Navigating the Deep: End-to-End Extraction on Deep Neural Networks

Haolin Liu, Adrien Siproudhis, Samuel Experton, Peter Lorenz, Christina Boura, Thomas Peyrin

Navigating the Deep: End-to-End Extraction on Deep Neural Networks

Haolin Liu[1,2], Adrien Siproudhis[2], Samuel Experton[3], Peter Lorenz[2], Christina Boura[3,4], and Thomas Peyrin[2]

Shanghai Jiao Tong University, China[1]
Nanyang Technological University, Singapore[2]
IRIF, Université Paris Cité, France[3]
Institut universitaire de France (IUF), Paris, France[4]

Abstract. Neural network model extraction has recently emerged as an important security concern, as adversaries attempt to recover a network's parameters via black-box queries. Carlini et al. proposed in CRYPTO'20 a model extraction approach inspired by differential cryptanalysis, consisting of two steps: signature extraction, which extracts the absolute values of network weights layer by layer, and sign extraction, which determines the signs of these signatures. However, in practice this signature-extraction method is limited to very shallow networks only, and the proposed sign-extraction method is exponential in time. Recently, Canales-Martínez et al. (Eurocrypt'24) proposed a polynomial-time sign-extraction method, but it assumes the corresponding signatures have already been successfully extracted and can fail on so-called low-confidence neurons.

In this work, we first revisit and refine the signature extraction process by systematically identifying and addressing for the first time critical limitations of Carlini et al.'s signature-extraction method. These limitations include rank deficiency and noise propagation from deeper layers. To overcome these challenges, we propose efficient algorithmic solutions for each of the identified issues, greatly improving the capabilities of signature extraction. Our approach permits the extraction of much deeper networks than previously possible.

In addition, we propose new methods to improve numerical precision in signature extraction, and enhance the sign extraction part by combining two polynomial methods to avoid exponential exhaustive search in the case of low-confidence neurons. This leads to the very first end-to-end model extraction method that runs in polynomial time.

We validate our attack through extensive experiments on ReLU-based neural networks, demonstrating significant improvements in extraction depth. For instance, our attack extracts consistently at least eight layers of neural networks trained on either the `MNIST` or `CIFAR-10` datasets, while previous works could barely extract the first three layers of networks of similar width. Our results represent a crucial step toward practical attacks on larger and more complex neural network architectures.

Keywords: ReLU-based neural networks · signature extraction · weight-recovery · sign extraction · end-to-end attack

1 Introduction

Neural Networks (NNs) are a class of machine learning models composed of layers of interconnected units (or neurons) that transform input data and learn to recognize patterns through a training process. Deep Neural Networks (DNNs) are a subset of NNs with multiple hidden layers, enabling them to model highly complex functions. DNNs have become indispensable in various applications, including computer vision (e.g., image recognition, and video analysis), natural language processing, automated medical diagnostics, and fraud detection, among others. However, training DNNs requires large datasets, significant computational resources, and carefully fine-tuned algorithms [5,26]. As a result, trained DNNs have become valuable intellectual assets, making them attractive targets for attackers seeking to extract the model rather than invest in training their own.

In many cases, DNNs are deployed as cloud-based or online services, allowing users to interact with them without direct access to their internal parameters [1,24]. A natural question, therefore, is how well different DNNs resist attacks in this black-box setting, where an adversary can query the network using random or carefully chosen inputs, potentially adaptively, and exploit the observed outputs to reconstruct either the exact internal parameters or an approximation sufficient to construct a functionally equivalent network.

Neural network model extraction is an old and well-studied problem. The earliest work dates back to 1994, when Fefferman [11] proved that the output of a sigmoid network uniquely determines its architecture and weights, up to trivial equivalences. Later, in 2005, Lowd and Meek [17] introduced new algorithms for reverse engineering linear classifiers and applied these techniques to spam filtering, marking one of the first practical extraction attacks. The field saw renewed interest in 2016 with the work of Tramèr et al. [25], who demonstrated attacks on deployed machine learning models accessible through APIs that output high-precision confidence values (also referred to as scores), in addition to class labels. By exploiting these confidence scores, they successfully mounted attacks on various model types, including logistic regression and decision trees.

For non-linear models, [25] and later [22] introduced the notion of *task accuracy extraction*, where the goal is to build a model that performs well on the same decision task, achieving high accuracy on predictions, without necessarily reproducing the exact outputs of the original model. This is different from *functionally equivalent extraction*, where the objective is to replicate the original model's outputs on all inputs, regardless of the ground truth. Jagielski et al. [14], who introduced this taxonomy, argued that learning-based approaches like [22,25] are fundamentally limited when it comes to achieving functionally equivalent extraction. Early attempts for functionally equivalent extraction relied on access to internal gradients of the model [19], side-channel information [3], or were limited to extracting only a small number of layers [14].

A significant step in model extraction came in 2020 with Carlini et al. [8], who studied ReLU networks in the S5 setting of [7], which exposes the network's pre-normalization outputs. In this work, the authors reframed the problem from a cryptanalytic perspective by observing that DNNs share key similarities

with block ciphers, including their iterative structure, alternating linear and non-linear layers, and the use in parallel of a small function (S-box in block ciphers, activation function in neural networks) to ensure non-linearity. Taking inspiration from cryptanalytic attacks on block ciphers, particularly differential cryptanalysis [4], Carlini et al. proposed a new two-step iterative approach for recovering a model's internal parameters. Their method decomposes the extraction process into two distinct phases: *signature extraction* and *sign extraction*. In the first step, the absolute values of the weights in a layer are reconstructed (signature extraction), followed by a second step where the correct signs are determined (sign extraction). This process is applied iteratively, proceeding layer by layer. This approach enabled them to successfully extract neural networks, including a model with 100,000 parameters trained on the `MNIST` digit recognition task, using only $2^{21.5}$ queries and less than an hour of computation. However, their method for extracting the signs had an exponential complexity in the number of neurons, limiting its feasibility to small networks with typically very few (at most 3) layers.

In 2024, Canales-Martínez et al. [6] extended the work of Carlini et al. [8] by introducing several new algorithms for the sign-extraction step in ReLU-based neural networks. Their improved sign-extraction methods permitted them to extract the parameters of significantly larger networks than those considered in [8]. For instance, they reported successfully extracting the parameters of a neural network with 8 hidden layers of 256 neurons each (amounting to 1.2 million parameters) trained on the `CIFAR-10` dataset [15] for image classification across 10 categories. It is important to note, however, that their experiments assumed the absolute values of the weights had already been recovered using the method of Carlini et al. [8] and were not implemented as full end-to-end attacks.

More recently, Foerster et al. [12] conducted a detailed analysis of the signature and sign extraction procedures, implementing and benchmarking them on various ReLU-based neural networks. Their work revealed that, contrary to what was previously believed, the main practical bottleneck preventing these attacks from advancing beyond the early layers lies in the signature-extraction step introduced in [8]. Because of this, both [8] and [12] failed in practice to recover weight parameters beyond the third layer. Although the authors of [6] reported successful extraction from deeper networks, this was assuming that signatures were already extracted with [8]'s method, which could not work on deeper parts of the networks. In addition, Foerster et al. [12] also identified a critical limitation in the sign-extraction method of [6]: for many networks, the confidence in the recovered signs of neurons does not increase even with additional iterations, contradicting the claim in [6] that wrong signs could be fixed by testing more critical points. To address this, they proposed applying exhaustive search, similar to [8], for neurons with low confidence. This indicates that, for many networks, sign extraction has not yet reached true polynomial-time extraction, leaving it as another unresolved bottleneck.

In parallel with these efforts to improve performance under full access to confidence scores, other members of the research community have turned their

attention to the more realistic "hard-label" scenario [7,9,29], (S1 setting in the taxonomy of [7]) where the DNN outputs only the predicted class label, corresponding to the highest confidence score, while the actual confidence values remain hidden. We show however that important problems persist even in the more informative S5 setting, and resolving them is a prerequisite for full extraction of deep networks. Moreover, because many techniques are shared across settings, advances achieved in S5 naturally carry over and improve extraction in the more restrictive scenarios as well.

To date, no method can efficiently recover a network's parameters beyond the third hidden layer in any setting (from S1 to S5)[1]. This highlights the need for significant improvements in the extraction to enable attacks on deeper networks, as well as an exploration of more challenging end-to-end attacks (attacks recovering a model's parameters solely from black-box input–output pairs, without any simplification such as artificially separating sign and signature extraction).

Our Contributions

Prior attacks do not scale to deep neural networks not merely because increased depth raises computational cost, but because their methods are incomplete and overlook issues that grow more severe in deeper models. In this work, we diagnose these challenges and propose practical solutions, thereby enabling effective end-to-end extraction on much deeper neural networks.

Reaching deeper layers. By identifying and addressing two critical limitations in signature extraction which arise as network depth increases, we allow signature extraction to progress beyond the first few layers:

- *Incorrect signature extraction due to rank-deficient systems.* The original attack produces incorrect signatures at deeper layers when the linear system used to compute signatures is rank-deficient. We provide a method to increase the rank, yielding correct signature extraction on deeper networks.
- *Overlooked influence of deeper layers.* Contrary to what was previously believed, we show that signatures originating from deeper layers than the one attacked can be misinterpreted as coming from the target layer. We propose strategies to filter them out depending on the neural network targeted.

End-to-end extraction. We present the first end-to-end model extraction that runs in polynomial time, achieved through notable enhancements to signature precision improvement and sign-extraction methods.

- *Signature precision improvement.* In addition to Carlini et al. [8]'s approach for improving signature precision, we identify and address two additional sources of imprecision in signature extraction, which improves extraction efficiency and supports wide layers.

[1] The authors of [7] report having successfully extracted the 4th layer of a single network. However, in that network, this 4th layer is very contractive, causing the issues identified in our work not to arise.

– *Confident sign extraction.* As noted by [12], the neuron wiggle method of [6] fails for the networks we consider on so-called low-confidence neurons. Instead of the exponential exhaustive search that [12] uses, we combine two polynomial methods from [6] to avoid relying on low-confidence neurons.

We validate all our improvements through extensive experiments on ReLU-based neural networks, showing significant improvements in extraction depth. Notably, we present the first end-to-end model extraction on eight-layer networks with eight neurons per layer trained on MNIST and CIFAR-10 datasets, extracting almost all the weights and achieving a very low relative error $\epsilon \leq 3.6 \times 10^{-4}$ on 73% of the input space, while previous works could barely extract the first three layers. These results mark a significant step toward practical attacks on larger, more complex neural network architectures. Our full code is accessible in `https://github.com/PsyduckLiu/End-to-End-Deep-Neural-Network-Extraction`.

The rest of the article is organized as follows. Section 2 introduces preliminaries: notations, the attack model, and a short overview of the original signature-extraction and sign-extraction methods. Section 3 presents our improvements to signature extraction. Section 4 describes our new sign-extraction strategy. Section 5 introduces our evaluation metrics and discusses a taxonomy of unrecovered weights. Section 6 reports our experimental evaluation and end-to-end results. We evaluate our attack on three different networks: one trained on `CIFAR-10` with architecture 3072-$8^{(8)}$-1 and two networks trained on `MNIST` with architectures 784-$8^{(8)}$-1 and 784-$16^{(8)}$-1, which we refer to as Model I, II, and III, respectively. Finally, Section 7 concludes and discusses limitations.

2 Preliminaries

In this section, we introduce important definitions and notations (Section 2.1), followed by assumptions regarding the attack setting and goal (Section 2.2) and finally an overview of the original attack of [8] and [6] (Section 2.3).

2.1 Notations and Definitions

This paper models neural networks as parametrized functions whose parameters are the unknowns we aim to extract. Our results do not depend on how neural networks are trained or applied. As a result, no prior knowledge about them is required to understand the attack.

A neural network consists of fundamental units called neurons, which are organized into layers and connected to other neurons in both the previous and next layers. Each neuron has an associated weight vector for its incoming neurons from the previous layer, along with a bias term that influences its output. Following the notations and definitions from [6,8], we now present several central definitions related to neural networks. A simple example to illustrate the definitions can be found in Appendix A.

Definition 1 (r-deep neural network). *An* r-deep fully connected neural network *of architecture* $[d_0, \cdots, d_{r+1}]$ *is a function* $f : \mathbb{R}^{d_0} \to \mathbb{R}^{d_{r+1}}$ *composed of alternating linear layers* $\ell^{(i)} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ *and a non-linear activation function* σ *acting component-wise such that:* $f = \ell^{(r+1)} \circ \sigma \circ \cdots \circ \sigma \circ \ell^{(2)} \circ \sigma \circ \ell^{(1)}$*, where* $\ell^{(i)}(x) = A^{(i)}(x) + b^{(i)}$*;* $A^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$*,* $b^{(i)} \in \mathbb{R}^{d_i}$*. We call:*

- r*: the number of layers (or* depth *of the network).*
- d_i*: the number of neurons in the* i*-th layer (or* width *of layer* i*).*
- $\ell^{(i)}$*: the* i*-th linear layer function.*
- $A^{(i)}$*: the* i*-th linear layer weight matrix.*
- $b^{(i)}$*: the* i*-th linear layer bias vector.*

To extend the notations, the architecture $[d_0, \ldots, d_{r+1}]$ can also be written as $d_0 - \cdots - d_{r+1}$. For consecutive layers with the same dimension, an exponential notation can be used for compactness, e.g., $20-10^{(3)}-1$ represents the architecture $20 - 10 - 10 - 10 - 1$.

As in [6,8,12], this research only considers fully connected neural networks using the widespread ReLU activation function [20] applied component-wise. The structure of a fully connected neural network resembles that of substitution-permutation networks (SPNs) such as the AES [2]. The component-wise application of the activation function is analogous to the use of S-boxes in SPNs, hence the analogy with cryptanalysis introduced in [8].

Definition 2 (ReLU neural network). *We say that a neural network* f *defined as above is a* ReLU neural network *if its non-linear activation function* σ *is*

$$\begin{aligned}\sigma(v) &= (\mathrm{ReLU}(v_1), \mathrm{ReLU}(v_2), \ldots, \mathrm{ReLU}(v_n)) \\ &= (\max(v_1, 0), \max(v_2, 0), \ldots, \max(v_n, 0))\end{aligned}$$

for $v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$*.*

In the following, we define a reduced-round neural network $F^{(i)}$ for $1 \leq i \leq r$ as a function $F^{(i)} : \mathbb{R}^{d_0} \to \mathbb{R}^{d_i}$ given by:

$$F^{(i)} = \sigma \circ \ell^{(i)} \circ \sigma \circ \cdots \circ \sigma \circ \ell^{(2)} \circ \sigma \circ \ell^{(1)}.$$

$F^{(i)}$ shares the same linear transformations and activation functions as the original function f, up to layer i. It will be used to track the transformation of an input vector as it goes through f. By $F^{(r+1)}$ and $F^{(0)}$, we mean respectively f and the identity function.

The *k-th neuron* of layer i is defined as the function $\eta_k^{(i)}(x) = A_k^{(i)} \cdot x + b_k^{(i)}$, where $A_k^{(i)}$ is the k-th row of $A^{(i)}$ and $b_k^{(i)}$ is the k-th coordinate of $b^{(i)}$.

A central notion in the analysis of [8] is that of *critical points* of neurons. They enable the extraction of their corresponding neuron's weights through carefully chosen queries.

Definition 3 (critical point). *An input point $x \in \mathbb{R}^{d_0}$ is called a* critical point *of a neuron $\eta_k^{(i)}$ if*

$$\eta_k^{(i)} \circ F^{(i-1)}(x) = 0$$

The activation pattern for a given input x is the set of all (active) neurons that contribute to the output $f(x)$.

Definition 4 (activation pattern). *The* activation pattern *of x through f at layer i, for $1 \leq i \leq r$, is the set*

$$\mathcal{S}_f^{(i)}(x) := \{\eta_k^{(j)} \mid \eta_k^{(j)} \circ F^{(j-1)}(x) > 0, \quad 1 \leq j \leq i, 1 \leq k \leq d_j\}.$$

If $i = r$, we note $\mathcal{S}_f^{(i)}$ as $\mathcal{S}_f$ and refer to it as the activation pattern of x through f. If $\eta_k^{(i)} \in \mathcal{S}_f(x)$ then we say that neuron $\eta_k^{(i)}$ is active*, otherwise we say that it is* inactive *(for x and f).*

Definition 5 (polytope). *The polytope of x at layer i, $\mathcal{P}_x^{(i)}$, is the largest connected open subset of*

$$\{x' \in \mathbb{R}^{d_0} \mid \mathcal{S}_f^{(i)}(x) = \mathcal{S}_f^{(i)}(x')\}$$

containing x. If $i = r$, we note $\mathcal{P}_x^{(i)}$ as $\mathcal{P}_x$ and refer to it as the polytope of x.

Therefore, $\mathcal{P}_x$ represents the region of the input space consisting of all points x' connected to x that have the same activation pattern. Let's now justify the use of the term *polytope* to describe this space.

Lemma 1 (local affine network). *First, for $x \in \mathbb{R}^{d_0}$, for all $1 \leq i \leq r$, the network $F^{(i)}$ is affine on $\mathcal{P}_x^{(i)}$, meaning that there exists $\Gamma_x^{(i)} \in \mathbb{R}^{d_i \times d_0}$, $\gamma_x^{(i)} \in \mathbb{R}^{d_i}$ such that $\forall x' \in \mathcal{P}_x^{(i)}$, we have $F^{(i)}(x') = \Gamma_x^{(i)} x' + \gamma_x^{(i)}$. We note:*

- *$F_x^{(i)}$ the function $x' \mapsto \Gamma_x^{(i)} x' + \gamma_x^{(i)}$.*
- *f_x and $F_x^{(r+1)}$ the function $x' \mapsto \ell^{(r+1)} \circ F_x^{(r)}(x')$.*

Second, it follows that if x is not a critical point, there exists $\epsilon > 0$ such that for all $x' \in \mathcal{B}(x; \epsilon)$,

$$f(x') = f_x(x') \quad \textit{and} \quad \forall i \in \{1, \ldots, r+1\}, \quad F^{(i)}(x') = F_x^{(i)}(x'),$$

where $\mathcal{B}(x; \epsilon) \subset \mathbb{R}^{d_0}$ is the ball of radius ϵ centered at x.

Though ReLUs are not linear, they are piecewise linear. This means that when moving near a non-critical point, all ReLUs act as linear functions. Recall that critical points are solutions of an equation of the form $A_k^{(i)} \cdot F^{(i-1)}(x) + b_k^{(i)} = 0$. Thus, they form hyperplanes which partition the input space into these different affine regions, hence referred to as *polytopes*.

We denote by $I_x^{(i)}$ the diagonal matrix representing which neurons in layer i are active for the input x: the k-th coefficient of the diagonal is 1 if the k-th neuron in layer i is active, and 0 otherwise. Thus,

$$\begin{aligned} F^{(i)}(x) &= I_x^{(i)}(A^{(i)} \cdots (I_x^{(2)}(A^{(2)}(I_x^{(1)}(A^{(1)}x + b^{(1)})) + b^{(2)}) \cdots + b^{(i)}) \\ &= I_x^{(i)}A^{(i)} \cdots I_x^{(2)}A^{(2)}I_x^{(1)}A^{(1)}x + \gamma_x^{(i)} = \Gamma_x^{(i)}x + \gamma_x^{(i)}. \end{aligned}$$

If x and x' are two points in the same polytope, then $\forall i, I_x^{(i)} = I_{x'}^{(i)}$ implying that $\Gamma_x^{(i)} = \Gamma_{x'}^{(i)}$ and $\gamma_x^{(i)} = \gamma_{x'}^{(i)}$, i.e., that $F_x^{(i)} = F_{x'}^{(i)}$.

2.2 Adversarial Resources and Goal

We consider two parties in this model extraction attack: an oracle $\mathcal{O}$ and an adversary. The adversary generates queries x and sends them to the oracle, which then responds with the correct output $f(x)$.

Adversarial resources. We make the following assumptions regarding the target neural network and the attacker's capabilities. These are the same assumptions as in [6,8,12]:

- **Fully connected ReLU network**. f is a fully connected ReLU neural network.
- **Known architecture**. The attacker knows the architecture of the target neural network f.
- **Unrestricted input access**. The attacker can query the network on any input $x \in \mathbb{R}^{d_0}$.
- **Raw output access**. The oracle returns the complete raw output $f(x)$, with no post-processing.
- **Precise computations**. The oracle computes $f(x)$ using 64-bit arithmetic.

Adversarial goal. The objective of the extraction is not to exactly replicate the target network, but rather to achieve what is known as an (ϵ, δ)-*functionally equivalent extraction* [8]. We slightly change the original definition to normalise the difference between the target and the extracted network.

Definition 6 (functionally equivalent extraction). *We say that two models f and $\hat{f}$ are* (ϵ, δ)-functionally equivalent *on an input space $S \subset \mathbb{R}^{d_0}$ if $\forall x \in S$, $\mathbb{P}(|\frac{\hat{f}(x)-f(x)}{f(x)}| \leq \epsilon) \geq 1 - \delta$.*

2.3 Overview of the Existing Signature Extraction

In this section, we recall the signature-extraction algorithm from [8]. The extraction is performed layer by layer: first, we recover the parameters of layer 1, then use them to reconstruct layer 2, and so on. We now explain the process of recovering layer i, assuming that the first $i-1$ layers of the target network f have been correctly extracted.

Signature extraction is carried out in five main steps: searching critical points, recovering partial signatures, merging partial signatures, finding missing entries in the signatures, and computing the bias.

Random search for critical points. In each polytope, the network behaves affinely, meaning that the derivative of f remains constant within the same polytope. Therefore, critical points can be identified when a change in the derivative occurs, indicating that a critical hyperplane has been crossed. To find these critical points, we apply a binary search along random lines in the input space. See Appendix B for a detailed explanation.

Partial signature recovery. When we cross a neuron's critical hyperplane, we move, depending on the sign of the neuron, from a region (polytope) where the neuron does not contribute to the network's output to one where it does. All other neurons remain unchanged. Consequently, by querying the network close to a critical point, we can construct a system of equations that isolates the contribution of the neuron in question, allowing us to recover its parameters up to a sign. We now describe this process in more detail.

Following [8], we define the *second-order differential operator* of f along direction Δ as

$$\partial^2_{\Delta} f(x) := \frac{1}{\epsilon_1}(f(x + \epsilon_2 \Delta) + f(x - \epsilon_2 \Delta) - 2f(x)).$$

Suppose the target layer is layer i. We assume that the previous layers have been extracted and thus that $\Gamma_x^{(i-1)}$ is known for all inputs x.

Lemma 2 ([8]). *Let x be a critical point for a neuron $\eta_k^{(i)}$ in layer $i \in \{1, \ldots, r\}$, and assume that x is not a critical point for the other neurons. Further, let $\Delta \in \mathbb{R}^{d_0}$. Then:*

$$\partial^2_{\Delta} f(x) = c_k^{(i)} \left| A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta) \right|$$

where $c_k^{(i)} \in \mathbb{R}$ is a constant.

Proof. See Appendix C.1.

By using Lemma 2 in two directions, Δ and Δ_0, we obtain

$$\frac{\partial^2_{\Delta} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{\left| A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta) \right|}{\left| A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta_0) \right|}.$$

Then, by comparing the absolute values between

$$\frac{\partial^2_{\Delta} f(x) + \partial^2_{\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)} \text{ and } \frac{\partial^2_{\Delta + \Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)},$$

we can eliminate the absolute value (see Appendix C.2), obtaining

$$\frac{A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta)}{A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta_0)}.$$

By taking enough directions Δ in the input space, we can solve for $A_k^{(i)}$ up to a constant $c_k^{(i)}$, reconstructing the signature (see Appendix C.3). We note $\hat{A}_k^{(i)}$ the recovered signature of $A_k^{(i)}$. As k remains unknown, the extraction permutes the rows of $A^{(i)}$: we do not know if $\hat{A}_1^{(i)} = A_1^{(i)}$ or if $\hat{A}_1^{(i)} = A_2^{(i)}$. When extracting the next hidden layer $\hat{A}^{(i+1)}$, the end-to-end attack naturally permutes the columns to match the permuted rows of $\hat{A}^{(i)}$, since $\hat{A}^{(i)}$, rather than $A^{(i)}$, is used in the extraction. In the same fashion, the next layer absorbs the constant $c_k^{(i)}$: $\hat{a}_{j,k}^{(i+1)} = a_{j,k}^{(i+1)} \cdot \frac{c_j^{(i+1)}}{c_k^{(i)}}$. The extracted last layer forces the permutation and scaling to match the target neural network, yielding a functionally equivalent extraction.

However, since $\Gamma_x^{(i-1)} = I_x^{(i-1)} A^{(i-1)} \Gamma_x^{(i-2)}$, ReLUs on layer $i-1$ are blocking some coefficients for all directions Δ we take around x. Therefore, instead of extracting a full (scaled) matrix row, for example $(a_1, a_2, a_3, a_4, a_5)$, we can only retrieve a partial signature, say, $(0, a_2, a_3, 0, a_5)$.

Merging signatures into components. To reconstruct the full signature, we need to merge partial signatures obtained from different critical points of the same neuron $\eta_k^{(i)}$. Each recovered signature from critical points of $\eta_k^{(i)}$ is $A_k^{(i)}$ with some missing entries, scaled to an unknown factor. Assuming that no two rows of the matrix are identical, merging two signatures requires only checking whether all their shared non-zero weights are proportional. For example, consider two signatures $(\lambda a, \lambda b, 0, \lambda d, 0)$ and $(\Lambda a, \Lambda b, 0, 0, \Lambda e)$ where λ and Λ are constants. If $\frac{\lambda a}{\lambda b} = \frac{\Lambda a}{\Lambda b}$, they can be merged to get $(a, b, 0, d, e)$, up to a constant (see Fig. 1). The resulting merged row is called a *component*. The *size of a component* is the number of critical points whose partial signatures merged to form that component.

If a component has a size greater than 2, the original attack from Carlini et al. [8] assigns it to the target layer, based on the belief that signature extraction would not produce consistent results on critical points from deeper layers (deeper layers are not expected to have such linear dependencies). However, we will explain later why this is not a correct perception, as critical points from deeper layers might indeed be merged. The process described above (randomly searching for critical points, extracting partial rows, and merging signatures) is repeated until the number of components of size > 2 is equal to d_i, the number of neurons at layer i.

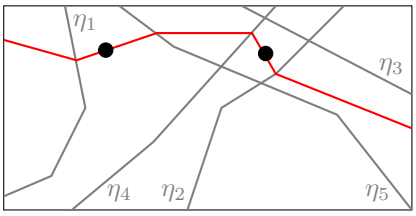


Fig. 1: Input space of a network where neurons on the previous layer are labelled in grey on their active side. In red is the neuron we aim to find. The critical points on the left and right yield respectively $(\Lambda a, \Lambda b, 0, 0, \Lambda e)$ and $(\lambda a, \lambda b, 0, \lambda d, 0)$. We can infer $(a, b, 0, d, e)$ up to a constant, even though no single polytope activates $\eta_1, \eta_2, \eta_4, \eta_5$ simultaneously.

Targeted search for critical points. At this stage of the attack, we know that all selected components correspond to neurons on the target layer. However, components recovered from random critical points often have weights missing, even after merging numerous partial signatures. To complete the partial rows, a more targeted search for critical points is necessary. We start from one of the critical points within the component. As we cross different hyperplanes associated with neurons in the previous layer, we activate different weights of the target neuron, ultimately enabling its full reconstruction.

For example, in Figure 1, we would follow the red hyperplane as it bends across the grey hyperplanes, retrieving enough critical points to reconstruct the full signature of our neuron. In higher dimensions, we follow directions that are more likely to trigger the weights we have not found yet. The exact procedure is described in [8] and further improved in [12].

Recovering the bias. Once we have recovered the signature of $\eta_k^{(i)}$, $c_k^{(i)} A_k^{(i)}$, it suffices to pick a critical point x of the component to compute the corresponding scaled bias $c_k^{(i)} b_k^{(i)}$ using the equation $(c_k^{(i)} A_k^{(i)}) \cdot F^{(i-1)}(x) + c_k^{(i)} b_k^{(i)} = 0$.

Last layer recovery. The oracle returns the complete raw outputs of f and the last output layer is a linear layer. Therefore, its extraction is straightforward. No additional query is needed for extraction, as we can reuse previously queried critical points to build the linear system and recover the last linear layer's weights.

2.4 Overview of Existing Sign-Extraction Methods

After recovering each neuron's signature in layer up to a constant $c_i^{(k)}$, we must determine the sign of each constant. Canales-Martínez et al. [6] propose two complementary sign-extraction techniques: the *system-of-equations* (SOE) approach and the *neuron wiggle*. We briefly summarize both methods.

SOE. The system-of-equations method aims at contractive layers, where it recovers the signs of every neuron in the target layer together. Locally around an input x, for a small perturbation direction Δ_k, we have

$$f(x + \Delta_k) - f(x) \;=\; G_x^{(i+1)}\, I_x^{(i)}\, A^{(i)}\, F_x^{(i-1)}\, \Delta_k,$$

where $G_x^{(i+1)}$ is the linear contribution of deeper layers at x, $A^{(i)}$ is the weight matrix of the target layer, $I_x^{(i)}$ is the diagonal activation mask of layer i at x, and $F_x^{(i-1)}$ is the local extracted linear map up to layer $i-1$. To find the signs of all neurons, we want to find $I_x^{(i)}$. Equivalently, we want to find which entries of $G_x^{(i+1)} I_x^{(i)}$ are zero. Therefore we build the following system:

$$\{c \cdot y_k = z_k\}_k$$

where $c = G_x^{(i+1)} I_x^{(i)}$ is the variable we are looking for, $y_k = A^{(i)} F_x^{(i-1)} \Delta_k$ and $z_k = f(x+\Delta_k) - f(x)$. To solve SOE, we thus need that $\text{rank}(F_x^{(i-1)}) \geq \text{rank}(A^{(i)})$. This technique can therefore be used as long as the network stays very contractive.

It is thus usually applicable only for the first layer.

Neuron Wiggle. For non-contractive layers, the neuron wiggle can be used to recover neuron signs individually. Consider a change in input Δ_k sufficiently small to stay within the linear region but which maximises $\hat{\eta_k}(x^* + \Delta_k)$. The general intuition is that the ReLU blocks the change caused by Δ_k on the inactive side of η. Therefore if our sign guess is correct, the output should display a large change. If our sign guess is incorrect, we shouldn't see a particular change. We can observe a correct change only if we can sufficiently maximise $\hat{\eta_k}(x^* + \Delta_k)$ without maximising at the same time other neurons. We refer the reader to the original article [6] for further details. This makes the neuron wiggle a probabilistic method which depends on many factors, notably the architecture of the network. As a consequence, the neuron wiggle is ran on many critical points x^* of η_k and the sign is chosen according to the overall results, with a certain confidence level α. $\alpha = 1$ indicates that all points agree, whereas $\alpha = 0$ implies an even split. Canales-Martínez et al. report very convincing results on a $3072 - 256^{(8)} - 10$ network, recovering the sign of all but ten neurons in this very large network. The neuron wiggle is further scrutinised in [12], where the authors recommend running an exhaustive search on the signs of the neurons with confidence level $\alpha < 0.75$.

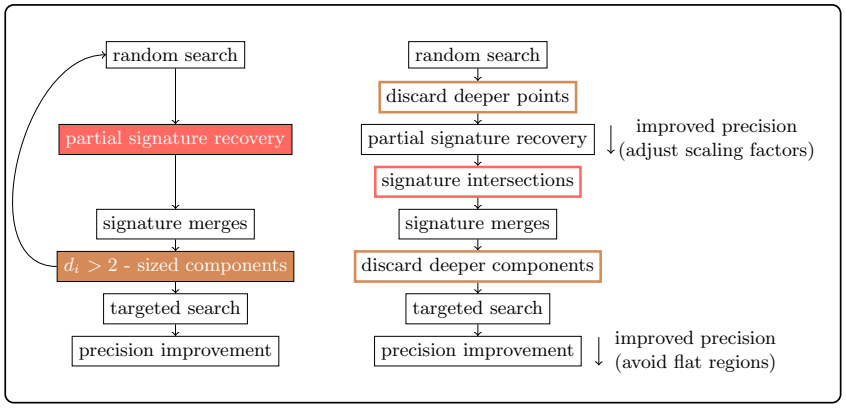


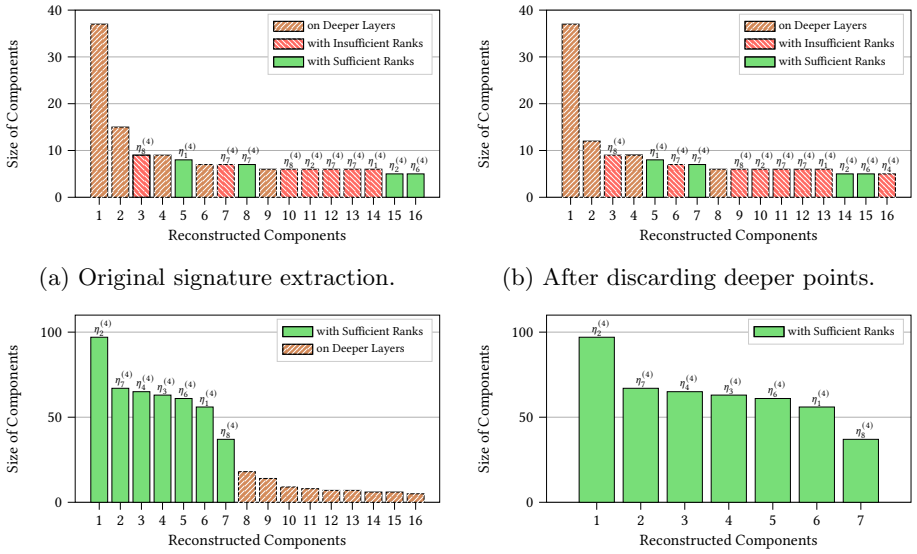
Fig. 2: Left: Original signature extraction from [8]. Right: Proposed improvements. Two error-inducing steps in the original attack are coloured on the left. Improvements match the colour of the step they address. Further precision improvements for signature extraction are marked on the right.

3 Improving Signature Extraction

To the best of our knowledge, all prior work has been able to apply signature extraction on very shallow neural networks only (3 layers maximum). Specifically,

according to Table 1 in [8], the deepest model they attempted to attack had an architecture of $40 - 20 - 10 - 10 - 1$, consisting of only 3 hidden layers. On the other hand, Table 2 in [12] shows that they attempted to attack a model with an architecture of $784 - 16^{(8)} - 1$, which had 8 hidden layers. However, despite running the attack for over 36 hours, they failed to recover the fourth hidden layer. Although [6] demonstrated that their sign extraction was effective across all layers of a complex $3072 - 256^{(8)} - 10$ model, they supposed in their experiments that the signatures had been at this stage correctly extracted, without running the two processes together.

Motivated by these limitations, we analyze the signature extraction method and identify two challenges preventing its successful application on deeper layers. We then propose solutions to overcome them. The left part of Fig. 2 illustrates the signature extraction workflow and highlights where these challenges arise.



(a) Original signature extraction.

(b) After discarding deeper points.

(c) After signature intersections.

(d) After discarding deeper components.

Fig. 3: Gradual improvement of signature extraction results on layer 4 of Model II with $3,000$ critical points (due to space constraints, only the largest 16 components are displayed). Each component on the target layer is labelled with its associated neuron on top. (a) Original signature extraction from [8]. (b) After discarding deeper critical points (see Section 3.2), deeper components either disappear or have a smaller size. (c) After intersecting critical points with insufficient ranks (see Section 3.1), components with rank deficiency disappear. (d) After discarding deeper components (see Section 3.2), all remaining components are in the target layer. The only unrecovered component corresponds to an always-off neuron $\eta_5^{(4)}$.

We provide now a brief overview of each failure. First, the signature extraction relies on solving a system of equations to recover the weights of the target neuron

associated with all active neurons in the previous layer. However, as we go deeper into the network, the rank of this system might not match the number of active neurons on the previous layer. This mismatch can lead to an underdetermined system, resulting in incorrect signature extraction. Such components are in red in Fig. 3.

Second, the component selection of the largest d_i components of size greater than 2 is not a good approach. Indeed, our experiments show that this strategy is only valid for shallow networks where there are comparatively fewer critical points on deeper layers. The right part of Fig. 2 gives an outline of our improvements. Components from deeper layers are in brown in Fig. 3.

Our methods for resolving the issues of incorrect signature extraction and deeper components are given respectively in Section 3.1 and Section 3.2. We show in Fig. 3, as an illustration, how each improvement we propose allows us to gradually reach a correct extraction the 4th layer of Model II with architecture $784 - 8^{(8)} - 1$. The same approach naturally extends to subsequent layers.

Moreover, we present in Section 3.3 several numerical precision-improvement strategies that increase extraction efficiency, and provide additional precision refinements for wide layers in the context of an end-to-end attack. The rightmost part of Fig. 2 indicates when they are applied.

3.1 Increasing Rank with Subspace Intersections

The issue: insufficient rank. When extracting the signature, our goal is to recover a partial signature $y_x \in \mathbb{R}^{d_{i-1}}$ of a neuron $\eta^{(i)}$ at its critical point x. Consequently, the solution space should be of dimension 1. To determine y_x, we solve $(\Gamma_x^{(i-1)}\Delta) \cdot y_x = \partial_\Delta^2 f(x)$ using a sufficient number of random directions $\Delta \in \mathbb{R}^{d_0}$. The resulting partial row y_x consists of the coefficients for the neurons in layer $i-1$ that are active at input x, entries on inactive indices are zero. We denote by $s_x^{(i-1)}$ the number of active neurons in layer $i-1$ and by $r_x^{(i-1)}$ the rank of the map $\Gamma_x^{(i-1)} \in \mathbb{R}^{d_{i-1} \times d_0}$. If $r_x^{(i-1)} < s_x^{(i-1)}$, the system is underdetermined and y_x cannot be uniquely determined. This typically occurs when some earlier layer $k < i-1$ has fewer active neurons than layer $i-1$ (see Fig. 4), since $\Gamma_x^{(i-1)} = I_x^{(i-1)} A^{(i-1)} \cdots I_x^{(1)} A^{(1)}$ and thus:

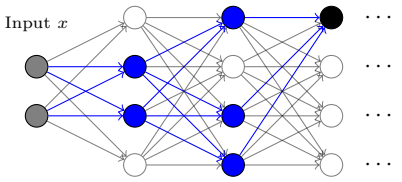


Fig. 4: Solving for a neuron's weights from certain critical points can yield an underdetermined system. The target neuron is shown in black; active neurons are in blue. At input x, $\operatorname{rank}(\Gamma_x^{(1)}) = 2$, so $\operatorname{rank}(\Gamma_x^{(2)}) \leq 2$ even though three neurons are active in layer 2; consequently, the layer-2 system has no unique solution.

$$\operatorname{rank}(\Gamma_x^{(i-1)}) \leq \min(\{\operatorname{rank}(I_x^{(k)})\}_{1 \leq k \leq i-1}) \neq \operatorname{rank}(I_x^{(i-1)}) \text{ in general.}$$

If X denotes the random variable representing the number of active neurons per layer, then $\mathbb{P}(r_x^{(i-1)} < s_x^{(i-1)}) = 1 - \mathbb{P}(X \geq s_x^{(i-1)})^{i-1}$. Because of the exponent $(i-1)$, this (failure) probability approaches 1 rapidly with depth, making extraction beyond the first few layers increasingly unlikely.

Signature intersections. If the rank is insufficient, we no longer have a unique solution but rather a space of solutions. For a critical point x in the layer i, the solution space is given by $\mathcal{S}_x = L_x + \ker(\Gamma_x^{(i-1)})$, where L_x is any particular solution and $\ker(\cdot)$ denotes the kernel. Suppose we have another critical point x' with solution space $\mathcal{S}_{x'} = L_{x'} + \ker(\Gamma_{x'}^{(i-1)})$. We want to intersect $\mathcal{S}_x$ and $\mathcal{S}_{x'}$ into a space of smaller dimension containing $A_k^{(i)}$. Doing this for multiple critical points should give us a space of dimension 1, yielding $A_k^{(i)}$ up to a scaling factor.

Let's study this in more details. Since we recover rows up to a scalar factor, we seek to compute the intersection of $\mathcal{S}_x$ and $\lambda\mathcal{S}_{x'}$, where λ is a scalar factor equal to the ratio of the row scalar factors of x and x'. Let $(e_1, \ldots, e_k)$ be a basis of $\ker(\Gamma_x^{(i-1)})$, and let $(f_1, \ldots, f_m)$ be a basis of $\ker(\Gamma_{x'}^{(i-1)})$. We then solve the following system in $\mathbb{R}^{d_{i-1}}$:

$$L_x + \mu_1 e_1 + \cdots + \mu_k e_k = \lambda L_{x'} + \lambda_1 f_1 + \cdots + \lambda_m f_m,$$

where the unknowns are $\mu_1, \ldots, \mu_k$ and $\lambda, \lambda_1, \ldots, \lambda_m \in \mathbb{R}$. This system consists of d_{i-1} equations in $\mathbb{R}$. However, only the equations corresponding to the active neurons in the layer $i-1$ for both inputs x and x' are relevant. To ensure that we do not merge spaces from critical points associated with different neurons, we overdetermine the system. Indeed, an overdetermined system with random coefficients is inconsistent with very high probability, and thus a system of $\ell > N$ equations with N unknowns typically doesn't have a solution except if there is some ground truth behind it. Therefore, we impose the following merging condition:

$$\ell > 1 + |\ker(\Gamma_x^{(i-1)})| + |\ker(\Gamma_{x'}^{(i-1)})| = 1 + k + m,$$

where ℓ is the number of relevant equations.

Intersections allow us to exploit at least 90% of critical points on the target layer when attacking for example the Model II. By comparison, the original attack gives a uniquely solvable system (and thus a correct signature) for only about 6% of layer-4 critical points, with even worse results in deeper layers. We could intersect subspaces three by three to have a lower condition for merging, allowing for the use of more critical points. However, this would considerably slow down the attack. For this reason, all our results use pairwise merges.

We provide in Fig. 3c an experiment showing how intersecting the critical-point solution spaces prunes components affected by rank deficiency, making the correct ones stand out.

3.2 Deeper Layers' Influence on Extraction

While going through the signature extraction procedure, we made the crucial assumption that x is a critical point of a neuron on the layer we are targeting.

Since we are trying to extract that layer, we cannot verify this assumption. We might be extracting a neuron on a deeper layer. The authors of [8] claim that it is exceedingly unlikely that signatures extracted from critical points on a deeper layer can be merged into a component. They conclude that a component of size greater than two is on the target layer. In this section, we first explain why signatures from deeper layers can indeed be merged. Second, we propose a two-step solution which consists in discarding deeper points, then deeper components. Finally we discuss how noise behaves in very large networks.

The issue: deeper merges. Assume that we are trying to extract layer i. From two critical points x_1 and x_2 of the same neuron in a deeper layer $i+t$, we solve respectively for partial weights y_1 and y_2 using the preceding layer $i+t-1$ in the systems below. Since x_1 and x_2 are critical points of the same neuron, and we are extracting from the preceding layer, we know that y_1 and y_2 can be merged. We'll see that under a specific scenario, the extraction from x_1 and x_2 can also be merged when using layer $i-1$ for extraction, inducing noise in our extraction.

$$\partial^2_{\Delta} f(x_1) = (\Gamma^{(i+t-1)}_{x_1} \Delta) \cdot y_1$$
$$\partial^2_{\Delta} f(x_2) = (\Gamma^{(i+t-1)}_{x_2} \Delta) \cdot y_2$$

Let's write A_{x_1} the matrix $I^{(i+t-1)}_{x_1} \circ A^{(i+t-1)} \circ I^{(i+t-2)}_{x_1} \circ \cdots \circ I^{(i)}_{x_1} \circ A^{(i)}$ and A_{x_2} the matrix $I^{(i+t-1)}_{x_2} \circ A^{(i+t-1)} \circ I^{(i+t-2)}_{x_2} \circ \cdots \circ I^{(i)}_{x_2} \circ A^{(i)}$. Thus, using A_{x_1} and A_{x_2}, we can rewrite

$$\partial^2_{\Delta} f(x_1) = [(A_{x_1} \circ \Gamma^{(i-1)}_{x_1}) \Delta] \cdot y_1$$
$$\partial^2_{\Delta} f(x_2) = [(A_{x_2} \circ \Gamma^{(i-1)}_{x_2}) \Delta] \cdot y_2$$

Now suppose that x_1 and x_2 have the same activation pattern between layers i and $i+t-1$, meaning they set the same neurons as active. In this case, $A_{x_2} = A_{x_1}$, which we write as A. Therefore, our systems are as follows.

$$\partial^2_{\Delta} f(x_1) = [(A \circ \Gamma^{(i-1)}_{x_1}) \Delta] \cdot y_1$$
$$\partial^2_{\Delta} f(x_2) = [(A \circ \Gamma^{(i-1)}_{x_2}) \Delta] \cdot y_2$$

These systems respectively yield the same solutions as:

$$\partial^2_{\Delta} f(x_1) = (\Gamma^{(i-1)}_{x_1} \Delta) \cdot (A^{\top} y_1)$$
$$\partial^2_{\Delta} f(x_2) = (\Gamma^{(i-1)}_{x_2} \Delta) \cdot (A^{\top} y_2).$$

Since y_1 and y_2 can merge, so can $A^{\top} y_1$ and $A^{\top} y_2$. These systems correspond exactly to the partial signature extraction from x_1 and x_2 when extracting layer i. This is why the partial signatures extracted from x_1 and x_2 can merge into a component even though they are not on the target layer. We call all these unwanted additional components from deeper layers *noise components*.

Discarding deeper points. To discard points on deeper layers, we recycle and improve an algorithm from Section 4.4.2 of [8] that was only used in the context of the targeted search. For each critical point found, the algorithm process involves: 1) computing a large number of distinct intersection points (we use 100 in practice) between the hyperplane extracted from the critical point under evaluation and the extracted network, and 2) verifying on the target neural network whether these intersection points still lie on the extracted hyperplane. One intersection point not belonging to the extracted hyperplane indicates that the hyperplane has bent on a neuron that has not yet been extracted, and hence that the extracted hyperplane is on a deeper layer. This test is not an if-and-only-if condition, as a hyperplane not breaking does not guarantee that it corresponds to a neuron on the target layer, see Fig. 5. This method reduces the noise, but it is far from sufficient, see Fig. 3b. We refer the reader to the original description for a more thorough account of the strategy.

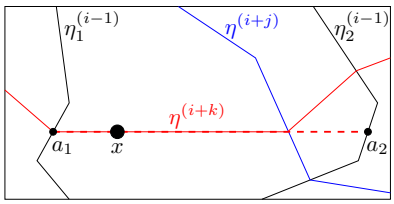


Fig. 5: Identifying if x is on the target layer. $0 \leq j < k$. By finding that the intersection point a_2 is not on the extracted hyperplane, we infer that the hyperplane we extracted from x (- - -) broke on a layer $i + j$ we did not extract (—). Thus, x cannot be on layer i. Yet, finding that a_1 is on the extracted hyperplane does not give any information about x's layer.

Discarding deeper components.
After discarding points on deeper layers, we compute intersections as described in Section 3.1, see Fig. 2. Intersections deal with components having insufficient rank. However, some components on deeper layers remain, see Fig. 3c. We use three criteria to discard them. First, we know that critical points that merge usually belong to the same neuron, regardless of whether they are in the target layer or a deeper layer. Therefore, the more critical points identified from deeper layers by the above test that merge with our candidate component, the more likely it is that this component belongs to a deeper layer. For this reason, we compute for each component a noise ratio:

$$\tau = \frac{\#\text{merges with deeper critical points}}{\text{size of the component}}.$$

Second, because merges of deeper critical points come from critical points with the same partial activation pattern, deeper components have a harder time obtaining a diverse set of critical points to yield a neuron with a very small number of entries unrecovered. Finally, because of the constraint on the activation pattern of deeper merges, deeper components tend to be smaller in size. While none of these criteria is an if-and-only-if test, combining them yields satisfying results.

Therefore, we first remove components with a size smaller than a fraction of the largest component (we found that in practice, 0.1 was sufficient for networks with eight hidden layers). Then, we discard components that either have a τ above a certain threshold (we use 0.1 and 0.2 for networks with width 8 and 16,

respectively) or a large number of entries unrecovered (at least half of them) if $\tau > 0$. Notably, making a strict association between deeper critical points and incorrect components can lead to mislabelling (example in Appendix D).

Noise in large networks. We have shown above that components coming from deeper layers result from critical points with the same activation pattern between the target layer and their layer. As the width of the network increases, the neurons on each layer cut the input space into smaller polytopes, making it more unlikely that critical points share the same partial activation pattern, up to the point of noise not being an issue anymore for a large enough width. For example, when extracting layer 4 of a $784 - 256^{(16)} - 1$ network with $100,000$ critical points, we get 189 components of size larger than 2. Only three of these do not belong to the target layer, with sizes 3, 3, and 4. By contrast, 177 components have size at least 5, and 157 have size at least 20. This shows that, in wide networks, imposing a minimum component size is an effective way to filter out noise.

As network width grows, intersection computations become a problem. With width 256, each critical point gives many recovered weights. This allows previously unavailable statistical arguments by viewing recovered weights as realisations of random variables. Indeed, the contribution of a neuron on a deeper layer corresponds to a very different mathematical expression from a neuron on the target layer. The deeper the neuron we are actually extracting upon, the higher the expected variance of the recovered weights is. Therefore, recovered neurons with unusual high variance can immediately be deemed to be on a deeper layer, without requiring any queries or computation, thereby increasing the proportion of critical points on the target layer. The time and queries for finding critical points grows linearly with the number of points we require, while computing intersections is comparatively very slow and grows polynomially with the number of points processed. For this reason increasing the proportion of critical points on the target layer, even if we discard some of them, results in a considerable speed-up of the attack. See Appendix E for proofs, experiments and more details.

3.3 Numerical Precision in the Context of an End-to-End Attack

In practical end-to-end attacks, model extraction relies on finite-precision arithmetic (typically 64-bit floating-point). However, many algorithms in the extraction process are numerically unstable, introducing small errors in the recovered weights. Such errors accumulate across layers and block the extraction of deeper layers. Thus, it is crucial to refine the recovered weights to minimise imprecision.

Carlini et al. [8] introduce a method to improve the precision of recovered signatures. It consists in finding more critical points, at least d_{i-1} (the number of neurons in layer $i-1$) for each neuron $\eta_k^{(i)}$ in the target layer i, and then solving the following linear system for its weight vector $\hat{A}_k^{(i)}$ and bias $\hat{b}_k^{(i)}$:

$$\begin{pmatrix} h_1 & 1 \\ h_2 & 1 \\ \cdots & \\ h_{d_{i-1}} & 1 \end{pmatrix} \begin{pmatrix} \hat{A}_k^{(i)} \\ \hat{b}_k^{(i)} \end{pmatrix} = 0 \qquad (\star)$$

where $h_j = \hat{F}^{(i-1)}(x_j)$ and x_j is a critical point for neuron $\eta_k^{(i)}$. This leads to precision improvements as weights given by components come from multiple imprecision-inducing computations (derivatives, system solving, etc.) which themselves depend on the precise identification of critical points.

To find more critical points, Carlini et al. [8] use a kind of targeted search. Foerster et al. [12] note that this search often struggles in practice and propose refinements, but these add numerous parameters hindering the reproducibility and substantially increasing the runtime. They report precision-improvement runtimes of 33 times that of signature extraction and 17 times that of sign extraction. For this reason, [12] recommends avoiding precision improvements, while noting that it is a "remaining concern" for an end-to-end attack.

Our improved signature extraction can efficiently avoid this issue. Specifically, we explicitly generate a larger pool of critical points at the beginning, a number far exceeding that required in Carlini et al. [8]. These points primarily serve to filter deep points and components in the early extraction steps, and are sufficient to be reused for precision refinement, eliminating the need for a dedicated, parameter-heavy targeted search. Our precision improvement step therefore remains a small fraction of the total extraction.

Carlini et al.'s precision improvement [8] is very potent but can only be applied at the end of the extraction of each layer. In other words, the attack per layer is still ran on imprecise numbers. Let's see why this can be a problem, before presenting our improvement to Carlini et al.'s method for wide layers.

Imprecision due to scaling factors affects extraction efficiency. When recovering a partial weight-vector y_x at a critical point x from a linear equation system $(\hat{\Gamma}_x^{(i-1)}\Delta) \cdot y_x = \partial_\Delta^2 f(x)$, each output from the previous layer is scaled by a neuron-specific factor, denoted as $(c_1^{(i)}, c_2^{(i)}, \cdots, c_{d_i}^{(i)})$. Large disparities among these scaling factors can make the matrix $\hat{\Gamma}_x^{(i-1)}\Delta$ ill-conditioned, amplifying numerical errors so that even tiny errors in recovered weights $\hat{\Gamma}_x^{(i-1)}$ cause large deviations in the partial weight-vector y_x. Critical points with such imprecise y_x cannot be merged into components, forcing the collection of more critical points initially to assemble a component capable of recovering all desired weights. As discussed in Section 3.2, computing intersections is already slow and grows polynomially with the number of points processed; imprecise partial weights further increase the number of queries and the cost of signature intersections, reducing overall efficiency.

To address this, we normalize each column of $\hat{\Gamma}_x^{(i-1)}\Delta$ by multiplying it by a secondary scaling factor $(c_1^{(i)'}, c_2^{(i)'}, \cdots, c_{d_i}^{(i)'})$, bringing all elements within $[-1, 1]$. After solving the normalized linear equation system, these factors are multiplied back onto the solution to obtain the recovered signature. This normalization ensures more stable and precise signature extraction, see Appendix F for an example. For instance, when our normalization method is applied to layer 3 of Model II, the precision of partial weights improves by roughly 10 times. The proportion of critical points merged into components increases from 88.9% to 96.2%, capturing nearly all points associated with the target-layer neurons.

Imprecision from critical points in large networks. We identify critical points by computing expected intersections of lines given by output derivatives, see Appendix B. The search for critical points can therefore be imprecise if the change in linearity caused by a critical hyperplane is small, i.e. if the output region is rather flat, see Fig. 6 for illustration. We thus classify critical points based on the angles between the second derivatives used to identify them. We then only stack up critical points with large derivative differences to construct the matrix in $(\star)$. For networks with a small width of 8, the number of flat critical points originally used in $(\star)$ is small and does not affect much the precision of the recovered neurons. Yet, we notice significant improvement for larger widths, for example a $\times 3$ precision improvement on the crucial first layer of our Model II.

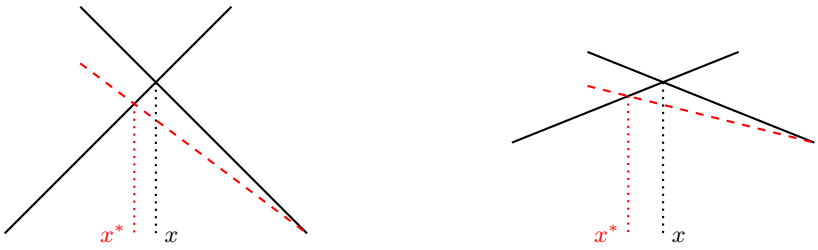


Fig. 6: Illustration of imprecisions in flat regions when finding critical points. In both cases the attacker computes a slightly imprecise derivative (dashed red line). Though the angular error (angle between the dashed red line and the black line next to it) is the same in the two cases, the distance between the extracted critical point x^* and the real one x changes depending on how flat the region is. The worst errors are caused by flat regions. For this reason we avoid them.

4 Confident Sign Extraction

In their original presentation of the neuron wiggle, the authors of [6] managed to recover almost all neuron signs for a $3072 - 256^{(8)} - 10$ neural network, assuming all signatures had been correctly recovered. It failed for neurons with so-called low confidence. This is why the authors of [6] recommend using more resources to ensure the sign extraction of the 10% lowest-confidence neurons. However, the authors of [12] found that in practice, on smaller architectures, many neurons have low confidence and, worse, increasing resources does not improve it. They resort to an exhaustive search over the signs of the low-confidence neurons to ensure successful recovery. Indeed, making a mistake on the sign of a neuron typically leads to a failure of the signature recovery of the following layer. This calls for a polynomial method to find the sign of low-confidence neurons.

We propose to combine SOE and the neuron wiggle to obtain a sign-extraction strategy which uses neurons with higher confidence than the one used by the neuron wiggle alone. We set-up the SOE under-determined system at a point x, and then use the result of the neuron wiggle on high-confidence neurons to

complete the rank: we choose the neuron inactive at x with the highest confidence, we eliminate its variable from the SOE unknowns. We iteratively add the most confident inactive neurons until the system reaches sufficient rank. Solving this system then gives the signs of the remaining low-confidence inactive neurons at x alongside all the active neurons, overcoming the limitations of the neuron wiggle. For this strategy to succeed, we need a point x where the number of neurons on the target layer inactive at x is at least d_i minus the SOE rank. We found such points easily for all networks considered by sending random input points through the extracted network. We also give a way to increase the rank of SOE. This rank-increasing method pushes further the value of combining SOE and the neuron wiggle, increasing the confidence of sign recovery.
As explained in Section 2.4, SOE consists in solving at a point x and for multiple directions Δ_k the system,

$$\{G_x^{(i+1)} I_x A^{(i)} F^{(i-1)} \Delta_k = f(x + \Delta_k) - f(x)\}_k,$$

where the unknown is $G_x^{(i+1)} I_x$. The rank of the system is limited by the rank of $F^{(i-1)}$ around x. Crucially, because the unknown is $G_x^{(i+1)} I_x$, any critical point with the same later activations (i.e., the same $G^{(i+1)} I^{(i)}$) can be used even if the previous activation patterns differ. This implies that we do not have to build the system around a single critical point. Instead, we search for different linear regions nearby which keep later activations fixed. This triggers many different activation patterns for $F^{(i-1)}$, their union having a higher rank, see Fig. 7. More details are given in Appendix G. We call this strategy of building the SOE system not from a single critical point but from a large zone an extension of SOE.

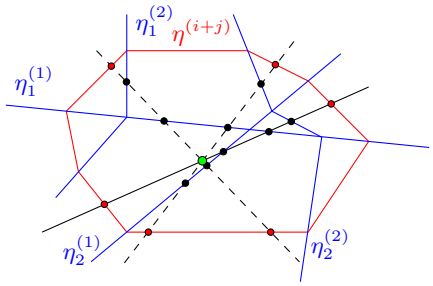


Fig. 7: Identifying critical points that share the same activation pattern after the target layer i. First, we randomly select a line (—) in the input space and identify the largest continuous segment of critical points (•) from previous layers that lies between two critical points (•) in deeper layers. Although these points differ in activation patterns in the previous layers, they share the same activation pattern in the deeper layers. We then take the midpoint (•) of this segment as the center of a sphere and explore multiple random directions (- - -) to find more critical points from previous layers within this region.

This yields two different strategies depending whether we favour a large zone to extend to, or an initial point with a high SOE rank and a large number of confident inactive neurons. Both strategies start by computing the neuron wiggle for all neurons. If we favour the former, the extension strategy, we first run random lines through the input space, then we perform the extension on the most promising zones (see Fig. 7). We then complete with the highest confidence neurons and select the zone with the highest lowest confidence used. The earlier we are in the extraction, the smaller the equivalence classes defined by $x_1 \sim x_2 \iff \{G_{x_1}^{(i+1)} I_{x_1} = G_{x_2}^{(i+1)} I_{x_2}\}$ are, and the less effective this increase in the rank of SOE is. This strategy works at its best in the deeper layers. If we favour the latter, the confidence strategy, we run random points through the network, compute the minimum number of active neurons per layer (the rank of SOE) and we select the point which maximises the lowest confidence used to solve the signs. We then perform an extension on that point before solving for the sign. This strategy works at its best in the first layers where the rank of SOE is still high. The authors of [6] argue that the rank of SOE stabilises in the deeper layers. Surprisingly, both strategies gave similar results in our experiments though they rely on two different structural properties of the target network. The extension strategy relies on the number of previous layer critical points within possible extensions whereas the confidence strategy relies on the rank of SOE as we progress through the network. An attacker can run both strategies and pick for each layer the one giving the more confident results. Variants between the two strategies can also be done (extension on more points than the x given by the confident strategy, assessing the SOE rank of the midpoint of the extension strategy etc.). We refer to all these new sign recovery strategies as SOE+Wiggle. For the sake of consistency, unless stated, all our experiments use the extension strategy. This is because for networks our end-to-end attack targets, the extension strategy can completely avoid the neuron wiggle for many layers, giving a deterministic method for sign recovery.

We compare our method with prior work on Model II in Table 1. Using the neuron wiggle alone, signs can be recovered only in layers 6 and 7; in these layers, the confidence falls below the 0.75 minimum threshold recommended by [12]. In contrast, SOE+Wiggle recovers all signs correctly across the network with high confidence and without resorting to exhaustive search. For large-scale networks, SOE+Wiggle remains effective. In Table 2, we compare our method with prior work on the $3072 - 256^{(8)} - 10$ model used in Table 4 of [6]. When computing each neuron's confidence using the neuron wiggle, we limit the total budget to 10^6 critical points, instead of collecting 200 points per neuron as in [6]. We set this limited budget because some neurons would require an excessive number of queries to obtain 200 critical points. With this budget, we still ensure that every neuron has a number of critical points that exceeds the threshold recommended in [12] for stable confidence estimates. Using the neuron wiggle alone, some signs in layers 5, 7, and 8 are incorrectly recovered, and the lowest confidence in each layer is low (around its minimum value 0.5). In contrast, our SOE+Wiggle method recovers all signs correctly across the network with high confidence.

Table 1: Comparison of sign-extraction methods on Model II. When recovering the signs of neurons on the target layer, we assume that all preceding weights have been correctly extracted.

Method	**Metric**	**L2***	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**
Neuron Wiggle [6]	correct sign extraction	7/8	5/8	5/8	6/8	8/8	8/8	5/8
Neuron Wiggle + exhaustive search [12]	low-confidence neurons ($\alpha < 0.75$)	4/8	5/8	5/8	4/8	4/8	3/8	4/8
SOE + Wiggle	correct sign extraction	8/8	8/8	8/8	8/8	8/8	8/8	8/8
(**our method**)	lowest confidence used	$\infty^\dagger$	0.81	∞	∞	0.75	1.0	∞

*L stands for layer. †When the SOE matrix reaches full rank after extension, the sign extraction is deterministic and the neuron wiggle is unnecessary.

Table 2: Comparison of different sign-extraction methods on the $3072-256^{(8)}-10$ model used in Table 4 of [6]. When recovering the signs of neurons on the target layer, we assume that all preceding weights have been correctly extracted.

Method	**Metric**	**CIFAR10-256$^{(8)}$-10**							
		L2*	**L3**	**L4**	**L5**	**L6**	**L7**	**L8**	**Recovery failures**
Neuron Wiggle [6]	lowest confidence used	0.53	0.53	0.51	0.50	0.50	0.51	0.51	15
Confidence strategy	lowest confidence used	0.87	0.83	0.75	0.67	0.64	0.62	0.78	0
	confidence increases from ext.$^\triangle$	–†	–	–	0.05	0.08	0.07	0.19	
Extension strategy	lowest confidence used	0.86	0.80	0.64	0.62	0.56	0.66	0.76	0
	confidence increases from ext.	–	–	0.09	$\infty^\ddagger$	∞	0.15	∞	

* L stands for layer. $^\triangle$ ext. stands for the extension of SOE. † When the SOE extension does not increase the SOE rank, the lowest confidence level used remains unchanged. ‡ Without SOE extension, the extension strategy fails as the SOE rank plus the number of inactive number at the point x is still smaller than d_i.

5 Evaluation Metrics and Unrecovered Weights

5.1 Evaluation Metrics

In practice, an attacker cannot query the entire input domain $\mathbb{R}^{d_0}$. Whether by design or due to constraints, extraction is restricted to a smaller region $S \subseteq \mathbb{R}^{d_0}$. On this space, which we require to be connected, some neurons do not have critical points and are thus always active (on) or always inactive (off). Always-off neurons can safely be ignored as they do not contribute to the network for inputs in S. Always-on neurons act as a fixed linear map on S. Under our goal of functional equivalence on S, this is naturally absorbed when recovering the final layer of f. There is no need to identify these neurons individually. If neuron $\eta_k^{(i)}$ is always off on S, then the k-th column of $A^{(i+1)}$ never contributes to the output on S, and recovering those weights is unnecessary. Many situations can cause a weight to never influence the network's output on S. Thus, we introduce two definitions formalizing what the attacker is, in practice, aiming to recover.

Definition 7 (effective architecture). *Let f be a ReLU network with architecture $[d_0, d_1, \ldots, d_r, d_{r+1}]$ and let $S \subseteq \mathbb{R}^{d_0}$. For $i = 1, \ldots, r$, let n_i denote the*

number of neurons $\eta^{(i)}$ in layer i whose pre-activation has a non-trivial *zero set on S, i.e., $\{ x \in S \mid \eta^{(i)} \circ F^{(i-1)}(x) = 0 \} \notin \{\emptyset, S\}$. Equivalently, n_i counts the neurons that are neither always off nor always on over S. The* effective architecture *of f on S is then defined as $[d_0, n_1, n_2, \ldots, n_r, d_{r+1}]_S$.*

Definition 8 (effective weight). *Let $A^{(i)} = (a^{(i)}_{j,k})$ be the weight matrix from layer $i-1$ to layer i and let $S \subseteq \mathbb{R}^{d_0}$. We say that the weight $a^{(i)}_{j,k}$ is* effective on *S if it can influence the network's output on S, that is, if there exists $x \in S$ such that*

$$\left(\eta^{(i-1)}_k \circ F^{(i-2)}(x) > 0\right) \quad \textit{and} \quad \left(\eta^{(i)}_j \circ F^{(i-1)}(x) > 0\right).$$

We also say that $a^{(i)}_{j,k}$ is effective on x as a shorthand for effective on $\{x\} \subseteq S$.

Now that we know what we aim to recover, i.e., effective weights in a network which behaves according to its effective architecture, we need metrics indicating the quality of the extraction. The number of effective weights recovered is not sufficient. Indeed, not all weights contribute equally to the network's behaviour: failing to recover 1% of the weights does not generally mean that the extracted model will behave correctly on 99% of S. While [8] introduced the notion of (ϵ, δ)-functional equivalence to go beyond weights count, we propose a complementary metric, that we call *coverage*, to quantify the fraction of S on which all effective weights are recovered.

Definition 9 (coverage). *Let $S' \subseteq S \subseteq \mathbb{R}^{d_0}$ be the set of all x for which no unrecovered weight of the model (resp. layer) is effective. The* model (resp. layer) coverage *is the proportion of points of S that are also in S'. We refer to S' as the* covered space*.*

Representing volumes, coverage is estimated by sending a large set of random input points through the network and checking how many of them rely on unrecovered weights. The main purpose of coverage is to distinguish errors due to missing information (unrecovered but effective weights) from those due to numerical imprecision, since coverage is independent of the extraction's numerical precision. Unlike an (ϵ, δ) assessment (see Def. 6), which requires choosing a tolerance ϵ and whose meaning depends on the task/scale (the same ϵ can have a very different interpretation across networks), coverage is parameter-free and task-agnostic. Setting δ to 1− coverage and computing ϵ over the covered space provides a cleaner view of precision-improvement strategies: in practice, ϵ stabilizes quickly once $1 - \delta$ falls below the coverage (Fig. 8).

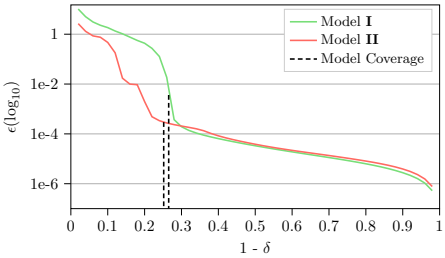


Fig. 8: (ϵ, δ) analysis of our end-to-end extractions of Models I and II. As the reduced-space proportion δ increases, the maximum output error ϵ decreases. The relationship is initially very sharp but smooths out after a transition phase, which aligns with the model coverage.

5.2 Unrecovered Effective Weights

Our aim is to increase the coverage by recovering as many effective weights as possible given an attack cost. Let's discuss the effective weights we do not recover. They fall in two distinct categories.

Definition 10 (taxonomy of unrecovered weights). *For a given weight $a_{j,k}^{(i)}$, let $S_{(+,+)} \subseteq S \subseteq \mathbb{R}^{d_0}$ be the set of input points activating the two neurons connected by $a_{j,k}^{(i)}$:*

$$S_{(+,+)} = \{x \in S | (\eta_k^{(i-1)} \circ F^{(i-2)}(x) > 0) \wedge (\eta_j^{(i)} \cdot F^{(i-1)}(x) > 0)\}$$

Similarly, let $S_{(+,-)} \subseteq S$ be the set of input points activating, out of the two neurons connected by $a_{j,k}^{(i)}$, only the neuron on layer $i-1$:

$$S_{(+,-)} = \{x \in S | (\eta_k^{(i-1)} \circ F^{(i-2)}(x) > 0) \wedge (\eta_j^{(i)} \cdot F^{(i-1)}(x) \leq 0)\}$$

We define the $(+,+)$ (resp. $(+,-)$) activation of $a_{j,k}^{(i)}$ as the proportion of points of S that are also in $S_{(+,+)}$ (resp. $S_{(+,-)}$).

For example, a weight has a $(+,+)$ activation equal to 0 if and only if $S_{(+,+)} = \emptyset$. Based on $(+,+)$ and $(+,-)$ activations, we give a classification of unrecovered weights as follows:

Taxonomy of unrecovered weights		$(+,+)$	$(+,-)$
Non-effective weights		0	≥ 0
Effective weights	unreachable weights	> 0	0
	query-intensive weights	> 0	> 0

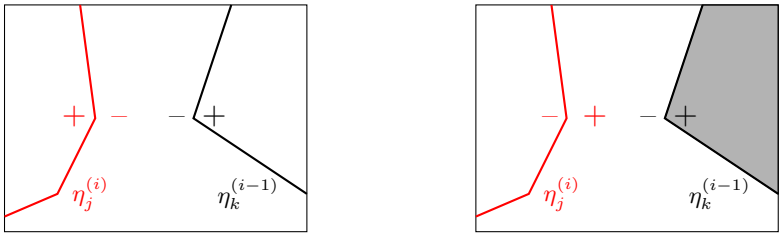
Query-intensive weights. To recover a weight, we must hit some specific hyperplanes. The budget for the attack is limited and thus we cannot grid the whole S with search lines. The targeted search, one of the few steps in the attack which we do not improve, has its limitations too. Therefore it is expected that we miss some effective weights. We refer to those weights as query-intensive weights. The number of critical points of a neuron we gather is not clearly correlated to how close it is to being always on/off. Despite this lack of correlation at the neuron level, we observe that in general increasing the number of critical points used reduces significantly the number of query intensive weights, see Table 3.

Unreachable weights. Suppose that all critical points of a neuron $\eta_j^{(i)}$ lie on the inactive side of a neuron $\eta_k^{(i-1)}$. In this case, the contribution of $\eta_k^{(i-1)}$ to the output of $\eta_j^{(i)}$ is always zero during extraction, leaving us with no information about the corresponding weight $a_{j,k}^{(i)}$. If no region of S exists where both neurons

Table 3: Taxonomy of unrecovered weights in all extracted models, under the assumption that all preceding layers have been perfectly extracted, for different attack costs. Model I having a much larger input size, we ran into memory limitations when performing the attack with 6,000 critical points.

Models	**I**	**II**		**III**	
Critical points used	3,000	3,000	6,000	3,000	6,000
non-effective weights	98	104	102	130	96
unreachable weights	5	8	8	0	0
query-intensive weights	18	11	5	25	7
Model Coverage	91.75%	74.12%	74.78%	71.35%	79.53%

are active (as illustrated in Fig. 9a), then $a_{j,k}^{(i)}$ is never activated and can be safely treated as non-effective. However, if a region does exist where both neurons are active (depicted in grey in Fig. 9b), then $a_{j,k}^{(i)}$ is effective, contributes to the output of the network for inputs in this region, and should ideally be recovered. In this case, we say that $a_{j,k}^{(i)}$ is an *unreachable* weight. Increasing the number of critical points does not diminish the number of unreachable weights, see Table 3.



(a) $a_{j,k}^{(i)}$ is non-effective. (b) $a_{j,k}^{(i)}$ is unreachable (and so effective).

Fig. 9: The arrangement of hyperplanes can cause some effective weights to not be recoverable by the current extraction strategies. We call them unreachable.

6 Experimental Results

All our experiments were conducted on an Intel Core i7-14700F CPU using networks trained on either the `MNIST` or `CIFAR-10` datasets, two widely used benchmarking datasets in computer vision. They have also been commonly used in related works [6,8,12] to evaluate model extraction methods. `MNIST` consists of 28×28 pixel grayscale images of handwritten digits, divided into ten classes ("0" through "9") [16]. In contrast, `CIFAR-10` contains 32×32 pixel RGB images of real-world objects across ten classes (e.g., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) [15]. The search space is the box centered at the origin of radius 10^3 so as to keep the effective architecture close to the original.

6.1 Comparison with the State-of-the-Art

On a $784-16^{(8)}-1$ `MNIST` network, the state-of-the-art extraction of [12] fails to recover any hidden layer beyond the third, even under the optimal assumption that all previous layers have been perfectly extracted. On the same architecture, with the same assumptions, our attack recovers at least 93% of the effective weights in each deeper layer. We further evaluate our method on other architectures and observe similar performance. The results are given in Table 4.

Table 4: Results demonstrating the weight extraction of layers deeper than those recovered by state-of-the-art methods in [12]. Each layer is attacked independently using $3,000$ critical points. Layer 9 is linear and can therefore be trivially recovered using previously issued queries.

Model	Signature Extraction	L1*	L2	L3	L4	L5	L6	L7	L8	L9	Entire Model
Ours											
`CIFAR-10`	Effective Neurons	8	8	8	8	6	8	8	6	10	-
3072-$8^{(8)}$-10	Eff.‡ Weights Recovered	100%	100%	90.32%	98.36%	89.74%	88.10%	94.92%	90.70%	100%	99.91%
(I)	Time	56m48s	1h33m	1h42m	4h43m	4h25m	5h2m	6h53m	2h23m	0.01s	27h38m
	Queries	$2^{23.55}$	$2^{26.15}$	$2^{26.15}$	$2^{27.10}$	$2^{27.04}$	$2^{27.16}$	$2^{27.56}$	$2^{26.45}$	0	$2^{29.72}$
`MNIST`	Effective Neurons	8	7	8	7	6	7	8	6	1	-
784-$8^{(8)}$-1	Eff. Weights Recovered	100%	100%	100%	100%	85.37%	90.91%	88.24%	92.86%	100%	99.71%
(II)	Time	3m24s	7m5s	7m37s	10m41s	11m46s	28m29s	20m38s	36m46s	0.01s	2h6m
	Queries	$2^{21.59}$	$2^{24.21}$	$2^{24.22}$	$2^{24.24}$	$2^{24.26}$	$2^{25.35}$	$2^{24.96}$	$2^{25.61}$	0	$2^{27.64}$
`MNIST`	Effective Neurons	16	16	16	16	16	15	16	12	1	-
784-$16^{(8)}$-1	Eff. Weights Recovered	100%	100%	100%	99.61%	99.21%	97.01%	98.64%	93.78%	100%	99.82%
(III)	Time	6m17s	26m13s	28m57s	39m33s	1h39m	2h13m	1h48m	3h56m	0.01s	11h17m
	Queries	$2^{21.60}$	$2^{24.22}$	$2^{24.24}$	$2^{24.66}$	$2^{25.63}$	$2^{26.02}$	$2^{25.78}$	$2^{26.93}$	0	$2^{28.48}$
Foerster et al. [12]											
`MNIST`	All Weights	100%	100%	-	$\geq 37.50\%^{\dagger}$	-	-	-	$\geq 0\%^{\dagger}$	100%	-
784-$16^{(8)}$-1	Time	2h46m	7m50s	-	>36 h	-	-	-	>36 h	0.01s	-
	Queries	$2^{22.36}$	$2^{19.01}$	-	-	-	-	-	-	0	-

*L stands for layer. **L1** is attacked with 500 critical points. ‡'Eff.' is short for effective.
†Foerster et al. recover 6/16 neurons on layer 4 and 0/16 neurons on layer 8, running the attack for 36 hours before stopping.

The taxonomy of these extraction results, as well as those obtained using 6,000 critical points, is given above in Table 3. These results confirm our expectations. First, despite a large S, we observe a large amount of non-effective weights, about as much as one or two hidden layers. This shows that previously used metrics in the literature, notably the numbers of total weights recovered, can be misleading. Second, increasing the attack cost has the intended effects on the quality of the extraction: query-intensive weights diminish unlike unreachable weights. By doubling the number of critical points , Model II's coverage barely changed from 74.12% to 74.78% while Model III's, which does not have any unreachable active weights, changed more substantially from 71.35% to 79.53%. For both models only 0.9% additional weights are recovered.

6.2 Results of the End-to-End Attack

We give in Table 5 the results for the end-to-end extraction. For sign recovery, we employ the extension strategy. In model III, the extraction process terminates at layer 7, where only 12 of 16 neuron signatures are recovered with low precision, thereby preventing sign extraction and the subsequent signature extraction at layer 8. Note how, contrary to the previous extraction, the proportion of effective weights recovered plunges from layer five onwards.

Table 5: End-to-end attack results using $3,000$ critical points for all but the first layer. Layer 9 is linear and can therefore be trivially recovered using previously made queries.

Model	End-to-end Extraction	L1*	L2	L3	L4	L5	L6	L7	L8	L9	Entire Model
	Effective Neurons	8	8	8	8	6	8	8	6	10	-
	Layer coverage	100%	100%	98.97%	75.11%	98.43%	95.06%	97.12%	99.06%	100%	73.47%
CIFAR-10	Eff.‡ Weights Recovered	100%	100%	90.32%	91.67%	92.86%	90.48%	94.92%	93.02%	100%	99.90%
3072-$8^{(8)}$-10	Time for Signature	1h14m	1h35m	1h38m	2h38m	5h35m	3h19m	1h41m	1h38m	0.01s	19h18m
(I)	Queries for Signature	$2^{24.56}$	$2^{26.15}$	$2^{26.15}$	$2^{26.58}$	$2^{27.4}$	$2^{26.76}$	$2^{26.18}$	$2^{26.22}$	0	$2^{29.42}$
	Time for Signs	1.69s	21s	43s	2m1s	2m25s	2m43s	2m30s	1m51s	0.01s	12m36s
	Queries for Signs	$2^{24.17}$	$2^{22.85}$	$2^{24.19}$	$2^{25.52}$	$2^{19.87}$	$2^{19.64}$	$2^{26.83}$	2^{19}	0	$2^{27.68}$
	Effective Neurons	8	7	8	7	6	7	8	6	1	-
	Layer coverage	100%	100%	100%	100%	93.75%	93.75%	76.21%	88.50%	100%	74.82%
MNIST	Eff. Weights Recovered	100%	100%	100%	100%	85.37%	87.5%	86.27%	92.86%	100%	99.68%
784-$8^{(8)}$-1	Time for Signature	12m8s	7m5s	7m46s	11m13s	13m23s	10m21s	22m7s	19m48s	0.01s	1h43m
(II)	Queries for Signature	$2^{24.36}$	$2^{24.21}$	$2^{24.22}$	$2^{24.24}$	$2^{24.25}$	$2^{24.27}$	$2^{24.97}$	$2^{24.78}$	0	$2^{27.44}$
	Time for Signs	0.19s	0.85s	47s	1.46s	1.33s	1m35s	1m24s	12s	0.01s	4m2s
	Queries for Signs	$2^{20.23}$	$2^{16.79}$	$2^{24.26}$	$2^{20.32}$	$2^{15.68}$	$2^{25.24}$	$2^{25.48}$	$2^{22.4}$	0	$2^{26.77}$
	Effective Neurons	16	16	16	16	16	15	16	-	-	-
	Layer coverage	100%	100%	100%	98.78%	88.28%	87.69%	28.09%	-	-	-
MNIST	Eff. Weights Recovered	100%	100%	100%	99.61%	91.63%	89.57%	76.39%	-	-	-
784-$16^{(8)}$-1	Time for Signature	21m35s	24m9s	24m47s	39m16s	1h59m	1h11m	2h44m	-	-	-
(III)	Queries for Signature	$2^{23.69}$	$2^{24.31}$	$2^{24.27}$	$2^{24.69}$	$2^{26.01}$	$2^{25.1}$	$2^{25.63}$	-	-	-
	Time for Signs	0.73s	7m44s	7m49s	5m32s	8m22s	7m11s	7m38s	-	-	-
	Queries for Signs	$2^{20.23}$	$2^{24.7}$	$2^{25.31}$	$2^{25.64}$	$2^{25.68}$	$2^{26.4}$	$2^{26.43}$	-	-	-

*L stands for layer. **L1** is attacked with 500 critical points.

The sudden drop in coverage in layer 4 of Model I is caused by a weight with a $(+,+)$ activation of 23% of an almost-always-on neuron (98% active). Either making more queries, or slightly reducing the search space S should increase substantially the coverage. The drop in coverage in layer 7 of Model II is caused by a particularly active unreachable weight with a $(+,+)$ activation of 11%. The (ϵ, δ) analysis of the extraction of Model I and II is given in Fig. 8. We reach a $(3.64 \times 10^{-4}, 0.26)$ extraction for Model I and a $(2.96 \times 10^{-5}, 0.25)$ extraction for Model II. The choice of δ values is guided by their respective model coverage.

7 Conclusion

In this work, we carried out an in-depth analysis of the extraction approach introduced in [8], reused in several follow-up works [6,7,12], and identified critical

limitations that prevented recovering weights beyond the third hidden layer. We then proposed improvements to the sign-recovery process by cleverly combining two methods introduced from Canales-Martínez et al. [6]. Finally, we introduced several techniques that together improve the numerical precision of the extracted signature. Taken as a whole, these advances permit, for the first time, polynomial end-to-end extraction of neural networks of significant depth and open the way to practical attacks on larger architectures.

Limitations. The main limitation we faced to going much deeper with the non end-to-end attack is the influence of deeper layers when targetting small-width networks, and the cost of the attack when targetting large-width networks. For the end-to-end attack, floating-point imprecisions remain the main limitation. Addressing these issues is a promising direction for future work.

Acknowledgements. The authors with NTU affiliation are supported by the Singapore NRF-NRFI08-2022-0013 grant. Christina Boura is partially supported by the France 2030 program under grant agreement No. ANR-22-PECY-0010 Cryptanalyse. Haolin Liu is also supported by the National Natural Science Foundation of China (No. 62372294) and the China Scholarship Council (No. 202406230017).

References

1. Google cloud platform. Website, `https://cloud.google.com/`, accessed February 12, 2025
2. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce (Nov 2001)
3. Batina, L., Bhasin, S., Jap, D., Picek, S.: {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 515–532 (2019)
4. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 2–21. Springer, Berlin, Heidelberg (Aug 1991). `https://doi.org/10.1007/3-540-38424-3_1`
5. Bishop: Bishop Book. `https://www.bishopbook.com/` (2025), accessed February 12, 2025
6. Canales-Martínez, I.A., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Satpute, N., Shamir, A.: Polynomial time cryptanalytic extraction of neural network models. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part III. LNCS, vol. 14653, pp. 3–33. Springer, Cham (May 2024). `https://doi.org/10.1007/978-3-031-58734-4_1`
7. Carlini, N., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Shamir, A.: Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. In: Fehr, S., Fouque, P.-A. (eds.) EUROCRYPT 2025, Part I. LNCS, vol. 15601, pp. 364–396. Springer, Cham (May 2025). `https://doi.org/10.1007/978-3-031-91107-1_13`
8. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS,

vol. 12172, pp. 189–218. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_7
9. Chen, Y., Dong, X., Guo, J., Shen, Y., Wang, A., Wang, X.: Hard-label cryptanalytic extraction of neural network models. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VIII. LNCS, vol. 15491, pp. 207–236. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0944-4_7
10. Daniely, A., Granot, E.: An exact poly-time membership-queries algorithm for extracting a three-layer relu network. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), https://openreview.net/forum?id=-CoNloheTs
11. Fefferman, C., et al.: Reconstructing a neural net from its output. Revista Matemática Iberoamericana **10**(3), 507–556 (1994)
12. Foerster, H., Mullins, R., Shumailov, I., Hayes, J.: Beyond slow signs in high-fidelity model extraction. Neurips **abs/2406.10011** (2024), https://api.semanticscholar.org/CorpusID:270521873
13. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) USENIX Security 2020. pp. 1345–1362. USENIX Association (Aug 2020)
14. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: 29th USENIX security symposium (USENIX Security 20). pp. 1345–1362 (2020)
15. Krizhevsky, A., Nair, V., Hinton, G.: CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/kriz/cifar.html **6**(1), 1 (2009)
16. LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L.: Handwritten digit recognition with a back-propagation network. In: Touretzky, D. (ed.) Advances in Neural Information Processing Systems. vol. 2. Morgan-Kaufmann (1989)
17. Lowd, D., Meek, C.: Adversarial learning. In: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. pp. 641–647 (2005)
18. Martinelli, F., Simsek, B., Gerstner, W., Brea, J.: Expand-and-cluster: Parameter recovery of neural networks. In: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net (2024), https://openreview.net/forum?id=3MIuPRJYwf
19. Milli, S., Schmidt, L., Dragan, A.D., Hardt, M.: Model reconstruction from model explanations. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. pp. 1–9 (2019)
20. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning (2010), https://api.semanticscholar.org/CorpusID:15539264
21. Oliynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: A survey on stealing machine learning models and defenses. ACM Computing Surveys **55**(14s), 1–41 (2023)
22. Papernot, N., Mcdaniel, P., Goodfellow, I.J., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (2016), https://api.semanticscholar.org/CorpusID:1090603
23. Reith, R.N., Schneider, T., Tkachenko, O.: Efficiently stealing your machine learning models. In: Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society. pp. 198–210 (2019)

24. Team, H.F.: Hugging face (nd), https://huggingface.co/, accessed February 12, 2025
25. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction APIs. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 601–618. USENIX Association (Aug 2016)
26. Xiao, T., Zhu, J.: Foundations of large language models (2025), https://api.semanticscholar.org/CorpusID:275570622
27. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Bach, F. and Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. pp. 448–456. (2015) https://proceedings.mlr.press/v37/ioffe15.html
28. Ba, J., Kiros, J. R., Hinton, G. E.: Layer Normalization. In: Advances in neural information processing systems 2016 deep learning symposium (2016), https://api.semanticscholar.org/CorpusID:8236317
29. Canales-Martínez, Isaac A., and David Santos. "Extracting Some Layers of Deep Neural Networks in the Hard-Label Setting." Cryptology ePrint Archive (2025).

A Example of a Small Neural Network

We illustrate here the definitions introduced in Section 2.1. Consider the $2-3-2-1$ neural network given by the following layers $A^{(i)}$, $b^{(i)}$:

$$A^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 0.5 & 2 \end{pmatrix}, b^{(1)} = \begin{pmatrix} 1 \\ -2 \\ -5 \end{pmatrix}$$

$$A^{(2)} = \begin{pmatrix} 2 & -4 & -1 \\ -3 & 4 & 5 \end{pmatrix}, b^{(2)} = \begin{pmatrix} -2 \\ -3 \end{pmatrix}$$

$$A^{(3)} = \begin{pmatrix} 1 & -1 \end{pmatrix}, b^{(3)} = \begin{pmatrix} 3 \end{pmatrix}$$

The architecture of this neural network is depicted in Fig. 10. Let the input to the neural network be $v = (x, y)$. The neurons in the first layer output $(\eta_1^{(1)}(v), \eta_1^{(2)}(v), \eta_1^{(3)}(v)) = (x + y + 1, x - y - 2, 0.5x + 2y - 5)$. Depending on (x, y), the ReLU activation function σ might set some entries to 0 before passing them to the second layer. For example, when the input is $v = (2, 2)$, we have: $F^{(1)}(v) = (5, -2, 0)$, $\sigma \circ F^{(1)}(v) = (5, 0, 0)$, $F^{(2)}(v) = 5 * (2, -3) + 0 * (-4, 4) + 0 * (-1, 5) + (-2, -3) = (8, -18)$, $\sigma \circ F^{(2)}(v) = (8, 0)$, and finally $f(v) = F^{(3)}(v) = 8 * 1 + 0 * (-1) + 3 = 11$. Since $F^{(1)}(v) = (5, -2, 0)$, it follows that v is a critical point of $\eta_3^{(1)}$.

This network can also be represented as a collection of polytopes (see Fig. 11). As can be seen in this figure, critical hyperplanes from the first layer make three straight lines, represented as dotted grey lines: $\eta_1^{(1)} : x + y + 1$, $\eta_2^{(1)} : x - y - 2$, and $\eta_3^{(1)} : 0.5x + 2y - 5$. The hyperplanes of $\eta_1^{(2)}$ and $\eta_2^{(2)}$ are shown in their respective colors, with their active side marked by $+$ and their inactive side by $-$. They indeed break on the grey lines as critical hyperplanes on deeper layers

break on all hyperplanes of previous layers: the input to $\eta_k^{(i)}$ is $F^{(i-1)}(v)$ rather than v for $i = 2, \cdots, r$.

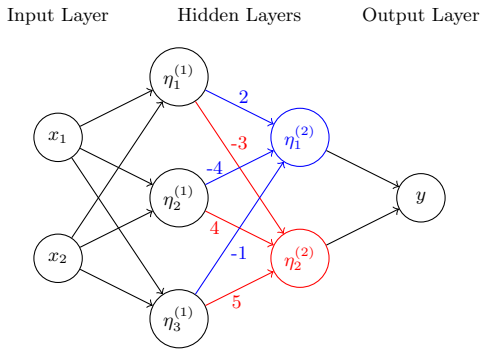


Fig. 10: Network representation of the neural network example, with the details of layer 2 highlighted in colour.

Let's compute f_v for $v = (2, 2)$ as an example:

$$\Gamma_v = A^{(3)} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} A^{(2)} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} A^{(1)} = (2.5, 4)$$

$$\gamma_v = f(v) - \Gamma_v \cdot v = 11 - 13 = 2.$$

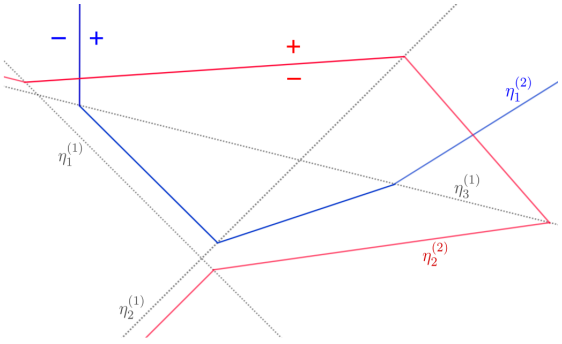


Fig. 11: The neural network example partitions the input space into 18 distinct polytopes.

B Finding Critical Points

The attack starts by searching for critical points of neurons on the layer we are targeting. Critical points are at the turning point of a ReLU and thus their left and right derivatives do not agree. We'll exploit this non-linearity to find critical points. We now describe the procedure developed in [8] and in [6]. Very intuitive illustrations are given in both papers.

Suppose we perform the search between two points a and b on the search line. First we compute the derivatives of f at a towards b and of f at b towards a, respectively m_a and m_b. If they are the same then we know that there are no critical points on the search line. If they are not the same we know that there is at least a critical point on the search line between a and b. We could directly reiterate the binary search on each half of the segment, etc. There is however a more efficient way to find the critical points. It relies on predicting where the critical point should be if it is the only one between a and b, and checking if it is indeed there.

If there is a unique critical point, we expect to find it where the derivatives cross at $x^* = a + \frac{f(b)-f(a)-m_b(b-a)}{a-b}$. Its expected value is $\hat{f}(x^*) = f(a) + m_a\frac{f(b)-f(a)-m_b(b-a)}{a-b}$. If $\hat{f}(x^*) = f(x)$ then the critical point is unique, otherwise we keep dividing the segment with dichotomy. This faster algorithm can generate pathological errors, see ([6], p.32) for more conditions to deal with them. In practice, we use [8]'s strategy.

C Details of Signature Extraction from [8]

C.1 Proof of Lemma 2

Let x be a critical point of layer $i \in \{1, \dots, r\}$ for neuron $\eta_k^{(i)}$ such that x is not a critical point of any other neuron, and $\Delta \in \mathbb{R}^{d_0}$. We have that:

$$\partial^2_\Delta f(x) = c_k^{(i)} \left| A_k^{(i)} \cdot \Gamma_x^{(i-1)} \Delta \right|$$

Proof. We take ϵ_2 sufficiently small so that $x + \epsilon_2\Delta$ and $x - \epsilon_2\Delta$ are in the neighbourhood of x. Thus $x + \epsilon_2\Delta$ and $x - \epsilon_2\Delta$ have the exact same activation pattern except for $\eta_k^{(i)}$. Let's write $G^{(i)}$ for the composition of layers starting

from i, i.e. $f = G^{(i+1)} \circ F^{(i)}$.

$$\begin{aligned}
\partial^2_{\Delta} f(x) &= \frac{1}{\epsilon_1}(f(x+\epsilon_2\Delta) + f(x-\epsilon_2\Delta) - 2f(x)) \\
&= \frac{1}{\epsilon_1}\Big(G^{(i+1)} \circ \sigma(A^{(i)}F^{(i-1)}(x+\epsilon_2\Delta) + b^{(i)}) \\
&+ G^{(i+1)} \circ \sigma(A^{(i)}F^{(i-1)}(x-\epsilon_2\Delta) + b^{(i)}) \\
&- 2\, G^{(i+1)} \circ \sigma(A^{(i)}F^{(i-1)}(x) + b^{(i)})\Big) \\
&= \Omega^{(i)}_x \Big(\sigma(A^{(i)}F^{(i-1)}(x+\epsilon_2\Delta) + b^{(i)}) \\
&+ \sigma(A^{(i)}F^{(i-1)}(x-\epsilon_2\Delta) + b^{(i)}) \\
&- 2\,\sigma(A^{(i)}F^{(i-1)}(x) + b^{(i)})\Big)
\end{aligned}$$

Where $\Omega^{(i)}_x$ is $\frac{1}{\epsilon_1}(c^{(i)}_1, \ldots, c^{(i)}_{d_i})$ for some values $c_n \in \mathbb{R}$. Let us analyze $C = \sigma(A^{(i)}F^{(i-1)}_x(x+\epsilon_2\Delta)+b^{(i)}) + \sigma(A^{(i)}F^{(i-1)}_x(x-\epsilon_2\Delta)+b^{(i)}) - 2\sigma(A^{(i)}F^{(i-1)}_x(x)+b^{(i)}) \in \mathbb{R}^{d_i \times 1}$.

The coefficients $j \neq k$ of C are zero. Indeed, since x, $x+\epsilon_2\Delta$, and $x-\epsilon_2\Delta$ belong to the same activation region except for $\eta^{(i)}_k$, the computation becomes affine, leading to zero. More precisely:

$$\begin{aligned}
C[j] \;=\; & A^{(i)}_j \cdot [\Gamma^{(i-1)}_x(x+\epsilon_2\Delta) + \gamma^{(i-1)}_x] \\
& + A^{(i)}_j \cdot [\Gamma^{(i-1)}_x(x-\epsilon_2\Delta) + \gamma^{(i-1)}_x] \\
& - A^{(i)}_j \cdot 2[\Gamma^{(i-1)}_x x + \gamma^{(i-1)}_x] \\
& = A^{(i)}_j \cdot \Gamma^{(i-1)}_x x + A^{(i)}_j \cdot \epsilon_2 \Gamma^{(i-1)}_x \Delta \\
& +\; A^{(i)}_j \cdot \Gamma^{(i-1)}_x x \;-\; A^{(i)}_j \cdot \epsilon_2 \Gamma^{(i-1)}_x \Delta - A^{(i)}_j \cdot 2\Gamma^{(i-1)}_x x \\
& = 0 \quad \text{when } \eta^{(i)}_j \text{ is active at } x.
\end{aligned}$$

$C[j] = 0 + 0 - 2 \times 0 = 0$ when $\eta^{(i)}_j$ is inactive at x.

For coefficient k, since x is a critical point of $\eta^{(i)}_k$, we have:

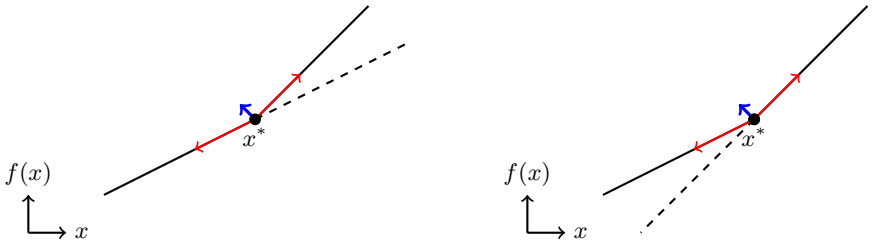
$$A^{(i)}_k \cdot F^{(i-1)}(x) \;+\; b^{(i)}_k = A^{(i)}_k \cdot [\Gamma^{(i-1)}_x x + \gamma^{(i-1)}_x] \;+\; b^{(i)}_k = 0$$

and since $A^{(i)}_k \cdot \Gamma^{(i-1)}_x \Delta$ is either positive or negative, we get:

$$C[k] = \left| A^{(i)}_k \cdot \Gamma^{(i-1)}_x \Delta \right|$$

Thus, by multiplying $\Omega^{(i)}_x$ by C, we obtain the expected result.

Intuition behind the above proof is given in Fig. 12 below.



(a) $\eta(x) \geq 0$ for $x \geq x^*$ (b) $\eta(x) \geq 0$ for $x \leq x^*$

Fig. 12: Illustration of how a neuron's activation change at a critical point creates a detectable break in the network's output, allowing the extraction of the neuron. The solid line shows the true output $f(x)$, while the dotted line shows the hypothetical output if the neuron η were always inactive. At the critical point x^*, η flips its state, creating a change in slope. Two cases can occur: (a) the neuron becomes active ($\eta(x) \geq 0$ for $x \geq x^*$), in which case the solid line bends away from the dotted line; or (b) the neuron becomes inactive ($\eta(x) \geq 0$ for $x \leq x^*$), in which case the solid line aligns with the dotted line after x^*. The slopes before and after x^* (red arrows) differ by the absolute weight of η, shown by the blue arrow. This indicates how much the neuron contributes, hence its weights, although an additional step is required to determine the correct sign of the contribution.

C.2 Correlating the Weights' Signs

We know by Lemma 2 that

$$\frac{\partial^2_{\Delta} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{\left|A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta)\right|}{\left|A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta_0)\right|} \coloneqq \frac{|\alpha|}{|\beta|}$$

Now consider on one hand,

$$\frac{\partial^2_{\Delta+\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{|\alpha + \beta|}{|\beta|}$$

And on the other,

$$\frac{\partial^2_{\Delta} f(x) + \partial^2_{\Delta_0} f(x)}{\partial^2_{\Delta_0} f(x)} = \frac{|\alpha| + |\beta|}{|\beta|}$$

The two values above are equal if and only if α and β have the same sign. This additional test allows us to remove the absolute values and correlate the sign of our result for each direction Δ we take to that of the first direction Δ_0 taken. This is why the sign extraction only has to find one sign per neuron rather than one sign per weight.

C.3 Solving for the Signature

By querying different directions $\{\Delta_m\}$ around x a critical point of $\eta_k^{(i)}$, we can obtain a set of $\{y_m\}$ such that,

$$y_m = \frac{A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta_m)}{c_k^{(i)}},$$

where $c_k^{(i)} = A_k^{(i)} \cdot (\Gamma_x^{(i-1)} \Delta_0)$. To solve for $A_k^{(i)}$, we build the following system of equations:

$$\begin{pmatrix} \Gamma_x^{(i-1)} \Delta_0 \\ \Gamma_x^{(i-1)} \Delta_1 \\ \Gamma_x^{(i-1)} \Delta_2 \\ \vdots \\ \Gamma_x^{(i-1)} \Delta_{d_{i-1}-1} \end{pmatrix} \cdot \frac{1}{c_k^{(i)}} \begin{pmatrix} a_{k,0}^{(i)} \\ a_{k,1}^{(i)} \\ a_{k,2}^{(i)} \\ \vdots \\ a_{k,d_{i-1}-1}^{(i)} \end{pmatrix} = \begin{pmatrix} 1 \\ y_1 \\ y_2 \\ \vdots \\ y_{d_{i-1}-1} \end{pmatrix}$$

When extracting the first layer, the input of the layer is also the input to the network, giving us full control over it. We can use the input basis vectors $\{e_i\}$ as directions Δ, allowing us to recover each entry of the signature independently from each equation. However, for deeper layers, we must first gather enough linearly independent conditions.

D Deeper Critical Points Merging with Components on the Target Layer

In this section, we show that a critical point x_1 of neuron $\eta_k^{(i)}$ on the target layer can merge with a critical point x_2 of a neuron in a deeper layer $i+t$ if x_2 causes $\eta_k^{(i)}$ to be the only active neuron on its layer, meaning that for some $a_k \in \mathbb{R}$,

$$\sigma \circ F^{(i)}(x_2) = (0, \dots, 0, a_k, 0, \dots, 0).$$

This indicates that a strict association between deeper critical points and incorrect components can lead to mislabelling. We note y_1 and y_2 the matrix rows extracted from x_1 and x_2 respectively. We therefore have,

$$\begin{aligned} \partial_\Delta^2 f(x_1) &= (\Gamma_{x_1}^{(i-1)} \Delta) \cdot y_1 \\ \partial_\Delta^2 f(x_2) &= (\Gamma_{x_2}^{(i+t-1)} \Delta) \cdot y_2 \end{aligned}$$

Let's write A_{x_2} the matrix $A'_{x_2} \circ I_{x_2}^{(i)} \circ A^{(i)}$, where $A'_{x_2} = A^{(i+t-1)} \circ I_{x_2}^{(i+t-2)} \circ \cdots \circ I_{x_2}^{(i+1)} \circ A^{(i+1)}$. Thus, for $c \in \mathbb{R}$,

$$\begin{aligned} (\Gamma_{x_2}^{(i+t-1)} \Delta) \cdot y_2 &= [(A_{x_2} \circ \Gamma_{x_2}^{(i-1)}) \Delta] \cdot y_2 \\ &= (\Gamma_{x_2}^{(i-1)} \Delta) \cdot (A_{x_2}^\top y_2) \\ &= (\Gamma_{x_2}^{(i-1)} \Delta) \cdot [(A^{(i)\top} \circ I_{x_2}^{(i)\top} \circ A'^\top_{x_2}) y_2] \\ &= (\Gamma_{x_2}^{(i-1)} \Delta) \cdot c y_1 \end{aligned}$$

Let's explain the last equality. Our assumption that x_2 causes $\eta_k^{(i)}$ to be the only active neuron on its layer implies that $I^{(i)}$ is diagonal with all entries zero except for a one at the k-th diagonal position. We also have that y_1 is the k-th row of the matrix $A^{(i)}$. Therefore, $(A^{(i)\top} \circ I_{x_2}^{(i)\top} \circ A_{x_2}'^{\top})y_2 = c \cdot y_1$, where c is a constant. The partial signatures y_1 and $c \cdot y_1$ extracted from x_1 and x_2 can be thus merged even though x_2 is on a deeper layer.

This case is very rare, even for small networks, but it might still happen. We encountered two such neurons in our experiments. The extracted component will have the right weights, but we might discard it because of a high τ. We check for this phenomenon on neurons with a low τ (neurons we are almost sure are on the target layer): first we finish the extraction of those neurons, we could compute their sign using the neuron wiggle and if a critical point is on the inactive side of all those neurons, we do not count it towards its component's τ.

On a side note, when this happens, the sign of all neurons on layer i can be recovered immediately with no queries or computations as we have an input x^* for which all neurons on the layer must be inactive (first order derivatives in $\mathcal{P}_{x^*}$ are equal to zero). We must have $\eta_k^{(i)} \circ F^{(i-1)}(x^*) < 0$, for all neurons on the layer, so we choose the sign of each $\eta_k^{(i)}$ to match this condition.

E Noise for Large Networks

In this section, we show that recovered partial neurons from deeper layers correspond to very different mathematical expressions from neurons on the target layer, leading to the possibility of identifying deeper critical point at no cost using statistical arguments. Indeed, we can view each recovered weight as a random variable.

Let's start with a neuron $\eta_j^{(i+1)}$ on the i+1'th layer. The equation of its hyperplane is given by:

$$A_j^{(i+1)} \cdot F^{(i)}(x) + b_j^{(i+1)} = 0$$

Which is, rewritten according to layer $i-1$,

$$A_j^{(i+1)} \cdot \left(I^{(i)} A^{(i)} F^{(i-1)}(x) + I^{(i)} b^{(i)}\right) + b_j^{(i+1)} = 0$$

$$\sum_{k=1}^{d_{i+1}} a_{j,k}^{(i+1)} \cdot \left[I^{(i)} A^{(i)} F^{(i-1)}(x)\right]_k + \mathcal{B} = 0$$

$$\sum_{k=1}^{d_{i+1}} a_{j,k}^{(i+1)} \cdot \left(\sum_{l=1}^{d_i} I_{k,k}^{(i)} \cdot a_{k,l}^{(i)} \cdot [F^{(i-1)}(x)]_l\right) + \mathcal{B} = 0$$

Expressing directly in terms of outputs of the extracted layer $i-1$:

$$\sum_{l=1}^{d_i} \left(\sum_{k=1}^{d_{i+1}} a_{j,k}^{(i+1)} \cdot I_{k,k}^{(i)} \cdot a_{k,l}^{(i)}\right) [F^{(i-1)}(x)]_l + \mathcal{B} = 0$$

and so, if it is not made inactive by a ReLU on layer $i-1$, the first weight extracted is:

$$\sum_{k=1}^{d_{i+1}} a_{j,k}^{(i+1)} \cdot I_{k,k}^{(i)} \cdot a_{k,1}^{(i)}$$

And therefore by a quick induction, the first extracted weight of a neuron $\eta_j^{(n)}$ when recovering layer i correspond to all the active paths between the weight we think we are extracting and $\eta_j^{(n)}$:

$$\sum_{k_n=1}^{d_n} a_{j,k_n}^{(n)} \cdot I_{k_n,k_n}^{(n-1)} \cdot (\sum_{k_{n-1}=1}^{d_{n-1}} a_{k_n,k_{n-1}}^{(n-1)} \cdot I_{k_{n-1},k_{n-1}}^{(n-2)}$$
$$\cdot (\dots (\sum_{k_{i+1}=1}^{d_{i+1}} a_{k_{i+2},k_{i+1}}^{(i+1)} \cdot I_{k_{i+1},k_{i+1}}^{(i)} \cdot a_{k_{i+1},1}^{(i)}) \dots))$$

As an example, suppose for simplicity that every layer has width d, that for any given critical point we expect half of the neurons to be active, and that all the weights of the network have the same mean and variance. Let's find the distribution of our extracted $w^{(n)}$ when attempting to find layer i. Let's denote the set of paths as $P = \{(k_{i+1}, k_{i+2}, \dots, k_n, j) | \forall m, k_m \in \{1, \dots, d\}\}$. Then,

$$w^{(n)} = \sum_{p \in P} a_{k_i+1,1}^{(i)} \cdot \prod_{l=i}^{n-1} (a_{p_{l-i+2},p_{l-i+1}}^{(l+1)} \cdot I_{p_{l-i+1},p_{l-i+1}}^{(l)}), \text{where } p_i \text{ is the } i\text{th entry of } p$$

Let's write L for $n-i$, and $\mathcal{C}_L$ for $\frac{d^L}{2^L}$ then we have that,

$$\begin{aligned}\mathbb{E}[w^{(n)}] &= d^L \cdot \mathbb{E}[w^{(i)}] \cdot (\mathbb{E}[w^{(i)}] \cdot \frac{1}{2})^L \\ &= \mathcal{C}_L \cdot \mathbb{E}[w^{(i)}]^{L+1}\end{aligned}$$

and,

$$\begin{aligned}\mathbb{E}[(w^{(n)})^2] = \mathbb{E}[\sum_{p \in P} \sum_{p' \in P} (a_{k_i+1,1}^{(i)})^2 \cdot \prod_{l=i}^{n-1} a_{p_{l-i+2},p_{l-i+1}}^{(l+1)} \\ \cdot I_{p_{l-i+1},p_{l-i+1}}^{(l)} \cdot a_{p'_{l-i+2},p'_{l-i+1}}^{(l+1)} \cdot I_{p'_{l-i+1},p'_{l-i+1}}^{(l)}]\end{aligned}$$

Let's separate between $p = p'$ and $p \neq p'$. For $p = p'$, we have just as above, d^L paths, $I_{p',p'}^{(l)} = I_{p,p}^{(l)}$ and $a_{p'}^{(l)} = a_p^{(l)}$. For $p \neq p'$, we have $d^{2L} - d^L$ possible path combinations, $I_{p',p'}^{(l)} \neq I_{p,p}^{(l)}$ and $a_{p'}^{(l)} \neq a_p^{(l)}$. By assumption $\mathbb{E}[I^{(l)}] = \frac{1}{2}$. We therefore get summing the $p = p'$ paths with the $p \neq p'$ paths,

$$\mathbb{E}[(w^{(n)})^2] = d^L \cdot \mathbb{E}[(w^{(i)})^2] \cdot (\mathbb{E}[(w^{(i)})^2] \cdot \frac{1}{2})^L + (d^{2L} - d^L) \cdot \mathbb{E}[w^{(i)}]^2 \cdot (\mathbb{E}[w^{(i)}]^2 \cdot \frac{1}{2^2})^L$$

Let's simplify notation and set $\mathbb{E}[w^{(i)}] = \mu$, $\mathbb{V}(w^{(i)}) = \sigma^2$ for all i using our assumption,

$$\begin{aligned}\mathbb{E}[(w^{(n)})^2] &= d^L \cdot (\sigma^2 + \mu^2) \cdot (\frac{\sigma^2 + \mu^2}{2})^L + (d^{2L} - d^L) \cdot \mu^2 \cdot (\frac{\mu}{2})^{2L} \\ &= \mathcal{C}_L \left[(\mathcal{C}_L - 1)\mu^{2L+2} + (\sigma^2 + \mu^2)^{L+1}\right]\end{aligned}$$

and therefore,

$$\begin{aligned}\mathbb{V}(w^{(n)}) &= \mathbb{E}[(w^{(n)})^2] - \mathbb{E}[w^{(n)}]^2 \\ &= \mathcal{C}_L \left[(\mathcal{C}_L - 1)\mu^{2L+2} + (\sigma^2 + \mu^2)^{L+1}\right] - \mathcal{C}_L^2 \mu^{2L+2} \\ &= \mathcal{C}_L[(\sigma^2 + \mu^2)^{L+1} - \mu^{2(L+1)}]\end{aligned}$$

Therefore for a given statistical distribution of $w^{(i)}$, the variance and the mean of weights recovered from a neuron on a deeper layer depends on how much deeper the neuron is compared to the target layer ($L = n - i$), as well as the dimension of each layer. In the attack, a partial neuron is extracted up to a scalar. Our experiments show that despite this scalar and underdetermined systems, variations in variance are enough to identify a large proportion of points on deeper layers, see Fig. 13. Being able to identify points on deeper layers leads to the speedup discussed in Section 3.2, see Table 6.

L4 points kept (%)	100	99	95	90	80	70	60	50	...	20	10
L4 points proportion (%)	11.8	14.5	16.6	17.6	19.2	20.2	21.1	22.2	...	26.5	24.5
Expected min speedup	1	1.54	2.00	2.27	2.69	2.96	3.26	3.58	...	5.10	4.37

Table 6: Effectiveness of discarding partial weights with high variance in tackling noise when extracting layer 4 in a $784-256^{(16)}-1$ network with 100,000 points. Initially, points on layer 4 represent 11.7% of all points. Imposing stricter upper bounds on the variance discards points predominantly on deeper layers, gradually increasing the proportion of points on layer 4. The expected minimum speed-up, (L4 points kept/All points kept)^2, is computed assuming the intersections and merging run in $\Omega(x^2)$ where x is the number of critical points collected. For example, if the procedure keeps 50% of the points on layer 4, while discarding 90% of all points, we would need to collect twice the number of points to run the attack with the same number of points on the target layer. Therefore we would end up merging 80% less points, meaning 5 times less. As the merging is at least quadratic, we would expect it to run at least 25 times faster. While trying to collect enough partial signatures to have a reasonable number of merges on the target layer, we ran into hardware limitations, making it impossible to compute empirical speedups (akin to [7]). Given the rank deficiencies, $\Omega(x^2)$ is far from being tight and empirical speedup factors should be much greater.

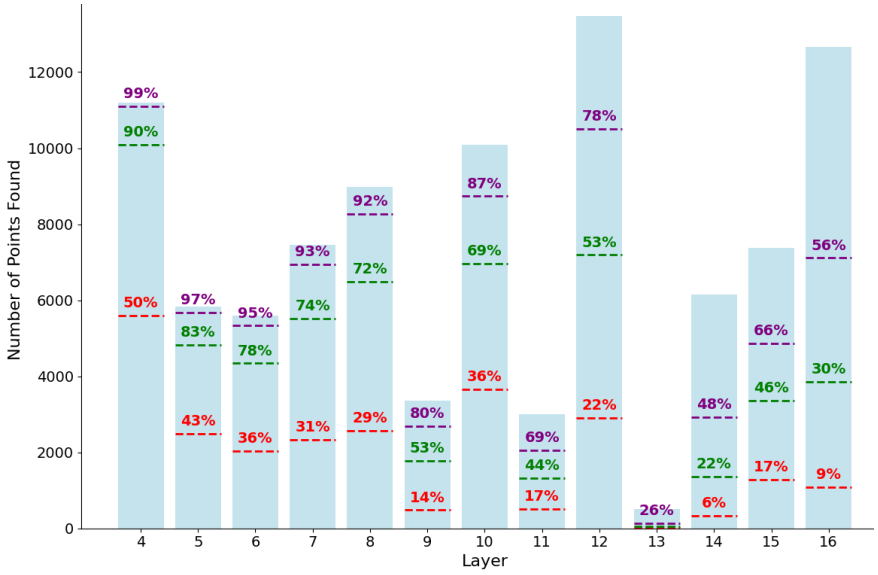


Fig. 13: For each layer, the number of critical points we discard on a $784\text{–}256^{(16)}\text{–}1$ network attacked with 100,000 points by imposing an upper bound on variance that would keep 99%, 90% and 50% of points on layer 4, respectively in purple, green and red. As implied by the mathematical analysis, critical points from deeper layers are easier to identify: the percentage of points on deeper layers kept is always lower than that of layer 4.

F Impact of Scaling Factors on Precision

In this section, we give an example showing how large disparities among scaling factors degrade the precision of signature recovery. Without normalization, the scaling factors for layer 2 in a $784 - 8^{(8)} - 1$ `MNIST` model are

$$(161.71, 32.11, 23.14, 1.07, 0.81, 36.64, 0.56, 0)$$

where values are rounded to two decimal places for simplicity. The ratio between the largest factor (161.71) and the smallest factor (0.56) is 288.77.

Next, we can identify a critical point for the 8th neuron in layer 3, for which the recovered weights are imprecise. The true weights for this neuron are

$$(0.51, 0.47, -0.55, 0.37, -0.28, 0.63, -0.23, -0.39)$$

with relative ratios (normalized to the first element)

$$(1, 0.92, -1.08, 0.73, -0.56, 1.25, -0.46, -0.77) \tag{a}$$

Without normalization, the recovered weights are

$$(-6.33\times10^{-4}, -3.04\times10^{-3}, 4.99\times10^{-3}, -7.29\times10^{-2}, 6.85\times10^{-2}, -3.46\times10^{-3}, 0, 0)$$

After multiplication by the scaling factors, they become

$$(-1.02\times10^{-1}, -9.77\times10^{-2}, 1.16\times10^{-1}, -7.79\times10^{-2}, 5.56\times10^{-2}, -1.27\times10^{-1}, 0, 0)$$

with relative ratios (normalized to the first element)

$$(1, 0.95, -1.13, 0.76, -0.54, 1.24, 0, 0) \tag{b}$$

After applying normalization, the recovered weights are

$$(-6.34\times10^{-4}, -2.95\times10^{-3}, 4.78\times10^{-3}, -7.04\times10^{-2}, 7.12\times10^{-2}, -3.51\times10^{-3}, 0, 0)$$

After multiplication by the scaling factors, they become

$$(-1.03\times10^{-1}, -9.46\times10^{-2}, 1.11\times10^{-1}, -7.52\times10^{-2}, 5.78\times10^{-2}, -1.28\times10^{-1}, 0, 0)$$

with relative ratios (normalized to the first element)

$$(1, 0.92, -1.08, 0.73, -0.56, 1.25, 0, 0) \tag{c}$$

With normalization, the recovered weights exhibit perfect relative ratios (c), matching the true ratios (a). In contrast, without normalization, the recovered weights are imprecise, as reflected in the relative ratios (b).

G Increasing the Rank of SOE

In this section, we analyze the rank of our improved SOE system, which stacks the SOE systems of multiple points. Consider n points $\{x_j\}_{j\in[1,n]}$ for sign extraction on layer i. These points share the same activation pattern on later layers, denoted as $G_{x_j}^{(i+1)}$ (simplified as $G^{(i+1)}$), but trigger many different activation patterns for $F_{x_j}^{(i-1)}$. For each point x_j, we can construct a SOE system:

$$\{G^{(i+1)}I^{(i)}A^{(i)}F_{x_j}^{(i-1)}\Delta_{k_j} = f(x_j+\Delta_{k_j}) - f(x_j)\}_{k_j}$$

by taking multiple input directions Δ_{k_j}, where our unknown is $G^{(i+1)}I^{(i)}$. Stacking these systems yields a larger SOE system:

$$G^{(i+1)}I^{(i)}\begin{pmatrix}\{A^{(i)}F_{x_1}^{(i-1)}\Delta_{k_1}\}_{k_1}\\ \{A^{(i)}F_{x_2}^{(i-1)}\Delta_{k_2}\}_{k_2}\\ \vdots\\ \{A^{(i)}F_{x_n}^{(i-1)}\Delta_{k_n}\}_{k_n}\end{pmatrix} = \begin{pmatrix}B_1\\ B_2\\ \vdots\\ B_n\end{pmatrix} = \begin{pmatrix}\{f(x_1+\Delta_{k_1})-f(x_1)\}_{k_1}\\ \{f(x_2+\Delta_{k_2})-f(x_2)\}_{k_2}\\ \vdots\\ \{f(x_n+\Delta_{k_n})-f(x_n)\}_{k_n}\end{pmatrix}$$

where each block matrix $\{A^{(i)}F_{x_j}^{(i-1)}\Delta_{k_j}\}_{k_j}$ is denoted as B_j.

In this stacked SOE system, our focus is on the rank of

$$C = \begin{pmatrix}B_1\\ B_2\\ \vdots\\ B_n\end{pmatrix}$$

as rank deficiency determines how many inactive neurons with high confidence must be recovered using neuron wiggle.

First, we give a trivial bound for $\text{rank}(C)$. Since C contains all rows of $\{B_j\}_{j\in[1,n]}$, $\text{rank}(C)$ is at least the largest rank among the individual blocks: $\max_{1\le j\le n}\text{rank}(B_j)$. On the other hand, the row space of C is spanned by the row spaces of all $\{B_j\}_{j\in[1,n]}$, so $\text{rank}(C)$ cannot exceed the sum of the ranks of these blocks: $\Sigma_{j=1}^{n}\text{rank}(B_j)$. Therefore, a loose bound for $\text{rank}(C)$ is:

$$\max_{1\le j\le n}\text{rank}(B_j)\le\text{rank}(C)\le\Sigma_{j=1}^{n}\text{rank}(B_j)$$

Next, we present a method for exact rank calculation. Since the row space of C is spanned by the row spaces of all $\{B_j\}_{j\in[1,n]}$, its rank is the sum of the ranks of the individual blocks minus the dimension of any shared row spaces, to avoid double-counting. This yields

$$\begin{aligned}\text{rank}(C)=&\Sigma_{j=1}^{n}\text{rank}(B_j)-\Sigma_{1\le j<k\le n}\dim(R(B_j)\cap R(B_k))\\&+\Sigma_{1\le j<k<l\le n}\dim(R(B_j)\cap R(B_k)\cap R(B_l))\\&-\cdots+(-1)^{n-1}\dim(R(B_1)\cap R(B_2)\cap\cdots\cap R(B_n))\end{aligned}$$

where $\dim(\cdot)$ denotes the dimension of a space, and $R(B_j)$ represents the row space spanned by B_j.