization algorithm itself. It would be interesting to have an adaptive descent, where all steps are made knowing the local curvature of the loss to better navigate the particularities of the loss landscape. In this section, we propose to use a second-order[9] optimization algorithm to serve this purpose. This section introduces the foundations of second-order optimization methods, their application to DL-SCA, and the challenges they face. SCOOP, our main contribution, is presented in Section 5.

3.1 Second-Order Optimization

Second-order optimization methods incorporate curvature information, typically represented by the Hessian matrix, in addition to the gradient. We first present the generic second-order optimization method: Newton's method, and then discuss its application to the DL-SCA context. Newton's method is based on the following update rule:

$$\theta^{(t+1)} = \theta^{(t)} - \nabla^2 L\left(\theta^{(t)}\right)^{-1} \nabla L\left(\theta^{(t)}\right),$$

where $\nabla^2 L\left(\theta^{(t)}\right)$ is the Hessian matrix of the loss function at step t. The inverse of the matrix induces large steps where the curvature of the loss is low, and small steps where the curvature is high. It is the optimal way to reach a local minimum in a convex scenario [N⁺18].

The deep learning community relies on estimates of the Hessian matrix as its full computation is not tractable. They also couple it with a learning rate [LN89], obtaining the following update rule with $\hat{h}_t = \mathbb{E}\left[\nabla^2 L\left(\theta^{(t)}\right)\right]$:

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \hat{h}_t \nabla L\left(\theta^{(t)}\right).$$

3.2 Second-Order Optimization on a Simplified Model

We verify the efficiency of Newton's method for DL-SCA against masking on the simplified model introduced in Section 2.3.2. We target second-order boolean masking as it already impacts the optimization process. Figure 3.2.1 compares the performance of Newton's method with the gradient descent on the simplified model. Dashed lines correspond to gradient descent, and solid lines to Newton's method. We can see that Newton's method converges faster than the gradient descent (Figure 3.2.1 (left)). Yet, due to the low entropy of the secret, the plateaus are not clearly visible by just looking at the loss. To estimate the length of the plateau, a good indicator is taking a look at the gradient of the loss with respect to the iterations $\delta_L = L\left(\theta^{(t)}\right) - L\left(\theta^{(t+1)}\right)$. Looking at Figure 3.2.1 (right), we can observe the end of the plateau at the local maxima of δ_L. We see then that the plateau length for the gradient descent is 380 iterations, whereas for Newton's method it is 50. The second-order optimization therefore reduced the plateau length by almost one order of magnitude.

[9]that uses second order derivative.

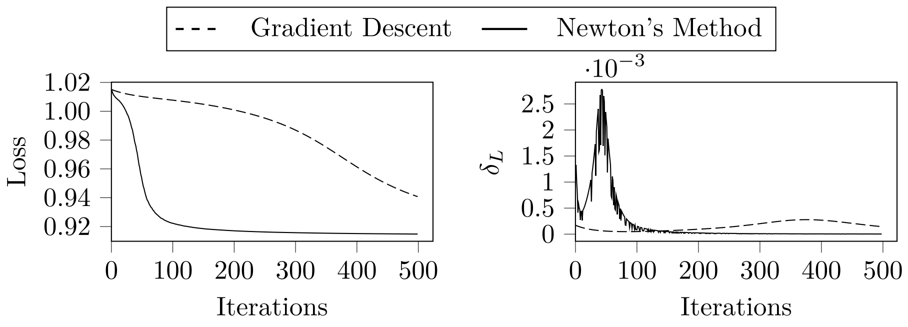


Figure 3.2.1: Loss convergence (left) and gradient of the loss w.r.t. the step (right) for simulated optimization on the simplified model considered masking scheme at order 2.

3.3 Challenges of Second-Order Optimization for Deep Learning

While Newton's method can be formulated simply, its application to deep learning is much more challenging. This section discusses these challenges, how they can be addressed, and proposes a novel Hessian estimation. This section does not tackle DL-SCA specific challenges, but the methods introduced here are later combined with other intrinsic DL-SCA aspects in Sections 4 and 5.

3.3.1 Ensuring Convergence

Unlike the gradient descent, Newton's method is prone to converge to local maxima, making its usage outside convex scenarios challenging. To prevent this, one needs to make sure that the estimation of the Hessian is positive definite (*i.e.,* all its eigenvalues are strictly positive) at each step. Some algorithms have been proposed to guarantee this properly, such as the Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm [LN89], which ensures each estimate of the Hessian matrix is positive definite as long as its first iterate also is. Usually one then picks the Identity matrix as the initializer since it is positive definite and does not privilege any direction. However, L-BFGS is both memory and computationally expensive, and hence not suitable for DL-SCA.

We propose another approach, giving more freedom to the computation of $\hat{h}_t$. Once the Hessian matrix is computed, we can shift its eigenvalues to ensure that the smallest eigenvalue is strictly positive. This can be done efficiently if we make sure enough *structure* to $\hat{h}$ (sparse, semi-separable, diagonal, *etc.*). Actually, this has already been applied in Section 3.2. However, by performing this spectral shifting, we considerably increase the *condition number* κ of $\hat{h}$ which leads to numerical instabilities when manipulating (*e.g* inversing) the Hessian. To illustrate this, let $\hat{h}$ be diagonalizable, then we have: $\kappa(\hat{h}) = |\max \lambda_i|/|\min \lambda_i|$, where λ_i are the eigenvalues of $\hat{h}$. Assuming there is at least one negative eigenvalue in $\hat{h}$, then by shifting them, we put $\min \lambda_i = \epsilon$ where ϵ is a small positive number. This means that $\kappa(\hat{h})$ becomes very large, and hence $\hat{h}$'s manipulation is numerically unstable. This issue can be mitigated by combining the spectral shifting with a *clipping* mechanism [LLH+23].

3.3.2 Efficient Estimation of the Hessian

Estimating the Hessian matrix in deep learning is challenging due to the high dimensionality and complexity of neural networks. The Hessian captures second-order derivatives of the loss function with respect to model parameters, resulting in a matrix whose size grows quadratically with the number of parameters, often being in the millions or billions for deep networks. Computing and storing such a large matrix is practically infeasible. This section aims at proposing an efficient solution to this problem.

Instead of trying to estimate the full Hessian matrix, we can focus on an approximating operator of the Hessian. This would result in an *inexact* Newton's Method that has been shown to still converge despite the bias in the Hessian estimate [DES82, BBN19]. Inexact Newton's Methods have already been explored in the context of deep learning, and have shown promising results [YGS+21, LLH+23]. Among all the approximating operators, the diagonal operator is the simplest to manipulate and to scale, and is hence the one we focus on. Liu *et al.* propose a computationally efficient diagonal Hessian estimator called *Gauss-Newton-Bartlett* (GNB) estimator [LLH+23]. Its name comes from the Gauss-Newton decomposition of the Hessian:

$$\nabla_\theta^2 L(\theta) = \mathbf{J}_\theta\left(\mathbf{F}(\mathbf{x}|\theta)\right)\frac{\partial^2 L(\theta)}{\partial y^2}\mathbf{J}_\theta\left(\mathbf{F}(\mathbf{x}|\theta)\right)^T + \mathbf{J}_\theta\left(\mathbf{F}(\mathbf{x}|\theta)\right)\frac{\partial L(\theta)}{\partial y},$$

where y are prediction logits. Note that the Gauss-Newton-Bartlett estimator is designed to work with the cross-entropy loss, which is the case in DL-SCA. In order to derive an efficient estimator, the authors assume that the second term of the Gauss-Newton decomposition is negligible compared to the first term and do not compute it. This results in a biased estimator (*i.e.*, $\mathbb{E}\left[\hat{h}\right] \neq \mathrm{diag}\left(\nabla_\theta^2 L(\theta)\right)$), but with intrinsic positive-definiteness. Liu *et al.* argue that this bias is small, however this assumption might not hold in a side-channel paradigm, where we have seen that well-adopted methods by the deep-learning community do translate poorly to. Instead, we propose to use the Hutchinson diagonal estimator.

Definition 1 (Hutchinson Diagonal Estimator [Hut89])**.** Let $\mathbf{z}$ be a *i.i.d.* random vector such that $\mathbb{E}[z_i] = 0$ and $\mathbb{V}[z_i] = 1$. Then the Hutchinson Hessian diagonal estimator is defined as:

$$\hat{h} = \mathbf{z} \odot \nabla_\theta^2 L(\theta)\,\mathbf{z}.$$

Interestingly, the Hutchinson estimator is unbiased. Furthermore, it relies on a simple Hadamard product as well as a Hessian-Vector product which can be computed efficiently (in linear time). The law of $\mathbf{z}$ is set freely as long as it is *i.i.d.*, has a zero mean and a unit variance. To have the lowest variance possible (and hence the fastest convergence), the Rademacher distribution is often chosen as per the following theorem.

Theorem 2 (Minimal Variance Diagonal Estimator [Hut89])**.** *Let us assume $\hat{h}_t$ is the Hutchinson estimator of the diagonal of the Hessian of $L(\theta)$, as per Definition 1. Then, the minimal variance estimator of the diagonal of the Hessian is reached when $\mathbf{z}$ follows a Rademacher distribution.*

Proof. The proof is recalled in Appendix B.2. □

Liu *et al.* argue that the Gauss-Newton Bartlett estimator might have a lower variance than the Hutchinson estimator, lacking however from theoretical or empirical evidence to back up this claim. Nevertheless, they are right to point out the issue of the variance, as the Hutchinson estimator is quite slow to converge. To address this issue, we propose a novel estimator (to the best of our knowledge) called the *Biased Hutchinson Estimator*. It is obtained by releasing the constraint on the variance on the law of $\mathbf{z}$ and allocating m iterations to the estimator. We establish the resulting optimal estimator in the following theorem.

Theorem 3 (Low Variance Biased Hutchinson Estimator)**.** *Let $\mathbf{z}$ be a random vector such that $\mathbb{E}[z_i] = 0$. Then, assuming m iterations of the Hutchinson estimator, then*

$$\mathbb{V}[z_i]^\star = \frac{\frac{m-1}{m}\left\|\mathrm{diag}\left(\nabla_\theta^2 L(\theta)\right)\right\|_2^2}{\frac{1}{m}\left\|\overline{\nabla_\theta^2 L(\theta)}\right\|_F^2 + \left\|\mathrm{diag}\left(\nabla_\theta^2 L(\theta)\right)\right\|_2^2},$$

is the optimal choice of variance to minimize the expected error at the m-th iteration. Here, $\overline{\nabla_\theta^2 L(\theta)}$ is the matrix $\nabla_\theta^2 L(\theta)$ with its diagonal set to zero, and $\|\cdot\|_F$ is the Frobenius norm.

Proof. The proof is given in Appendix B.3. $\square$

Corollary 1. *(Probabilistic Bound) Let $\hat{h}$ be the biased estimator of the diagonal of the Hessian of $L(\theta)$ as per Theorem 3. Then, let $\delta \in [0,1]$ and let m be a fixed number of iterations, such that $\hat{h}_m$ is the estimator of the diagonal of the Hessian after m iterations. Then, the following holds with probability $1-\delta$:*

$$\left\|\hat{h}_m - \operatorname{diag}\left(\nabla_\theta^2 L(\theta)\right)\right\|_2 \leq \sqrt{\frac{1}{m\delta}\mathbb{E}\left[\|\epsilon\|_2^2\right]},$$

where $\epsilon = \mathbf{z} \odot \nabla_\theta^2 L(\theta)\,\mathbf{z} - \operatorname{diag}\left(\nabla_\theta^2 L(\theta)\right) + Bias(\mathbf{z})$.

Proof. The proof is given in Appendix B.3. $\square$

We see then that choosing the variance of $\mathbf{z}$ different from 1 can have a positive effect on the convergence of the Hutchinson estimator. Probabilistic guarantees are also given in Corollary 1, which ensures that the error of the estimator is bounded. Unfortunately, the optimal choice of $\mathbb{V}[z_i]$ as per Theorem 3 requires the knowledge of the *exact* Hessian matrix, which we do not have access to. However, we can still use the theorem to guide the choice of $\mathbb{V}[z_i]$, and we propose to use a scaled Rademacher distribution with values -0.9 and 0.9 as a heuristic choice.[10] We empirically verified that the convergence of the Hutchinson estimator is improved by using the biased estimator with the scaled Rademacher distribution, and follows the theoretical bounds given in Corollary 1 (Appendix C).

4 Sparse Stochastic Mirror Descent for DL-SCA

The previous section explains how to effectively estimate the Hessian matrix in the context of deep learning, and hence how to improve the optimization process regardless of the DL-SCA context. In this new section, we propose to leverage a prior knowledge singular to DL-SCA. Both those aspects are combined in the Scoop algorithm presented in Section 5.

We propose to use the Stochastic Mirror Descent (SMD) algorithm, which is a generalization of the stochastic gradient descent that allows for a more flexible optimization process. We first formalize a property of DL-SCA that can be exploited by SMD, and then introduce the SMD algorithm.

4.1 Sparsity of the DL-SCA Formulation

Real traces of a device contain few samples related to the secret. This means that the ideal model $\mathbf{F}^\star$ should be a combination of a few features. This is a commonly made assumption [MDP19b]. Hence, we assume that the weights of $\mathbf{F}^\star$'s first layer should be sparse, which has been observed by Zaid *et al.* [ZBHV19]. Considering the high dimensionality of the input, we assume most of the weights of $\mathbf{F}^\star$ are used for the first layer, hence targeting weights sparsity in the entire model should be reasonable choice.

To put a specific constraint on the weights such as sparsity, deep learning is often trained by solving the following optimization problem:

$$\arg\min_\theta L(\theta) + \lambda R(\theta) \tag{4.1.1}$$

[10]Naturally, if we could perform an infinite number of iterations, choosing a variance of 1 for z would be ideal, since Theorem 3 dictates that $\lim_{m\to\infty} \mathbb{V}[z]^\star = 1$.

where R is a *regularization* term. λ is a hyperparameter that controls the trade-off between the loss and the regularization term. This formulation is known as *explicit* regularization. However, choosing a specific optimization algorithm can converge to the same solution without the need of a regularization term. This is called *implicit* regularization.

4.2 Stochastic Mirror Descent

The Stochastic Gradient descent operates in an Euclidean space. While not trivial by looking at Equation 2.4.2, it turns out that this update equation is the solution of the following optimization problem:

$$\theta^{(t+1)} = \arg\min_{\theta} \eta_t \theta^T \nabla_\theta L(\theta) + \frac{1}{2} \left\| \theta - \theta^{(t)} \right\|_2^2 ,$$

where $\left\| \theta - \theta^{(t)} \right\|_2^2$ is the Euclidean distance. If one wants the optimization algorithm to operate in a different space, we can modify the update rule replacing the Euclidean distance by a *Bregman* divergence denoted D_ψ, such that the update equation becomes:

$$\begin{cases} \theta^{(t+1)} = \arg\min_{\theta} \eta_t \theta^T \nabla_\theta L(\theta) + D_\psi\left(\theta, \theta^{(t)}\right) \\ D_\psi\left(\theta, \theta^{(t)}\right) = \psi(\theta) - \psi\left(\theta^{(t)}\right) - \left\langle \nabla_\theta \psi\left(\theta^{(t)}\right), \theta - \theta^{(t)} \right\rangle \end{cases} . \tag{4.2.1}$$

This formulation is called Stochastic Mirror Descent (SMD) and is, in fact, a generalized SGD. ψ is called the potential function. The Bregman divergence shares many properties with the Euclidean distance, such as non-negativity, convexity, *etc.* The choice of ψ determines the space in which the algorithm operates in. Interestingly, it has been shown that the Mirror Descent algorithm is an implicit ψ-regularizer [SGAA23] and this property has been extended to the Stochastic Mirror Descent algorithm for some families of models [ALH22].

To induce sparsity in the weights of the model, it is often considered doing ℓ_1-regularization [Goo16]. A problem however arises when choosing ψ to be the ℓ_1 norm. To understand let us look at the solution of Equation 4.2.1 in Equation 4.2.2.

$$\nabla\psi\left(\theta^{(t+1)}\right) = \nabla\psi\left(\theta^{(t)}\right) - \eta_t \nabla L\left(\theta^{(t)}\right) \tag{4.2.2}$$

We see that the iterative process of SMD requires the gradient of ψ. However, the ℓ_1 norm is not differentiable at 0. Azizan *et al.* suggest using the $\ell_{1+\epsilon}$ norm as an alternative with $\epsilon < 1$ [ALH22].

Hence SMD with a potential function ψ that is the $\ell_{1+\epsilon}$ norm is a good candidate to induce sparsity in the weights of the model.

5 Scoop: An Optimizer against Higher-Order Masking

In this section, we introduce a second-order optimization algorithm called Scoop: SeCond-Order precOnditioned sParse stochastic mirror descent. In other-words, Scoop is a Stochastic Mirror Descent algorithm, where the potential function is the $\ell_{1+\epsilon}$ norm, and where the descent is preconditioned by the inverse of an estimation of the diagonal Hessian matrix. The algorithm is presented in Algorithm 1.

In Algorithm 1, β_1 and β_2 are the momenta hyperparameters, η_t is the learning rate, and ϵ is the parameter of the $\ell_{1+\epsilon}$ norm (set to 0.1 by default [ALH22]). The function $c(\cdot)$ is a clipping function that ensures the stability of the update rule [LLH+23], as described in Section 3.3.1.

Algorithm 1 Scoop: SeCond-Order precOnditioned sParse stochastic mirror descent

```
1: for $t = 1, 2, \dots, T$ do
2:     Compute mini-batch loss $L(\theta_t)$
3:     Sample $v$ from a scaled Rademacher distribution
4:     $\tilde{H}_t \leftarrow v \odot \nabla (\langle \nabla L(\theta_t), v \rangle)$    ▷ Hutchinson estimator
5:     $h_{t+1} \leftarrow \beta_2 h_t + (1 - \beta_2) \tilde{H}_t$    ▷ EMA[11]of the Hessian
6:     $g_{t+1} \leftarrow \beta_1 g_t + (1 - \beta_1) \nabla L(\theta_t)$    ▷ EMA of the gradient
7:     Step $\leftarrow (1 + \epsilon) |\theta_t|^\epsilon \operatorname{sign}(\theta_t) - \eta_t c \left( h_{t+1}^{-1} g_{t+1} \right)$    ▷ Step in the dual space
8:     $\theta_{t+1} \leftarrow |\text{Step}/(1 + \epsilon)|^{1/\epsilon} \operatorname{sign}(\text{Step})$    ▷ Projection back onto the primal space
9: end for
```

While more expensive than Adam, Scoop is still computationally efficient as the computation of $\tilde{H}_t$ is done efficiently in $\mathcal{O}(n)$ where n is the number of parameters.

The source code and guidelines of Scoop are available at https://github.com/nathan-rousselot/scoop.

6 Experimental Results

In this section, we evaluate the performance of DL-SCA using Scoop as an optimizer, and show that the combination of the Hessian preconditioning with SMD brings an edge to DL-SCA profiling. We first present the results on a simulated scenario where we study high-order masking schemes. We then evaluate Scoop on the ASCADv1 dataset, which is protected by a first-order boolean masking scheme. We compare Scoop to Adam, which is the most commonly used optimizer in the side-channel community. We also compare Scoop to the state-of-the-art attacks on ASCADv1. Finally, we evaluate Scoop on the ASCADv2 dataset, which is protected by an affine masking scheme and loop shuffling. All models are trained using a single Nvidia RTX 4500 Ada Generation with 24GB RAM.

6.1 Attack Against High-Order Simulated Masking Schemes

To properly assess the optimizer's efficiency, we first consider noise-free exhaustive datasets, *i.e.,* datasets where all the possible combinations of values are present without temporal noise. In other words, we reduce the DL-SCA problem to a simple optimization problem. We consider a boolean masking scheme at order n for an 8-bit secret. The leakage model is the Hamming weight, and we target the output of the SBox of an AES with identity labelling.

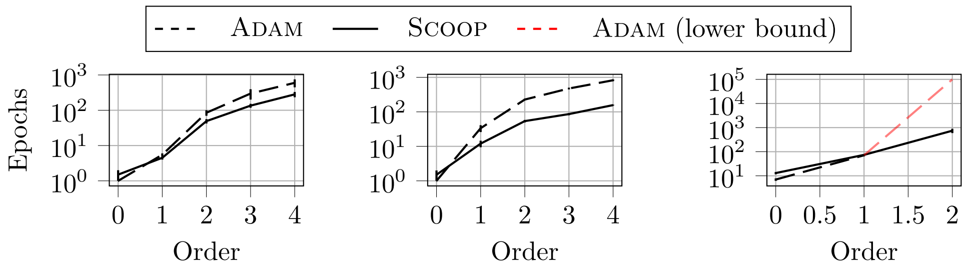
We target those simulated traces with three different architectures: MLP, CNN-VGG [ZBHV19] and Transformer [HCM24]. The formulation of this problem follows the exact same procedure given in Section 5.2.1 of [MCLS23]. We train each model on the dataset for up to 10^5 epochs with Adam and Scoop. Then, we measure the plateau length as the number of epochs required to reach a validation loss of $\mathbb{H}[s] - \epsilon$, where $\epsilon = 0.05$ in our setup. All experiments are averaged over 100 runs. Due to the limited GPU memory of our hardware setup, we can not run the Transformer model on the exhaustive dataset above order 2.

Figure 6.1.1 compares the plateau length of the training of multiple models (MLP, CNN and Transformer) when trained with Adam and Scoop. We see that Scoop effectively reduces the plateau length as soon as there is a masking countermeasure enabled, which is consistent with our theoretical findings. Table 1 shows the average plateau reduction using Scoop compared to Adam. We can see that Scoop is particularly efficient on high-order masking schemes as the plateau length is reduced by a factor up to 5 for the

[11]Exponential Moving Average.

Table 1: Average plateau reduction (in percentage) using Scoop compared to Adam on an exhaustive simulated dataset with MLP and CNN.

Model	**n=0**	**n=1**	**n=2**	**n=3**	**n=4**
MLP	−50%	18.18%	42.15%	54.57%	52.89%
CNN	−50%	64.22%	76.27%	81.87%	80.97%



(a) MLP (b) CNN (c) Transformer

Figure 6.1.1: Comparison of the plateau length with Adam (dotted lines) and Scoop (full lines) on an exhaustive simulated dataset for three architectures. For the transformer (c), Adam was unsuccessful in retrieving the secret within the 10^5 allocated epochs at order 2, hence we plot a lower bound instead.

CNN architecture at order 3 and higher. It seems that the plateau reduction gets more important as the order of the masking increases, and is also sensitive to the architecture of the model. Meanwhile, we have observed a time increase of approximately 5% only using Scoop in comparison to Adam. Furthermore, for the Transformer model, Adam is unable to train a successful model within the 10^5 allocated epochs at order 2, while Scoop was able to do it in less than 10^3 epochs (Figure 6.1.1c). This confirms the importance of the optimizer in the profiling step.

Additionally, we compare the weights' distributions of the MLP model trained with Adam and Scoop against 4th order masking scheme. The results are shown in Figure 6.1.2. We can see that the weights' distribution is more concentrated around zero when using Scoop, which is consistent with Section 4.2 since Scoop incorporates $\ell_{1+\epsilon}$-SMD.

6.2 Attacking First-Order Boolean Masking Scheme: ASCADv1 Dataset

In the side-channel community, there exist different public datasets that serve as benchmarks. The most popular one is the ASCADv1 dataset [BPS+20] which is a first-order

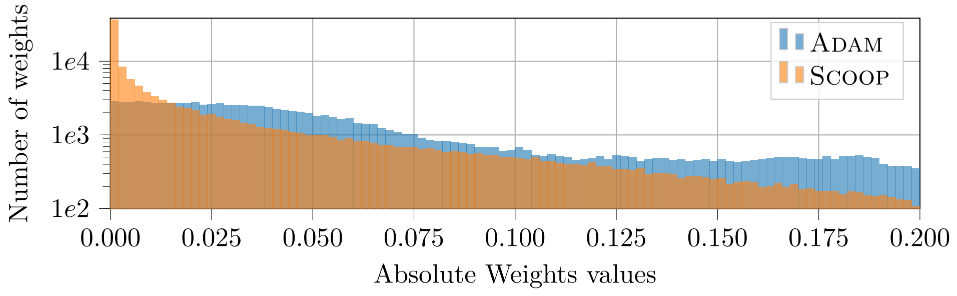


Figure 6.1.2: Comparison of weights' distributions in training an MLP against 4th order masking scheme when using Scoop and Adam.

Table 2: Comparison of the state of the art on ASCADv1.

	Na	Plateau Size	Val Loss	GPU Characteristics	GPU Time (h)[12]
CNN-VGG [ZBHV19]	191	40	7.78	8*V100 (256GB)	10,000
MLP [Wei20]	$\approx$ 250	-	-	-	-
Transnet [HSAM22]	$\approx$ 230	-	-	Google TPU	-
AutoSCA CNN [WPP22]	158	-	-	1080Ti (11GB)	10
AutoSCA MLP [WPP22]	129	-	-	1080Ti (11GB)	10
Ensemble [LCE+23]	203	-	-	-	-
Multi-Task [MO24]	-	25/50	-	-	-
MLP-Scoop (this work)	110	3	7.69	RTX 4500 Ada (24GB)	0.5
CNN-Scoop (this work)	73	11	7.65	RTX 4500 Ada (24GB)	3

boolean masked AES software implementation. Within this dataset, one can use raw traces, which are longer but easy to break since many leakage points are present, or the most common *extracted* dataset which focuses on the third sub-byte calculation, which has no first-order leakages. We focus on the later one, which is more challenging.

6.2.1 State of the Art of DL-SCA against ASCADv1

ASCADv1 has been extensively explored using both classical approaches and deep learning. Gaussian modeling with PCA (Template Attack) can outperform DL-based methods in certain settings [BPS+20]. Among neural architectures, MLPs have been thoroughly studied [Wei20], with Wu *et al.* introducing a Bayesian hyperparameter optimization that currently achieves the SOTA key-recovery results [WPP22]. VGG-like CNNs [SZ15] remain popular for their desynchronization robustness, originally proposed by Zaid *et al.* [ZBHV19] and further analyzed in [BS23]. Ensemble learning has also shown promise on ASCADv1 [LCE+23], while transformer-based models leverage shift-invariance [HSAM22, HCM24] to target ASCADv1 with additional desynchronization. Finally, multitask approaches have been studied for full key recovery and to reduce the plateau length [MO24]. In this paper, we compare ourselves with those works (Table 2). However, some of those do not aim at producing the most efficient attack, hence many of them did not share all the data of interest.

6.2.2 Results

Best Model Evaluation. We train and fine-tune a VGG-like CNN inspired by Zaïd *et al.* [ZBHV19], using Scoop as the optimizer instead of Adam. The Hessian is estimated using a low-variance biased Hutchinson estimator. Hyperparameter tuning is performed over three hours using a tree-structured Parzen estimator [ASY+19]. The hyperparameters search grid is similar to Wu *et al.* [WPP22], besides Scoop specific hyperparameters and a finer grid for most of the hyperparameters. The exact hyperpameters grid is given in Appendix D. We use 50,000 traces for the profiling dataset, and 10,000 traces for the validation dataset.

The results of the attack are shown in Table 2, where the column labeled *Na* contains the number of traces required to reduce the guessing entropy to zero. The results show that the trained model outperforms the state of the art by reaching a successful attack in 73 traces, a plateau size of 11 epochs and a validation loss of 7.65 which are all beyond the previous results we are aware of.

Comparison Between the Different Optimization Methods. We compare here the difference in performance between Adam (SGD), Sophia [LLH+23] (second-order SGD with GNB Hessian estimator), SMD and Scoop with the different Hessian methods discussed in Sections 3 and 4. The results are shown in Table 3. We see that both second order SGD and SMD alone outperform Adam (SGD) in all categories (besides training

[12]The GPU time includes the hyperparameter optimization time.

Table 3: Comparison of the different methods on ASCADv1, with a CNN.

	Na	**Plateau Size**	**Val Loss**	**Train Loss**
Adam (SGD)	180	23	7.89	7.63
Sophia (2nd Order SGD) [LLH+23]	123	11	7.74	7.65
SMD	105	11	7.68	**7.11**
Scoop (GNB+SMD)	130	12	7.69	7.13
Scoop (biased Hutch. + SMD)	**73**	**11**	**7.65**	7.12

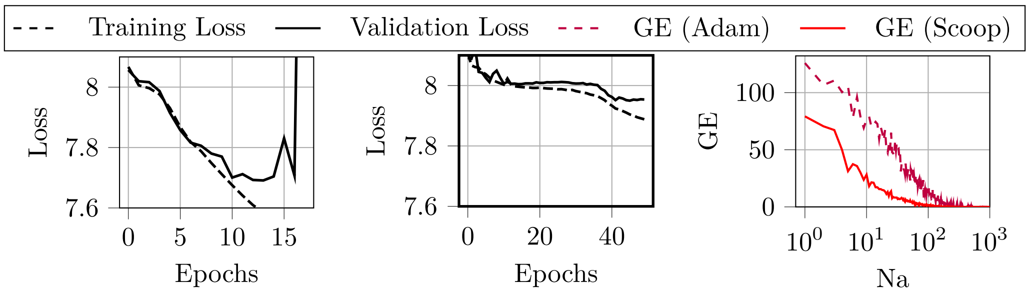


Figure 6.2.1: Profiling loss curves of a finetuned MLP trained with Scoop against ASCADv1 (left). The same model trained with Adam is shown in (center). Both models Guessing Entropies are compared in (right).

loss for second-order SGD). Their combination, Scoop, is also a step forward. It seems that using a biased Hutchinson estimator ends in a stronger attack *Na*-wise, yet stronger evidences are needed to confirm if using a biased estimator gives a significant edge over an unbiased one.

Multi-Layer Perceptrons. We use the same fine-tuning approach as for the CNN, and obtained an MLP model, reported in Table 2, which is also above current state of the art in terms of *Na* (110 traces), plateau length (3 epochs) and validation loss (7.69). The exact hyperpameters grid is given in Appendix E.2.

Interestingly, we can see on its training curves (Figure 6.2.1 (left)) that the plateau is almost non-existent for this MLP trained with Scoop. Yet, training the exact same model using the same seed with Adam (Figure 6.2.1 (center)) shows a much more prominent plateau of 38 epochs.

Weights' Distributions. Similarly to Section 6.1, we compare the weights' distributions of the MLP model trained with Adam and Scoop against ASCADv1. The results are shown in Figure 6.2.2. There again, Scoop's resulting weights are more concentrated around zero, which is consistent with Section 4.2 since Scoop incorporates $\ell_{1+\epsilon}$-SMD.

6.2.3 Hyperparameter Optimization Performance

We previously discussed about hyperparameters optimization without going into details. And for a fact, it is well studied in the literature, and available tools allow for effortless models finetuning [ASY+19]. Furthermore, Wu *et al.* deeply investigated it in the context of DL-SCA [WPP22]. The hyperparameters optimization step is often constrained by a *time budget* as this phase is the most expensive step in a deep learning training pipeline. A solution to reduce the time spent on this step is to use coarser and narrower search grids. In addition to requiring prior knowledge on the optimal architecture, this can lead to suboptimal models or induce a bias in the results. In this section, we evaluate the impact of using Scoop during the hyperparameters optimization step.

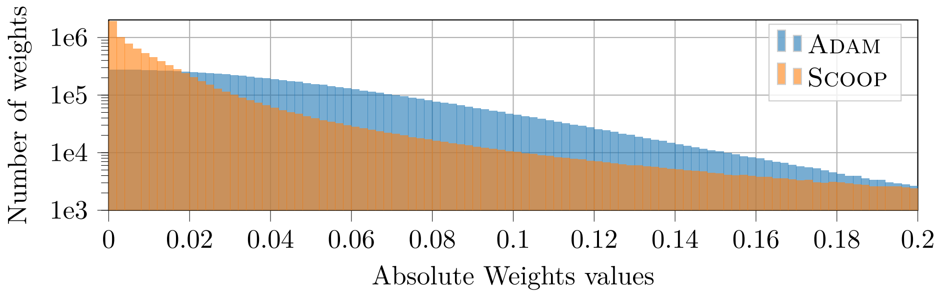


Figure 6.2.2: Comparison of weights' distributions in training an MLP against ASCADv1. (red): Adam, (blue): Scoop.

Table 4: Comparison of the hyperparameters optimization performance using Scoop and Adam on ASCADv1.

	MLP		CNN	
	Adam	Scoop	Adam	Scoop
Rate of Successful Models[13]	2.1%	**4.1%**	7.7%	**15.4%**
Avg Val Loss	7.93	**7.83**	7.87	**7.86**
Best Val Loss	7.92	**7.69**	7.81	**7.76**
Avg Cost Per Epoch (s)	**0.31**	0.91	**3.17**	3.74
Total Duration of finetuning (h)	**0.51**	0.82	4.18	**2.99**
Relative Finetuning Cost	1	**0.82**	1	**0.358**
Relative Attack Performance	1	**3.87**	1	**1.26**

To evaluate this, we finetune a MLP and a CNN against ASCADv1. We use the same hyperparameters grid as before, which is wide and fine. We sample the grids randomly, sampling more than 150 MLP architectures, and 100 CNN architectures. Using deterministic options of PyTorch [PGM+19], we trained each model using Scoop and Adam.

Table 4 displays the results of this study. It contains the rate of successful models (RoS), the average validation loss, the best validation loss, the average cost per epoch and the total duration of the finetuning (TD). The relative finetuning cost is computed as follows:

$$\text{Relative Finetuning Cost} = \frac{\text{RoS}_{\text{Adam}}}{\text{RoS}_{\text{Scoop}}} \times \frac{\text{TD}_{\text{Scoop}}}{\text{TD}_{\text{Adam}}}.$$

The relative attack performance is computed as $\text{PI}_{\text{Scoop}}/\text{PI}_{\text{Adam}}$. Looking at MLPs, we see that using Adam, 2.1% of the trained models turn successful, and reach an average validation loss of 7.93. On the other hand, 4.1% of models trained with Scoop turn successful eventually, and reach a much better validation loss of 7.83 on average. This means that, on average, Scoop manages to bring random architectures to working models twice as much as Adam, and they are stronger. For the CNNs, the quality of the models is similar between the two optimizers (with a slight edge for Scoop) but there again, Scoop manages to bring random architectures to working models twice as much as Adam.

Interestingly, the training time cost remains similar between the two optimizers. We estimate the total reduction of finetuning cost to be 18% for MLPs and 64% for CNNs using Scoop, while gaining a 287% and 26% improvement in attack performance, respectively. This allows, for example, for a faster security assessment of a new dataset, or could allow for more explorative works on hyperparameters.

[13]Reached when PI ≥ 0.05 bits.

6.3 Attacking Against Second-Order Affine Masking Scheme and Loop Shuffling: ASCADv2 Dataset

Another well-known dataset is ASCADv2 [MS23], an affine masked software AES implementation (pseudo-second order [FMPR11]). It incorporates a loop shuffling mechanism that consists in permuting the order in which bytes are processed in the subbytes loop. To our knowledge, no non-worst-case attack have been successful on ASCADv2. Only one attack has been successful so far [WPP23], the authors exploited a weakness emerging from the affine masking scheme [FMPR11], namely that the multiplicative mask is shared among all the elements in the AES state. Moreover, Wu *et al.* deactivated the loop shuffling mechanism to demonstrate the effectiveness of their attack. Hereafter, we do not rely on any of these assumptions.

6.3.1 Approach and Architecture

ASCADv2 is a challenging dataset not only by its countermeasures, but also by its dimension. Each trace is 15,000 samples long, and models trained to fit such high dimensional data require a lot of computational power and memory capacity. Considering our hardware setup, we discard costly architectures such as CNNs and Transformers. Instead, we decide to focus on MLPs. Considering the challenging nature of the dataset, we decide to split the dataset in three parts: a profiling dataset of 200,000 traces, a validation dataset of 10,000 traces, and an attack dataset of 300,000 traces. The profiling and attack datasets are both from the "profiling traces" and are respectively the 200,000 first and 300,000 last traces of the set of traces. The validation dataset is the same as the one provided by the ASCADv2 authors. This allows to mount an attack with a large number of traces, which is necessary to break the dataset. Having separate validation and attack datasets is key for complex problems to ensure the model is not overfitting the validation dataset during the fine-tuning process. We employ two different approaches to define the architecture.

First Approach: Fine-tuning a deep MLP. For the first approach, similarly to ASCADv1, we rely on the work of [Wei20], which is a hyperparameters' optimization approach using a tree-structured Parzen estimator [ASY+19]. We choose to fine-tune a narrow MLP with n hidden layers of size 2^8. We fine-tune the model for 3 hours, and the best model is selected based on the validation loss. The model takes about 10 minutes to train on our hardware.

Second Approach: Single hidden layer MLP. Despite having a deep MLP might seem as a natural choice, empirical studies have shown that it is hard to justify the need for more than one or two hidden layers in a multi-layer perceptrons [BL07]. That is why we propose a second, finetuning-free, approach. We train a single-hidden-layer MLP of size $n_{\text{hidden}} = 2/3 \times (\text{input size} + \text{output size})$. This model has a total of approximately 231 millions parameters.

6.3.2 Results

Fine-tuned MLP. The fine-tuned MLP results in a successful attack in 3 hours of fine-tuning and 10 minutes of training. The guessing entropy is reduced to 0 after the complete 300k attack traces in the dataset, with a plateau size of 25 epochs. Full results are shown in Table 5.

Single Hidden Layer MLP. The single hidden layer MLP results in a successful attack in 10 hours of training. The guessing entropy is reduced to 0 after only 150k attack traces in the dataset, at probability 1. Sometimes, less than 50k attack traces are enough to retrieve the key. The training has a plateau size of 12 epochs. Full results are shown in Table 5, and the training example is shown in Figure 6.3.1.

Table 5: Results of the two approaches.

	Na	Plateau Size	Val Loss	GPU Time (h)
MLP (fine-tuned)	300k	25	7.9998	3
MLP (single hidden layer)	150k	12	7.9996	10

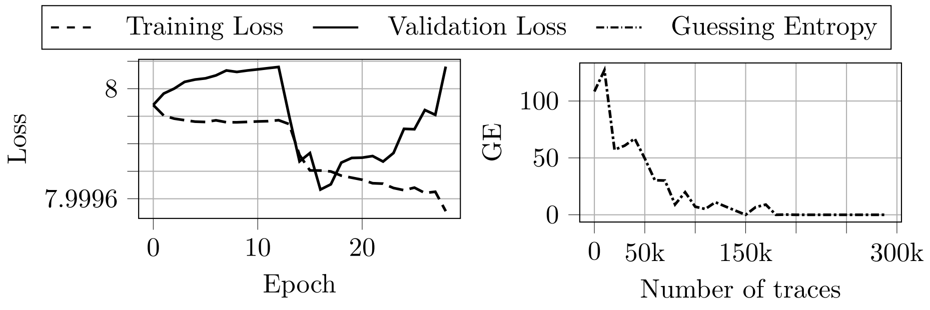


Figure 6.3.1: Training example of a successful model on ASCADv2 dataset using Scoop with a biased Hutchinson estimator (left) and the corresponding guessing entropy (right).

What about Adam? Sophia? SMD? We try the same approaches (same architectures, hyperparameters, allocated time and hardware) with Adam, Sophia and SMD, but none of them ends in a successful attack, which is consistent with the current literature.

7 Conclusion

In this paper, we provided both theoretical and empirical evidences that deep learning-based profiling attacks are significantly hindered by masking schemes, especially as the masking order increases. Through a visualization of the loss landscape in this context, we showed that higher-order masking leads to less intense gradients, and apparition of interfering saddle points. Those two phenomena are responsible for the plateau effect observed in the training of deep learning models for side-channel analysis.

To address these challenges, we introduced Scoop, a novel optimization algorithm combining second-order optimization techniques with sparse stochastic mirror descent. Despite the plateau effect still affecting Scoop, our approach effectively reduces the plateau length by a factor up to 5 on exhaustive simulated traces, while incurring minimal additional computational cost. Experimental results on the ASCADv1 and ASCADv2 datasets confirm that models trained with Scoop not only outperform the current state of the art in amount of traces required for a successful key recovery and plateau length, but also achieves the first successful non-worst-case attack on ASCADv2. Additionally, we showed that by using Scoop, we can significantly improve the hyperparameters optimization process, leading to better models in a shorter amount of time.

Interestingly, to achieve this attack, Scoop hyperparameters optimization converged towards a model with a strong ℓ_2 regularization, which, to the best of our knowledge, remains at small values in the current DL-SCA literature [RB24].

These findings show the importance of considering novel optimization strategies for DL-SCA, and not only the architecture. A potential future research direction would be to investigate the role of ϵ which is the SMD parameter of Scoop. We believe that putting ϵ as a hyperparameter, and maybe in a schedule, could lead to even better results.

A Proof that $\mathbf{F}$ has a unique optimum for a two-share secret

Proof. We want $\forall(s_0, s_1) \in \{0,1\}$ *s.t.* $s_0 \oplus s_1 = 0 (i.e. s_0 = s_1) \Rightarrow \{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0 \approx 1$. We note $\underline{\theta} = (\theta_0, \theta_1)$.

We recall the expression of the model:

$$\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0 = \frac{e^{2\alpha_0 s_0 \theta_0 + 2\alpha_1 s_1 \theta_1} + e^2}{\left(e^{2\alpha_0 s_0 \theta_0} + e\right)\left(e^{2\alpha_1 s_1 \theta_1} + e\right)} \tag{A.0.1}$$

Let us set $\theta_0 = \alpha_0 \lambda$ and $\theta_1 = \alpha_1 \lambda$ where λ is a constant. Then, we have:

$$\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0 = \frac{e^{4\lambda s_0} + e^2}{\left(e^{2\lambda s_0} + e\right)^2} \tag{A.0.2}$$

$$\mathbb{E}\left[\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0\right] = \frac{1}{2} + \frac{e^{4\lambda + e^2}}{2\left(e^{2\lambda} + e\right)^2} \underset{\lambda \to +\infty}{\longrightarrow} 1 \tag{A.0.3}$$

Now, let us set $\theta_0 = -\alpha_0 \lambda$ and $\theta_1 = -\alpha_1 \lambda$, then we have:

$$\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0 = \frac{e^{-4\lambda s_0} + e^2}{\left(e^{-2\lambda s_0} + e\right)^2} \tag{A.0.4}$$

and

$$\mathbb{E}\left[\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0\right] = \frac{1}{2} + \frac{e^2 + e^{-4\lambda}}{2\left(e^{-2\lambda} + e\right)^2} \underset{\lambda \to +\infty}{\longrightarrow} c < 1 \tag{A.0.5}$$

Now let us set $\theta_0 = \alpha_0 \lambda$ and $\theta_1 = -\alpha_1 \lambda$, then we have:

$$\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0 = \frac{e^{2\lambda s_0} + e^2}{\left(e^{2\lambda s_0} + e\right)\left(e^{-2\lambda s_1} + e\right)} \tag{A.0.6}$$

$$\mathbb{E}\left[\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0\right] = \frac{1}{2} + \frac{e^{2\lambda} + e^2}{2\left(e^{2\lambda} + e\right)\left(e^{-2\lambda} + e\right)} \underset{\lambda \to +\infty}{\longrightarrow} c < 1 \tag{A.0.7}$$

And now

$$\{\mathbf{F}(\mathbf{x}| - \underline{\theta})\}_0 = \frac{e^{-2\lambda s_0} + e^2}{\left(e^{-2\lambda s_0} + e\right)\left(e^{2\lambda s_1} + e\right)} \tag{A.0.8}$$

$$\mathbb{E}\left[\{\mathbf{F}(\mathbf{x}| - \underline{\theta})\}_0\right] = \frac{1}{2} + \frac{e^{-2\lambda} + e^2}{2\left(e^{-2\lambda} + e\right)\left(e^{2\lambda} + e\right)} \underset{\lambda \to +\infty}{\longrightarrow} \frac{1}{2} \tag{A.0.9}$$

Thus, $\{\mathbf{F}(\mathbf{x}|\underline{\theta})\}_0 - \{\mathbf{F}(\mathbf{x}| - \underline{\theta})\}_0 \neq 0$ and the loss landscape is not symmetric.

Hence $\mathbf{F}$ has a unique optimum. $\square$

B Proof of Theorems

B.1 Proof that Hutchinson Diagonal Estimator is unbiased

Before proving the theorems, we recall the proof of the Hutchinson diagonal estimator unbiasedness.

Let us first introduce Lemma 1.

Lemma 1. *(Hessian Vector Product Identity) The Hessian vector product can be computed as:*

$$\nabla_\theta^2 L(\theta)\,\mathbf{z} = \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right] \tag{B.1.1}$$

Proof. By the definition of the derivative:

$$\nabla_\theta^2 L(\theta)\,\mathbf{z} = \lim_{\epsilon\to 0} \frac{\nabla_\theta L(\cdot + \epsilon\mathbf{z}) - \nabla_\theta L(\theta)}{\epsilon} = \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right] \tag{B.1.2}$$

□

Theorem 4 (Hutchinson Diagonal Estimator is unbiased [Hut89])**.** *Let* $\mathbf{z}$ *be a i.i.d. random vector such that* $\mathbb{E}[z_i] = 0$ *and* $\mathbb{V}[z_i] = 1$*. Then the Hutchinson diagonal estimator is an unbiased estimator of the diagonal of the Hessian of* $L(\theta)$*:*

$$\mathbb{E}\left[\mathbf{z} \odot \nabla_\theta^2 L(\theta)\,\mathbf{z}\right] = \operatorname{diag}\left(\nabla_\theta^2 L(\theta)\right) \tag{B.1.3}$$

Proof. From Lemma 1, we have that:

$$\begin{aligned}
\mathbf{z} \odot \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right] &= \mathbf{z} \odot \nabla_\theta^2 L(\theta)\,\mathbf{z} && \text{(B.1.4)}\\
&= \mathbf{z} \odot \left\{ \sum_{j=1}^{n} \left\{\nabla_\theta^2 L(\theta)\right\}_{ij} z_j \right\}_{i=1}^{n} && \text{(B.1.5)}\\
&= \left\{ \sum_{j=1}^{n} \left\{\nabla_\theta^2 L(\theta)\right\}_{ij} \mathbb{E}[z_j z_i] \right\}_{i=1}^{n} && \text{(B.1.6)}\\
& && \text{(B.1.7)}
\end{aligned}$$

since z_i are sampled independently, $\mathbb{E}[z_j z_i] = \mathbb{E}[z_j]\,\mathbb{E}[z_i] = 0$ if $i \neq j$ and $\mathbb{E}[z_j z_i] = \mathbb{E}\left[z_j^2\right] = \mathbb{V}[z] + \mathbb{E}[z]^2 = 1$ if $i = j$. It follows that

$$\mathbb{E}\left[\mathbf{z} \odot \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right]\right] = \left\{\left\{\nabla_\theta^2 L(\theta)\right\}_{ii}\right\}_{i=1}^{n} = \operatorname{diag}\left(\nabla_\theta^2 L(\theta)\right) \tag{B.1.8}$$

□

B.2 Proof of Theorem 2

Proof. Let $\mathbf{z}$ be a random vector such that $\mathbb{E}[z_i] = 0$ and $\mathbb{V}[z_i] = 1$. And let $\hat{h}_t$ be the Hutchinson estimator of the diagonal of the Hessian of $L(\theta)$. Then, the variance of $\hat{h}_t$ is:

$$\Sigma_{ij} = \operatorname{Cov}\left(\left\{\mathbf{z} \odot \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right]\right\}_i, \left\{\mathbf{z} \odot \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right]\right\}_j\right) \tag{B.2.1}$$

For the sake of simplicity, let us denote $H = \mathbf{z} \odot \nabla_\theta\left[\langle \nabla L(\theta), \mathbf{z}\rangle\right]$. Then,

$$\begin{aligned}
\Sigma_{ij} &= \mathbb{E}\left[\{\mathbf{z} \odot H\mathbf{z}\}_i \{\mathbf{z} \odot H\mathbf{z}\}_j\right] - \mathbb{E}\left[\{\mathbf{z} \odot H\mathbf{z}\}_i\right]\mathbb{E}\left[\{\mathbf{z} \odot H\mathbf{z}\}_j\right]\\
&= \mathbb{E}\left[\sum_{k=1}^{n}\sum_{l=0}^{n} H_{ik}H_{jl} z_i z_k z_j z_l\right] - H_{ii}H_{jj}\\
&= \left(\sum_{k=1}^{n}\sum_{l=0}^{n} H_{ik}H_{jl}\mathbb{E}\left[z_i z_k z_j z_l\right]\right) - H_{ii}H_{jj}
\end{aligned}$$

What is important to note is that the diagonal of the covariance involve the fourth order moment of $\mathbf{z}$ (since $i = j = k = l$). Recalling that $\mathbb{E}\left[z_i^4\right]$ is the kurtosis of the distribution of $\mathbf{z}$. Hence, if $z_i \sim$ Rademacher, then $\operatorname{Tr}(\Sigma)$ is minimal. Non-diagonal elements do not depend on intrinsic properties of the distribution of $\mathbf{z}$ besides its expectation and variance which are fixed. □

B.3 Proof of Theorem 3 and Corollary 1

Proof of Theorem 3. This theorem is a generalization of Lemma 4.1 from [DM23].

Let $\mathbf{z}$ be a random vector with *i.i.d* entries, such that $\mathbb{E}[z_i] = 0$ and $\mathbb{V}[z_i]$ is set freely. Let $H \in \mathbb{R}^{n\times n}$ be the Hessian matrix of interest. Then, defining ϵ as the error of estimation of the diagonal of the Hessian, we have that:

$$\epsilon = \mathbf{z} \odot H\mathbf{z} - \operatorname{diag}(H) + \operatorname{Bias}(\mathbf{z}) \tag{B.3.1}$$

Where $\operatorname{Bias}(\mathbf{z})$ is the bias of the estimator. Let us note $h_i = H_{ii} - H_{ii}z_i^2$. Then assuming m iterations of the estimator, we have that:

$$\begin{aligned}
\mathbb{E}\left[\|\epsilon\|_2^2\right] &= \frac{1}{m}\mathbb{E}\left[\sum_{i=1}^n\left(h_i + \sum_{i\neq j} H_{ij}z_iz_j\right)^2\right] + \mathbb{E}[\operatorname{Bias}(\mathbf{z})] \\
&= \frac{1}{m}\mathbb{E}\left[\sum_{i=1}^n h_i^2 + 2\sum_{i=1}^n h_i \sum_{i\neq j} H_{ij}z_iz_j + \sum_{i=1}^n\left(\sum_{i\neq j} H_{ij}z_iz_j\right)^2\right] + \mathbb{E}[\operatorname{Bias}(\mathbf{z})] \\
&= \frac{1}{m}\mathbb{E}\left[\sum_{i=1}^n h_i^2 + 2\sum_{i=1}^n h_i \sum_{i\neq j} H_{ij}z_iz_j + \sum_{i=1}^n\sum_{i\neq j}\sum_{i\neq k} H_{ij}H_{ik}z_iz_iz_jz_k\right] + \mathbb{E}[\operatorname{Bias}(\mathbf{z})]
\end{aligned}$$

Let us break down each term separately.

$$\begin{aligned}
\mathbb{E}\left[h_i^2\right] = \mathbb{E}\left[\left(H_{ii} - H_{ii}z_i^2\right)^2\right] &= \mathbb{E}\left[H_{ii}^2 - 2H_{ii}^2z_i^2 + H_{ii}^2z_i^4\right] \\
&= H_{ii}^2 - 2H_{ii}^2\mathbb{E}\left[z_i^2\right] + H_{ii}^2\mathbb{E}\left[z_i^4\right] \\
&= H_{ii}^2\left(1 - 2\mathbb{V}[z_i] + \operatorname{Kurtosis}(z_i)\right)
\end{aligned}$$

Thus,

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^n h_i^2\right] &= \sum_{i=1}^n H_{ii}^2\left(1 - 2\mathbb{V}[z_i] + \operatorname{Kurtosis}(z_i)\right) \\
&= \left(1 - 2\mathbb{V}[z_i] + \operatorname{Kurtosis}(z)\right)\sum_{i=1}^n H_{ii}^2 \\
&= \left(1 - 2\mathbb{V}[z_i] + \operatorname{Kurtosis}(z)\right)\|\operatorname{diag}(H)\|_2^2
\end{aligned}$$

The second term is pretty straightforward, by the *i.i.d* hypothesis, we have that $\mathbb{E}[z_iz_j] = 0$ if $i \neq j$. Hence,

$$\begin{aligned}
\mathbb{E}\left[\sum_{i=1}^n H_{ii}(1 - z_i^2)\sum_{i\neq j} H_{ij}z_iz_j\right] &= \mathbb{E}\left[\sum_{i=1}^n\sum_{j\neq i} H_{ii}H_{ij}z_iz_j - \sum_{i=1}^n\sum_{j\neq i} H_{ii}H_{ij}z_i^3z_j\right] \\
&= \sum_{i=1}^n\sum_{j\neq i} H_{ii}H_{ij}\mathbb{E}[z_iz_j] - \sum_{i=1}^n\sum_{j\neq i} H_{ii}H_{ij}\mathbb{E}\left[z_i^3z_j\right] \quad = 0
\end{aligned}$$

And for the third term, if $j \neq k$, then $\mathbb{E}\left[z_i z_j z_i z_k\right] = 0$ which nullifies the term. If $j = k$, then $\mathbb{E}\left[z_i z_j z_i z_k\right] = \mathbb{E}\left[z_i^2 z_j^2\right] = \mathbb{E}\left[z_i^2\right]\mathbb{E}\left[z_j^2\right] = \mathbb{V}\left[z_i\right]^2$. Hence,

$$\begin{aligned}\mathbb{E}\left[\sum_{i=1}^{n}\sum_{i\neq j}\sum_{i\neq k} H_{ij}H_{ik}z_i z_i z_j z_k\right] &= \sum_{i=1}^{n}\sum_{i\neq j} H_{ij}H_{ij}\mathbb{V}\left[z_i\right]^2 \\ &= \mathbb{V}\left[z_i\right]^2 \sum_{i=1}^{n}\sum_{i\neq j} H_{ij}H_{ij} \\ &= \mathbb{V}\left[z_i\right]^2 \left\|\bar{H}\right\|_F^2\end{aligned}$$

Finally, the bias term is easily computed as:

$$\begin{aligned}\mathbb{E}\left[\operatorname{Bias}(\mathbf{z})\right] &= \mathbb{E}\left[\sum_{j=1}^{n}\left(\left(1-\mathbb{V}\left[z_j\right]\right)H_{jj}\right)^2\right] \\ &= \left(1-\mathbb{V}\left[z_j\right]\right)^2 \left\|\operatorname{diag}(H)\right\|_2^2\end{aligned}$$

Putting it all together, we have that:

$$\mathbb{E}\left[\|\epsilon\|_2^2\right] = \frac{1}{m}\left[\left(1-2\mathbb{V}\left[z_i\right]+\operatorname{Kurtosis}(z_i)\right)\left\|\operatorname{diag}(H)\right\|_2^2 + \mathbb{V}\left[z_i\right]^2\left\|\bar{H}\right\|_F^2\right] + \left(1-\mathbb{V}\left[z_i\right]\right)^2\left\|\operatorname{diag}(H)\right\|_2^2$$

Observing that $\nabla^2_{\mathbb{V}[z_i]}\mathbb{E}\left[\|\epsilon\|_2^2\right] = 2\left\|\bar{H}\right\|_F^2 + 2\left\|\operatorname{diag}(H)\right\|_2^2 > 0$, we know that the expected error is convex, and we can minimize it using the first order condition. We have that:

$$\mathbb{V}\left[z_i\right]^\star = \frac{\frac{m-1}{m}\left\|\operatorname{diag}(H)\right\|_2^2}{\frac{1}{m}\left\|\bar{H}\right\|_F^2 + \left\|\operatorname{diag}(H)\right\|_2^2}$$

□

We can also prove the probabilistic bound of Corollary 1, which is in fact derived from Markov's inequality.

Proof of Corollary 1. Recall that, from Theorem 3, the expected error of the estimator is given by:

$$\mathbb{E}\left[\|\epsilon\|_2^2\right] = \frac{1}{m}\left[\left(1-2\mathbb{V}\left[z_i\right]+\operatorname{Kurtosis}(z)\right)\left\|\operatorname{diag}(H)\right\|_2^2 + \mathbb{V}\left[z_i\right]^2\left\|\bar{H}\right\|_F^2\right] + \left(1-\mathbb{V}\left[z\right]\right)^2\left\|\operatorname{diag}(H)\right\|_2^2$$

If we are given m iterations. We can formulate the Markov inequality:

$$\mathbb{P}\left[\left\|\hat{h}_m - \operatorname{diag}(H)\right\|_2 \geq \delta\right] \leq \frac{1}{m\delta}\mathbb{E}\left[\|\epsilon\|_2^2\right]$$

Recalling that $\mathbb{E}\left[\|\epsilon\|_2^2\right] > 0$, we can rewrite the inequality as:

$$\mathbb{P}\left[\left\|\hat{h}_m - \operatorname{diag}(H)\right\|_2 \leq \sqrt{\frac{1}{m\delta}\mathbb{E}\left[\|\epsilon\|_2^2\right]}\right] \geq 1-\delta$$

□

C Empirical Convergence Analysis of the Biased Hutchinson Estimator

We empirically study the convergence of the different estimators. Namely, we compare three cases, two unbiased ones: $z \sim \mathcal{N}(0,1)$ and $z \sim$ Rademacher and the biased one z as per Theorem 3. We draw $H \in \mathbb{R}^{n \times n}$ at random K times and compute the empirical mean of the estimators. We then measure their relative errors. Figure C.0.1 shows the convergence of the different estimators. We see that the biased one is much more accurate in the first iterations, while the unbiased ones take much more time to converge. We see that the Rademacher estimator converges faster than the Gaussian one, which is in line with the theoretical results (the Kurtosis of the Rademacher distribution is lower than the one of the Gaussian distribution). While not included in the figure, we have also confirmed that when $z \sim \mathcal{L}(0,1)$, the estimator converges slower than any other one : it is the worst Kurtosis distribution. Recall we want a good estimator, but as inferring is epensive, we want to best possible estimator in few iterations. Actually, most of second-order optimization algorithms use just one iteration of the estimator [LLH+23]. The biased estimator is then the best choice, as it is the most accurate in the first iterations.

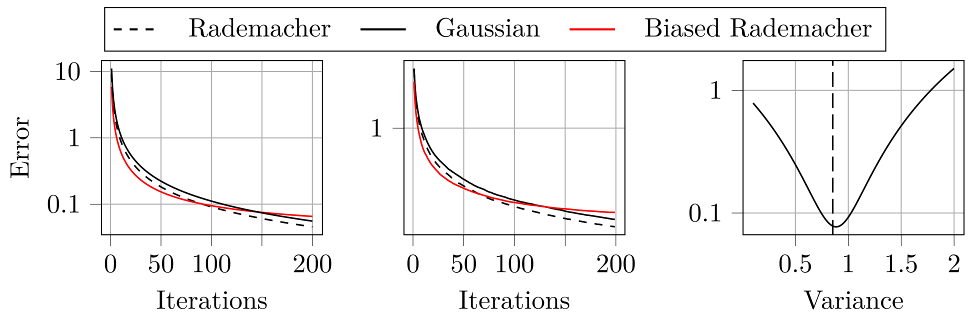


Figure C.0.1: Theoretical expected relative error (left) and empirical mean relative error (center) of the different estimators of the diagonal of the Hessian : Gaussian, Rademacher and biased Rademacher. For a fixed iteration $m = 50$, the impact of $\mathbb{V}[z]$ on the relative error is shown (right).

D ASCADv1 CNN Hyperparameters

The hyperparameters grid for the fine-tuning of the CNN on ASCADv1 is shown in Table 6.

Table 6: Hyperparameters grid for the fine-tuning of the CNN on ASCADv1

Hyperparameter	**Range**	**Step**	**Description**
η_0	[1e-4, 1e-2]	Continuous log	Initial learning rate
β_1, β_2	[0.9,0.99]	0.01	Gradient and Hessian momentum
$\phi(\cdot)$	[ReLU, SeLU]	-	Activation function
λ	[0, 0.3]	0.1	ℓ_2 regularization / Weight-Decay
n_linear	[2, 5]	1	Number of linear layers
n_conv	[1, 5]	1	Number of convolutional layers
global_pool	[True, False]	-	Global pooling
conv_filter_type	[“increasing”, “increasing_clipped”, “same”]	-	Convolutional filter type
kernel_size	[5, 17]	2	Kernel size
conv_filter_size	[10,20]	10	Convolutional filter size
pooling_size	[2, 3]	1	Pooling size
input_bn	[True, False]	-	Batch Normalization on input
conv_bn	[True, False]	-	Batch Normalization on convolutional layers
linear_bn	[True, False]	-	Batch Normalization on linear layers

The conv_filter_type parameter defines the behavior of the convolutional filters' sizes throughout the network. The “increasing” option doubles the filter size at each layer, while the “increasing_clipped” option also doubles the filter size at each layer, but clips it to 100. The “same” option keeps the same filter size at each layer.

E ASCADv2

E.1 Labels Construction on ASCADv2 through SNR study

ASCADv2 dataset does not explicitly contain non-worst-case labels. Hence, we need to generate them. As we target a classic AES SBox, we can generate the labels as follows:

$$y_i = \mathrm{SBox}[\mathrm{pt}_i \oplus \mathrm{key}_i] \tag{E.1.1}$$

where SBox is the SBox of the AES, pt_i the plaintext byte at position i and key_i the key byte at position i. To verify if our labels are correctly generated, we study the Signal-to-Noise Ratio (SNR) which, in a SCA paradigm, is actually a variance analysis (ANOVA) metric instead of a signal-to-noise ratio as per the classical signal processing definition. It is a metric that we can use to identify leakage points, given that we have access to the values manipulated during the execution. The SNR is defined as:

$$\mathrm{SNR} = \frac{\mathbb{V}\left[\mathbb{E}\left[\mathcal{L}\left(\mathbf{x}\right)|\mathbf{x}\right]\right]}{\mathbb{E}\left[\mathbb{V}\left[\mathcal{L}\left(\mathbf{x}\right)|\mathbf{x}\right]\right]}. \tag{E.1.2}$$

Where $\mathcal{L}\left(\mathbf{x}\right)$ is the leakage model of $\mathbf{x}$, being the targeted value. Since ASCADv2 is a masked implementation, the targeted value should never be leaking in the trace, and hence we expect the SNR not to emphasize any peak. Figure E.1.1 shows the SNR for different label constructions. In the first setting (Figure E.1.1 (left)) we deactivate both masking and loop shuffling, and we see that indeed the sensitive value leaks at a single point. In the second setting (Figure E.1.1 (center)), we activate the loop shuffling, and we see 16 leakage points, being the number of iterations of the subbyte loop. Finally, in the last setting (Figure E.1.1 (right)), we activate the masking and loop shuffling, and we can see that the SNR is just noise.

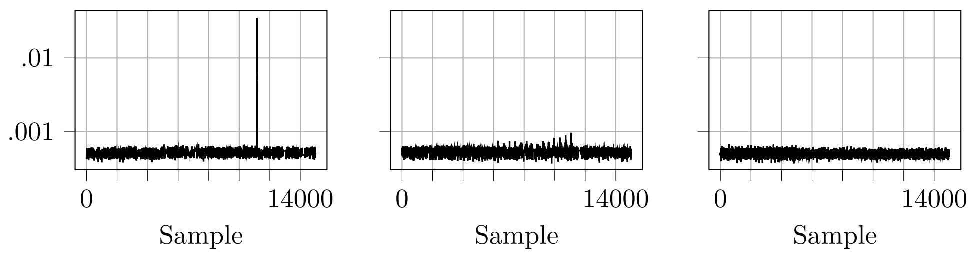


Figure E.1.1: Signal-to-noise ratio of the ASCADv2 dataset with different countermeasures configurations on byte 15. (left): No masking nor shuffling, (center): Shuffling, (right): Affine masking + shuffling.

E.2 ASCADv2 MLP Hyperparameters

The hyperparameters grid for the fine-tuning of the MLP on ASCADv2 is shown in Table 7.

Table 7: Hyperparameters grid for the fine-tuning of the MLP on ASCADv2.

Hyperparameter	**Range**	**Step**	**Description**
n	[3, 8]	1	Number of hidden layers
η_0	[1e-4, 1e-2]	Continuous log	Initial learning rate
β_1, β_2	[0.9,0.99]	0.01	Gradient and Hessian momentum
$\phi(\cdot)$	[ReLU, ELU, SeLU, Tanh]	-	Activation function
λ	[0, 0.3]	0.1	ℓ_2 regularization / Weight-Decay
InputBN	[True, False]	-	Batch Normalization on input
HiddenBN	[True, False]	-	Batch Normalization on hidden layers

The best found hyperparameters are: $n = 6$, $\eta_0 = 2.2e-4$, $\beta_1 = 0.97$, $\beta_2 = 0.92$, $\phi(\cdot) =$ ReLU, $\lambda = 0.2$, InputBN=False, HiddenBN=True. This model takes 3 hours to finetune and 10 minutes to fully train. To make the fine-tuning process more efficient, we employ a pruning strategy aimed at removing the worst performing models early in the process.

References

[ALH22] Navid Azizan, Sahin Lale, and Babak Hassibi. Stochastic mirror descent on overparameterized nonlinear models. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):7717–7727, 2022.

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[BBN19] Raghu Bollapragada, Richard H Byrd, and Jorge Nocedal. Exact and inexact subsampled newton methods for optimization. *IMA Journal of Numerical Analysis*, 39(2):545–578, 2019.

[BL07] Yoshua Bengio and Yann LeCun. Scaling learning algorithms toward ai. 2007.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, June 2020.

[BS23] Yohaï-Eliel Berreby and Laurent Sauvage. Investigating efficient deep learning architectures for side-channel attacks on aes. *arXiv preprint arXiv:2309.13170*, 2023.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 45–68. Springer, Cham, September 2017.

[CLM20] Valence Cristiani, Maxime Lecomte, and Philippe Maurine. Leakage assessment through neural estimation of the mutual information. In *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18*, pages 144–162. Springer, 2020.

[CLM23] Valence Cristiani, Maxime Lecomte, and Philippe Maurine. Revisiting mutual information analysis: Multidimensionality, neural estimation and optimality proofs. *Journal of Cryptology*, 36(4):38, October 2023.

[DES82] Ron S Dembo, Stanley C Eisenstat, and Trond Steihaug. Inexact newton methods. *SIAM Journal on Numerical analysis*, 19(2):400–408, 1982.

[DM23] Prathamesh Dharangutte and Christopher Musco. A tight analysis of hutchinson's diagonal estimator. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 353–364. SIAM, 2023.

[FMPR11] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 262–280. Springer, Berlin, Heidelberg, August 2011.

[Goo16] Ian Goodfellow. Deep learning, 2016.

[GSW23] Benjamin Grimmer, Kevin Shu, and Alex L Wang. Accelerated gradient descent via long steps. *arXiv preprint arXiv:2309.09961*, 2023.

[HCM24] Suvadeep Hajra, Siddhartha Chowdhury, and Debdeep Mukhopadhyay. EstraNet: An efficient shift-invariant transformer network for side-channel analysis. *IACR TCHES*, 2024(1):336–374, 2024.

[HSAM22] Suvadeep Hajra, Sayandeep Saha, Manaar Alam, and Debdeep Mukhopadhyay. TransNet: Shift invariant transformer network for side channel analysis. In Lejla Batina and Joan Daemen, editors, *AFRICACRYPT 22*, volume 2022 of *LNCS*, pages 371–396. Springer, Cham, July 2022.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[Hut89] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[KB15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

[Lan20] Guanghui Lan. *First-order and stochastic optimization methods for machine learning*, volume 1. Springer, 2020.

[LCE$^+$23] Dorian Llavata, Eleonora Cagli, Rémi Eyraud, Vincent Grosso, and Lilian Bossuet. Deep stacking ensemble learning applied to profiling side-channel attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 235–255. Springer, 2023.

[LLH$^+$23] Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. *arXiv preprint arXiv:2305.14342*, 2023.

[LN89] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[LZC$^+$21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR TCHES*, 2021(3):235–274, 2021.

[MCLS23] Loïc Masure, Valence Cristiani, Maxime Lecomte, and François-Xavier Standaert. Don't learn what you already know scheme-aware modeling for profiling side-channel analysis against masking. *IACR TCHES*, 2023(1):32–59, 2023.

[MDP19a] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR TCHES*, 2020(1):348–375, 2019.

[MDP19b] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *COSADE 2019*, volume 11421 of *LNCS*, pages 145–167. Springer, Cham, April 2019.

[MO24] Thomas Marquet and Elisabeth Oswald. Exploring multi-task learning in the context of masked AES implementations. In Romain Wacquez and Naofumi Homma, editors, *COSADE 2024*, volume 14595 of *LNCS*, pages 93–112. Springer, Cham, April 2024.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings 6*, pages 3–26. Springer, 2016.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against ANSSI's protected AES implementation on ARM: end-to-end attacks with multi-task learning. *Journal of Cryptographic Engineering*, 13(2):129–147, June 2023.

[MSP$^+$17] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[N$^+$18] Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[PP20] Guilherme Perin and Stjepan Picek. On the influence of optimizers in deep learning-based side-channel analysis. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 615–636. Springer, Cham, October 2020.

[RB24] Azade Rezaeezade and Lejla Batina. Regularizers to the rescue: fighting overfitting in deep learning-based side-channel analysis. *Journal of Cryptographic Engineering*, 14(4):609–629, 2024.

[SGAA23] Haoyuan Sun, Khashayar Gatmiry, Kwangjun Ahn, and Navid Azizan. A unified approach to controlling implicit regularization via mirror descent. *Journal of Machine Learning Research*, 24(393):1–58, 2023.

[SSSS17] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pages 3067–3075. PMLR, 2017.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR TCHES*, 2019(2):107–131, 2019.

[Wei20] Leo Weissbart. Performance analysis of multilayer perceptron in profiling side-channel analysis. In *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18*, pages 198–216. Springer, 2020.

[WPP22] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IEEE Transactions on Emerging Topics in Computing*, 2022.

[WPP23] Lichao Wu, Guilherme Perin, and Stjepan Picek. Not so difficult in the end: Breaking the lookup table-based affine masking scheme. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *SAC 2023*, volume 14201 of *LNCS*, pages 82–96. Springer, Cham, August 2023.

[YGS$^+$21] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. In *proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10665–10673, 2021.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR TCHES*, 2020(1):1–36, 2019.