Cryptanalytic Extraction of Neural Networks with Various Activation Functions

Xiaokang Qi[1,2,3], Hao Lei[1,2,3], Longxiang Wei[1,2,3], Xiaohan Sun[1,2,3] and Meiqin Wang[1,2,3](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China
xiaokangqi@mail.sdu.edu.cn,leihao@mail.sdu.edu.cn,longxiangwei@mail.sdu.edu.cn,
xhansun@mail.sdu.edu.cn,mqwang@sdu.edu.cn

[2] State Key Laboratory of Cryptography and Digital Economy Security, Shandong University, Qingdao, 266237, China

[3] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China.

Abstract.
Originally introduced as a machine learning problem in 1991, model extraction was explicitly cast as a cryptanalytic challenge at CRYPTO 2020 and has since gained increasing prominence in this context. While early work focused on ReLU-based neural networks, recent studies have investigated model extraction in the *raw-output* setting for PReLU-based models. However, research on other activation functions remains largely unexplored. In modern deep learning, activation functions beyond ReLU are widely used, thereby creating a need for extraction techniques that can accommodate a wider variety of activation functions. This paper broadens the scope of model extraction by introducing a systematic framework for parameter recovery that is specifically tailored to different categories of activation functions. In addition to ReLU and PReLU, we investigate several other activation functions, including Leaky ReLU, HardTanh, ELU, and the Step function. To the best of our knowledge, this is the first study to explore model extraction for these activation functions and for PReLU-based models in the *hard-label* setting. We provide a detailed theoretical analysis of the properties of each activation function, propose novel attack strategies, and offer new theoretical insights. The effectiveness of our approach is demonstrated through model extraction attacks in both the raw-output and hard-label settings. Moreover, we discuss the security implications of activation functions for neural network design and explore how composite or mixed activation functions may enhance security. This work provides valuable insights into model extraction and introduces a flexible framework that may have meaningful implications for both the cryptographic and machine learning communities.

Keywords: Various Activation Functions · Cryptanalysis · Model Parameter Recovery Attack · Activation Function Security · Hard-Label.

1 Introduction

Artificial intelligence (AI) continues to advance and increasingly permeates modern technological ecosystems. Within AI, *machine learning* enables computational systems to learn from large-scale data and to make predictions or decisions on new inputs [Hea18]. *Deep neural networks* (DNNs) have emerged as a central paradigm due to their ability to solve complex tasks, including image and speech recognition, autonomous driving, natural language processing, and computer assisted medical diagnosis.

Training state-of-the-art DNNs typically requires massive datasets, careful hyperparameter tuning, and substantial computational resources [KMH+20, JYP+17]. Consequently, trained models represent valuable intellectual property, and protecting the confidentiality of neural network parameters is critical in both industrial deployments and academic settings [TZJ+16].

Problem statement and isomorphisms. Let the victim model be a neural network f_θ with parameters θ that is accessible only through a black-box oracle. An attacker may submit arbitrary inputs $X \in \mathcal{X}$ and receive feedback ζ. The goal is to collect input–output pairs (X, ζ) and to use them to recover θ. In practice, parameters are typically encoded as 64-bit floating-point numbers. It is also crucial to account for *network isomorphisms*, such as neuron permutations in fully connected layers, which admit multiple parameterizations that realize the same function. Accordingly, the practical objective is to recover a *functionally equivalent* network: if $f_\theta(X) = f_{\hat{\theta}}(X)$ for all $X \in \mathcal{X}$, then f_θ and $f_{\hat{\theta}}$ are isomorphic [CJM20, MCH+24, CDG+24, CCH+25, CDM+25, LSE+25, CMS26, JCB+20, RK20]. The difficulty of this task depends primarily on the model architecture and activation functions, as well as on the *attack scenario*.

Oracle feedback scenarios. For black-box classifiers, oracle feedback is commonly categorized into five scenarios [JCB+20]: (1) only the most likely class label ("S1"); (2) the most likely label together with its probability ("S2"); (3) the top-m labels with probabilities ("S3"); (4) all labels with probabilities ("S4"); and (5) the raw output $f_\theta(X)$ ("S5"). Among these, S5, which corresponds to the raw-output setting, conveys the most information, whereas S1, which corresponds to the hard-label setting, conveys the least.

Related work. Model extraction dates back to early work by Baum [Bau91]. More recently, Carlini et al. (CRYPTO 2020) argued that model extraction can be fruitfully viewed as *cryptanalysis* [CJM20]: DNNs interleave global linear transformations with local nonlinear operators, mirroring the structure of block ciphers, in which the nonlinear operators are public and the linear components contain the "secret" parameters. From this perspective, an adversary adaptively crafts queries to reveal the internal structure and recover the parameters.

A substantial body of work focuses on ReLU networks in the information-rich S5 setting. By traversing input lines that cross *critical points* and exploiting derivative discontinuities, one can detect internal ReLU state flips and convert these observations into linear constraints, enabling recovery in polynomial time [CJM20, MCH+24]. While the original *sign* recovery step incurred exponential complexity, subsequent work by Canales-Martínez et al. (EUROCRYPT 2024) reduced this to polynomial time by leveraging neuron fluctuations that induce linear equations [MCH+24]. In deeper networks, however, end-to-end *signature* recovery becomes fragile due to noise propagation and rank deficiencies; Liu et al. identified this bottleneck and proposed algorithmic corrections that improve the depth and stability of extraction at scale [LSE+25].

The S1 setting, which returns only the predicted label, remains a significant challenge and is representative of many deployed services. Chen et al. (ASIACRYPT 2024) first addressed S1 by exploiting decision boundary points to efficiently recover signatures. However, their approach was limited to small, shallow networks with binary outputs and did not provide a complete end-to-end attack; instead, it relied on exponential time sign guessing [CDG+24]. Subsequently, Carlini et al. (EUROCRYPT 2025) introduced a geometric analysis of the decision boundary, identifying *transition points* that align with a neuron's critical hyperplane. Although critical points are not directly observable in S1, tracing the decision boundary and identifying these alignments enables the indirect recovery of weight directions and polynomial time layerwise reconstruction; this approach has been

empirically validated on large models [CCH+25]. Building on this, Canales-Martínez and Santos [CMS26] further advanced output layer recovery by treating the output layer as an unknown linear mapping that can be recovered from linear constraints at boundary points and by analyzing contractive architectures to reduce computational overhead.

Beyond ReLU. Until recently, research on model extraction has focused almost exclusively on ReLU, leveraging its piecewise linearity to enable layerwise recovery strategies [CJM20, MCH+24, LSE+25]. Chen et al. [CDM+25] took an initial step beyond this line of work by analyzing PReLU, whose learnable negative slope preserves information from negative inputs. They explained why primitives tailored to ReLU fail and proposed techniques for recovering signatures, signs, and slopes, with extensions to top-m feedback. These developments motivate a broader and previously underexplored research direction: the systematic extraction of activation functions beyond ReLU across diverse feedback scenarios. The present work takes a further step in this direction.

The field has evolved from early work on small ReLU networks in the raw-output setting along three complementary routes:

- **(1) Deepening progress under S5**, focusing on improved signature and sign recovery for ReLU networks, with the goal of achieving end-to-end recovery in larger and more realistic architectures.
- **(2) Expanding model extraction across more settings**, with a particular emphasis on improving methods under S1, the most information-limited and challenging scenario.
- **(3) Extending beyond ReLU to a broader family of activation functions**, focusing on more complex and practically relevant activation functions and developing new strategies for recovering signatures, signs, and slopes.

These advances indicate that model extraction is moving beyond toy settings toward the security boundary of real-world deep networks.

1.1 Our Contributions

Motivated by the observation that prior work has focused primarily on ReLU, while model extraction for other activation functions remains largely unexplored, we study model extraction for networks with activation functions beyond ReLU. We present a framework for parameter recovery across activation functions, along with specialized algorithms and empirical validation in both the raw-output (S5) and hard-label (S1) settings. Our analysis and recovery methods are organized into three broad categories based on the structure of activation functions: *piecewise linear* activation functions, which are linear within each region between breakpoints; *piecewise nonlinear* activation functions, which combine linear and nonlinear segments across several regions; and *special* activation functions, such as PReLU with learnable slope parameters and the Step function[1] with discrete $\{0, 1\}$ outputs, which are essentially piecewise linear but require recovery procedures that differ from the general case. Fully nonlinear activation functions are outside the scope of this work, as we currently lack suitable analytical methods to capture and exploit their nonlinear features. For piecewise linear activation functions, we refine methods originally developed for ReLU networks; for piecewise nonlinear activation functions such as ELU, we introduce a derivative difference strategy that exploits the function's behavior in nonlinear regions; and for special activation functions like PReLU and the Step function, we develop new extraction procedures. The attack results for these activation functions are summarized

[1] In this paper, "Step function" refers to the Heaviside step function, which is used consistently throughout the text.

in Table 1. Our goal is to provide a unified treatment of recovery methods and security considerations across diverse families of activation functions. The main contributions are as follows.

- **Piecewise linear activation functions (Leaky ReLU, HardTanh).** We present the first attacks targeting neural network models with Leaky ReLU and HardTanh activation functions. By combining properties specific to each activation function with prior techniques, we demonstrate effective recovery under both S1 and S5. For Leaky ReLU in S5, we enhance sign recovery in *contractive networks*. For HardTanh in S1, we introduce an improved method that addresses the challenge of multiple critical points for a single neuron. These advancements represent a first step toward extending extraction attacks to more general piecewise linear activation functions.
- **Piecewise nonlinear activation functions (ELU).** We present the first recovery method specifically designed for piecewise nonlinear activation functions. In this context, the curvature of the decision boundary complicates classical dual points localization, and under S5, the nonconstant differences in the first derivative invalidate previous recovery methods. We propose an approach based on *activation patterns*, leveraging carefully designed patterns to enable effective recovery in S5.
- **PReLU activation function.** We present the first S1 extraction attacks for networks with PReLU activation function. We introduce a novel method for recovering the slope parameter α in S1, exploiting *decision points* and *affine transformations* on both sides of the activation boundary to recover the per-neuron slope parameter α.
- **Step activation function.** We introduce extraction attacks for networks with the Step activation function. For the Step function, we leverage the $\{0, 1\}$ code structure and the piecewise constant output patches to extract model parameters. This approach successfully recovers the target network's signature and sign information in both S1 and S5.

Table 1 reports the current best-known recovery results for different activation functions. We present recovery performance in both S5 and S1, providing a unified empirical view across multiple activation families.

Positioning and significance. Moving beyond methods centered on ReLU, we address the practical diversity of activation functions by introducing a unified classification and a framework for parameter recovery that works across various scenarios. In addition to ReLU and PReLU, we investigate Leaky ReLU, HardTanh, ELU, and the Step function. To the best of our knowledge, this is the first study to explore model extraction for these activation functions, and in particular for PReLU-based models in the S1 setting. Our analysis provides techniques for recovering parameters specific to each activation function and offers new theoretical insights applicable in both S5 and S1, thereby extending extraction beyond the ReLU case. We further examine the security implications of activation function choices and explore composite and mixed activation functions as a means of strengthening resistance to extraction attacks. Collectively, these contributions provide valuable insights into model extraction and introduce a flexible framework that may have meaningful implications for both the cryptographic and machine learning communities.

1.2 Organization

The remainder of this paper is organized as follows. Section 2 introduces the notation and definitions used throughout and provides a technical review of prior work on model extraction and activation functions. Section 3 presents extraction methods for piecewise linear activation functions. Section 4 develops methods for piecewise nonlinear activation

Table 1: Model extraction results for neural networks with different activation functions. $\max|\theta - \hat{\theta}|$: maximum coordinate-wise parameter error. ✓ successful end-to-end attacks; ✓ successful layerwise (not end-to-end) attacks; ✓ restricted layerwise attacks; ✗ no known successful attacks. *poly*: polynomial time; *exp*: exponential time. $a - b^i - c$: input dimension a, i hidden layers of width b, and output dimension c. In Leaky $\text{ReLU}_{0.1}$ and ELU_1, the subscripts denote the parameter α.

	Neural Network				**Parameter Extraction**			
Function	Setting		Architecture	Params	Signature	Sign	$\max\|\theta - \hat{\theta}\|$	**Approach**
ReLU	S5	✓	$10 - 10^2 - 1$	231	*poly*	*exp*†	$2^{-36.00}$	[CJM20]
	S1	✓	$64 - 64^4 - 10$	17290	*poly*	*poly*	$2^{-20.41}$	[CCH+25]
Leaky $\text{ReLU}_{0.1}$	S5	✓	$10 - 10^2 - 1$	231	*poly*	*poly*‡	$2^{-38.56}$	*This work*
	S1	✓	$64 - 64^4 - 10$	17290	*poly*	*poly*	$2^{-15.04}$	*This work*
HardTanh	S5	✓	$64 - 64^2 - 10$	8970	*poly*	$O(1)$	$2^{-25.57}$	*This work*
	S1	✓	$64 - 64^4 - 10$	17290	*poly*	$O(1)$	$2^{-20.62}$	*This work*
PReLU	S5	✓	$64 - 32^2 - 1$	3233	*poly*	*poly*	$2^{-27.44}$	[CDM+25]
	S1	✓	$64 - 64^4 - 10$	17546	*poly*	*poly*	$2^{-8.19}$	*This work*
ELU_1	S5	✓	$64 - 2^2 - 1$	139	*poly*	*poly*	$2^{-7.45}$	*This work*
	S1	✗	–	–	–	–	–	–
Step	S5	✓	$5 - 3 - 2 - 3$	35	*poly*	*exp*	$2^{-12.52}$	*This work*
	S1	✓	$5 - 3 - 2 - 3$	35	*poly*	*exp*	$2^{-9.76}$	*This work*

† Dedicated polynomial time sign recovery assuming real weights in [MCH+24]. ‡ Polynomial time is guaranteed only in the contractive case; otherwise, exponential time is required. The two ✓ in HardTanh indicate that not all neurons can be fully recovered, with $\max|\theta - \hat{\theta}|$ computed only over the recovered neurons. The ✓ in PReLU implies that recovery is possible only for contractive networks. The ✓ in ELU indicates that it can only handle networks with relatively small widths and depths.

functions. Section 5 addresses the extraction methods for the PReLU activation function. Section 6 discusses extraction methods for the Step activation function. Experiments discusses the experimental setup and empirical results. Finally, Section 7 concludes the paper.

2 Preliminary

2.1 Basic Definitions and Notations

This subsection presents the basic definitions and notation used throughout. We next introduce several important formal definitions and largely follow the notation adopted in previous work [CCH+25, MCH+24, CDM+25, CDG+24, CJM20].

A deep neural network is a layered computational model that maps an input vector to an output through successive linear transformations interleaved with nonlinear activation functions. In S1, a deep neural network takes an input x and processes it layer by layer to produce a final hard-label prediction (e.g., "dog"), as illustrated in Figure 1.

Definition 1. (Model architecture) A deep neural network (DNN) is a neural network with multiple layers of interconnected neurons. Its architecture determines the form of the function f and is specified by:

(a) the number of layers, i.e., how many transformations the input undergoes as it passes through the network (typically including an input layer, one or more hidden layers, and an output layer);

(b) the dimension d_i of each layer, where $\{d_i\}_{i=0}^{k}$ denotes the number of neurons in the i-th layer, with d_0 as the input dimension and d_k as the output dimension.

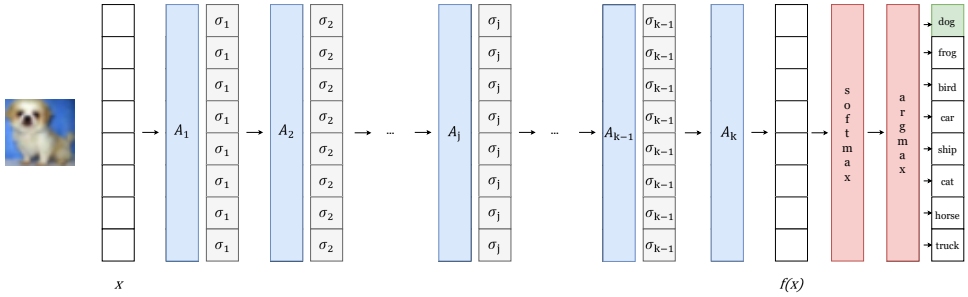


Figure 1: Fully connected ReLU neural network.

In this paper, we work with fully connected deep neural networks, where the j-th layer computes $f_j(x) = A^{(j)}x + b^{(j)}$ followed by an activation function. We use the term *signature* to refer to the exact weights and biases of each neuron, and *sign* to refer to the sign pattern of these parameters.

Definition 2. (Activation layer) The k-th activation layer σ_k of a neural network consists of d_k nonlinear activation functions $\sigma_{k,i}$, for $i \in \{1, \ldots, d_k\}$.

In this paper, we focus on a range of activation functions whose mathematical properties are discussed in detail in Section 3, Section 4, Section 5 and Section 6.

For brevity, we omit several standard definitions, as our framework closely follows prior work; the full statements can be found in Appendix A.1. Since we explore activation functions beyond ReLU, we must extend some of these definitions and present their generalized forms below. In the S5 setting, Carlini et al. [CJM20] showed that model extraction can exploit special inputs at which a neuron's state changes. By moving along lines in the input space that cross such *critical points*, they observed jumps in the derivative, which can then be converted into linear constraints on the weights. The set of all inputs that place a given neuron exactly at its switching threshold forms the corresponding *critical hyperplane*. We now formalize these notions.

Definition 3. (Critical point) A critical point is an input $x \in \mathbb{R}^{d_0}$ that makes the input to the activation function of a neuron equal to a threshold value t. Formally, when $V(\eta; x) = t$, we say that neuron η is at a critical point and that the input x is a *witness* to the critical point of neuron η, denoted

$$x \in W_t(\eta),$$

where $W_t(\eta)$ denotes the set of witnesses to the critical point of neuron η corresponding to the threshold value t. For example, if

$$A_k^i \sigma_{k-1}(Y_{k-1}) + b_k^i = t,$$

then x is a witness to a critical point of the i-th neuron in layer k. Here, Y_{k-1} is the pre-activation of the $(k-1)$-th layer, i.e., the output after passing through the previous layers, excluding the activation function.

Definition 4. (Critical hyperplane) The critical hyperplane of a neuron η at threshold t is the set of all inputs $x \in \mathbb{R}^{d_0}$ for which the pre-activation equals t:

$$\mathcal{H}_\eta(t) = \{x \in \mathbb{R}^{d_0} \mid V(\eta; x) = t\}.$$

When $x \in \mathcal{H}_\eta(t)$, neuron η is at a critical point corresponding to the threshold value t.

In the S1 setting, only the predicted label is observable, so internal critical points cannot be accessed directly. Carlini et al. [CCH+25] instead work with the decision boundary: by tracing points where the label changes (decision points) and following the induced decision surface (decision hyperplane), they identify intersections with a neuron's critical hyperplane. These dual points, which lie in a low dimensional dual space, reveal the corresponding weight directions. We next formalize these notions.

Definition 5. (Decision point) A decision point is an input $x \in \mathbb{R}^{d_0}$ at which the predicted class of the neural network changes. More precisely, it lies on the boundary between two regions of the input space that are assigned different classes. Formally, a point x is a decision point if

$$z(f(x+\epsilon)) \neq z(f(x-\epsilon))$$

for some sufficiently small $\epsilon > 0$, where z denotes the predicted label.

Definition 6. (Decision hyperplane) A decision hyperplane is the set of all decision points forming the boundary between two classification regions in the input space. It separates regions of the input space on which the network's predicted class differs between two labels.

Definition 7. (Dual point) A dual point is an input $x \in \mathbb{R}^{d_0}$ that lies simultaneously on a critical hyperplane and a decision hyperplane. Equivalently, x is a dual point if it is a critical point of some neuron; that is, $V(\eta; x) = t$ for some threshold t, and at the same time, it lies on the decision boundary where the network's predicted class changes.

Definition 8. (Dual space) The dual space D_η of a neuron η is the locally linear $(d_0 - 2)$-dimensional subspace obtained as the intersection of the $(d_0 - 1)$-dimensional critical hyperplane of η and the $(d_0 - 1)$-dimensional decision hyperplane in a neighborhood of a dual point x_{dual}.

A **critical hyperplane** determines the state of a neuron. When the input crosses the critical hyperplane, the neuron's state changes. It partitions the input space into regions, where each region corresponds to a distinct affine transformation. The properties of these regions depend on the neuron's state.

A **decision hyperplane** determines the final classification. When the input crosses the decision hyperplane, the predicted class changes. It partitions the input space into regions, where each region is assigned a single output label, and different regions correspond to different labels.

In fact, the critical hyperplane and the decision hyperplane are independent, as illustrated in Figure 2. Each partition divides the input space into regions according to a different criterion. The critical hyperplane divides the input space so that every region corresponds to a single affine transformation, whereas the decision hyperplane divides the input space so that every region is associated with a single predicted class label.

2.2 Overview of Prior Work

Although our methods differ from those of Carlini et al., we analyze them within the same general framework. To clarify the underlying principles and make our contributions easier to follow, we briefly review prior attacks in the S5 and S1 settings [CJM20, CCH+25].

2.2.1 Overview of the S5 Attack

We first recall the signature and sign recovery methods for ReLU networks introduced by Carlini et al.; full details are given in [CJM20, Section 4].

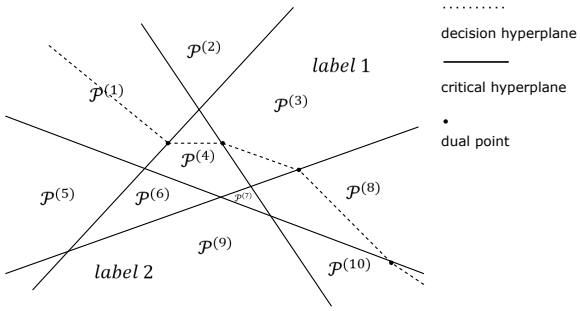


Figure 2: The decision hyperplane and the critical hyperplane partition the input space into distinct regions.

Signature recovery. We recover the first-layer signature by using differences of first-order derivatives along orthogonal directions (see Figure 3). For any orthogonal direction e_i and small $\varepsilon > 0$, define

$$\alpha_i^+ = \left.\frac{\partial f(x)}{\partial e_i}\right|_{x=x^*+\varepsilon e_i}, \qquad \alpha_i^- = \left.\frac{\partial f(x)}{\partial e_i}\right|_{x=x^*-\varepsilon e_i}.$$

For ReLU, the activation has slopes 1 and 0 on the two sides of the critical point. Therefore, for the neuron j that switches at x^*, we have

$$\alpha_i^+ - \alpha_i^- = A_{j,i}^{(1)}\, A^{(2)} \quad \text{or} \quad \alpha_i^- - \alpha_i^+ = A_{j,i}^{(1)}\, A^{(2)}.$$

For any other neuron $\eta' \neq \eta$ whose pre-activation $V(\eta'; x)$ does not cross its own critical hyperplane at x^*, the contribution to $\alpha_i^+ - \alpha_i^-$ is either $1 - 1 = 0$ or $0 - 0 = 0$, so interference from other neurons cancels out. Consequently,

$$\frac{\alpha_k^+ - \alpha_k^-}{\alpha_i^+ - \alpha_i^-} = \frac{A_{j,k}^{(1)}}{A_{j,i}^{(1)}},$$

which yields the signed relative weights of neuron j in the first layer. Once the direction of $A_j^{(1)}$ is known, the bias is recovered via

$$b_j^{(1)} = -\hat{A}_j^{(1)} \cdot x^*,$$

since the pre-activation at the critical point x^* is zero. For deeper networks, this approach is extended by using second-order derivatives: one samples directions δ_i and evaluates mixed second derivatives

$$y_i = \left.\frac{\partial^2 f(x)}{\partial \delta_1\, \partial \delta_i}\right|_{x=x^*}$$

and solves the resulting linear system to recover the corresponding weight vectors.

Sign recovery. In the contractive case, we consider a deep fully connected ReLU network with layer widths

$$d_k \leq d_{k-1} \leq \cdots \leq d_1 \leq d_0,$$

so that each linear map $A^{(i)}$ is surjective. Because ReLU maps negative pre-activations to zero, any preimage must lie in the nonnegative orthant. Given a hidden vector $h_i \in \mathbb{R}^{d_i}$, one searches for $h_{i-1} \in \mathbb{R}^{d_{i-1}}$ such that

$$f_{i-1}(h_{i-1}) = h_i \quad \text{and} \quad h_{i-1} \geq 0.$$

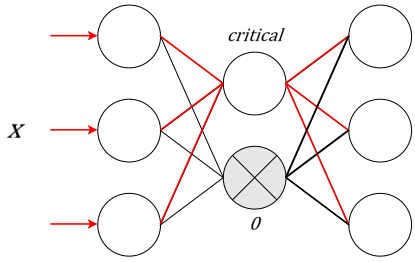


Figure 3: Recovering the weights through differences of first-order derivatives.

In the single layer case, we consider a neuron η_i and a witness point x_i such that

$$h = f_1(x_i)$$

has its i-th coordinate equal to zero. Since $A^{(1)}$ is a contraction, the affine map $\hat{f}_1$ admits preimages x^+ and x^- for

$$\hat{f}_1(x^+) = h + e_i \quad \text{and} \quad \hat{f}_1(x^-) = h - e_i,$$

where e_i is the i-th standard basis vector in $\mathbb{R}^{d_1}$. Comparing the outputs $f(x^+)$ and $f(x^-)$ with $f(x_i)$ then determines whether the current sign of the i-th row of $A^{(1)}$ is correct or must be flipped. If the recursive preimage construction succeeds for generic choices of h_j, it becomes possible to inductively recover a point $x \in \mathcal{X}$ such that $f_{1..j}(x) = h_j$; in this case, the network is treated as contractive for the purposes of sign recovery.

In the non-contractive case, sign recovery relies on exhaustive search over the 2^{d_j} possible sign patterns in layer j, which requires exponential work in the layer width.

2.2.2 Overview of the S1 Attack

We next recall the geometric attack in the S1 setting proposed by Carlini et al.; full details are given in [CCH+25, Sections 5 and 6].

Signature recovery. In the S1 setting, only the predicted label is observable, so internal critical points cannot be accessed directly; instead, it is necessary to work with the decision boundary. By tracing points where the predicted label changes and following the resulting decision hyperplane, local bends caused by intersections with a neuron's critical hyperplane can be detected, as the normal vectors of the decision surfaces on either side of the critical hyperplane differ. Such intersection points are referred to as dual points. Around each dual point, a small neighborhood can be constructed on the decision boundary that lies simultaneously on the decision surface and on the neuron's critical hyperplane, which we refer to as the dual space D_η of that neuron. If two dual spaces D_{η_i} and D_{η_j} correspond to the same neuron η, their intersection spans the neuron's critical hyperplane H_η.

$$\operatorname{span}\big(D_{\eta_i} \cap D_{\eta_j}\big) = H_\eta,$$

and the normal vector of H_η yields the direction of the corresponding weight vector. Dual spaces arising from different neurons generically do not share a common hyperplane, so by aggregating sufficiently many dual spaces, each neuron's critical hyperplane can be recovered, and thus its weight direction.

Sign recovery. The Neuron Wiggle Technique of Carlini et al. is based on applying a small, carefully chosen perturbation (a "neuron wiggle") near a dual point and observing

how this perturbation affects activations in the target layer. The perturbation is tuned so that the target neuron changes by an amount ε, while all other neurons in the same layer change only by approximately $\pm\varepsilon/\sqrt{d_i}$, where d_i denotes the width of the target layer. More concretely, for a target layer of width d_i, the induced changes on the two sides satisfy the following relations:

$$s_{\text{on}} \propto \|v_{\text{on}}\|_2 \approx \left\| \left(\pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}}, \ldots, \varepsilon, \ldots, \pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}} \right) \right\|_2, \tag{1}$$

$$s_{\text{off}} \propto \|v_{\text{off}}\|_2 \approx \left\| \left(\pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}}, \ldots, 0, \ldots, \pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}} \right) \right\|_2. \tag{2}$$

In the above expressions, s_{on} (s_{off}) refers to the speed with which the average future neurons change their value on the on-side (off-side) of the target neuron, and $\|v_{\text{on}}\|_2$ ($\|v_{\text{off}}\|_2$) refers to the output norm in the corresponding states. At a dual point x_{dual} on a neuron's critical hyperplane, the method involves moving a small distance along the boundary on both sides and measuring the shortest distances, Δ_{on} and Δ_{off}, until a neuron in a later layer flips its activation. The side on which downstream activations change more rapidly tends to be the neuron's positive side; therefore, it typically holds that

$$\Delta_{\text{off}} > \Delta_{\text{on}}.$$

The side with the shorter distance is labeled as positive, while the other side is labeled as negative. By repeating this test at multiple dual points and aggregating the outcomes, an estimate of the sign of each neuron's weight vector can be obtained. This behavior is illustrated in Figure 4.

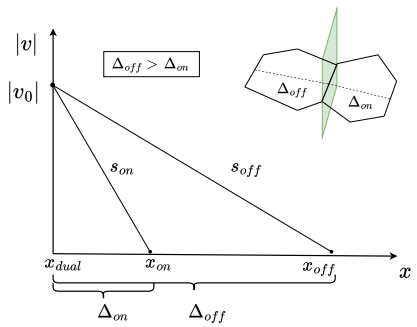


Figure 4: A higher average speed on the on-side s_{on} compared to the off-side s_{off} results in a measurable difference, where $\Delta_{\text{off}} > \Delta_{\text{on}}$.

2.3 Adversarial Goals and Assumptions

Adversarial goals. As discussed in Section 1, given a collection of input-output pairs (x, ζ), the adversary's goal is to recover a set of parameters $\hat{\theta}$ such that $f_{\hat{\theta}}(x)$ matches $f_\theta(x)$ as closely as possible according to some predefined metrics, effectively recovering a functionally equivalent model.

Assumptions. Building on the setting introduced by Carlini et al. [CJM20], we largely follow the standard assumptions on the oracle O and the adversary's knowledge that are now common in the field. These assumptions, together with the specific variations considered in this work, are described in detail in Appendix A.2. In this paper, we systematically extend this framework along three axes: we relax the assumption of full raw-output access by considering both S5 and S1; we move beyond scalar outputs by

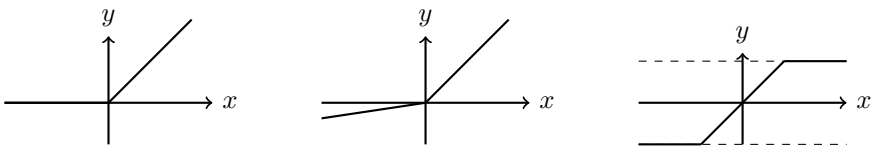
treating both binary and multiclass classification; and we generalize from ReLU to a variety of activation functions. This allows us to systematically analyze the security of activation function choices in neural network design and refine certain theoretical aspects of model extraction.

3 Piecewise Linear Activation Functions

We organize our analysis and recovery methods into three broad categories based on the structure of activation functions: *piecewise linear* activation functions, which are linear within each region between breakpoints; *piecewise nonlinear* activation functions, which combine linear and nonlinear segments; and *special* activation functions, such as PReLU with learnable slope parameters and the Step function with discrete $\{0, 1\}$ outputs, whose recovery procedures differ from the general case. These three categories are treated in Section 3, Section 4, Section 5, and Section 6, respectively.

In this section, we focus on piecewise linear activation functions, which are linear within each segment and exhibit jumps in the first derivative at segment boundaries; these functions typically share many properties with ReLU. ReLU is the prototypical example and underlies much of the existing work on model extraction. Our analysis of other piecewise linear activation functions follows the same framework as in the ReLU case, and we refer the reader to Section 2.2 for a detailed overview of the ReLU setting. Figure 5 illustrates typical profiles of such activations.

$$\mathrm{ReLU}(x) = \begin{cases} x, & x \geq 0, \\ 0, & x < 0, \end{cases} \quad \mathrm{LeakyReLU}_\alpha(x) = \begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0, \end{cases} \quad \mathrm{HardTanh}(x) = \begin{cases} -1, & x < -1, \\ x, & -1 \leq x \leq 1, \\ 1, & x > 1. \end{cases}$$



(a) ReLU (b) Leaky ReLU (c) HardTanh

Figure 5: Piecewise linear activations: ReLU, Leaky ReLU, and HardTanh.

3.1 Model Extraction in the Leaky ReLU Setting

In this subsection, we analyze the Leaky ReLU activation function and present its corresponding attack method. Leaky ReLU employs a small slope on the negative side, which prevents neurons from shutting down and helps preserve gradients [MHN+13, XWCL15]. We define

$$V(\eta; x) = w_\eta^\top x + b_\eta, \qquad h_\eta(x) = \sigma\big(V(\eta; x)\big), \qquad \mathcal{H}_\eta(t) = \{x \in \mathbb{R}^{d_0} : V(\eta; x) = t\}.$$

Let $x^\star$ satisfy $V(\eta; x^\star) = 0$, i.e., $x^\star \in \mathcal{H}_\eta(0)$. Consider an infinitesimal displacement across $\mathcal{H}_\eta(0)$ in the normal direction of this hyperplane:

$$\Delta x = \varepsilon\, w_\eta, \quad \varepsilon \in \mathbb{R},\ |\varepsilon| \ll 1.$$

We then consider the directional derivatives of h_η along w_η at $x^\star$ from the positive and negative sides:

$$D_\eta^+ = \lim_{\varepsilon \downarrow 0} \frac{h_\eta(x^\star + \varepsilon w_\eta) - h_\eta(x^\star)}{\varepsilon}, \qquad D_\eta^- = \lim_{\varepsilon \uparrow 0} \frac{h_\eta(x^\star + \varepsilon w_\eta) - h_\eta(x^\star)}{\varepsilon}.$$

For Leaky ReLU, we obtain

$$D_\eta^+ - D_\eta^- = (1-\alpha)\,\|w_\eta\|_2.$$

3.1.1 Model Extraction in S5

Signature recovery. In the Leaky ReLU case, we use the same signature recovery procedure as in Section 2.2.1, based on differences of first order derivatives along orthogonal directions. Using the notation α_i^+ and α_i^- from the ReLU setting, we only need to adjust the slopes of the activation function. The Leaky ReLU activation has slopes 1 and α on the two sides of the critical point, so for the neuron j that switches at x^* we obtain

$$\alpha_i^+ - \alpha_i^- = (1-\alpha)\,A_{j,i}^{(1)}\,A^{(2)} \quad \text{or} \quad \alpha_i^- - \alpha_i^+ = (1-\alpha)\,A_{j,i}^{(1)}\,A^{(2)}.$$

For any other neuron $\eta' \neq \eta$ whose pre-activation $V(\eta'; x)$ does not cross its own critical hyperplane at x^*, the corresponding contribution to $\alpha_i^+ - \alpha_i^-$ is either $1-1=0$ or $\alpha-\alpha=0$, so interference from other neurons cancels out exactly as in the ReLU case. The additional factor $(1-\alpha)$ is common to all coordinates; hence, the ratios

$$\frac{\alpha_k^+ - \alpha_k^-}{\alpha_i^+ - \alpha_i^-} = \frac{A_{j,k}^{(1)}}{A_{j,i}^{(1)}}$$

still reveal the signed relative weights of neuron j.

Sign recovery. Existing sign recovery methods have limitations. Carlini et al. [CJM20] proposed a polynomial time sign recovery procedure for contractive ReLU networks; however, this method relies on strong contraction assumptions and is primarily of theoretical interest. In their experimental evaluation, the polynomial time sign recovery method is effective only for the first layer, where contraction-based recovery can be applied; for deeper layers, because ReLU discards all information on the negative side, sign recovery requires an exhaustive search over sign patterns at critical points, resulting in exponential running time. The polynomial time sign recovery algorithm for ReLU proposed in [MCH+24] is also not applicable to Leaky ReLU, since its nonzero negative branch violates the assumptions underlying the neuron toggling method. More generally, activation functions with nonzero negative halves cannot use this technique.

To overcome these limitations, we introduce an improved sign recovery method for Leaky ReLU that is effective for contractive networks. In our experiments, this method is highly efficient since Leaky ReLU preserves information on the negative side, unlike ReLU.

1-deep case. As in the ReLU case, we first consider a single neuron η_i with a critical point x_i. Let $h = f_1(x_i)$, where at least one component of h is zero. Since $A^{(1)}$ is a contraction, we can find preimages x^+ and x^- such that

$$\hat{f}_1(x^+) = h + e_i \quad \text{and} \quad \hat{f}_1(x^-) = h - e_i,$$

where e_i is the i-th standard basis vector in $\mathbb{R}^{d_1}$. We then compute

$$\frac{|f(x^+) - f(x_i)|}{|f(x^-) - f(x_i)|} = \frac{1}{\alpha} \quad \text{or} \quad \frac{|f(x^+) - f(x_i)|}{|f(x^-) - f(x_i)|} = \alpha.$$

One of these equalities holds because, when $h_i = 0$, the i-th component of $\sigma(h - e_i)$ is αe_i, whereas the i-th component of $\sigma(h + e_i)$ is e_i. If the first equality holds, the extracted sign is correct; if the second holds, we flip the sign (including $b_i^{(1)}$). Since piecewise linear activation functions have slope discontinuities at critical points, the same method efficiently recovers signs for all such functions.

k-deep case. For multilayer networks, unlike in the ReLU setting, networks with Leaky ReLU activations do not require the constraint $h_{i-1} \geq 0$, since information on the negative

side is preserved. As before, we work in the contractive setting, so that each linear map $A^{(i)}$ is surjective and every hidden vector admits a preimage. To compute preimages at layer i, we first invert the activation on the negative coordinates. Given a hidden vector h_i, we define an intermediate vector $\hat{h}_i$ by

$$(\hat{h}_i)_\ell = \begin{cases} h_{i,\ell}, & h_{i,\ell} \geq 0, \\ \dfrac{1}{\alpha}\, h_{i,\ell}, & h_{i,\ell} < 0, \end{cases}$$

which follows from the Leaky ReLU rule $\sigma(t) = t$ for $t \geq 0$ and $\sigma(t) = \alpha t$ for $t < 0$. We then solve

$$f_{i-1}(h_{i-1}) = \hat{h}_i$$

for a preimage h_{i-1}. Iterating this construction from layer j down to layer 0 yields a point $x \in \mathcal{X}$ such that $f_{1..j}(x) = h_j$. Combined with the ratio based test in the single layer case, this additional rescaling step enables complete sign recovery for contractive Leaky ReLU networks.

3.1.2 Model Extraction in S1

For signature recovery, we adopt the same procedure as in Section 2.2.2. For piecewise linear activation functions such as ReLU and Leaky ReLU, critical points induce critical hyperplanes that partition the input space into linear regions. Dual points can be identified, and the corresponding hyperplanes can be fully recovered using existing techniques. The normal vector of a critical hyperplane then yields the signature of the associated neuron. In the ReLU case, many coordinates of a dual point may be jointly inactive, so the consistency test for "same neuron" should check whether the resulting linear system attains the *maximum rank allowed* by these shared inactive coordinates rather than full rank. For Leaky ReLU, since the negative side does not vanish, full rank can be tested directly.

For sign recovery, the Neuron Wiggle Technique remains applicable. However, the nonzero slope on the negative side attenuates the amplitude of the oscillation, so the theoretical accuracy of sign recovery decreases as the negative side slope parameter increases; the larger this parameter is, the less effective the recovery becomes. More concretely, for a target layer with width d_i, the expected changes on the two sides satisfy:

$$s_{\text{on}} \propto \|v_{\text{on}}\| \approx \left\| \left(\pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}}, \ldots, \varepsilon, \ldots, \pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}} \right) \right\|, \tag{3}$$

$$s_{\text{off}} \propto \|v_{\text{off}}\| \approx \left\| \left(\pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}}, \ldots, \alpha\varepsilon, \ldots, \pm\frac{\varepsilon}{\sqrt{d_i}}, \pm\frac{\varepsilon}{\sqrt{d_i}} \right) \right\|. \tag{4}$$

By (3) and (4), we have the following: $\dfrac{\|v_{\text{on}}\|}{\|v_{\text{off}}\|} = \dfrac{\sqrt{(1-\frac{1}{d_i})+1}}{\sqrt{(1-\frac{1}{d_i})+\alpha^2}} \leq \dfrac{\sqrt{(1-\frac{1}{d_i})+1}}{\sqrt{1-\frac{1}{d_i}}} \approx \sqrt{2}$

That is, the ratio between the two speed estimates above is approximately $\sqrt{2} \approx 1.4$.

3.2 Model Extraction in the HardTanh Setting

In this subsection, we analyze the HardTanh activation function and present its corresponding attack method. HardTanh is a three-segment piecewise linear approximation to tanh with saturation, which yields bounded outputs and stable gradients under amplitude constraints [Hea18]. Being an odd function, HardTanh satisfies $\text{HardTanh}(-x) = -\text{HardTanh}(x)$; consequently, for networks with HardTanh as the activation function, we have $f_{1..j}(x) = -f_{1..j}(-x)$. For a neuron η, the pre-activations thresholds

$$V(\eta; x) = -1 \quad \text{and} \quad V(\eta; x) = 1$$

induce two critical hyperplanes:

$$\mathcal{H}_\eta(-1) = \{x \in \mathbb{R}^{d_0} : w_\eta^\top x + b_\eta = -1\}, \qquad \mathcal{H}_\eta(1) = \{x \in \mathbb{R}^{d_0} : w_\eta^\top x + b_\eta = 1\}.$$

These hyperplanes are parallel and share the same normal vector w_η. Observing several such parallel hyperplanes yields multiple affine constraints of the form

$$w_\eta^\top x + b_\eta = t, \quad t \in \{-1, 1\},$$

which collectively stabilize the recovery of both w_η and b_η for neuron η.

For the odd activation HardTanh, the critical points of a neuron are symmetric with respect to the origin. Specifically, any critical point with $V(\eta; x) = 1$ has a symmetric counterpart with $V(\eta; x') = -1$, which lies on the parallel critical hyperplane $\mathcal{H}_\eta(-1)$. In other words, every critical point on $\mathcal{H}_\eta(1)$ has a symmetric counterpart on $\mathcal{H}_\eta(-1)$ for the same neuron η, with the threshold reversed from 1 to -1.

Sign recovery. For odd activation functions, relative sign differences between neurons in a layer can be absorbed into the weights of the neurons in the subsequent layer. In the case of HardTanh, sign recovery is therefore not required, and the effective complexity of sign recovery is $O(1)$. Since HardTanh is an odd function with central symmetry, flipping the sign of a neuron's weight vector does not change the absolute value of its output. As illustrated in Figure 6, when the relative signs between neurons η_i, η_j, η_m, and η_n are not recovered, it suffices to adjust the entries of an incoming weight vector in the subsequent layer at the positions corresponding to i, j, m, and n, effectively transferring the sign ambiguity to a single neuron in the next layer. More generally, for activation functions with odd or even symmetry, sign recovery is not an issue: for odd functions, any global sign choice per neuron can be reparameterized into the subsequent layer, while for even functions, sign flips leave the output unchanged and can therefore be ignored altogether.

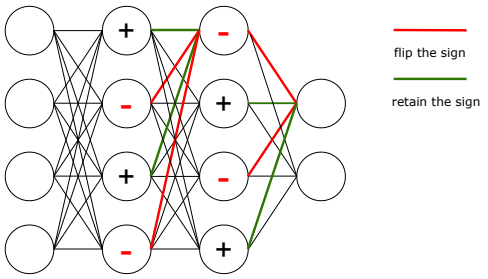


Figure 6: For odd activation functions, the sign of different neurons can be equivalently represented as the sign of the same neuron in the subsequent layer.

In S5, it is natural to attempt to recover all information by reusing the machinery of the S5 attack reviewed in Section 2.2.1; however, for neural networks with HardTanh as the activation function, this approach fails in practice. The activation range of HardTanh is limited to $[-1, 1]$, so information outside this range is discarded when calculating the weights using derivatives. In particular, many coordinates of deeper critical points, which are originally identified using derivative based methods, lie in the saturation region. The original method relies on second order derivatives of the form

$$y_i = \left.\frac{\partial^2 f(x)}{\partial \delta_1 \partial \delta_i}\right|_{x=x^*}, \qquad i = 1, \ldots, d_{j+1}.$$

From these quantities, we construct a system of linear equations by setting $h_i = \sigma(f_{1..j-1}(x + \delta_i))$ and solving for a vector w such that $h_i \cdot w = y_i$. However, in the HardTanh case,

most components of $\sigma(f_{1..j-1}(x+\delta_i))$ lie in the saturation region and therefore provide no useful variation, making it impossible to extract meaningful information from the resulting system. As a consequence, when using a method similar to that in Section 2.2.1 to handle HardTanh, the weights cannot be effectively recovered. In our experiments, this approach successfully recovers only the weights of the first layer, where $\sigma(f_{1..j-1}(x+\delta_i))$ coincides with δ_i before HardTanh is applied. For subsequent layers, the available critical points do not reveal sufficiently rich information for weight recovery. Although some correct partial signatures can be recovered, complete signatures cannot be obtained.

In S1, by contrast, the dual point based methods reviewed in Section 2.2.2 cannot be directly applied. A natural idea is to use dual points x^* to recover the dual space and the corresponding critical hyperplane, but the point selection procedure fails to distinguish between the two critical hyperplanes $\mathcal{H}_{\eta_i}(-1)$ and $\mathcal{H}_{\eta_i}(1)$ associated with the same neuron. In particular, it cannot determine whether a dual point $x^* \in \mathbb{R}^{d_0}$ lies in $\mathcal{H}_{\eta_i}(-1)$ or in $\mathcal{H}_{\eta_i}(1)$. In our experiments, this leads to a complete failure of the SVD step: the sampled points do not sufficiently explore the dual space, resulting in poor fitting due to a "thickness" effect between the two parallel hyperplanes. Consequently, no neuron in any layer is recovered correctly. To address this issue, we introduce a method that separates these two types of critical points and treats them independently.

Algorithm 1: Separation of Critical Points via SVD

Input : Dual points $P = \{p_1, \ldots, p_n\}$
Output : Normal vector δ of the critical hyperplane

Compute a center point $p_{\text{center}} \in P$;
Construct the difference matrix $A \in \mathbb{R}^{n \times d_0}$ with rows $A_i = p_i - p_{\text{center}}$;
Compute the SVD $A = U\Sigma V^\top$;
Let δ be the right singular vector corresponding to the smallest singular value;
Initialize $S_+ \leftarrow \emptyset$ and $S_- \leftarrow \emptyset$;
foreach $p_i \in P$ **do**
    Compute $t_i \leftarrow \langle p_i - p_{\text{center}}, \delta \rangle$;
    if $t_i > 0$ **then**
        $S_+ \leftarrow S_+ \cup \{p_i\}$;
    else
        $S_- \leftarrow S_- \cup \{p_i\}$;
Refine δ using either S_+ or S_-;
return δ;

Algorithm 2: Separation of Critical Points via Random Directions

Input : Dual points $P = \{p_1, \ldots, p_n\}$; number of trials K
Output : Recovered normal vector δ_{best}

Compute a center point $p_{\text{center}} \in P$;
Initialize candidate set $\mathcal{C} \leftarrow \emptyset$;
for $k \leftarrow 1$ **to** K **do**
    Sample a random direction $v_k \sim \mathcal{N}(0, I_{d_0})$ and normalize it;
    Initialize $S_+ \leftarrow \emptyset$ and $S_- \leftarrow \emptyset$;
    foreach $p_i \in P$ **do**
        Compute $t_i \leftarrow \langle p_i - p_{\text{center}}, v_k \rangle$;
        if $t_i > 0$ **then**
            $S_+ \leftarrow S_+ \cup \{p_i\}$;
        else
            $S_- \leftarrow S_- \cup \{p_i\}$;
    foreach $S \in \{S_+, S_-\}$ **do**
        if $|S| < 3$ **then**
            continue;
        Construct the difference matrix $A^S \in \mathbb{R}^{n \times d_0}$ with rows $A^S_i = p_i - p_{\text{center}}$ for $p_i \in S$;
        all_zero $\leftarrow |\{j : \|(A^S)_{:,j}\|_\infty < \varepsilon\}|$;
        shared_coords $\leftarrow |\{j : (A^S)_{a,j} \neq 0$ and $(A^S)_{b,j} \neq 0$, for some $a \neq b\}|$;
        if *shared_coords* ≤ 3 **then**
            continue;
        Compute the SVD $A^S = U\Sigma V^\top$ with singular values $\sigma_1 \geq \cdots \geq \sigma_{d_0}$ and right singular vectors $v_1, \ldots, v_{d_0}$;
        Set $\delta_k \leftarrow v_{d_0}$ and $e_k \leftarrow \sigma_{d_0}$;
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\delta_k, e_k)\}$;
Select δ_{best} from $\mathcal{C}$ with the smallest error e_k;
return δ_{best};

We compute a center point and project all points along each chosen direction, thereby

separating them into two subsets on opposite sides while simultaneously filtering out coordinates that carry no information and thus act as free dimensions. The detailed procedures are given in Algorithm 1 and Algorithm 2. In our experiments, we use Algorithm 2, which reliably separates the two types of critical points, discards redundant or uninformative samples, and recovers the desired signatures in our setting. For deeper layers, full recovery is not achieved, but the method still recovers the majority of signatures; the corresponding results are summarized in Table 3.

Limitation. Our improved method does not fully recover all signatures. The effective information for the HardTanh activation function is concentrated within the $[-1, 1]$ range, and the saturation region is overly broad, which makes it difficult to recover signatures in deeper layers. As the network depth increases, the coordinates of critical points often lie outside the $[-1, 1]$ range, causing the performance of our method to diminish in the later layers. If, for a given dimension i, all critical points found for that dimension have coordinates of either 1 or -1, this dimension provides no information and becomes a free dimension. Additionally, due to the random nature of the point-finding method, the points may lack sufficient diversity, leading to an incomplete separation of critical points.

For neural networks with ReLU as the activation function, recovering the sign has been the most challenging aspect of parameter recovery, which is why previous work on S1 did not provide an end-to-end attack. In contrast, for networks with HardTanh as the activation function, sign recovery is not an issue. Consequently, if future work can reliably recover all signatures in networks with HardTanh, a fully end-to-end attack would become possible.

4 Piecewise Nonlinear Activation Functions

For piecewise nonlinear activation functions, which combine linear and nonlinear segments and thus contain both linear and nonlinear branches, classical strategies based on "critical points" or "dual points" often fail. Although the pre-activation $a_i(x)$ is linear in x, so that the set $\{x : a_i(x) = t\}$ forms a hyperplane in the input space, the corresponding decision level sets after applying a nonlinear σ are, in general, curved analytic surfaces rather than hyperplanes. This curvature makes the localization of dual points effectively impossible, so the resulting geometry no longer reveals useful information. Figure 7 illustrates typical profiles of such activations.

$$\mathrm{ELU}_\alpha(x) = \begin{cases} x, & x \geq 0, \\ \alpha(e^x - 1), & x < 0. \end{cases} \qquad \mathrm{ELU}'_\alpha(x) = \begin{cases} 1, & x \geq 0, \\ \alpha e^x, & x < 0. \end{cases}$$

$$\mathrm{Swish}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}}, \qquad \mathrm{GELU}(x) = \frac{1}{2} x \left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right).$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic sigmoid anderf$(\cdot)$ is the Gaussian error function.



(a) ELU (b) Derivative (c) Swish (d) GELU

Figure 7: Smooth activations: ELU, its derivative, Swish, and GELU.

4.1 Model Extraction in the ELU Setting

In this subsection, we analyze the ELU activation function and present its corresponding attack method. ELU (also referred to as EReLU in some works) combines a linear positive branch with an exponential negative branch, whose negative mean facilitates normalization but at the cost of additional exponential computation [CUH16]. It is a piecewise defined activation function that becomes a C^1 function when $\alpha = 1$, where C^1 denotes a function that is continuous and has continuous first derivatives. Let

$$V(\eta; x) = w_\eta^\top x + b_\eta, \qquad h_\eta(x) = \begin{cases} V(\eta; x), & V(\eta; x) \geq 0, \\ \alpha\big(e^{V(\eta;x)} - 1\big), & V(\eta; x) < 0, \end{cases}$$

Define the directional derivative of h_η at x along w_η by

$$D_\eta(x; w_\eta) := \lim_{\varepsilon \to 0} \frac{h_\eta(x + \varepsilon w_\eta) - h_\eta(x)}{\varepsilon}.$$

For $V(\eta; x) > 0$, the point lies on the linear branch and

$$D_\eta(x; w_\eta) = \|w_\eta\|_2^2,$$

For $V(\eta; x) < 0$, the point lies on the nonlinear branch and

$$D_\eta(x; w_\eta) = \alpha\, e^{V(\eta;x)}\, \|w_\eta\|_2^2.$$

At a critical point x^* with $V(\eta; x^*) = 0$, the one sided limits are

$$D_\eta^+ := \lim_{\substack{x \to x^* \\ V(\eta;x) > 0}} D_\eta(x; w_\eta) = \|w_\eta\|_2^2, \qquad D_\eta^- := \lim_{\substack{x \to x^* \\ V(\eta;x) < 0}} D_\eta(x; w_\eta) = \alpha\, \|w_\eta\|_2^2.$$

Thus, the jump at the kink of the ELU activation is

$$D_\eta^+ - D_\eta^- = (1 - \alpha)\, \|w_\eta\|_2^2.$$

For activation functions of this type, the dual point method reviewed in Section 2.2.2 is no longer applicable. Although the critical hyperplane still forms an affine subspace that encodes weight information, in the S1 setting, the strategy of traversing the decision boundary in order to detect local bends and then locate dual points breaks down. Because the decision boundary is itself curved, local exploration along the boundary, starting from a single decision point, cannot separate the curvature induced by a neuron's critical hyperplane from the curvature inherent in the nonlinear decision surface, as illustrated in Figure 8.

In the S5 setting, the method reviewed in Section 2.2.1 is no longer effective. In this case, the first order derivative difference can no longer eliminate the influence of neurons other than the target neuron: on the negative branch, the derivative difference is nonzero and nonconstant, so residual contributions from other neurons cannot be canceled. Consequently, a more stringent condition is required. When all neurons in a layer are in the positive branch, the layer is locally linear. If we craft an input such that only the target neuron satisfies $a_\eta(x) < 0$ while all other neurons in the same and subsequent layers satisfy $a_j(x) \geq 0$, then under this special activation pattern the first order derivative difference can still isolate the contribution of the target neuron, as illustrated in Figure 9. Our method exploits the coexistence of linear and nonlinear regions of the activation function and remains applicable even when the first order derivative is continuous at the critical point.

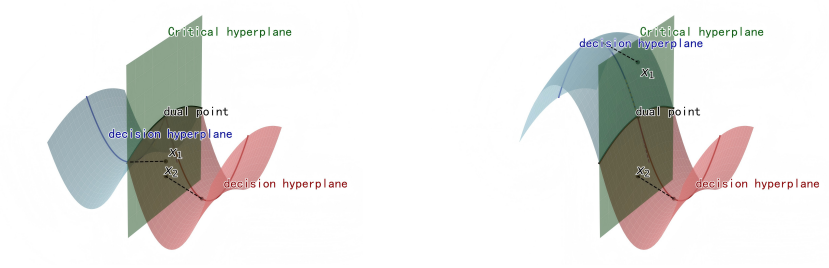


Figure 8: Relationship between the decision boundary and a neuron's critical hyperplane in the nonlinear regime: the decision boundary is curved, and crossing the critical hyperplane changes the local orientation of nearby boundary patches.

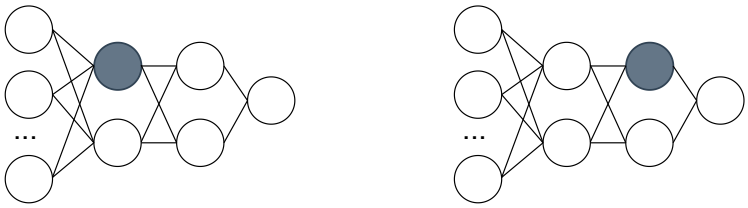


Figure 9: An example with two hidden layers, where white points lie on the positive branch and blue points lie on the negative branch.

Searching for points. We begin by explaining how to locate points that exhibit the required activation pattern.

Step 1: Locating a suitable starting point. Find a point at which all second order partial derivatives vanish, i.e.,

$$\nabla^2 f(x) = 0.$$

Such a point necessarily lies in a region where, in every layer, every neuron is in its positive branch.[2]

Step 2: Advancing to the first nonzero curvature. After locating such a point x_0, randomly choose a direction v and move along the ray

$$x(t) = x_0 + t\,v, \qquad t \geq 0.$$

Define

$$t_* = \inf\{\, t > 0 : \nabla^2 f(x(t)) \neq 0 \,\},$$

and set $x_* := x(t_*)$. The point x_* is the first point along the ray at which a second derivative becomes nonzero. At x_*, at least one neuron must be in its negative branch. Moreover, because x_* is the closest point with nonzero curvature along the ray, there is a high probability that *exactly one* neuron is in the negative branch. This point x_* is the desired point for the subsequent analysis.

Clustering method. After obtaining points with the desired activation pattern, it is necessary to determine the layer to which the responsible neuron belongs. Since our recovery procedure proceeds layer by layer, interference from earlier layers is eliminated. However, this criterion alone is insufficient to distinguish whether the neuron lies in the

[2]Two theoretical exceptions are: (i) neurons in the negative branch whose downstream weights are exactly zero; (ii) the exact algebraic cancellation of second order terms along multiple paths. Both events are extremely unlikely in practice and can be ignored.

current layer or in a subsequent one. We now describe how to cluster the collected candidate points $\{x_*\}$ and associate each cluster with an individual neuron. This procedure groups points according to the directions of their gradient jumps.

We consider a neural network with ELU activation defined as follows:

$$\begin{aligned} z^1 &= A^1 x + b^1, \quad a^1 = \sigma(z^1), \\ z^2 &= A^2 a^1 + b^2, \quad a^2 = \sigma(z^2), \\ &\vdots \\ z^k &= A^k a^{k-1} + b^k, \quad a^k = \sigma(z^k), \\ y &= {w^{\text{out}}}^{\mathrm{T}} a^k + b^{\text{out}}, \end{aligned}$$

where $x \in \mathbb{R}^d$ is the input, $A^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ for $\ell = 1, \ldots, k$ with $n_0 = d$, $w^{\text{out}} \in \mathbb{R}^{n_k}$ is the output layer weight vector, and σ denotes the ELU activation function. Let

$$D^\ell(x) := \operatorname{diag}(\sigma'(z^\ell(x))) \in \mathbb{R}^{n_\ell \times n_\ell}, \qquad M^\ell(x) := D^\ell(x) A^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}.$$

Then, for any input x, the Jacobian of the mapping $x \mapsto y(x)$ is

$$J(x) := \nabla_x y(x) = {w^{\text{out}}}^{\mathrm{T}} M^k(x) M^{k-1}(x) \cdots M^1(x) \in \mathbb{R}^{1 \times d}.$$

Theorem 1. *Under the above conditions, if only the j-th diagonal entry of D^ℓ differs between the two sides,*[3] *then there exist scalars $\delta, \lambda \in \mathbb{R}$ and a vector $v \in \mathbb{R}^d$ such that*

$$\Delta g = \delta \lambda v^\top,$$

i.e., Δg is collinear (as a row vector) with a fixed direction $v^\top \in \mathbb{R}^{1 \times d}$.

Proof. See Appendix B.1. □

We cluster the points so that each cluster corresponds to a single neuron. By Theorem 1, the gradient difference

$$\Delta g = \left.\frac{\partial y}{\partial x}\right|_{\text{right}} - \left.\frac{\partial y}{\partial x}\right|_{\text{left}}$$

is proportional to the weight vector w of that neuron, and neurons with different weights yield gradient differences with distinct directions. We therefore extract the principal direction of Δg and cluster the points according to a directional similarity metric. Next, we apply directional consistency filtering within each cluster to remove outliers and improve cluster purity. Ultimately, each neuron is represented by a "dual triple" $(x_{\text{left}}, x_{\text{mid}}, x_{\text{right}})$, where x_{left} and x_{right} lie on opposite sides of the neuron's activation boundary, with one side fully positive and the other containing exactly one neuron switch. See Algorithm 4 for details.

Recovery method. The principal direction of Δg recovered during the clustering process corresponds to the neuron signature scaled by a constant. The recovery is performed layer by layer, mapping the input space of the i-th layer to its corresponding input space. The focus is on identifying points with positive pre-activation outputs in subsequent layers, without considering the signs of pre-activation outputs in earlier layers. For the recovery of the bias term b_i associated with neuron η_i, we utilize the relation at the point x_{middle} in the corresponding neuron cluster, which satisfies

$$w_i^\top x_{\text{middle}} + b_i = 0.$$

[3] More precisely, $\Delta D^\ell := D^\ell(x_{\text{right}}) - D^\ell(x_{\text{left}})$ is diagonal with only the j-th entry nonzero. If multiple neurons switch branches, Δg is a sum of several rank one terms.

Here, w_i is the recovered, scaled weight vector corresponding to neuron η_i, enabling the recovery of b_i. Since the process proceeds layer by layer, interference from earlier layers is avoided. However, points from later layers may be included during the recovery of a given layer, complicating the process. Additionally, the nonlinear nature of the network introduces challenges in scale alignment. These issues are addressed below.

In prior work on networks with ReLU activation, it suffices to recover the neuron signature scaled by a constant. The relative magnitudes between neurons in the current layer can be linearly mapped to the relative magnitudes of the corresponding neuron signature components in the next layer. Therefore, recovering the scaled signature alone is sufficient to reconstruct an equivalent network. However, for networks with nonlinear activations, the exact neuron signature must be recovered to obtain an equivalent network. This is because the relative proportions between neurons in the previous layer cannot be linearly propagated through the nonlinear activation and thus cannot be compressed into the next layer.

To address this, we develop a method to extract the exact scaling between the recovered neuron signature and the true network signature. By clustering directional derivatives, we first recover the neuron signature up to an unknown scaling. To further recover parameters strictly equal to the true parameters, we compute the gradient difference along a randomly chosen perturbation direction,

$$Cw\big(e^{wx_1+b} - e^{wx_2+b}\big),$$

where w denotes the neuron parameter vector, C is a scalar factor from the subsequent layer, and x_1, x_2 are two points along the perturbation direction used to approximate the directional derivative.

After recovering the neuron signature scaled by a constant, we compute the scalar projections

$$X_j := wx_j, \quad j = 1, 2, 3, 4,$$

which represent the projection of the perturbation directions onto w. Using these projections, the derivative differences along the perturbation directions can be written as

$$Cw\big(e^{aX_1+b} - e^{aX_2+b}\big) \quad \text{and} \quad Cw\big(e^{aX_3+b} - e^{aX_4+b}\big).$$

To obtain a scalar ratio suitable for scaling recovery, we select a single component i:

$$\frac{\big[Ce^b w(e^{aX_1} - e^{aX_2})\big]_i}{\big[Ce^b w(e^{aX_3} - e^{aX_4})\big]_i} = k.$$

This defines the function

$$F(a) := \frac{e^{aX_1} - e^{aX_2}}{e^{aX_3} - e^{aX_4}} - k,$$

which is used to determine the scaling factor a.

Theorem 2. *Let $a > 0$ and assume $X_1 < X_2$, $X_3 < X_4$. Suppose that the intervals $[X_1, X_2]$ and $[X_3, X_4]$ satisfy*

$$\big((X_1 + X_2) - (X_3 + X_4)\big) \cdot \big((X_2 - X_1) - (X_4 - X_3)\big) > 0.$$

Then $F(a)$ is strictly monotonic for $a > 0$ and therefore possesses at most one zero.

Proof. See Appendix B.2. □

By Theorem 2, there exists a unique $a > 0$ such that $F(a) = 0$. This value can be efficiently computed using the bisection method and provides the exact scaling factor

between the recovered and true parameters. Multiplying the recovered parameters by this factor fully restores the true network parameters.

Hence, the neuron's signature is recovered up to a global sign. To recover this sign, we evaluate the recovered signature matrix A_j^i of neuron η_j on the specific input points x_j that were used to cluster this neuron. If the resulting pre-activation value $\langle A_j^i, x_j \rangle + b_j$ is positive, the sign of A_j^i is flipped; otherwise, it is retained. This procedure is valid because these clustering points are constructed such that the pre-activation of the current neuron is negative, while the pre-activations of all other neurons in the same layer remain positive.

During clustering, points originating from the current layer dominate due to their abundance. However, points associated with later layers may also be included, which can result in more clusters than the number of neurons in the current layer. These additional clusters correspond to the case where neurons in the current layer produce positive outputs, while exactly one neuron in a subsequent layer produces a negative output. For such clusters, the associated direction vector is given by $e_j^T A^\ell \left(\prod_{i=1}^{\ell-1} M_+^i \right)$. To identify the correct neuron signatures for the current layer, we consider combinations of rows from the recovered weight matrices A^i, where each row corresponds to a candidate neuron signature. Specifically, we enumerate all $\binom{n}{d_i}$ possible combinations, where n is the total number of clusters and d_i is the number of neurons in layer i. Each candidate combination is then validated using input points x drawn from the remaining clusters: we compute $A^i x + b$ and retain only those combinations for which all entries are strictly positive. This filtering step ensures that only the true weight matrix corresponding to the current layer is preserved, thereby enabling consistent end-to-end recovery of the network parameters.

Combining the above steps results in a complete end-to-end attack, the detailed procedure of which is outlined in Algorithm 3. In practice, obtaining points with the required activation pattern is challenging, particularly in larger networks. Hence, we restrict experiments to a toy example that demonstrates the feasibility of the approach.

Algorithm 3: End-to-End Attack Procedure for ELU-Activated Neural Networks

Input: $f(\cdot)$: Neural network with ELU activation; $\{\mathcal{L}_j\}_{j=1}^n$: Network layers; x: Input domain.

Output: $\{(w_i^{(j)}, b_i^{(j)})\}_{i=1}^{d_j}$: Recovered parameters for each layer $j = 1, \ldots, n$, where d_j is the number of neurons in layer j.

Function `EndToEndAttack`($f(\cdot)$, $\{\mathcal{L}_j\}_{j=1}^n$, x)**:**

  for $j = 1$ **to** $n - 1$ **do**

    Step 1: Sampling ;

    Sample a set of points x_{random} from the input space of layer $\mathcal{L}_j$[4].

    Step 2: Signature Recovery ;

    Cluster the points x_{random} to obtain the neuron signatures for layer $\mathcal{L}_j$.

    Step 3: Bias Recovery ;

    For each neuron cluster, find the triplet $(x_{\text{middle}}, x_{\text{right}}, x_{\text{left}})$ such that:

$$w_i^\top x_{\text{middle}} + b_i = 0.$$

    Recover the bias term b_i for the neuron.

    Step 4: Sign and Scale Recovery ;

    Recover the sign and scale of each neuron by applying sign recovery and scaling recovery algorithms.

    Step 5: Exclusion ;

    Remove inconsistent candidate weight matrices A_i^j that do not belong to the current neuron cluster.

  Final Layer: ;

  For the last layer $\mathcal{L}_n$, directly recover the weights and biases from the input-output relations (no activation function applied).

  return *Recovered network parameters* $\{(w_i^{(j)}, b_i^{(j)})\}_{i=1}^{d_j}$ *for all layers* $j = 1, \ldots, n$.

[4]In the case of uniform network activation probabilities, random point selection can provide a sufficient

Limitation. Using the above procedure, we can effectively distinguish points corresponding to different neurons; however, clustering alone cannot determine whether a given cluster corresponds to the current layer or to a subsequent layer. Since the network architecture is assumed to be known, we enumerate all feasible combinations of layer and neuron assignments and, together with the sign recovery step described earlier, test whether a solution exists. This exhaustive search has exponential time complexity in the worst case. The correct combination, namely the one for which a solution is found, simultaneously determines both the neuron and the corresponding sign configuration. In addition, points that exhibit the required activation pattern are difficult to obtain in practice, especially in wider networks; therefore, we restrict our experiments to a toy example that demonstrates the feasibility of the approach.

4.2 Analysis on Smooth Functions

Swish and GELU are smooth, softly gating activation functions: Swish is mildly non-monotonic near zero, whereas GELU gates inputs via a Gaussian CDF. They are widely used in modern deep learning models. Compared with ReLU, their main advantages are: (i) smoother gradients and improved optimization stability; (ii) better signal flow with fewer "dead" units; and (iii) slightly higher expressivity through soft gating, which enables finer-grained, continuous control over activation strength compared to hard thresholding. The tradeoff is a modest increase in computation and potentially slower inference on constrained hardware [RZL17, Hen16].

For smooth activation functions such as Swish and GELU, the nonlinearity is C^∞ on $\mathbb{R}$. Specifically, both $\mathrm{Swish}(x) = x\,\sigma(x)$ and $\mathrm{GELU}(x) = \frac{1}{2}x\big(1 + \mathrm{erf}(x/\sqrt{2})\big)$ are compositions and products of C^∞ functions and are therefore C^∞ on $\mathbb{R}$. As a result, all derivatives of every order exist and are continuous. There are no kinks similar to ReLU, and there is no constant first order derivative jump across any hyperplane. Consequently, for fixed parameters, networks that use only these activation functions are C^∞ in their inputs.

In particular, there are no explicit critical hyperplanes with a constant first order derivative gap across the boundary, and the gradient difference between nearby points varies smoothly with the pre-activations rather than reducing to a fixed constant multiple of a weight vector. Consequently, methods predicated on piecewise linearity or on constant first order derivative differences, such as boundary traversal with dual points and our earlier ELU based treatment, no longer apply in this regime. Within our threat model, we regard this family of smooth activation functions as among the most secure: to the best of our knowledge, no successful model extraction attack has been demonstrated to date against networks with multiple hidden layers that use them, and our own attempts have not yielded a practical attack.

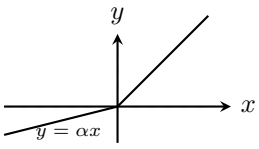
5 PReLU Activation Function

Beyond ReLU, PReLU introduces a learnable parameter α on the negative half axis, which enhances model expressivity. PReLU employs a learnable negative slope that adapts across channels and layers, which invalidates extraction methods designed for ReLU networks. As a result, new procedures are required for recovering signatures, signs, and slope parameters [HZRS15, XWCL15, CDM+25]. The PReLU activation function is defined as:

$$\mathrm{PReLU}_\alpha(x) = \begin{cases} x, & x \geq 0, \\ \alpha x, & x < 0, \end{cases}$$

number of points with the desired activation pattern. While not all points are useful, the clustering algorithm efficiently removes the incorrect ones to complete signature recovery.

where $0 < \alpha < 1$ is a learnable slope parameter that varies across neurons. Figure 10 illustrates the typical profile of the PReLU activation function.



(a) PReLU

Figure 10: PReLU activation function.

The derivative of the PReLU activation is given by:

$$\frac{\mathrm{d}}{\mathrm{d}x}\mathrm{PReLU}_\alpha(x) = \mathbf{1}_{\{x\geq 0\}} + \alpha\,\mathbf{1}_{\{x<0\}},$$

making the function continuous on $\mathbb{R}$ and differentiable on $\mathbb{R}\setminus\{0\}$. At a pre-activation hyperplane $V(\eta; x) = 0$, the one-sided jump in the directional derivative along the normal w_η is equal to $(1-\alpha)\,\|w_\eta\|_2^2$.

PReLU shares many properties with ReLU. Chen et al. [CDM+25] demonstrated that, in the S5 setting, the additional parameter α can be recovered by taking ratios of the first derivatives, while no attacks are currently known for S1. We now explain how to extract PReLU parameters in the S1 setting.

The decision hyperplanes and critical hyperplanes induced by PReLU share the same fundamental properties as those of other piecewise linear activation functions. In fact, PReLU is a special case of a piecewise linear function. Consequently, the signature recovery method based on dual points, reviewed in Section 2.2.2, remains applicable: the parameter α does not affect the piecewise linear structure or the geometry of the critical hyperplanes. Hence, regardless of the value of α, the signatures of the network can still be recovered exactly as in the ReLU case. The remaining task is to recover the additional parameter α given the known weight matrices.

5.1 Slope Recovery

To mount an extraction attack on PReLU networks in the S1 setting, it suffices to recover the per neuron negative slope parameter α. Our method exploits local dual points and the fact that, across the critical hyperplane of a target neuron in layer i, only a single diagonal entry of $D^{(i)}$ changes while all other components remain fixed; the resulting change in the local orientation of the decision hyperplane is illustrated in Figure 11.

For the i-th layer with the pre-activation vector $Y_i = [y_1^{(i)}, \ldots, y_{d_i}^{(i)}]^\top$, the PReLU activation function is given by:

$$\sigma_i(Y_i) = D_i Y_i, \quad D_i = \mathrm{diag}(\tau_1^{(i)}, \ldots, \tau_{d_i}^{(i)}), \quad \tau_j^{(i)} = \begin{cases} 1, & y_j^{(i)} \geq 0, \\ s_j^{(i)}, & y_j^{(i)} < 0. \end{cases}$$

A i-layer deep neural network $f_\theta(x)$ is expressed as:

$$f_\theta(x) = \Gamma_{\mathcal{P}} x + B_{\mathcal{P}} = G^{(i)} A^{(i)} C^{(i-1)} x + B_{\mathcal{P}},$$

where $G^{(i)}$ and $C^{(i-1)}$ represent the uncovered and recovered parts of the network, respectively:

$$G^{(i)} = \begin{cases} A^{(k+1)} D_{\mathcal{P}}^{(k)} A^{(k)} \cdots D_{\mathcal{P}}^{(i+1)} A^{(i+1)} D_{\mathcal{P}}^{(i)}, & i \in \{1, \ldots, k\}, \\ 1, & i = k+1, \end{cases}$$

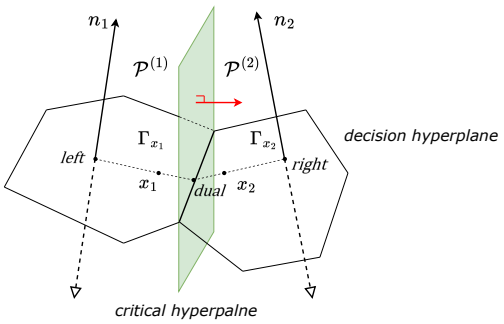


Figure 11: The critical hyperplane of a neuron alters the local orientation of the decision hyperplane: on each side, the normal to the decision hyperplane coincides with the direction vector of the affine transformation associated with the corresponding pattern $\mathcal{P}^{(1)}$ or $\mathcal{P}^{(2)}$.

$$C^{(i-1)} = D_{\mathcal{P}}^{(i-1)} A^{(i-1)} \cdots D_{\mathcal{P}}^{(1)} A^{(1)} \sigma, \quad i \in \{2, \ldots, k+1\}.$$

At a dual point x^* associated with a target neuron η_l in layer i, consider two nearby points x_1 and x_2 in a small neighborhood of x^*, lying on the decision hyperplanes on the positive and negative activation sides of η_l, respectively. These decision hyperplanes satisfy

$$\begin{aligned} \left[G_+^{(i)} D_+^{(i)} \left(A^{(i)} x_1 + b^{(i)}\right)\right]_m &= \left[G_+^{(i)} D_+^{(i)} \left(A^{(i)} x_1 + b^{(i)}\right)\right]_n, \\ \left[G_-^{(i)} D_-^{(i)} \left(A^{(i)} x_2 + b^{(i)}\right)\right]_m &= \left[G_-^{(i)} D_-^{(i)} \left(A^{(i)} x_2 + b^{(i)}\right)\right]_n. \end{aligned} \tag{5}$$

where $G_+^{(i)} \approx G_-^{(i)}$, and $D_+^{(i)}$ and $D_-^{(i)}$ differ only in the l-th diagonal entry corresponding to the target neuron η_l. We define

$$D_+^{(i)} = \operatorname{diag}\left(\tau_1^{(i)}, \tau_2^{(i)}, \ldots, 1, \ldots, \tau_{n_i}^{(i)}\right), \qquad D_-^{(i)} = \operatorname{diag}\left(\tau_1^{(i)}, \tau_2^{(i)}, \ldots, \alpha_l^{(i)}, \ldots, \tau_{n_i}^{(i)}\right),$$

where the entries at position l correspond to the positive (1) and negative ($\alpha_l^{(i)}$) activation sides of the target neuron.

Let n_+ and n_- denote the input-space normals of the two adjacent decision hyperplanes. Then, we have:

$$n_+^\top \propto \left[G_+^{(i)} D_+^{(i)} A^{(i)}\right]_{m-n}, \quad n_-^\top \propto \left[G_-^{(i)} D_-^{(i)} A^{(i)}\right]_{m-n},$$

and $G_+^{(i)} \approx G_-^{(i)}$. Right-multiplying by $(A^{(i)})^{-1}$ cancels $A^{(i)}$, yielding:

$$\tilde{n}_+ = n_+ (A^{(i)})^{-1} = \left[\left(G_+^{(i)} D_+^{(i)} A^{(i)}\right)_m - \left(G_+^{(i)} D_+^{(i)} A^{(i)}\right)_n\right] (A^{(i)})^{-1} = \left[G_+^{(i)} D_+^{(i)}\right]_{m-n},$$

$$\tilde{n}_- = n_- (A^{(i)})^{-1} = \left[\left(G_-^{(i)} D_-^{(i)} A^{(i)}\right)_m - \left(G_-^{(i)} D_-^{(i)} A^{(i)}\right)_n\right] (A^{(i)})^{-1} = \left[G_-^{(i)} D_-^{(i)}\right]_{m-n}.$$

Let $G^{(i)}$ be a matrix whose m-th and n-th rows are given by $\mathbf{g}_m = [g_{m1}, g_{m2}, \ldots, g_{mn_i}]$ and $\mathbf{g}_n = [g_{n1}, g_{n2}, \ldots, g_{nn_i}]$, respectively. Multiplying $G^{(i)}$ with $D_1^{(i)}$ and $D_2^{(i)}$ results in the following expressions for the m-th and n-th rows of the matrices:

$$\tilde{n}_+ = \left[\tau_1^{(i)}(g_{m1} - g_{n1}), \ldots, 1 \cdot (g_{ml} - g_{nl}), \ldots, \tau_{n_i}^{(i)}(g_{mn_i} - g_{nn_i})\right],$$

$$\tilde{n}_- = \left[\tau_1^{(i)}(g_{m1} - g_{n1}), \ldots, \alpha_l^{(i)}(g_{ml} - g_{nl}), \ldots, \tau_{n_i}^{(i)}(g_{mn_i} - g_{nn_i})\right].$$

Thus, the term $[G^{(i)}D^{(i)}]_{m-n}$ captures the difference. Since $D_+^{(i)}$ and $D_-^{(i)}$ differ only in the l-th diagonal entry, their ratio directly reveals $\alpha_l^{(i)}$:

$$\alpha_l^{(i)} \approx \left[\frac{\tilde{n}_-}{\tilde{n}_+}\right]_l = \left[\frac{n_-(A^{(i)})^{-1}}{n_+(A^{(i)})^{-1}}\right]_l. \tag{6}$$

Each dual point locally isolates a single neuron and provides an independent estimate of its parameter α. In our experiments, we do not know a priori which side corresponds to the positive branch and which to the negative branch. Accordingly, after computing a raw estimate $\hat{\alpha}_1^{(i)}$ from (6), we enforce the convention $\alpha \in (0, 1]$ by symmetry:

$$\hat{\alpha}_1^{(i)} \leftarrow \min\{\hat{\alpha}_1^{(i)}, (\hat{\alpha}_1^{(i)})^{-1}\},$$

which compensates for a possible swap of the two sides.

Limitation. The derivation above assumes that every layer, except the last one, has full row rank, ensuring that $(A^{(i)})^{-T}$ exists or is well-conditioned. As a practical surrogate, we assume equal or contractive layer widths.[5]

Sign Recovery. For sign recovery under S1, we again rely on the Neuron Wiggle Technique described above. Notably, recovery of the additional slope parameter α depends only on the relative signs within each neuron and is therefore independent of the sign assignment across neurons. This phenomenon was empirically confirmed by Chen et al. [CDM+25] using their Neuron Wiggle Technique:

$$s_{\text{on}} \propto \|v_{\text{on}}\|_2 \approx \left\| \left(\frac{\pm\tau_1^{(j)}\varepsilon}{\sqrt{d_i}}, \frac{\pm\tau_2^{(j)}\varepsilon}{\sqrt{d_i}}, \ldots, \varepsilon, \ldots, \frac{\pm\tau_{d_i-1}^{(j)}\varepsilon}{\sqrt{d_i}}, \frac{\pm\tau_{d_i}^{(j)}\varepsilon}{\sqrt{d_i}} \right) \right\|_2, \tag{7}$$

$$s_{\text{off}} \propto \|v_{\text{off}}\|_2 \approx \left\| \left(\frac{\pm\tau_1^{(j)}\varepsilon}{\sqrt{d_i}}, \frac{\pm\tau_2^{(j)}\varepsilon}{\sqrt{d_i}}, \ldots, s_i^{(j)}\varepsilon, \ldots, \frac{\pm\tau_{d_i-1}^{(j)}\varepsilon}{\sqrt{d_i}}, \frac{\pm\tau_{d_i}^{(j)}\varepsilon}{\sqrt{d_i}} \right) \right\|_2. \tag{8}$$

6 Step Activation Function

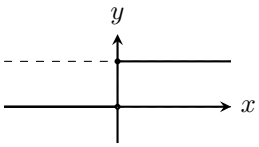
The Heaviside step function, denoted as Step in this paper, outputs 0 for negative inputs and 1 for positive inputs, resulting in binary-valued features from the second layer onward. The Step function is typically trained using straight-through estimators (STEs), which may introduce bias and instability, limiting the model's expressive power [Ros58, BLC13]. The Step function is defined as:

$$\text{Step}(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0. \end{cases}$$

Figure 12 illustrates the typical profile of the Step activation function.

In terms of its mathematical properties, the Step function is discontinuous at $x = 0$, which results in a derivative that is 0 almost everywhere (a.e.) and undefined at $x = 0$. As an activation function, it maps pre-activations to binary vectors in $\{0, 1\}^{d_k}$ for a layer of width d_k. Despite its nondifferentiability, this binary mapping admits a form of structural identifiability: as long as the pre-activations in the preceding layer do not cross a discontinuity, the outputs of subsequent layers remain invariant, a property that can be exploited by an attacker.

[5] Our parameters are all 64-bit real numbers, and due to the probabilistic properties of such matrices, it is almost certainly full rank, especially as the matrix size increases.



(a) Step ($H(0) = 1$)

Figure 12: Step activation function.

Let $\sigma(x) = \mathbf{1}_{\{x \geq 0\}}$. We consider a network defined by

$$a^0 = x, \qquad z^j = A^j a^{j-1} + b^j, \qquad a^j = \sigma(z^j) \quad (j = 1, \ldots, k),$$

followed by an output layer

$$y = w^{\text{out}^{\mathrm{T}}} a^k + b^{\text{out}}.$$

Then, for every $j \geq 1$,

$$a^j(x) \in \{0, 1\}^{d_j}.$$

Define the first-layer code map $\pi : \mathbb{R}^{d_0} \to \{0, 1\}^{d_1}$ by

$$\pi(x) = a^1(x) = \sigma(A^1 x + b^1),$$

and the cells

$$\mathcal{C}_s \;:=\; \{\, x \in \mathbb{R}^{d_0} :\; \pi(x) = s \,\}, \qquad s \in \{0, 1\}^{d_1}.$$

Consequently, the network factors as

$$f \;=\; \Phi \circ \pi, \qquad \Phi : \{0, 1\}^{d_1} \to \mathcal{Y}.$$

Thus, the Step network maps the input space $\mathbb{R}^{d_0}$ onto a *finite collection of constant plateaus*: for each s, every connected component of $\mathcal{C}_s$ is a region on which the output is constant, and different cells may map to different constants.

Let $\gamma : [0, 1] \to \mathbb{R}^{d_0}$ be a continuous path. This refers to a smooth function where each value $t \in [0, 1]$ maps to a point $\gamma(t)$ in the input space $\mathbb{R}^{d_0}$, with no jumps or discontinuities along the path. If no neuron in layer 1 crosses the boundary of its Step activation function along $\gamma([0, t])$, then for all $k \geq 2$, the codes $a^k(\gamma(s))$ remain constant on $[0, t]$. Hence, if

$$t_* = \inf\{\, t \in [0, 1] :\; y(\gamma(t)) \neq y(\gamma(0)) \,\},$$

then necessarily

$$\exists\, j \text{ such that } z_j^1(\gamma(t_*)) = 0.$$

Let

$$H := \{x :\; w^\top x + b = 0\}, \qquad (w, b) = (A^1_{j,:}, b^1_j),$$

denote the corresponding first-layer decision hyperplane. Thus, the first output change occurs when γ crosses H. Moreover, at this first change, exactly one first-layer neuron crosses the boundary of its Step activation function, and (w, b) can be recovered (up to positive scaling) directly from the flip point $x_* = \gamma(t_*)$.

Searching for points. Our method first locates such points and then constructs linear systems to recover the signatures. A key requirement is to obtain a sufficient number of qualifying points for each neuron, which leads to a clustering subproblem. We address this through the following three steps:

Step 1: Locating a label-change point. Fix a line segment in the input space and apply a bisection search to locate a point at which the predicted label changes; denote this boundary point by x_*.

Step 2: Seeding a local neighborhood. Around x_*, construct a local sample set by small perturbations along random unit directions u:

$$S \;:=\; \{\, x_* + \varepsilon u : \|u\|_2 = 1,\ |\varepsilon| \ll 1 \,\}.$$

Step 3: Local exploration from the seed set. From each $s \in S$, move along random unit directions v with very small step sizes τ and record any additional points $x'_*(s, v, \tau)$ at which the predicted label changes. Collect these into

$$S^* \;:=\; \{\, x'_*(s, v, \tau) : s \in S,\ \|v\|_2 = 1,\ |\tau| \ll 1 \,\}.$$

Because the seed set S lies near a critical hyperplane, the subsequent local exploration returns *critical points* that, with high probability, belong to the *same* critical hyperplane, as illustrated in Figure 13. By contrast, since the Step activation maps most of the input space onto large constant plateaus, starting from an arbitrary point and following a random direction with small steps is unlikely to encounter any qualifying point within a finite iteration budget.

Recovery method. Using the resulting cluster S^* associated with each neuron, we assemble an appropriate system of linear equations and obtain the corresponding signature by solving this system. Since we recover an *equivalent* model, neuron permutations within a layer do not need to be resolved; it suffices to identify the multiset $\{(A^1_{j,:}, b^1_j)\}_{j=1}^{d_1}$. Given the known architecture, we repeatedly apply the above procedure to obtain distinct weight vectors w until the required number of neurons in the first layer has been recovered.[6]

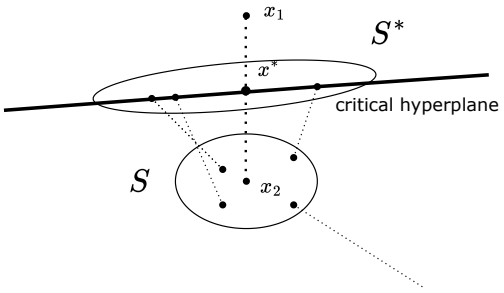


Figure 13: Schematic illustration of locating critical points associated with a single neuron.

Because the Step activation produces binary outputs from the second layer onward, the network loses almost all continuous information and exact recovery of later-layer parameters is generally infeasible. Nevertheless, since the set of first-layer codes $\{0, 1\}^{d_1}$ is finite, for moderate d_1 the mapping from these codes to outputs can be enumerated; in

[6]The above method applies to both S1 and S5. Although under S1 the exact final output value is unobserved, the label transitions obey the same property: the point at which the label first changes occurs when the path crosses a decision hyperplane, i.e., at the jump point $x_* = \gamma(t_*)$.

this sense recovering (W^1, b^1) together with the finite map yields a functionally equivalent model (see Theorem 3).

Theorem 3. *Let $a^1(x) = \sigma(A^1 x + b^1) \in \{0,1\}^{d_1}$ denote the first-layer code. Define*

$$\Psi : \{0,1\}^{d_1} \to \mathcal{Y}, \qquad \Psi(s) = f(x) \text{ for any } x \text{ such that } a^1(x) = s.$$

Then the network factors as $f = \Psi \circ a^1$. In particular, once A^1 and b^1 are recovered, learning Ψ on the finite domain $\{0,1\}^{d_1}$ yields a model that is functionally equivalent to the original network.

Proof. See Appendix B.3. □

Experiments

We evaluate our attacks on both pseudo-trained neural networks and real-world networks trained on standard datasets. For the pseudo-trained setting, network parameters are generated through a pseudo-training procedure rather than learned from real data. In the S5 experiments, we adopt the training pipeline proposed by Carlini et al. [CJM20] to validate the effectiveness of our methods across different activation functions. Experiments are conducted under both the S1 and S5 settings on pseudo-trained networks with varying architectures, using the activation functions discussed above. Our experiments on signature recovery for LeakyReLU, HardTanh, and PReLU under S1 follow the setup of Carlini et al. [CCH+25], where similar white-box information is used, including classifying dual points by the toggled neuron, computing the normal vector m to the decision plane, and walking along the decision plane until it bends.

In addition to pseudo-trained networks, we also perform experiments on real models trained on the MNIST dataset. MNIST consists of grayscale images of handwritten digits from ten classes ("0"–"9"), where each image has a resolution of 28×28, with 60,000 training samples and 10,000 test samples. All images are converted to floating-point tensors and normalized to the range $[-1, 1]$ by first rescaling pixel values to $[0, 1]$ and then applying standard mean-variance normalization. Models are trained using the cross-entropy loss and the Adam optimizer [Kin14] with the default settings provided by Keras [Fra15]. The learning rate is set to 10^{-3}, training is performed for 20 epochs with a batch size of 32, and model performance is evaluated on the test set after each epoch. For the Leaky ReLU and ELU experiments under the S5 setting, we follow the experimental setup of Chen et al. [CDM+25] and construct multiple binary classification tasks based on different pairs of classes (e.g., "0" vs. "1", "2" vs. "3").

All experiments can be completed in a practical amount of time, typically within a few hours when run on a single core of a modern desktop machine. The results are summarized in **Table 1** and **Table 2**, which report the results on pseudo-trained networks and MNIST-trained networks, respectively. The code is available at `https://github.com/qixiaokang1-stack/cryptanalytic-model-various-functions.git`.

As summarized in **Table 3**, our experiments on HardTanh do not achieve full recovery; the detailed recovery outcomes are reported there. Our attack follows the layerwise recovery assumption of Carlini et al., recovering parameters one layer at a time and treating all preceding layers as perfectly recovered when attacking the next layer. For comparison, if we omit our proposed refinements and instead apply the original method of Carlini et al. [CCH+25], our experiments show that no neuron is correctly recovered in any layer.

Security Analysis. From the perspective of activation-specific security, piecewise linear activations are generally less secure than genuinely nonlinear ones, as the latter tend to provide stronger resistance to parameter recovery attacks. When piecewise linear activations are unavoidable, multi-segment designs are typically more difficult to extract

Table 2: Model extraction results for neural networks trained on the MNIST dataset with different activation functions. All experimental settings and parameter definitions follow those in **Table 1**, except that the networks are trained on the real MNIST dataset.

	Neural Network				**Parameter Extraction**			
Function	Setting		Architecture	Params	Signature	Sign	$\max\|\theta - \hat{\theta}\|$	**Approach**
Leaky $\text{ReLU}_{0.1}$	S5	✓	$784 - 10 - 1$	7861	*poly*	*poly*	$2^{-32.16}$	*This work*
	S1	✓	$784 - 64^4 - 10$	63370	*poly*	*poly*	$2^{-24.89}$	*This work*
HardTanh	S5	✓	$784 - 128^2 - 10$	118282	*poly*	$O(1)$	$2^{-25.57}$	*This work*
	S1	✓	$78 - 64^4 - 10$	63370	*poly*	$O(1)$	$2^{-24.46}$	*This work*
PReLU	S5	✓	$784 - 40 - 1$	31481	*poly*	*poly*	$2^{-34.34}$	[CDM+25]
	S1	✓	$784 - 64^4 - 10$	63626	*poly*	*poly*	$2^{-7.48}$	*This work*
ELU_1	S5	✓	$784 - 2^2 - 1$	1579	*poly*	*poly*	$2^{-6.72}$	*This work*
	S1	✗	–	–	–	–	–	–
Step	S5	✓	$784 - 3 - 2 - 10$	2393	*poly*	*exp*	$2^{-12.43}$	*This work*
	S1	✓	$784 - 3 - 2 - 10$	2393	*poly*	*exp*	$2^{-8.08}$	*This work*

Table 3: Signature recovery of neural networks with HardTanh activation function. ✓ denotes correct recovery, and × denotes that the collected points are insufficient to fully extract the signature, i.e., they do not fit a single hyperplane. Here, $\max|\theta - \hat{\theta}|$ denotes the maximum coordinate-wise parameter error among the recovered parameters.

Architecture	Params	**Layer1** ✓ / ×	**Layer2** ✓ / ×	**Layer3** ✓ / ×	**Layer4** ✓ / ×	$\max\|\theta - \hat{\theta}\|$	**Approach**
$64 - 64^2 - 10$	8970	64 / 0	62 / 2	–	–	$2^{-25.57}$	*This work*
$128 - 128^2 - 10$	34314	128 / 0	126 / 2	–	–	$2^{-23.71}$	*This work*
$32 - 32^4 - 10$	4554	32 / 0	32 / 0	26 / 6	25 / 7	$2^{-21.2}$	*This work*
$64 - 64^4 - 10$	17290	64 / 0	63 / 1	58 / 6	61 / 3	$2^{-20.62}$	*This work*
$784 - 128^2 - 10$(MNIST)	118282	122 / 6	86 / 42	–	–	$2^{-24.80}$	*This work*
$784 - 64^4 - 10$(MNIST)	63370	62 / 2	52 / 12	52 / 12	42 / 22	$2^{-24.46}$	*This work*

than their two-segment counterparts. For activation functions exhibiting odd or even symmetry, sign recovery becomes unnecessary, which further weakens the associated security guarantees. Beyond the choice of individual activation functions, appropriate *compositions* or *mixtures* of activation functions can further enhance overall model security.

We classify composite and mixed activation functions into three categories: (i) *composite activations*, in which different activation functions are applied to distinct regions of the input domain; (ii) *mixed activation layers*, where neurons within the same layer employ different activation functions; and (iii) *layerwise mixtures*, in which different layers adopt different activation families. These designs are primarily motivated by the goals of increasing expressivity and improving predictive accuracy, as demonstrated in prior work on composite and mixed activations [SSAL16, AHSB15, Rad15]. From a security perspective, however, all three categories introduce additional challenges for parameter recovery attacks. In particular, the selection and interaction of activation functions are critical factors in determining the feasibility of such attacks: if any constituent activation within a layer falls outside the solvable regimes, the entire layer may become intractable. Furthermore, even combinations of individually solvable activations can hinder joint recovery due to misaligned critical points and incompatible signatures. A systematic investigation of attacks targeting more complex activation functions, including fully composite or extensively mixed activation designs, is therefore deferred to future work.

7 Conclusions

In this paper, we advance cryptanalytic model extraction for neural networks. We introduce a systematic framework that extends model extraction to a broad range of activation functions, covering both piecewise linear and piecewise nonlinear cases, and we develop extraction procedures tailored to different activation families. In addition to ReLU and PReLU, we investigate several other activation functions, including Leaky ReLU, HardTanh, ELU, and the Step function. To the best of our knowledge, this is the first study to explore model extraction for these activation functions and for PReLU-based models in the S1 setting. Our experiments validate the effectiveness of our methods in recovering model parameters. We further analyze the security of different activation functions and show how composite or mixed activation configurations can strengthen neural network security, thereby clarifying how activation design influences the overall security of modern architectures. Overall, our framework provides a unified treatment of model extraction across diverse activation functions and offers guidance for the design of more secure neural networks.

Our work opens several promising directions for future research. One direction is to extend our model extraction framework to neural networks with more complex activation functions, including highly smooth families with C^∞ regularity, as well as composite and mixed activation functions. Another direction is to design variants of our attacks that scale to larger models and rely on weaker architectural assumptions, ideally yielding extraction procedures that translate into practical attacks against models deployed in real-world systems.

Acknowledgments. The authors would like to thank Kai Hu for his contributions to this paper. Due to his submission limits to ToSC, he opted not to be included as an author. We also appreciate the feedback from the anonymous shepherd and reviewers, whose comments have significantly improved the quality of this paper. This work is supported by the National Key R&D Program of China (Grant No. 2024YFA1013000, 2023YFA1009500), the National Natural Science Foundation of China (Grant No. 62032014, U2336207), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM114), and the National Cryptologic Science Fund of China (Grant No. 2025NCSF01013).

References

[AHSB15] Forest Agostinelli, Matthew D. Hoffman, Peter J. Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015.

[Bau91] Eric B. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks*, 2(1):5–19, 1991.

[BLC13] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

[CCH+25] Nicholas Carlini, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, and Adi Shamir. Polynomial time cryptanalytic extraction of deep neural networks in the hard-label setting. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part I*, volume 15601 of *Lecture Notes in Computer Science*, pages 364–396. Springer, 2025.

[CDG+24] Yi Chen, Xiaoyang Dong, Jian Guo, Yantian Shen, Anyu Wang, and Xiaoyun Wang. Hard-label cryptanalytic extraction of neural network models. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9-13, 2024, Proceedings, Part VIII*, volume 15491 of *Lecture Notes in Computer Science*, pages 207–236. Springer, 2024.

[CDM+25] Yi Chen, Xiaoyang Dong, Ruijie Ma, Yantian Shen, Anyu Wang, Hongbo Yu, and Xiaoyun Wang. Delving into cryptanalytic extraction of prelu neural networks. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part II*, volume 16246 of *Lecture Notes in Computer Science*, pages 576–607. Springer, 2025.

[CJM20] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 189–218. Springer, 2020.

[CMS26] Isaac A. Canales-Martínez and David Santos. Extracting some layers of deep neural networks in the hard-label setting. In Daniel Escudero and Ivan Damgård, editors, *Progress in Cryptology – LATINCRYPT 2025*, pages 399–421, Cham, 2026. Springer Nature Switzerland.

[CUH16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[Fra15] Chollet François. Keras. *https://github. com/fchollet/keras*, 2015.

[Hea18] Jeff Heaton. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618. *Genetic programming and evolvable machines*, 19(1):305–307, 2018.

[Hen16] D Hendrycks. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015.

[JCB+20] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1345–1362. USENIX Association, 2020.

[JYP+17] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[Kin14] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KMH+20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[LSE+25] Haolin Liu, Adrien Siproudhis, Samuel Experton, Peter Lorenz, Christina Boura, and Thomas Peyrin. Navigating the deep: Signature extraction on deep neural networks. *arXiv preprint arXiv:2506.17047*, 2025.

[MCH+24] Isaac Andrés Canales Martinez, Jorge Chávez-Saab, Anna Hambitzer, Francisco Rodríguez-Henríquez, Nitin Satpute, and Adi Shamir. Polynomial time cryptanalytic extraction of neural network models. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part III*, volume 14653 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2024.

[MHN+13] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[Rad15] Alec Radford. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[RK20] David Rolnick and Konrad Kording. Reverse-engineering deep relu networks. In *International conference on machine learning*, pages 8178–8187. PMLR, 2020.

[Ros58] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[SSAL16] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pages 2217–2225. PMLR, 2016.

[TZJ+16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX security symposium (USENIX Security 16)*, pages 601–618, 2016.

[XWCL15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Appendix

A Appendix: Preliminary

In this appendix, we provide the definitions used to describe our networks and the assumptions that were omitted from the main text.

A.1 Definition

Definition 9. (k-Deep neural network) A k-layer deep neural network $f_\theta(x)$ is a function determined by parameters θ, which takes an input from the input space X and returns a value in the output space Y. The function f consists of a sequence of alternating linear layers f_j and element-wise nonlinear activation functions σ_i, and can be written as:

$$f = f_{k+1} \circ \sigma_k \circ \cdots \circ \sigma_2 \circ f_2 \circ \sigma_1 \circ f_1,$$

where f_j represents the linear transformation at the j-th layer, and σ_i denotes the nonlinear activation function at the i-th layer.

Definition 10. (Fully connected layer) The j-th layer f_j of the neural network is given by a transformation:

$$f_j(x) = A^{(j)}x + b^{(j)},$$

where the weight matrix $A^{(j)} \in \mathbb{R}^{d_j \times d_{j-1}}$ is a $d_j \times d_{j-1}$ matrix, and the bias vector $b^{(j)} \in \mathbb{R}^{d_j}$ is a d_j-dimensional vector. Specifically, we denote the weight matrix $A^{(j)}$ at the j-th layer and the bias vector $b^{(j)}$ as:

$$A^{(j)} = \begin{bmatrix} a_j^{1,1} & a_j^{1,2} & \cdots & a_j^{1,d_{j-1}} \\ a_j^{2,1} & a_j^{2,2} & \cdots & a_j^{2,d_{j-1}} \\ \vdots & \vdots & \ddots & \vdots \\ a_j^{d_j,1} & a_j^{d_j,2} & \cdots & a_j^{d_j,d_{j-1}} \end{bmatrix}, \quad B^{(j)} = \begin{bmatrix} b_j^1 \\ b_j^2 \\ \vdots \\ b_j^{d_j} \end{bmatrix},$$

where $a_j^{i,j}$ and b_j^i are floating point numbers. In the following, we will denote A_j^i as the i-th row of $A^{(j)}$, and $A_j^{:,i}$ as the i-th column of $A^{(j)}$. Each layer applies these linear transformations followed by a nonlinear activation function σ.

Definition 11. (Neuron) A neuron is a function determined by the corresponding weight matrix, bias vector, and activation function. The set of neurons $\{\eta_i\}_{i=1}^N$ represents a collection of functions that receive inputs and apply an activation function σ. Formally, the i-th neuron in layer j, denoted $\eta_{j,i}$, is given by:

$$\eta_{j,i}(\sigma_{j-1}(Y_{j-1})) = \sigma_{j,i}(A^{(j)}x + b^{(j)}),$$

where $A^{(j)}$ is the weight matrix, $b^{(j)}$ is the bias vector, and $\sigma_{j,i}$ is the activation function. The total number of neurons is given by

$$N = \sum_{j=1}^{k-1} d_j.$$

Definition 12. (Parameters) The parameters θ of the neural network refer to the specific assignments of the weights $A^{(j)}$ and biases $b^{(j)}$, which are typically obtained during the training process. We denote θ as the original parameters of the network, and $\hat{\theta}$ as the extracted or recovered equivalent parameters, where $\hat{A}^{(j)}$ and $\hat{b}^{(j)}$ represent the recovered weight matrix and bias vector at the j-th layer, respectively.

Definition 13. (Classification) A deep neural network (DNN) for classification can be used for binary or multiclass tasks.

Binary classification. In the binary case, the DNN has a single output neuron with a sigmoid activation function, producing a value in $(0, 1)$ that represents the predicted class probability. The output is

$$f_\theta(x) = \sigma\big(A^{(k)}x + b^{(k)}\big).$$

Multiclass classification. In the multiclass case, the DNN has one output neuron per class, and a softmax activation function is applied to obtain a probability distribution over all classes:

$$f_\theta(x) = \text{softmax}\big(A^{(k)}x + b^{(k)}\big).$$

Definition 14. (Hidden state) The function $f_{1..j}$ represents the computation of the first j layers of the network, including the linear transformations f_j but excluding the nonlinear activation functions σ. The hidden state of the j-th layer is the output of $f_{1..j}$, prior to applying σ. Specifically,

$$f = f_{1..k}.$$

Definition 15. (Affine transformation) In a neural network, we say the network is affine in a neighborhood P_x of $x \in \mathbb{R}^{d_0}$ if there exist matrices Γ_x and vectors γ_x such that for all $x' \in P_x$, we have:

$$f(x') = \Gamma_x x' + \gamma_x.$$

This affine transformation refers to the composite transformation from the input to the output of the network, which includes both linear transformations and activation functions. Specifically, the transformation from the first layer to the output is expressed as:

$$F_1(x) = f_1 \circ \sigma_1 \circ f_2 \circ \sigma_2 \circ \cdots \circ f_k(x),$$

For transformations starting from layer j, we define:

$$F_j(x) = f_j \circ \sigma_j \circ f_{j+1} \circ \sigma_{j+1} \circ \cdots \circ f_k(x).$$

Definition 16. (Neuron state) Let $V(\eta; x)$ denote the input to neuron η for input x, before applying the activation function σ. If $V(\eta; x) > 0$, the neuron's state is positive; if $V(\eta; x) < 0$, the neuron's state is negative. The neuron is at a critical point if $V(\eta; x) = 0$. The state of η on input x (i.e., active, inactive, or critical) is denoted by $S(\eta; x)$. Additionally, let $L(\eta)$ denote the layer to which neuron η belongs, where the first layer is indexed by 1.

Definition 17. (Linear neighbourhood) Let $x \in X$. The linear neighborhood of a point x is defined as the set

$$\{u \in X \mid S(\eta; x) = S(\eta; u) \text{ for all neurons } \eta \text{ in the network}\},$$

where $S(\eta; x)$ represents the state of neuron η when the input is x.

Definition 18. (Model activation pattern) Consider a k-layer deep neural network f_θ with $n = \sum_{i=1}^{k} d_i$ neurons. The model activation pattern of f_θ for an input $x \in X$ is a global description of the states of all n neurons, denoted as:

$$\mathcal{P}(x) = (\mathcal{P}^{(1)}(x), \ldots, \mathcal{P}^{(k)}(x)),$$

where $\mathcal{P}^{(i)}(x) \in \mathbb{R}^{d_i}$ is the vector of neuron states in layer i, and $\mathcal{P}^{(i)}_j(x)$ represents the state of the j-th neuron in layer i, for $i \in \{1, \ldots, d_i\}$.

A.2 Assumptions

Assumptions by Carlini et al. in [CJM20]:

- **Architecture Knowledge:** The adversary has knowledge of the neural network architecture.
- **Global Input:** The adversary can provide arbitrary inputs from the input space X.
- **Full Output:** The adversary receives the full output from the model f without any further processing.
- **Exact Computation:** The model f is specified and evaluated using 64-bit floating-point arithmetic.
- **Scalar Output:** Without loss of generality, we assume the output dimension is 1, i.e., $Y = \mathbb{R}$.
- **ReLU Activation:** All activation functions σ are ReLU functions.

B Appendix: Complete Proofs

In this appendix, we provide the full proofs of all results discussed in the main text.

B.1 Proof of Theorem 1

The proof of Theorem 1 is as follows:

Proof. Write $M_{\pm}^{\ell} := M^{\ell}(x_{\text{positive/negative}})$. Since only D^{ℓ} differs,

$$\begin{aligned}\Delta g &= w^{\text{out}\,T}\big(M_{+}^{\kappa}\cdots M_{+}^{\ell+1}\big)\big(M_{+}^{\ell}-M_{-}^{\ell}\big)\big(M_{+}^{\ell-1}\cdots M_{+}^{1}\big)\\ &= w^{\text{out}\,T}\Big(\prod_{i=\ell+1}^{\kappa} M_{+}^{i}\Big)\big(\Delta D^{\ell}A^{\ell}\big)\Big(\prod_{i=1}^{\ell-1} M_{+}^{i}\Big).\end{aligned}$$

Because ΔD^{ℓ} is diagonal with only the j-th entry nonzero, we can write $\Delta D^{\ell} = \delta\, e_j e_j^{\top}$. Thus

$$\Delta g = \delta\,\Big(w^{\text{out}\,T}\prod_{i=\ell+1}^{\kappa} M_{+}^{i}\,e_j\Big)\,\Big(e_j^{\top}A^{\ell}\prod_{i=1}^{\ell-1} M_{+}^{i}\Big).$$

Let

$$\alpha := w^{\text{out}\,T}\Big(\prod_{i=\ell+1}^{\kappa} M_{+}^{i}\Big)e_j \in \mathbb{R}, \qquad v^{\top} := e_j^{\top}A^{\ell}\Big(\prod_{i=1}^{\ell-1} M_{+}^{i}\Big) \in \mathbb{R}^{1\times d}.$$

Then $\Delta g = \delta\,\alpha\,v^{\top}$. □

B.2 Proof of Theorem 2

The proof of Theorem 2 is as follows:

Proof. Define

$$f(a;u,v) := e^{av} - e^{au}, \quad r(a;u,v) := \frac{f'(a;u,v)}{f(a;u,v)} = \frac{ve^{av}-ue^{au}}{e^{av}-e^{au}}.$$

The derivative of $r(a; u, v)$ with respect to a is

$$r'(a; u, v) = -\frac{(v-u)^2 e^{a(v+u)}}{(e^{av} - e^{au})^2} < 0.$$

To analyze the monotonicity of $F(a)$, reparameterize each interval $[u, v]$ by its midpoint m and half-length $d > 0$:

$$m = \frac{u+v}{2}, \quad d = \frac{v-u}{2}, \quad \text{so that} \quad u = m - d,\ v = m + d.$$

Substituting $u = m - d$ and $v = m + d$ into $r(a; u, v)$ gives

$$r(a; u, v) = \frac{(m+d)e^{a(m+d)} - (m-d)e^{a(m-d)}}{e^{a(m+d)} - e^{a(m-d)}}.$$

Factor out e^{am} from numerator and denominator:

$$r(a; u, v) = \frac{(m+d)e^{ad} - (m-d)e^{-ad}}{e^{ad} - e^{-ad}} = m + d\,\frac{e^{ad} + e^{-ad}}{e^{ad} - e^{-ad}} = m + d\,\coth(ad),$$

where $\coth(x) = \frac{\cosh(x)}{\sinh(x)} = \frac{e^x + e^{-x}}{e^x - e^{-x}}$. This shows why $r(a; u, v)$ can be expressed in the simple form $m + d\coth(ad)$.

For fixed $a > 0$, define

$$\phi(d) := d\,\coth(ad),$$

so that $r(a; u, v) = m + \phi(d)$ depends only on the interval length $2d$. Its derivative with respect to d is

$$\phi'(d) = \coth(ad) - \frac{ad}{\sinh^2(ad)} = \frac{\cosh(ad)\sinh(ad) - ad}{\sinh^2(ad)}.$$

Let $g(x) = \cosh(x)\sinh(x) - x$. Then $g'(x) = 2\sinh^2(x) > 0$ for all $x > 0$ and $g(0) = 0$, so $g(x) > 0$ for all $x > 0$. It follows that $\phi'(d) > 0$ for all $d > 0$, implying that $r(a; u, v) = m + d\coth(ad)$ is strictly increasing in the interval length $v - u = 2d$.

Now, for two intervals $[X_1, X_2]$ and $[X_3, X_4]$, let $m_1 = \frac{X_1+X_2}{2}, d_1 = \frac{X_2-X_1}{2}$ and $m_2 = \frac{X_3+X_4}{2}, d_2 = \frac{X_4-X_3}{2}$. Then

$$r(a; X_1, X_2) - r(a; X_3, X_4) = (m_1 - m_2) + (\phi(d_1) - \phi(d_2)).$$

By the assumption

$$\big((X_1 + X_2) - (X_3 + X_4)\big) \cdot \big((X_2 - X_1) - (X_4 - X_3)\big) > 0,$$

the quantity $(m_1 - m_2) + (\phi(d_1) - \phi(d_2))$ is strictly positive or strictly negative for all $a > 0$. Consequently,

$$\frac{d}{da}\log(F(a) + k) = r(a; X_1, X_2) - r(a; X_3, X_4)$$

maintains a constant, nonzero sign for all $a > 0$. Therefore, $F(a)$ is strictly monotonic on $a > 0$, completing the proof. □

B.3 Proof of Theorem 3

The proof of Theorem 3 is as follows:

Proof. For any input x, let $s = a_1(x) \in \{0,1\}^{d_1}$. By definition of Ψ, the original network can be written as

$$f(x) = \Psi\big(a_1(x)\big).$$

Once the first-layer parameters (A_1, b_1) are recovered, we can compute

$$a_1(x) = \sigma(A_1 x + b_1)$$

for every input x.

For each code $s \in \{0,1\}^{d_1}$, fix an input x_s with $a_1(x_s) = s$ and define

$$\Psi(s) := f(x_s).$$

This definition is well defined: any two inputs with the same first-layer code produce the same activations in all subsequent layers, hence the same output.

The reconstructed model is therefore

$$x \xmapsto{(A_1,b_1)} \tilde{z} \xmapsto{\text{Step}} a_1(x) \xmapsto{\Psi} y,$$

which coincides with $f(x)$ for every input x. Thus, recovering A_1 and b_1 yields a network that is functionally equivalent to the original one. □

C Clustering Algorithm

In this appendix, we provide the clustering Algorithm discussed in the main text.

Algorithm 4: Clustering Algorithm

Input : Dual triples $\mathcal{T} = \{(x^{(i)}_{\text{left}}, x^{(i)}_{\text{mid}}, x^{(i)}_{\text{right}})\}_{i=1}^{N}$; Jacobian oracle $J(x) = \nabla_x y(x)$; rank threshold $\kappa > 1$; merge threshold $t \in (0, 1)$; minimum cluster size $m_{\min}$; optional consistency threshold $\tau \in (0, 1)$

Output : Clusters $\{C_r\}$ of dual triples and corresponding direction estimates $\{v_r\}$

Step 1: Rank-1 screening and principal direction extraction;
Initialize $U \leftarrow [\,]$ and index set $\mathcal{I} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** N **do**
  $\Delta g^{(i)} \leftarrow J(x^{(i)}_{\text{right}}) - J(x^{(i)}_{\text{left}})$;
  if $\|\Delta g^{(i)}\| = 0$ **then**
    continue;
  Compute the SVD $A^{(i)} = U^{(i)}\Sigma^{(i)}V^{(i)\top}$;
  if $\sigma^{(i)}_2 = 0$ ***or*** $\sigma^{(i)}_1/\sigma^{(i)}_2 \geq \kappa$ **then**
    $u_i \leftarrow V^{(i)}_{:,1}/\|V^{(i)}_{:,1}\|$;
    $U \leftarrow U \cup \{u_i\}$; $\mathcal{I} \leftarrow \mathcal{I} \cup \{i\}$;

Step 2: Pairwise distance computation;
foreach $i, j \in \mathcal{I}$ **do**
  $D_{ij} \leftarrow 1 - \left|\langle u_i, u_j\rangle\right|$;

Step 3: Agglomerative clustering;
Initialize cluster set $\mathcal{C} \leftarrow \left\{\{i\} : i \in \mathcal{I}\right\}$;
while $\exists\, A, B \in \mathcal{C}$ *with* $d(A, B) \leq t$ **do**
  $(A^\star, B^\star) \leftarrow \arg\min_{A \neq B \in \mathcal{C}} d(A, B)$;
  if $d(A^\star, B^\star) \leq t$ **then**
    $\mathcal{C} \leftarrow \left(\mathcal{C} \setminus \{A^\star, B^\star\}\right) \cup \{A^\star \cup B^\star\}$;

Step 4: Consistency filtering and direction estimation;
Initialize output sets $\{C_r\} \leftarrow \emptyset$ and $\{v_r\} \leftarrow \emptyset$;
foreach $C \in \mathcal{C}$ **do**
  if $|C| \geq m_{\min}$ **then**
    $s(C) \leftarrow \dfrac{2}{|C|(|C|-1)} \sum_{i<j \in C} \left|\langle u_i, u_j\rangle\right|$;
    if τ not provided ***or*** $s(C) \geq \tau$ **then**
      Select a reference direction $u_{\text{ref}} \in \{u_i : i \in C\}$;
      $v \leftarrow \dfrac{1}{|C|} \sum_{i \in C} \operatorname{sgn}\left(\langle u_i, u_{\text{ref}}\rangle\right) u_i$;
      $v \leftarrow v/\|v\|$;
      Append C to $\{C_r\}$ and v to $\{v_r\}$;

return $\{C_r\}$, $\{v_r\}$