







Hard-Label Cryptanalytic Extraction of Neural Network Models

Yi Chen¹ , Xiaoyang Dong^{2,5} , Jian Guo³ , Yantian Shen⁴ ,
Anyu Wang^{1,5} , and Xiaoyun Wang^{1,5,6} 

¹ Institute for Advanced Study, Tsinghua University, Beijing, China,
chenyi2023@mails.tsinghua.edu.cn, {anyuwang, xiaoyunwang}@tsinghua.edu.cn

² Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University,
Beijing, China, xiaoyangdong@tsinghua.edu.cn

³ School of Physical and Mathematical Sciences, Nanyang Technological University,
Singapore, guojian@ntu.edu.sg

⁴ Department of Computer Science and Technology, Tsinghua University, Beijing,
China, shenyt22@mails.tsinghua.edu.cn

⁵ Zhongguancun Laboratory, Beijing, China

⁶ Shandong Key Laboratory of Artificial Intelligence Security, Shandong, China

Abstract. The machine learning problem of extracting neural network parameters has been proposed for nearly three decades. Functionally equivalent extraction is a crucial goal for research on this problem. When the adversary has access to the raw output of neural networks, various attacks, including those presented at CRYPTO 2020 and EURO-CRYPT 2024, have successfully achieved this goal. However, this goal is not achieved when neural networks operate under a hard-label setting where the raw output is inaccessible.

In this paper, we propose the first attack that theoretically achieves functionally equivalent extraction under the hard-label setting, which applies to ReLU neural networks. The effectiveness of our attack is validated through practical experiments on a wide range of ReLU neural networks, including neural networks trained on two real benchmarking datasets (MNIST, CIFAR10) widely used in computer vision. For a neural network consisting of 10^5 parameters, our attack only requires several hours on a single core.

Keywords: Cryptanalysis · ReLU Neural Networks · Functionally Equivalent Extraction · Hard-Label.

1 Introduction

Extracting all the parameters (including weights and biases) of a neural network (called victim model) is a long-standing open problem which is first proposed by cryptographers and mathematicians in the early nineties of the last century [2, 7], and has been widely studied by research groups from both industry and academia [1, 3, 4, 11, 14, 16, 18, 20].

In previous research [3, 4, 11, 14, 16, 18, 20], one of the most common attack scenarios is as follows. The victim model (denoted by f_θ where θ denotes the parameters) is made available as an Oracle \mathcal{O} , then the adversary generates inputs x to query \mathcal{O} and collects the feedback ζ to extract the parameters. This is similar to a cryptanalysis problem: θ is considered the secret key, and the adversary tries to recover the secret key θ , given the pairs (x, ζ) [4]. If f_θ contains m parameters (64-bit floating-point numbers), then the secret key θ contains $64 \times m$ bits, and the computation complexity of brute force searching is $2^{64 \times m}$.

Consider that there may be isomorphisms for neural networks, e.g., permutation and scaling for ReLU neural networks [18]. An important concept named *functionally equivalent extraction* is summarized and proposed in [11].

Functionally Equivalent Extraction. Denote by \mathcal{X} the input space of the victim model f_θ . Functionally equivalent extraction aims at generating a model $f_{\hat{\theta}}$ (i.e., the extracted model), such that $f_\theta(x) = f_{\hat{\theta}}(x)$ holds for all $x \in \mathcal{X}$, where $f_\theta(x)$ and $f_{\hat{\theta}}(x)$ are, respectively, the raw output of the victim model and the extracted model [11]. Such extracted model $f_{\hat{\theta}}$ is called the functionally equivalent model of f_θ (also say that f_θ and $f_{\hat{\theta}}$ are isomorphic [18]). Since $f_{\hat{\theta}}$ behaves the same as f_θ , the adversary can explore the properties of f_θ by taking $f_{\hat{\theta}}$ as a perfect substitute ¹.

Functionally equivalent extraction is hard [11]. Consider the isomorphisms (permutation and scaling) introduced in [18]. Scaling can change parameters and permutation does not. For a ReLU neural network f_θ that contains m parameters (64-bit floating-point numbers) and n neurons, from the perspective of the above cryptanalysis problem, even though *scaling can be applied to each neuron once*, the adversary still needs to recover $64 \times (m - n)$ secret key bits. Note that the case of $m \gg n$ is common for neural networks. For example, for the ReLU neural networks extracted by Carlini et al. at CRYPTO 2020 [4], the pairs (m, n) are (25120, 32), (100480, 128), (210, 20), (420, 40), and (4020, 60). Besides, even if we only recover a few bits (instead of 64 bits) of each parameter, the number of secret key bits to be recovered is still large, particularly in the case of large m .

When a functionally equivalent model $f_{\hat{\theta}}$ is obtained, the adversary can do more damage (e.g., adversarial attack [5]) or steal user privacy (e.g., property inference attack [9]). Thus, even though the cost is expensive, the adversary has the motivation to achieve functionally equivalent extraction.

Hard-label Setting. According to the taxonomy in [11], when the Oracle is queried, there are 5 types of feedback given by the Oracle: (1) label (the most likely class label, also called hard-label), (2) label and score (the most-likely class label and its probability score), (3) top-k scores, (4) all the label scores, (5) the raw output (i.e., $f_\theta(x)$). When the Oracle only returns the hard-label, we say

¹ Due to the isomorphisms introduced in [18], the parameters $\hat{\theta}$ of the extracted model may be different from that θ of the victim model, but it does not matter as long as $f_{\hat{\theta}}$ is the functionally equivalent model of f_θ .

that the victim model f_θ (i.e., neural networks in this paper) works under the hard-label setting [8].

To the best of our knowledge, there are no functionally equivalent extraction attacks that are based on the first four types of feedback so far. From the perspective of cryptanalysis, the raw output $f_\theta(x)$ is equivalent to the complete ciphertext corresponding to the plaintext (i.e., the query x). The other four types of feedback only reveal some properties of the ciphertext (raw output $f_\theta(x)$). For example, when $f_\theta(x) \in \mathbb{R}$, the hard-label only tells whether $f_\theta(x) > 0$ holds or not [8]. Among the five types of feedback, the raw output leaks the most information, while the hard-label (i.e., the first type of feedback) leaks the least [11].

Assuming that the feedback of the Oracle is the raw output, Jagielski et al. propose the first functionally equivalent extraction attack against ReLU neural networks with one hidden layer [11], which is extended to deeper neural networks in [18]. At CRYPTO 2020, Carlini et al. propose a differential extraction attack [4] that requires fewer queries than the attack in [18]. However, the differential extraction attack requires an exponential amount of time, which is addressed by Canales-Martínez et al. at EUROCRYPT 2024 [3]. Note that the extraction attacks in [3, 4, 18] are also based on the assumption that the feedback is the raw output. Due to the dependence on the raw output, all the authors in [3, 4, 11, 18], state that the hard-label setting (i.e., the feedback is the first type) is a defense against functionally equivalent extraction.

The above backgrounds lead to the question not studied before

Is it possible to achieve functionally equivalent extraction against neural network models under the hard-label setting?

1.1 Our Results and Techniques

Results. We have addressed this question head-on in this paper. In total, the answer is yes, and we propose the first functionally equivalent extraction attack against ReLU neural networks under the hard-label setting. Here, the definition of functionally equivalent extraction proposed in [4] is extended reasonably.

Definition 1 (Extended Functionally Equivalent Extraction) *The goal of the extended functionally equivalent extraction is to generate a model $f_{\hat{\theta}}$ (i.e., the extracted model), such that $f_{\hat{\theta}}(x) = c \times f_\theta(x)$ holds for all $x \in \mathcal{X}$, where $c > 0$ is a fixed constant, $f_\theta(x)$ and $f_{\hat{\theta}}(x)$ are, respectively, the raw output of the victim model and the extracted model. The extracted model $f_{\hat{\theta}}$ is the functionally equivalent model of the victim model f_θ .*

Since $f_{\hat{\theta}}(x) = c \times f_\theta(x)$ holds for all $x \in \mathcal{X}$, i.e., $f_{\hat{\theta}}$ is a simple scalar product of f_θ , the adversary still can explore the properties of the victim model f_θ by taking $f_{\hat{\theta}}$ as a perfect substitute. This is why we propose this extended definition. From the perspective of cryptanalysis, this extended definition allows the adversary not to guess the 64 bits of the constant c . To evaluate the efficacy of our model extraction attacks, and quantify the degree to which a model extraction

attack has succeeded in practice, we generalize the metric named (ε, δ) -functional equivalence proposed in [4].

Definition 2 (Extended (ε, δ) -Functional Equivalence) *Two models $f_{\hat{\theta}}$ and f_{θ} are (ε, δ) -functional equivalent on \mathcal{S} if there exists a fixed constant $c > 0$ such that*

$$\Pr_{x \in \mathcal{S}} [|f_{\hat{\theta}}(x) - c \times f_{\theta}(x)| \leq \varepsilon] \geq 1 - \delta$$

In this paper, we propose two model extraction attacks, one of which applies to 0-deep neural networks, and the other one applies to k -deep neural networks. The former attack is the basis of the latter attack. Our model extraction attacks theoretically achieve functionally equivalent extraction described in Definition 1, where the constant $c > 0$ is determined by the model parameter θ .

We have also performed numerous experiments on both untrained and trained neural networks, for verifying the effectiveness of our model extraction attacks in practice. The untrained neural networks are obtained by randomly generating model parameters. To fully verify our attacks, we also adopt two real benchmarking image datasets (i.e., MNIST and CIFAR10) widely used in computer vision, and train many classifiers (i.e., trained neural networks) as the victim model. Our model extraction attacks show good performances in experiments. The complete experiment results refer to Tables 1 and 2 in Section 7. The number of parameters of neural networks in our experiments is up to 10^5 , but the runtime of the proposed extraction attack on a single core is within several hours. Our experiment code is uploaded to GitHub (https://github.com/AI-Lab-Y/NN_cryptanalytic_extraction).

The analysis of the attack complexity is presented in Appendix B. For the extraction attack on k -deep neural networks, its query complexity is about $\mathcal{O}\left(d_0 \times 2^n \times \log_2^{\frac{1}{\epsilon}}\right)$, where d_0 and n are, respectively, the input dimension (i.e., the size of x) and the number of neurons, ϵ is a precision chosen by the adversary. The computation complexity is about $\mathcal{O}\left(n \times 2^{n^2+n+k}\right)$, where n is the number of neurons and k is the number of hidden layers. The computation complexity of our attack is much lower than that of brute-force searching.

Techniques. By introducing two new concepts, namely model activation pattern and model signature, we obtained some findings as follows. A ReLU neural network is composed of a certain number of affine transformations corresponding to model activation patterns. Each affine transformation leaks partial information about neural network parameters, which is determined by the corresponding model activation pattern. Most importantly, for a neural network that contains n neurons, $n + 1$ special model activation patterns will leak all the information about the neural network parameters.

Inspired by the above findings, we design a series of methods to find decision boundary points, recover the corresponding affine transformations, and further extract the neural network parameters. These methods compose the complete model extraction attacks.

Organization. The basic notations, threat model, attack goal and assumptions are introduced in Section 2. Section 3 introduces some auxiliary concepts. Then we introduce the overview of our model extraction attacks, the idealized model extraction attacks, and some refinements in practice in the following three sections respectively. Experiments are introduced in Section 7. At last, we present more discussions about our work and conclude this paper.

2 Preliminaries

2.1 Basic Definitions and Notations

This section presents some necessary definitions and notations.

Definition 3 (*k*-Deep Neural Network [4]) A *k*-deep neural network $f_\theta(x)$ is a function parameterized by θ that takes inputs from an input space \mathcal{X} and returns values in an output space \mathcal{Y} . The function $f: \mathcal{X} \rightarrow \mathcal{Y}$ is composed of alternating linear layers f_i and a non-linear activation function σ :

$$f = f_{k+1} \circ \sigma \circ \dots \circ \sigma \circ f_2 \circ \sigma \circ f_1. \quad (1)$$

In this paper, we exclusively study neural networks over $\mathcal{X} = \mathbb{R}^{d_0}$ and $\mathcal{Y} = \mathbb{R}^{d_{k+1}}$, where d_0 and d_{k+1} are positive integers. As in [3, 4], we only consider neural networks using the ReLU [15] activation function, given by $\sigma: x \mapsto \max(x, 0)$.

Definition 4 (Fully Connected Layer [4]) The *i*-th fully connected layer of a neural network is a function $f_i: \mathbb{R}^{d_{i-1}} \rightarrow \mathbb{R}^{d_i}$ given by the affine transformation

$$f_i(x) = A^{(i)}x + b^{(i)}. \quad (2)$$

where $A^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ is a $d_i \times d_{i-1}$ weight matrix, $b^{(i)} \in \mathbb{R}^{d_i}$ is a d_i -dimensional bias vector.

Definition 5 (Neuron [4]) A neuron is a function determined by the corresponding weight matrix, bias vector, and activation function. Formally, the *j*-th neuron of layer *i* is the function η given by

$$\eta(x) = \sigma \left(A_j^{(i)}x + b_j^{(i)} \right), \quad (3)$$

where $A_j^{(i)}$ and $b_j^{(i)}$ denote, respectively, the *j*-th row of $A^{(i)}$ and *j*-th coordinate of $b^{(i)}$. In a *k*-deep neural network, there are a total of $\sum_{i=1}^k d_i$ neurons.

Definition 6 (Neuron State [3]) Let $\mathcal{V}(\eta; x)$ denote the value that neuron η takes with $x \in \mathcal{X}$ before applying σ . If $\mathcal{V}(\eta; x) > 0$, then η is active, i.e., the neuron state is active. Otherwise, the neuron state is inactive². The state of the *j*-th neuron in layer *i* on input x is denoted by $\mathcal{P}_j^{(i)}(x) \in \mathbb{F}_2$. If $\mathcal{P}_j^{(i)}(x) = 1$, the neuron is active. If $\mathcal{P}_j^{(i)}(x) = 0$, the neuron is inactive.

² In [3, 4], the authors defined one more neuron state, namely critical, i.e., $\mathcal{V}(\eta; x) = 0$, which is a special inactive state since the output of neuron η is 0.

Definition 7 (Neural Network Architecture [4]) *The architecture of a fully connected neural network captures the structure of f_θ : (a) the number of layers, (b) the dimension d_i of each layer $i = 0, \dots, k+1$. We say that d_0 is the dimension of the input to the neural network, and d_{k+1} denotes the number of outputs of the neural network.*

Definition 8 (Neural Network Parameters [4]) *The parameters θ of a k -deep neural network f_θ are the concrete assignments to the weights $A^{(i)}$ and biases $b^{(i)}$ for $i \in \{1, 2, \dots, k+1\}$.*

When neural networks work under the hard-label setting, the raw output $f_\theta(x)$ is processed before being returned [8]. This paper considers the most common processing. The raw output $f_\theta(x) \in \mathbb{R}^{d_{k+1}}$ is first transformed into a category probability vector $\mathbf{P} \in \mathbb{R}^{d_{k+1}}$ by applying the Sigmoid (when $d_{k+1} = 1$) or Softmax (when $d_{k+1} > 1$) function to $f_\theta(x)$ [6]. Then, the category with the largest probability is returned as a hard-label. Definition 9 summarizes the hard-label and corresponding decision conditions on the raw output $f_\theta(x)$.

Definition 9 (Hard-Label) *Consider a k -deep neural network $f: \mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{Y} \in \mathbb{R}^{d_{k+1}}$. The hard-label (denoted by z) is related to the outputs $f_\theta(x)$. When $d_{k+1} = 1$, the hard-label $z(f_\theta(x))$ is computed as*

$$z(f_\theta(x)) = \begin{cases} 1, & \text{if } f_\theta(x) > 0, \\ 0, & \text{if } f_\theta(x) \leq 0. \end{cases} \quad (4)$$

*When $d_{k+1} > 1$, the output $f_\theta(x)$ is a d_{k+1} -dimensional vector. The hard-label $z(f_\theta(x))$ is the coordinate of the maximum of $f_\theta(x)$.*³

2.2 Adversarial Goals and Assumptions

There are two parties in a model extraction attack: the oracle \mathcal{O} who possesses the neural network $f_\theta(x)$, and the adversary who generates queries x to the Oracle. Under the hard-label setting, the Oracle \mathcal{O} returns the hard-label $z(f_\theta(x))$ in Definition 9.

Definition 10 (Model Parameter Extraction Attack) *A model parameter extraction attack receives Oracle access to a parameterized function f_θ (i.e., a k -deep neural network in our paper) and the architecture of f_θ , and returns a set of parameters $\hat{\theta}$ with the goal that $f_{\hat{\theta}}(x)$ is as similar as possible to $c \times f_\theta(x)$ where $c > 0$ is a fixed constant.*

In this paper, we use the $\hat{\cdot}$ symbol to indicate an extracted parameter. For example, θ is the parameters of the victim model f_θ , and $\hat{\theta}$ stands for the parameters of the extracted model $f_{\hat{\theta}}$.

³ If there are ties, i.e., multiple items of $f_\theta(x)$ share the same maximum, the hard-label is the smallest one of the coordinates of these items.

Assumptions. We make the following assumptions of the Oracle \mathcal{O} and the capabilities of the attacker:

- **Architecture knowledge.** We require knowledge of the neural network architecture.
- **Full-domain inputs.** We can feed arbitrary inputs from $\mathcal{X} = \mathbb{R}^{d_0}$.
- **Precise computations.** f_θ is specified and evaluated using 64-bit floating-point arithmetic.
- **Scalar outputs.** The output dimensionality is 1, i.e., $\mathcal{Y} = \mathbb{R}$.⁴
- **ReLU Activations.** All activation functions (σ 's) are the ReLU function.

Compared with the work in [4], we remove the assumption of requiring the raw output $f_\theta(x)$ of the neural network. Now, after querying the Oracle \mathcal{O} , the attacker obtains the hard-label $z(f_\theta(x))$. In other words, the attacker only knows whether $f_\theta(x) > 0$ holds or not.

3 Auxiliary Concepts

To help understand our attacks, this paper proposes some auxiliary concepts.

3.1 Model Activation Pattern

To describe all the neuron states, we introduce a new concept named *Model Activation Pattern*.

Definition 11 (Model Activation Pattern) Consider a k -deep neural network f_θ with $n = \sum_{i=1}^k d_i$ neurons. The model activation pattern of f_θ over an input $x \in \mathcal{X}$ is a global description of the n neuron states, and is denoted by $\mathcal{P}(x) = (\mathcal{P}^{(1)}(x), \dots, \mathcal{P}^{(k)}(x))$ where $\mathcal{P}^{(i)}(x) \in \mathbb{F}_2^{d_i}$ is the concatenation of d_i neuron states (i.e., $\mathcal{P}_j^{(i)}(x), i \in \{1, \dots, d_i\}$) in layer i .

In the rest of this paper, the notations $\mathcal{P}_j^{(i)}(x)$, $\mathcal{P}^{(i)}(x)$, and $\mathcal{P}(x)$ are simplified as $\mathcal{P}_j^{(i)}$, $\mathcal{P}^{(i)}$, and \mathcal{P} respectively, when the meaning is clear in context. Besides, $\mathcal{P}^{(i)} \in \mathbb{F}_2^{d_i}$ is represented by a d_i -bit integer. For example, $\mathcal{P}^{(i)} = 2^{j-1}$ means that only the j -th neuron in layer i is active, and $\mathcal{P}^{(i)} = 2^{d_i} - 1$ means that all the d_i neurons are active.

When the model activation pattern is known, one can precisely determine which neural network parameters influence the output $f_\theta(x)$. Consider the j -th neuron η in layer i . Due to the ReLU activation function, if the neuron state is *inactive*, neuron η does not influence the output $f_\theta(x)$. As a result, all the weights $A_{?,j}^{(i+1)}$ and $A_{j,?}^{(i)}$ (i.e., the elements of the j -th column of $A^{(i+1)}$, and the j -th row of $A^{(i)}$ respectively) and the bias $b_j^{(i)}$ do not affect the output $f_\theta(x)$.

⁴ This assumption is fundamental to our work. Our attack only applies to the case of scalar outputs.

Special ‘neuron’. For the convenience of introducing model extraction attacks later, we regard the input $x \in \mathbb{R}^{d_0}$ and the output $f_\theta(x) \in \mathbb{R}$ as, respectively, d_0 and 1 special ‘neurons’ that are always active. So we adopt two extra notations $\mathcal{P}^{(0)} = 2^{d_0} - 1$ and $\mathcal{P}^{(k+1)} = 2^1 - 1 = 1$, for describing the states of the special $d_0 + 1$ ‘neurons’. But if not necessary, we will omit the two notations.

3.2 Model Signature

Consider a k -deep neural network f_θ . For an input $x \in \mathcal{X}$, f_θ can be described as an affine transformation

$$\begin{aligned} f_\theta(x) &= A^{(k+1)} \dots \left(I_{\mathcal{P}}^{(2)} \left(A^{(2)} \left(I_{\mathcal{P}}^{(1)} \left(A^{(1)} x + b^{(1)} \right) \right) + b^{(2)} \right) \right) \dots + b^{(k+1)} \\ &= A^{(k+1)} I_{\mathcal{P}}^{(k)} A^{(k)} \dots I_{\mathcal{P}}^{(2)} A^{(2)} I_{\mathcal{P}}^{(1)} A^{(1)} x + B_{\mathcal{P}} = \Gamma_{\mathcal{P}} x + B_{\mathcal{P}}, \end{aligned} \quad (5)$$

where \mathcal{P} is the model activation pattern over x , $\Gamma_{\mathcal{P}} \in \mathbb{R}^{d_0}$, and $B_{\mathcal{P}} \in \mathbb{R}$. Here, $I_{\mathcal{P}}^{(i)} \in \mathbb{R}^{d_i \times d_i}$ are 0-1 diagonal matrices with a 0 on the diagonal’s j -th entry when the neuron state $\mathcal{P}_j^{(i)}$ is 0, and 1 when $\mathcal{P}_j^{(i)} = 1$.

The affine transformation is denoted by a tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$. Except for \mathcal{P} , the value of the tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ is only determined by the neural network parameters, i.e., $A^{(i)}$ and $b^{(i)}$, $i \in \{1, \dots, k+1\}$. Once the value of any neural network parameters is changed, the value of the tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ corresponding to some \mathcal{P} ’s will change too⁵. Therefore, we regard the set of all the possible tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ as a unique *model signature* of the neural network.

Definition 12 (Model Signature) For a k -deep neural network $f_\theta(x)$, the model signature denoted by \mathcal{S}_θ is the set of affine transformations

$$\mathcal{S}_\theta = \{(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \text{ for all the } \mathcal{P} \text{’s}\}.$$

In [3], Canales-Martínez et al. use the term ‘signature’ to describe the weights related to a neuron, which is different from the model signature. Except for the model signature, we propose another important concept, namely *normalized model signature*.

Definition 13 (Normalized Model Signature) Consider a victim model f_θ and its model signature $\mathcal{S}_\theta = \{(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \text{ for all the } \mathcal{P} \text{’s}\}$. Denote by $\Gamma_{\mathcal{P},j}$ the j -th element of $\Gamma_{\mathcal{P}}$ for $j \in \{1, \dots, d_0\}$. Divide the set of \mathcal{P} ’s into two subsets \mathcal{Q}_1 and \mathcal{Q}_2 . For each $\mathcal{P} \in \mathcal{Q}_1$, $\Gamma_{\mathcal{P},j} = 0$ for $j \in \{1, \dots, d_0\}$. For each $\mathcal{P} \in \mathcal{Q}_2$, there is at least one non-zero element in $\Gamma_{\mathcal{P}}$, without loss of generality, assume that $\Gamma_{\mathcal{P},1} \neq 0$. Let \mathcal{S}_θ^N be the following set

$$\mathcal{S}_\theta^N = \left\{ (\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \text{ for } \mathcal{P} \in \mathcal{Q}_1, \left(\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}, \frac{B_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|} \right) \text{ for } \mathcal{P} \in \mathcal{Q}_2 \right\}.$$

The set \mathcal{S}_θ^N is the normalized model signature of f_θ .

⁵ We do not consider the case of some neurons being always inactive, since such neurons are redundant and usually deleted by various network pruning methods (e.g., [10]) before the neural network is deployed as a prediction service.

Shortly, the difference between the normalized model signature \mathcal{S}_θ^N and the initial model signature \mathcal{S}_θ is as follows. For each $\mathcal{P} \in \mathcal{Q}_2$, i.e., there is at least one non-zero element in $\Gamma_{\mathcal{P}}$ (without loss of generality, assume that the first element is non-zero, i.e., $\Gamma_{\mathcal{P},1} \neq 0$), we transform the parameter tuple into $\left(\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}, \frac{B_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}\right)$.

In our attacks, the normalized model signature plays two important roles. First, the recovery of all the weights $A^{(i)}$ relies on the subset \mathcal{Q}_2 . Second, our attacks will produce many extracted models during the attack process while at most only one is the functionally equivalent model of f_θ , and the normalized model signature is used to filter functionally inequivalent models.

3.3 Decision Boundary Point

Our attacks exploit a special class of inputs named *Decision Boundary Points*.

Definition 14 (Decision Boundary Point) *Consider a neural network f_θ . If an input x makes $f_\theta(x) = 0$ hold, x is a decision boundary point.*

The extraction attacks presented at CRYPTO 2020 [4] and EUROCRYPT 2024 [3] exploit a class of inputs, namely critical points. Fig. 1 shows the difference between critical points and decision boundary points.

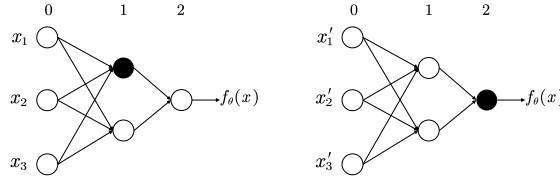


Fig. 1. Left: the critical point $x = [x_1, x_2, x_3]^\top$ makes the output of one neuron (e.g., the solid black circle) 0. Right: the decision boundary point $x' = [x'_1, x'_2, x'_3]^\top$ makes the output of the neural network 0.

Critical points leak information on the neuron states, i.e., whether the output of a neuron is 0, which is the core reason why the differential extraction attack can efficiently extract model parameters [4]. As a comparison, decision boundary points do not leak information on the neuron states.

Finding critical points relies on computing partial derivatives based on the raw output $f_\theta(x)$, refer to the work in [3, 4]. Thus, under the hard-label setting, we can not exploit critical points.

4 Overview of Our Cryptanalytic Extraction Attacks

Under the hard-label setting, i.e., the Oracle returns the most likely class $z(f_\theta(x))$ instead of the raw output $f_\theta(x)$, only decision boundary points x will leak the value of $f_\theta(x)$, since $f_\theta(x) = 0$. Motivated by this truth, our cryptanalytic extraction attacks focus on decision boundary points.

Attack Process. At a high level, the complete attack contains five steps.

- **Step 1: collect decision boundary points.** The algorithm for finding decision boundary points will be introduced in Section 6.1. Suppose that M decision boundary points are collected.
- **Step 2: recover the normalized model signature.** Recover the tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ corresponding to the M decision boundary points. After filtering duplicate tuples, regard the set of the remaining tuples as the (partial) normalized model signature \mathcal{S}_{θ}^N . Suppose that the size of \mathcal{Q}_2 is N , refer to Definition 13. It means that there are N decision boundary points that can be used to recover weights $A^{(i)}$.
- **Step 3: recover weights layer by layer.** Suppose that there are $n = \sum_{i=1}^k d_i$ neurons in the neural network. Randomly choose $n + 1$ out of N decision boundary points each time, assign a specific model activation pattern \mathcal{P} to each selected decision boundary point, and recover the weights $A^{(1)}, \dots, A^{(k+1)}$.
- **Step 4: recover all the biases.** Based on recovered weights, recover all the biases $b^{(i)}, i \in \{1, \dots, k + 1\}$ simultaneously.
- **Step 5: filter functionally inequivalent models.** As long as $N \geq n + 1$ holds, we will obtain many extracted models, but it is expected that at most only one is the functionally equivalent model. Thus, we filter functionally inequivalent models in this step.

Some functionally inequivalent models may not be filtered. For each surviving extracted model, we test the *Prediction Matching Ratio* (PMR, introduced in Section 6.3) over randomly generated inputs, and take the one with the highest PMR as the final candidate.

In Step 2, we recover the tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ by the extraction attack on 0-deep neural networks. In Step 3, for layer $i > 1$, the weight vector $A_j^{(i)}$ of the j -th neuron ($j \in \{1, \dots, d_i\}$) is recovered by solving a system of linear equations. For layer 1, except for selecting d_1 decision boundary points, the recovery of the weights $A^{(1)}$ does not use any extra techniques. In Step 4, all the biases are recovered by solving a system of linear equations.

5 Idealized Hard-Label Model Extraction Attack

This section introduces $(0, 0)$ -functionally equivalent model extraction attacks under the hard-label setting, which assumes infinite precision arithmetic and recovers the functionally equivalent model. We first introduce the 0-deep neural network extraction attack, which is used in the k -deep neural network extraction attack to recover the normalized model signature.

Note that this section only (partially) involves Steps 2, 3, and 4 introduced in Section 4. In the next section, we introduce the remaining steps and refine the idealized attacks to work with finite precision.

5.1 Zero-Deep Neural Network Extraction

According to Definition 3, zero-deep neural networks are affine functions $f_\theta(x) \equiv A^{(1)} \cdot x + b^{(1)}$ where $A^{(1)} \in \mathbb{R}^{d_0}$, and $b^{(1)} \in \mathbb{R}$. Let $A^{(1)} = [w_1^{(1)}, \dots, w_{d_0}^{(1)}]$, and $x = [x_1, x_2, \dots, x_{d_0}]^\top$. The model signature is $\mathcal{S}_\theta = (A^{(1)}, b^{(1)})$.

Our extraction attack is based on a decision boundary point x (i.e., $f_\theta(x) = 0$), and composed of 3 steps: (1) recover weight signs, i.e., the sign of $w_i^{(1)}$; (2) recover weights $w_i^{(1)}$; (3) recover bias $b^{(1)}$.

Recover Weight Signs. Denote by $e_i \in \mathbb{R}^{d_0}$ the basis vector where only the i -th element is 1 and other elements are 0.

Let the decision boundary point x move along the direction e_i , $i \in \{1, \dots, d_0\}$, and the moving stride is $s \in \mathbb{R}$ where $|s| > 0$. Query the Oracle and obtain the hard-label $z(f(x + se_i))$, then the sign of $w_i^{(1)}$ is

$$\text{sign}(w_i^{(1)}) = \begin{cases} 1, & \text{if } s > 0 \text{ and } z(f_\theta(x + se_i)) = 1, \\ -1, & \text{if } s < 0 \text{ and } z(f_\theta(x + se_i)) = 1. \end{cases} \quad (6)$$

When $z(f(x + se_i)) = 1$, we have $f(x + se_i) > 0$, i.e., $w_i^{(1)} \times s > 0$. Thus, the sign of $w_i^{(1)}$ is the same as that of s . If $z(f(x + se_i)) = 0$ always holds, no matter if s is positive or negative, then we have $w_i^{(1)} = 0$.

Recover Weights. Without loss of generality, assume that $w_1^{(1)} \neq 0$.

At first, let the decision boundary point x move along e_1 with a moving stride s_1 , such that the hard-label of the new point $x + s_1 e_1$ is 1, i.e., $z(f(x + s_1 e_1)) = 1$. Then, let the new point $x + s_1 e_1$ move along e_i with a moving stride s_i where $i \neq 1$ and $w_i^{(1)} \neq 0$, such that $x + s_1 e_1 + s_i e_i$ is a decision boundary point too. As a result, we have

$$s_1 w_1^{(1)} + s_i w_i^{(1)} = 0, \quad (7)$$

and obtain the weight ratio $\frac{w_i^{(1)}}{w_1^{(1)}}$. Since the signs of $w_i^{(1)}$ are known, the final extracted weights are

$$\widehat{A}^{(1)} = \left[\frac{w_1^{(1)}}{|w_1^{(1)}|}, \frac{w_2^{(1)}}{|w_1^{(1)}|}, \dots, \frac{w_{d_0}^{(1)}}{|w_1^{(1)}|} \right]. \quad (8)$$

Recover Bias. The extracted bias is $\widehat{b}^{(1)} = -\widehat{A}^{(1)} \cdot x = \frac{b^{(1)}}{|w_1^{(1)}|}$.

Thus, the model signature of $f_{\widehat{\theta}}$ is $\mathcal{S}_{\widehat{\theta}} = \left(\frac{A^{(1)}}{|w_1^{(1)}|}, \frac{b^{(1)}}{|w_1^{(1)}|} \right)$, and $f_{\widehat{\theta}}(x) = \frac{f(x)}{|w_1^{(1)}|}$.

Remark 1. In [14], the authors propose different methods to extract the parameters of linear functions $f_\theta(x) = A^{(1)} \cdot x$. Since this paper mainly focuses on the extraction attack on k -deep neural networks, we do not deeply compare our attack with the methods in [14].

5.2 k -Deep Neural Network Extraction

Basing the 0-deep neural network extraction attack, we develop an extraction attack on k -deep neural networks. Recall that, the expression of k -deep neural networks is

$$\begin{aligned} f_\theta(x) &= A^{(k+1)} \dots \left(I_{\mathcal{P}}^{(2)} \left(A^{(2)} \left(I_{\mathcal{P}}^{(1)} \left(A^{(1)}x + b^{(1)} \right) \right) + b^{(2)} \right) \right) \dots + b^{(k+1)} \\ &= \Gamma_{\mathcal{P}}x + B_{\mathcal{P}} \end{aligned} \quad (9)$$

where the model activation pattern is $\mathcal{P} = (\mathcal{P}^{(0)}, \mathcal{P}^{(1)}, \dots, \mathcal{P}^{(k)}, \mathcal{P}^{(k+1)})$ and

$$\Gamma_{\mathcal{P}} = A^{(k+1)} I_{\mathcal{P}}^{(k)} A^{(k)} \dots I_{\mathcal{P}}^{(2)} A^{(2)} I_{\mathcal{P}}^{(1)} A^{(1)}. \quad (10)$$

Notations. Our attack recovers weights layer by layer. Assuming that the weights of the first $i - 1$ layers have been recovered and we are trying to recover $A^{(i)}$ where $i \in \{1, \dots, k + 1\}$, we describe k -deep neural networks as:

$$f_\theta(x) = \Gamma_{\mathcal{P}}x + B_{\mathcal{P}} = \mathcal{G}^{(i)} A^{(i)} C^{(i-1)}x + B_{\mathcal{P}}, \quad (11)$$

where $\mathcal{G}^{(i)} \in \mathbb{R}^{d_i}$ and $C^{(i-1)} \in \mathbb{R}^{d_{i-1} \times d_0}$ are, respectively, related to the unrecovered part (excluding $A^{(i)}$) and recovered part of the neural network f_θ .

The values of $\mathcal{G}^{(i)}$ and $C^{(i-1)}$ are

$$\begin{aligned} \mathcal{G}^{(i)} &= \begin{cases} A^{(k+1)} I_{\mathcal{P}}^{(k)} A^{(k)} \dots I_{\mathcal{P}}^{(i+1)} A^{(i+1)} I_{\mathcal{P}}^{(i)}, & \text{if } i \in \{1, \dots, k\} \\ 1, & \text{if } i = k + 1 \end{cases} \\ C^{(i-1)} &= \begin{cases} I, & \text{if } i = 1 \\ I_{\mathcal{P}}^{(i-1)} A^{(i-1)} \dots I_{\mathcal{P}}^{(1)} A^{(1)}, & \text{if } i \in \{2, \dots, k + 1\} \end{cases} \end{aligned} \quad (12)$$

where $C^{(0)} = I \in \mathbb{R}^{d_0 \times d_0}$ is a diagonal matrix with a 1 on each diagonal entry.

Core Idea of Recovering Weights Layer by Layer. To better grasp the attack details presented later, we first introduce the core idea of recovering weights layer by layer. Assuming that the extracted weights of the first $i - 1$ layers are known, i.e., $\widehat{A}^{(1)}, \dots, \widehat{A}^{(i-1)}$ are known, we try to recover the weights in layer i .

To obtain the weight vector $\widehat{A}_j^{(i)}$ of the j -th neuron (denoted by $\eta_j^{(i)}$) in layer $i \in \{1, \dots, k + 1\}$ ⁶, we exploit a decision boundary point with the model activation pattern $\mathcal{P} = (\mathcal{P}^{(0)}, \mathcal{P}^{(1)}, \dots, \mathcal{P}^{(k)}, \mathcal{P}^{(k+1)})$ where

$$\mathcal{P}^{(i-1)} = 2^{d_{i-1}} - 1, \quad \mathcal{P}^{(i)} = 2^{j-1}. \quad (13)$$

It means that, in layer i , only the j -th neuron is active, and all the d_{i-1} neurons in layer $i - 1$ are active. Fig 2 shows a schematic diagram under this scenario.

Since $\mathcal{P}^{(i)} = 2^{j-1}$, all the $k - i$ layers starting from layer $i + 1$ collapse into a direct connection from $\eta_j^{(i)}$ to the output $f_\theta(x)$. The weight of this connection

⁶ when $i = k + 1$, it means that we are trying to recover the weights $A^{(k+1)}$.

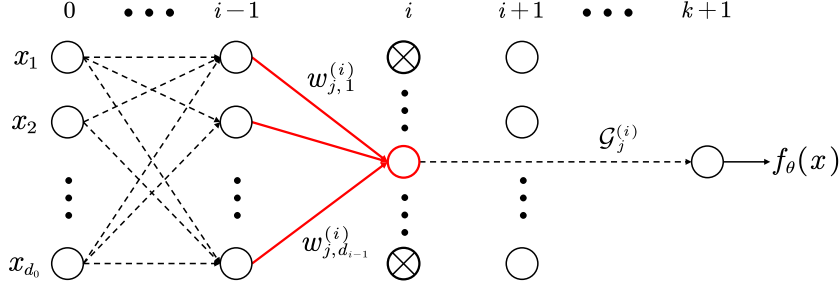


Fig. 2. The core idea of recovering the weight vector of the j -th neuron in layer i . Let $x = [x_1, \dots, x_{d_0}]^\top$ be a decision boundary point with $\mathcal{P}^{(i-1)} = 2^{d_{i-1}} - 1$, $\mathcal{P}^{(i)} = 2^{d_i} - 1$, i.e., in layer i , only the j -th neuron (the red hollow circle) is active, and in layer $i-1$, all the neurons are active. The first $i-1$ layers have been extracted, and collapse into one layer. All the layers starting from layer $i+1$ collapse into a direct connection between the j -th neuron in layer i and the final output.

is $\mathcal{G}_j^{(i)}$, i.e., the j -th element of $\mathcal{G}^{(i)}$ (see Eq. (12)). The expression (see Eq. (11)) of the k -deep neural network further becomes

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

where $\mathcal{G}_j^{(i)} \in \mathbb{R}$ and $A_j^{(i)} \cdot C^{(i-1)} \in \mathbb{R}^{d_0}$.

In Step 2 (see Section 4), applying the extraction attack on zero-deep neural networks, we can obtain the tuple $\left(\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}, \frac{B_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}\right)$ where

$$\Gamma_{\mathcal{P}} = \mathcal{G}_j^{(i)} A_j^{(i)} \cdot C^{(i-1)}, \quad \Gamma_{\mathcal{P},v} = \mathcal{G}_j^{(i)} A_j^{(i)} \cdot C_{?,v}^{(i-1)}. \quad (14)$$

Here the symbol $C_{?,v}^{(i-1)}$ stands for the v -th column vector of $C^{(i-1)}$.

According to Eq. (14), the value of each element of $\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}$ is not related to the absolute value of $\mathcal{G}_j^{(i)}$, i.e., the unrecovered part does not affect the affine transformation. Then, basing the vector $\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|}$ and the extracted weights $\widehat{A}^{(1)}, \dots, \widehat{A}^{(i-1)}$, we build a system of linear equations and solve it to obtain $\widehat{A}_j^{(i)}$. Next, we introduce more attack details.

Recover Weights in Layer 1. To recover the weight vector of the j -th neuron in layer 1, we exploit the model activation pattern \mathcal{P} where

$$\mathcal{P}^{(1)} = 2^{j-1}; \quad \mathcal{P}^{(i)} = 2^{d_i} - 1, \quad \text{for } i \in \{0, 2, 3, \dots, k+1\}. \quad (15)$$

It means that, in layer 1, only the j -th neuron is active, and all the neurons in other layers are active.

Under this model activation pattern, according to Eq. (12), we have

$$\mathcal{G}^{(1)} = A^{(k+1)} A^{(k)} \dots A^{(2)} I_{\mathcal{P}}^{(1)}. \quad (16)$$

Now, the expression of the k -deep neural network is

$$f_\theta(x) = \mathcal{G}_j^{(1)} \left(A_j^{(1)} x + b_j^{(1)} \right) + B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})} = \mathcal{G}_j^{(1)} A_j^{(1)} x + B_{\mathcal{P}} \quad (17)$$

where $A_j^{(1)} = [w_{j,1}^{(1)}, \dots, w_{j,d_0}^{(1)}]$, $\mathcal{G}_j^{(1)} \in \mathbb{R}$ is the j -th element of $\mathcal{G}^{(1)}$. As for $B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})} \in \mathbb{R}$, it is a constant determined by $(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})$. In other words, when $\mathcal{P}^{(1)}$ changes, the value of $B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})}$ does not change.

Recall that, in Step 2, we have recovered the following weight vector

$$\frac{\Gamma_{\mathcal{P}}}{|\Gamma_{\mathcal{P},1}|} = \left[\frac{\mathcal{G}_j^{(1)} w_{j,1}^{(1)}}{|\mathcal{G}_j^{(1)} w_{j,1}^{(1)}|}, \dots, \frac{\mathcal{G}_j^{(1)} w_{j,d_0}^{(1)}}{|\mathcal{G}_j^{(1)} w_{j,1}^{(1)}|} \right], j \in \{1, \dots, d_1\}. \quad (18)$$

In this step, our target is to obtain $\widehat{A}_j^{(1)}$ where

$$\widehat{A}_j^{(1)} = [\widehat{w}_{j,1}^1, \dots, \widehat{w}_{j,d_0}^1] = \left[\frac{w_{j,1}^1}{|w_{j,1}^1|}, \dots, \frac{w_{j,d_0}^1}{|w_{j,1}^1|} \right], j \in \{1, \dots, d_1\}. \quad (19)$$

Therefore, we need to determine d_1 signs, i.e., the signs of $\mathcal{G}_j^{(1)}$ for $\mathcal{P}^{(1)} = 2^{j-1}$ where $j \in \{1, \dots, d_1\}$.

Since $A_j^{(1)} x + b_j^{(1)} > 0$, we know that $\mathcal{G}_j^{(1)} \times B_{(\mathcal{P}^{(2)}, \dots, \mathcal{P}^{(k)})} < 0$ holds for $j \in \{1, \dots, d_1\}$, which tells us that the above d_1 signs are the *same*. Thus, by guessing 1 sign, i.e., the sign of $\mathcal{G}_j^{(1)}$ for $\mathcal{P}^{(1)} \in \{2^{1-1}, \dots, 2^{d_1-1}\}$, we obtain d_1 weight vectors presented in Eq. (19).

Recover Weights in Layer i ($i > 1$). To recover the weight vector of the j -th neuron in layer i , we exploit the model activation pattern \mathcal{P} where

$$\mathcal{P}^{(i)} = 2^{j-1}; \mathcal{P}^{(q)} = 2^{d_q} - 1, \text{ for } q \in \{0, \dots, i-1, i+1, \dots, k+1\}. \quad (20)$$

It means that, in layer i , only the j -th neuron is active, and all the neurons in other layers are active.

Under this model activation pattern, according to Eq. (12), we have

$$\begin{aligned} \mathcal{G}^{(i)} &= \begin{cases} A^{(k+1)} A^{(k)} \dots A^{(i+2)} A^{(i+1)} I_{\mathcal{P}}^{(i)}, & \text{if } i \in \{2, \dots, k\}, \\ 1, & \text{if } i = k+1, \end{cases} \\ C^{(i-1)} &= A^{(i-1)} A^{(i-2)} \dots A^{(1)}, \text{ if } i \in \{2, \dots, k+1\}. \end{aligned} \quad (21)$$

Now, the expression of k -deep neural networks becomes

$$\begin{aligned} f_\theta(x) &= \mathcal{G}_j^{(i)} \left(A_j^{(i)} C^{(i-1)} x + B_{(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(i)})} \right) + B_{(\mathcal{P}^{(i+1)}, \dots, \mathcal{P}^{(k)})} \\ &= \mathcal{G}_j^{(i)} A_j^{(i)} C^{(i-1)} x + B_{\mathcal{P}}, \end{aligned} \quad (22)$$

When $d_0 \geq d_{i-1}$ ⁷, we obtain $\widehat{A}_j^{(i)} = [\widehat{w}_{j,1}^{(i)}, \dots, \widehat{w}_{j,d_{i-1}}^{(i)}]$ by solving the above system of linear equations. Lemma 1 summarizes the expression of extracted weight vectors $\widehat{A}_j^{(i)}, j \in \{1, \dots, d_i\}, i \in \{2, \dots, k+1\}$.

Lemma 1. *Based on the system of linear equations presented in Eq. (26), for $i \in \{2, \dots, k+1\}$ and $j \in \{1, \dots, d_i\}$, the extracted weight vector $\widehat{A}_j^{(i)} = [\widehat{w}_{j,1}^{(i)}, \dots, \widehat{w}_{j,d_{i-1}}^{(i)}]$ is*

$$\widehat{A}_j^{(i)} = \left[\frac{w_{j,1}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{1,v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{w_{j,d_{i-1}}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{d_{i-1},v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], \quad (27)$$

where $C_{v,1}^{(q)} = A_v^{(q)} A^{(q-1)} \dots A^{(2)} [A_{1,1}^{(1)}, \dots, A_{d_1,1}^{(1)}]^\top$.

Proof. The proof refers to Appendix A.

In Lemma 1, for the consistency of the mathematical symbols, the weights $A^{(k+1)}$ are denoted by $[w_{1,1}^{(k+1)}, \dots, w_{1,d_k}^{(k+1)}]$ instead of $[w_1^{(k+1)}, \dots, w_{d_k}^{(k+1)}]$.

Recover All the Biases. Since $\widehat{A}^{(i)}$ for $i \in \{1, \dots, k+1\}$ have been obtained, we can extract all the biases by solving a system of linear equations.

Concretely, for the $\sum_{i=1}^k d_i + 1$ decision boundary points, $f_{\widehat{\theta}}(x) = 0$ should hold. Thus, we build a system of linear equations: $f_{\widehat{\theta}}(x) = 0$ where the expression of $f_{\widehat{\theta}}(x)$ refers to Eq. (9). Combining with Lemma 1, by solving the above system, we will obtain

$$\begin{cases} \widehat{b}^{(i)} = \left[\frac{b_1^{(i)}}{\left| \sum_{v=1}^{d_{i-1}} w_{1,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{b_{d_i}^{(i)}}{\left| \sum_{v=1}^{d_{i-1}} w_{d_i,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], i \in \{1, \dots, k\} \\ \widehat{b}^{(k+1)} = \frac{b^{(k+1)}}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|}. \end{cases} \quad (28)$$

Based on the extracted neural network parameters (see Eq. (19), Eq. (27), and Eq. (28)), the model signature of the extracted model $f_{\widehat{\theta}}$ is

$$\mathcal{S}_{\widehat{\theta}} = \left\{ (\widehat{\Gamma}_{\mathcal{P}}, \widehat{B}_{\mathcal{P}}) = \left(\frac{\Gamma_{\mathcal{P}}}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|}, \frac{B_{\mathcal{P}}}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|} \right) \text{ for all the } \mathcal{P}'\text{s} \right\}.$$

Consider the j -th neuron η in layer i . For an input $x \in \mathcal{X}$, denote by $h(\eta; x)$ the output of the neuron of the victim model. Based on the extracted neural network parameters (see Eq. (19), Eq. (27), and Eq. (28)), the output of the neuron of the extracted model is $\frac{h(\eta; x)}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}$. At the same time, all the

⁷ The case of $d_0 \geq d_{i-1}$ is common in various applications, particularly in computer vision [9, 13, 17], since the dimensions of images or videos are often large.

weights $w_{?,j}^{(i+1)}$ in layer $i + 1$ are increased by a factor of $\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|$. Thus, for the victim model and extracted model, the j -th neuron in layer i has the same influence on all the neurons in layer $i + 1$. As a result, for any $x \in \mathcal{X}$, the model activation pattern of the victim model is the same as that of the extracted model. Combining with $\mathcal{S}_{\hat{\theta}}$, for all $x \in \mathcal{X}$, we have

$$f_{\hat{\theta}}(x) = \frac{1}{\left| \sum_{v=1}^{d_k} w_v^{(k+1)} C_{v,1}^{(k)} \right|} \times f_{\theta}(x). \quad (29)$$

In Appendix C, we apply the extraction attack on 1-deep neural networks and directly present the extracted model, which helps further understand our attack.

Remark 2. Except for the $n + 1$ model activation patterns as shown in Eq. (15) and Eq. (20), the adversary could choose a new set of $n + 1$ model activation patterns. The reason is as follows. Consider the recovery of the weight vector of the j -th neuron in layer i , and look at Fig. 2 again. Our attack only requires that: (1) in layer i , only the j -th neuron is active; (2) in layer $i - 1$, all the d_{i-1} neurons are active. The neuron states in other layers do not affect the attack. Thus, there are more options for the $n + 1$ model activation patterns, and the rationale does not change.

Discussion on The Computation Complexity. Once $n + 1$ decision boundary points and k sign guesses are selected, to obtain an extracted model, we just need to solve $n + 2 - d_1$ systems of linear equations. However, since the model activation pattern of a decision boundary point is unknown, we have to traverse all the possible combinations of $n + 1$ decision boundary points (see Step 3 in Section 4), which is the bottleneck of the total computation complexity. The complete analysis of the attack complexity is presented in Appendix B.

6 Instantiating the Extraction Attack in Practice

Recall that, the complete extraction attack contains 5 steps introduced in Section 4. To obtain a functionally equivalent model, the adversary also needs three auxiliary techniques: *finding decision boundary points* (related to Steps 1 and 2), *filtering duplicate affine transformations* (related to Step 2), and *filtering functionally inequivalent models* (related to Step 5).

The idealized extraction attack introduced in Section 5 relies on decision boundary points x that make $f_{\theta}(x) = 0$ strictly hold. This section will propose a binary searching method to find decision boundary points under the hard-label setting. Under finite precision, it is hard to find decision boundary points x that make $f_{\theta}(x) = 0$ strictly hold. Therefore, the proposed method returns input points x close to the decision hyperplane as decision boundary points. As a result, the remaining two techniques need to consider the influence of finite precision. This ensures our model extraction attacks work in practice, for producing a $(\varepsilon, 0)$ -functionally equivalent model.

6.1 Finding Decision Boundary Points

Let us see how to find decision boundary points under the hard-label setting. Fig. 3 shows a schematic diagram in a 2-dimensional input space.

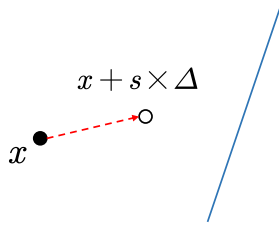


Fig. 3. A schematic diagram of finding decision boundary points. The blue solid line stands for the decision hyperplane composed of decision boundary points. The red dashed line stands for a direction vector $\Delta \in \mathbb{R}^{d_0}$. The starting point $x \in \mathbb{R}^{d_0}$ (i.e., the solid black circle) moves along the direction Δ , and arrives at $x + s \times \Delta$ (i.e., the hollow black circle) where $s \in \mathbb{R}$ is the moving stride.

We first randomly pick a starting point $x \in \mathbb{R}^{d_0}$ and non-zero direction vector $\Delta \in \mathbb{R}^{d_0}$. Then let the starting point move along the direction Δ or the opposite direction $-\Delta$. It is expected that the starting point will eventually cross the decision hyperplane in one direction, as long as Δ and $-\Delta$ are not parallel to the decision hyperplane.

Denote by $s \in \mathbb{R}$ the moving stride of the starting point, which means that the starting point arrives at $x + s \times \Delta$. After querying the Oracle with x and $x + s \times \Delta$, if $z(f_\theta(x)) \neq z(f_\theta(x + s \times \Delta))$ (i.e., the two labels are different), we know that the starting point has crossed the decision hyperplane when the moving stride is s . Now, the core of finding decision boundary points is to determine a suitable moving stride s , such that the starting point reaches the decision hyperplane, i.e., $f_\theta(x + s \times \Delta) = 0$ holds.

This task is done by *binary search*. Concretely, randomly choose two different moving strides s_{slow} and s_{fast} at first, such that

$$\begin{aligned} z(f_\theta(x + s_{\text{slow}} \times \Delta)) &= z(f_\theta(x)), \\ z(f_\theta(x + s_{\text{slow}} \times \Delta)) &\neq z(f_\theta(x + s_{\text{fast}} \times \Delta)). \end{aligned} \quad (30)$$

Then, without changing the conditions presented in Eq. (30), we dynamically adjust s_{slow} and s_{fast} until their absolute difference is close to 0, i.e., $|s_{\text{slow}} - s_{\text{fast}}| < \epsilon$ where ϵ is a precision defined by the adversary. Finally, return $x + s_{\text{slow}} \times \Delta$ as a decision boundary point.

Since the precision ϵ is finite, $x + s_{\text{slow}} \times \Delta$ is not strictly at the decision boundary, which will inevitably introduce minor errors (equivalent to noises) into the extracted model. If ϵ decreases, then $x + s_{\text{slow}} \times \Delta$ will be closer to the decision boundary, which is helpful to the model extraction attack, refers to the experiment results in Section 7.

6.2 Filtering Duplicate Affine Transformations

For a k -deep neural network f_θ consisting of $n = \sum_{i=1}^k d_i$ neurons, the idealized extraction attack exploits special $n + 1$ model activation patterns.

To ensure that the required $n + 1$ model activation patterns occur with a probability as high as possible, in Step 1 introduced in Section 4, we collect M decision boundary points where $M \gg n + 1$, e.g., $M = c_n 2^n$ and c_n is a small factor. As a result, there are many collected decision boundary points with duplicate model activation patterns. Therefore, in Step 2, after recovering the parameter tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$ (i.e., the affine transformation) corresponding to each decision boundary point, we need to filter the decision boundary points with duplicate affine transformations, since their model activation patterns should be the same. When filtering duplicate affine transformations, we consider two possible cases.

Filtering Correctly Recovered Affine Transformations. In the first case, assume that two affine transformations are both correctly recovered.

However, recovering affine transformations (i.e., 0-deep neural network extraction attack) relies on finding decision boundary points, which introduces minor errors. This is equivalent to adding noises to the recovered affine transformations, i.e., the tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}})$. To check whether two noisy affine transformations are the same, we adopt the checking rule below.

Comparing two vectors. Consider two vectors with the same dimension, e.g., $V^1 \in \mathbb{R}^d, V^2 \in \mathbb{R}^d$. Set a small threshold φ . If the following d inequations hold simultaneously

$$|V_j^1 - V_j^2| < \varphi, j \in \{1, \dots, d\} \quad (31)$$

where V_j^1 and V_j^2 are, respectively, the j -th element of V^1 and V^2 , the two vectors are considered to be the same.

Filtering Wrongly Recovered Affine Transformations. In the second case, assume that one affine transformation is correctly recovered and another one is partially recovered.

For the extraction attack on k -deep neural networks, when recovering the affine transformation corresponding to an input by the 0-deep neural network extraction attack (see Section 5.1), the process of binary search should not change the model activation pattern. Otherwise, the affine transformation may be wrongly recovered. Recall that, in the 0-deep neural network extraction attack, the d_0 elements of $\Gamma_{\mathcal{P}}$ are recovered one by one independently. Thus, the wrong recovery of one element of $\Gamma_{\mathcal{P}}$ does not influence the recovery of other elements.

As a result, we have to consider the case that one transformation is partially recovered. In this case, the filtering method is as follows. Consider two vectors $V^1 \in \mathbb{R}^d$ and $V^2 \in \mathbb{R}^d$. If $|V_j^1 - V_j^2| < \varphi$ holds for at least $(d - d_\varphi)$ j 's where $j \in \{1, \dots, d\}$ and $d_\varphi \in \mathbb{N}$ is a threshold, the two vectors are considered to be the same. Suppose that the occurrence frequencies of V^1 and V^2 are o_1 and

o_2 respectively, we regard V^1 as the correctly recovered affine transformation if $o_1 \gg o_2$, and vice versa.

6.3 Filtering Functionally Inequivalent Extracted Models

Consider k -deep neural networks consisting of $n = \sum_{i=1}^k d_i$ neurons. As introduced in Section 4, each time we randomly choose $n + 1$ out of N collected decision boundary points to generate an extracted model. Moreover, according to Section 5.2, in the extraction attack, we need to guess k signs, i.e., the sign of $\mathcal{G}_j^{(i)}$, $i \in \{1, \dots, k\}$.

When the model activation patterns of the selected $n + 1$ decision boundary points are not those required in the extraction attack, or at least one of the k sign guesses is wrong, the resulting extracted model $f_{\hat{\theta}}$ is not a functionally equivalent model of the victim model f_{θ} . Thus, we will get many functionally inequivalent extracted models.

Besides, due to the minor errors introduced by the finite precision used in finding decision boundary points, the parameters of the extracted model may be slightly different from the theoretical values (see Eq. (19), Eq. (27), and Eq. (28)). This subsection introduces three methods to filter functionally inequivalent extracted models, one of which considers the negative influence of finite precision together.

Filtering by the Normalized Model Signature. Before introducing the filtering method, we discuss how many possible model activation patterns there are at most for a k -deep neural network. Lemma 2 answers this question.

Lemma 2. *For a k -deep neural network consisting of $n = \sum_{i=1}^k d_i$ neurons, the upper bound of the number of possible model activation patterns is*

$$H = \left(\prod_{i=1}^k (2^{d_i} - 1) \right) + \sum_{i=2}^k \left(\prod_{j=1}^{i-1} (2^{d_j} - 1) \right), \quad (32)$$

where d_i is the number of neurons in layer i .

Proof. If all the d_i neurons in layer i are inactive, i.e., the outputs of these neurons are 0, then the neuron states of all the $\sum_{j=i+1}^k d_j$ neurons in the last $k - i$ layers are deterministic. In this case, the number of possible model activation patterns is decided by the first $i - 1$ layers, i.e., the maximum is $\prod_{j=1}^{i-1} (2^{d_j} - 1)$. If there is at least one active neuron in each layer, then there are at most $\prod_{i=1}^k (2^{d_i} - 1)$ possible model activation patterns.

After all the weights $\hat{A}^{(i)}$ and biases $\hat{b}^{(i)}$, $i \in \{1, \dots, k + 1\}$ are obtained, we assume that all the H model activation patterns are possible, and compute the resulting normalized model signature $\mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$. Denote by $\mathcal{S}_{\theta}^{\mathcal{N}}$ the normalized model signature recovered in Step 2 (see Section 4). If $\mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$ is not a subset of $\mathcal{S}_{\theta}^{\mathcal{N}}$, we regard $f_{\hat{\theta}}$ as a functionally inequivalent model.

Due to the minor errors caused by finite precision, i.e., the slight difference between the extracted parameters $\hat{\theta}$ and the theoretical values (see Eq. (19), Eq. (27), and Eq. (28)), when checking whether a tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \in \mathcal{S}_{\theta}^{\mathcal{N}}$ is equal to a tuple $(\hat{\Gamma}_{\mathcal{P}}, \hat{B}_{\mathcal{P}}) \in \mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$ or not, we adopt the checking rule presented in Section 6.2, refers to Eq. (31).

Besides, the filtering method in Section 6.2 does not ensure that all the wrongly recovered affine transformations are filtered. To avoid the functionally equivalent model being filtered, we adopt a flexible method.

Recall that, in Step 1, we collect a sufficient number of decision boundary points. For each tuple $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \in \mathcal{S}_{\theta}^{\mathcal{N}}$, denote by $m_{\mathcal{P}}$ the frequency that the tuple occurs in the collected decision boundary points. Suppose that the number of $m_{\mathcal{P}}$ where $m_{\mathcal{P}} > 1$ is N_{valid} . Then when at least $0.95 \times N_{\text{valid}}$ tuples $(\Gamma_{\mathcal{P}}, B_{\mathcal{P}}) \in \mathcal{S}_{\theta}^{\mathcal{N}}$ are in the set $\mathcal{S}_{\hat{\theta}}^{\mathcal{N}}$, the extracted model $f_{\hat{\theta}}$ is regarded as a candidate of the functionally equivalent model. Here, We call the ratio $0.95 \times \frac{N_{\text{valid}}}{|\mathcal{S}_{\theta}^{\mathcal{N}}|}$ the adaptive threshold.

Filtering by Weight Signs. After $\hat{A}^{(i)}, \hat{b}^{(i)}$ for $i \in \{1, \dots, k+1\}$ are obtained, we compute the matrices $\hat{\mathcal{G}}^{(i)}$ and check whether the k signs, i.e., the sign of $\hat{\mathcal{G}}_j^{(i)}, i \in \{1, \dots, k\}$ are consistent with the k guesses. If at least one sign is not consistent with the guess, the extracted model is not the functionally equivalent model.

Interestingly, except for handling wrong sign guesses, this method also shows high filtering effectiveness when the model activation patterns of the selected $n+1$ decision boundary points are not those required by our extraction attacks. This is not strange, since our extraction attack is designed for a specific set of model activation patterns. For wrong model activation patterns, whether the sign of $\hat{\mathcal{G}}_j^{(i)}, i \in \{1, \dots, k\}$ is 1 or -1 is a random event.

Filtering by Prediction Matching Ratio. The above two filtering methods are effective, but we find that some functionally inequivalent models still escape from the filtering. Therefore, the third method is designed to perform the last filtering on extracted models surviving from the above two filtering methods. This method is based on the *prediction matching ratio*.

Prediction Matching Ratio. Randomly generate N_1 inputs, query the extracted model $f_{\hat{\theta}}$ and the victim model f_{θ} . Suppose that the two models return the same hard-label for N_2 out of N_1 inputs. The ratio $\frac{N_2}{N_1}$ is called the prediction matching ratio.

According to Definition 1 and Definition 2, for a functionally equivalent model, the prediction matching ratio should be high, or even close to 100%. Note that many random inputs x and corresponding hard-label $z(f_{\theta}(x))$ are collected during the attack process (see Steps 1 and 2 in Section 4). Thus, we can exploit these inputs.

7 Experiments

Our model extraction attacks are evaluated on both untrained and trained neural networks. Concretely, we first perform experiments on untrained neural networks with diverse architectures and randomly generated parameters. Then, based on two typical benchmarking image datasets (i.e., MNIST, CIFAR10) in visual deep learning, we train a series of neural networks as classifiers and evaluate the model extraction attacks on these trained neural networks.

For convenience, denote by ‘ $d_0-d_1-\dots-d_{k+1}$ ’ the victim model, where d_i is the dimension of each layer. For example, the symbol 1000-1 stands for a 0-deep neural network with an input dimension of 1000 and an output dimension of 1.

Partial Universal Experiment Settings. Some settings are used in all the following experiments. For k -deep neural network extraction attacks, in Step 1, we randomly generate 8×2^n pairs of starting point and moving direction, where $n = \sum_{i=1}^k d_i$ is the number of neurons. The prediction matching ratio is estimated over 10^6 random inputs.

7.1 Computing $(\varepsilon, 0)$ -Functional Equivalence

To quantify the degree to which a model extraction attack has succeeded, the method (i.e., error bounds propagation [4]) proposed by Carlini et al. is adopted to compute $(\varepsilon, 0)$ -functional equivalence.

Error bounds propagation. To compute $(\varepsilon, 0)$ -functional equivalence of the extracted neural network $f_{\hat{\theta}}$, one just needs to compare the extracted parameters (weights $\hat{A}^{(i)}$ and biases $\hat{b}^{(i)}$) to the real parameters (weights $A^{(i)}$ and biases $b^{(i)}$) and analytically derive an upper bound on the error when performing inference [4].

Before comparing the neural network parameters, one must ‘align’ them [4]. This involves two operations: (1) adjusting the order of the neurons in the network, i.e., the order of the rows or columns of $A^{(i)}$ and $b^{(i)}$, (2) adjusting the values of $A^{(i)}$ and $b^{(i)}$ to the theoretical one (see Eq. (19), Eq. (27), and Eq. (28)) obtained by the idealized model extraction attacks. This gives an aligned $\tilde{A}^{(i)}$ and $\tilde{b}^{(i)}$ from which one can analytically derive upper bounds on the error. Other details (e.g., propagating error bounds layer-by-layer) are the same as that introduced in [4], and not introduced again in this paper.

7.2 Experiments on Untrained Neural Networks

Table 1 summarizes the experimental results on different untrained neural networks which demonstrates the effectiveness of our model extraction attacks.

According to Appendix B, the computation complexity of our model extraction attack is about $\mathcal{O}\left(n \times 2^{n^2+n+k}\right)$, where n is the number of neurons. Thus, we limit the number of neurons, which does not influence the verification of our

Table 1. Experiment results on untrained k -deep neural networks.

Architecture	Parameters	ϵ	PMR	Queries	$(\epsilon, 0)$	$\max \theta - \hat{\theta} $
512-2-1	1029	10^{-12}	100%	$2^{19.35}$	$2^{-12.21}$	$2^{-16.88}$
		10^{-14}	100%	$2^{19.59}$	$2^{-19.84}$	$2^{-24.62}$
2048-4-1	8201	10^{-12}	99.98%	$2^{23.32}$	$2^{-3.77}$	$2^{-10.44}$
		10^{-14}	100%	$2^{23.51}$	$2^{-13.70}$	$2^{-17.75}$
25120-4-1	100489	10^{-14}	99.98%	$2^{26.42}$	$2^{-2.99}$	$2^{-14.67}$
		10^{-16}	100%	$2^{26.67}$	$2^{-13.01}$	$2^{-23.19}$
50240-2-1	100485	10^{-14}	99.99%	$2^{25.85}$	$2^{-7.20}$	$2^{-15.58}$
		10^{-16}	100%	$2^{26.31}$	$2^{-14.44}$	$2^{-22.67}$
32-2-2-1	75	10^{-12}	100%	$2^{17.32}$	$2^{-10.99}$	$2^{-14.78}$
		10^{-14}	100%	$2^{17.56}$	$2^{-18.21}$	$2^{-20.61}$
512-2-2-1	1035	10^{-12}	99.99%	$2^{21.39}$	$2^{-10.34}$	$2^{-14.01}$
		10^{-14}	100%	$2^{21.59}$	$2^{-14.17}$	$2^{-17.29}$
1024-2-2-1	2059	10^{-12}	99.99%	$2^{22.38}$	$2^{-6.10}$	$2^{-13.77}$
		10^{-14}	100%	$2^{22.49}$	$2^{-14.16}$	$2^{-20.38}$

ϵ : the precision used to find decision boundary points.

$\max|\theta - \hat{\theta}|$: the maximum extraction error of model parameters.

PMR: prediction matching ratio.

model extraction attack. Note that the number of parameters is not limited. All the attacks can be finished within several hours on a single core.

The results in Table 1 also support our argument in Remark 2. For the 2-deep neural networks (e.g., 32-2-2-1), when recovering the weights in layer 1, we require that only one neuron in layer 2 is active, instead of all the 2 neurons being active. Our extraction attacks also achieve good performance.

The influence of the precision ϵ . A smaller ϵ will make the returned point $x + s_{\text{slow}} \times \Delta$ (see Section 6.1) closer to the decision boundary, which helps reduce the extraction error of affine transformations. As a result, the model extraction attack is expected to perform better. For example, for the 1-deep neural network 2048-4-1, when ϵ decreases from 10^{-12} to 10^{-14} , the value ϵ (respectively, $\max|\theta - \hat{\theta}|$) decreases from $2^{-3.77}$ to $2^{-13.70}$ (respectively, from $2^{-10.44}$ to $2^{-17.75}$), which is a significant improvement.

At the same time, using a smaller precision ϵ does not increase the attack complexity significantly. According to Appendix B, the query complexity is about $\mathcal{O}\left(d_0 \times 2^n \times \log_2^{\frac{1}{\epsilon}}\right)$. Thus, decreasing ϵ has little influence on the query complexity. Look at the neural network 2048-4-1 again. When ϵ decreases from 10^{-12} to 10^{-14} , the number of queries only increases from $2^{23.32}$ to $2^{23.51}$. Besides, when n (i.e., the number of neurons) is large, ϵ almost does not influence the computation complexity, since ϵ only influences Steps 1 and 2 (see Section 4), while the computation complexity is mainly determined by other steps (refer to Appendix B). When n is small, the practical runtime is determined by the query complexity, then decreasing ϵ also has little influence on the runtime.

Choosing an appropriate ϵ is simple. In our experiments, we find that a smaller ϵ should be used, when the prediction matching ratio estimated over 10^6 random inputs is not 100%, and the gap (e.g., 0.02%, see the third or fifth row) is not negligible.

7.3 Experiments on Trained Neural Networks

The MNIST and CIFAR10 Dataset. MNIST (respectively, CIFAR10) is one typical benchmarking dataset used in visual deep learning. It contains ten-class handwriting number gray images [12] (resp., real object images in a realistic environment [19]). Each of the ten classes, i.e., ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, and ‘9’ (resp., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), contains 28×28 pixel gray images (resp., 32×32 pixel RGB images), totaling 60000 (resp., 50000) training and 10000 (resp. 10000) testing images.

Neural Network Training Pipelines. When classifying different classes of objects, the decision boundary of trained neural networks will be different. To fully verify our model extraction attack, for MNIST (respectively, CIFAR10), we divide the ten classes into five groups and build a binary classification neural network for each group. All the neural networks share the same architecture d_0 -2-1, where $d_0 = 28 \times 28$ for MNIST (respectively, $32 \times 32 \times 3$ for CIFAR10). On the MNIST and CIFAR10 datasets, we perform a standard rescaling of the pixel values from $0 \cdots 255$ to $0 \cdots 1$. For the model training, we choose typical settings (the loss is the cross-entropy loss; the optimizer is standard stochastic gradient descent; batch size 128). The first four columns of Table 2 summarize a detailed description of the neural networks to be attacked in this section.

Experiment Results. The last four columns of Table 2 summarize the experiment results. Our extraction attack still achieves good performance when an appropriate precision ϵ is used, which further verifies its effectiveness.

The experimental results presented in Table 1 and Table 2 show that the attack performance (i.e., the value of ϵ and $\max|\theta - \hat{\theta}|$) is related to the precision ϵ and the properties of the decision boundary. However, we do not find a clear quantitative relationship between the attack performance and the precision ϵ (or some unknown properties of the decision boundary). Considering that the unknown quantitative relationships do not influence the verification of the model extraction attack, we leave the problem of exploring the unknown relationships as a future work.

8 Conclusion

In this paper, we have studied the model extraction attack against neural network models under the hard-label setting, i.e., the adversary only has access to the most likely class label corresponding to the raw output of neural network models. We propose new model extraction attacks that theoretically achieve

Table 2. Experiment results on neural networks trained on MNIST or CIFAR10.

task	architecture	accuracy	parameters	ϵ	Queries	$(\epsilon, 0)$	$\max \theta - \hat{\theta} $
'0' vs '1'	784-2-1	0.9035	1573	10^{-12}	$2^{20.11}$	$2^{-16.39}$	$2^{-17.85}$
				10^{-14}	$2^{20.32}$	$2^{-20.56}$	$2^{-22.81}$
'2' vs '3'	784-2-1	0.8497	1573	10^{-12}	$2^{20.11}$	$2^{-7.00}$	$2^{-7.80}$
				10^{-14}	$2^{20.32}$	$2^{-14.32}$	$2^{-15.06}$
'4' vs '5'	784-2-1	0.8570	1573	10^{-12}	$2^{20.02}$	$2^{-8.47}$	$2^{-8.82}$
				10^{-14}	$2^{20.32}$	$2^{-15.62}$	$2^{-15.81}$
'6' vs '7'	784-2-1	0.9290	1573	10^{-12}	$2^{20.11}$	$2^{-7.02}$	$2^{-7.93}$
				10^{-14}	$2^{20.32}$	$2^{-12.00}$	$2^{-12.91}$
'8' vs '9'	784-2-1	0.9501	1573	10^{-12}	$2^{20.11}$	$2^{-10.58}$	$2^{-11.62}$
				10^{-14}	$2^{20.32}$	$2^{-19.63}$	$2^{-21.72}$
airplane vs automobile	3072-2-1	0.8120	6149	10^{-12}	$2^{22.08}$	$2^{-4.84}$	$2^{-7.48}$
				10^{-14}	$2^{22.29}$	$2^{-12.41}$	$2^{-15.20}$
bird vs cat	3072-2-1	0.6890	6149	10^{-12}	$2^{22.07}$	$2^{-8.37}$	$2^{-9.80}$
				10^{-14}	$2^{22.29}$	$2^{-12.27}$	$2^{-14.73}$
deer vs dog	3072-2-1	0.6870	6149	10^{-12}	$2^{22.01}$	$2^{-9.55}$	$2^{-13.25}$
				10^{-14}	$2^{22.22}$	$2^{-13.19}$	$2^{-15.82}$
frog vs horse	3072-2-1	0.8405	6149	10^{-12}	$2^{22.08}$	$2^{-9.56}$	$2^{-10.71}$
				10^{-14}	$2^{22.29}$	$2^{-13.58}$	$2^{-15.58}$
ship vs truck	3072-2-1	0.7995	6149	10^{-12}	$2^{22.08}$	$2^{-8.63}$	$2^{-8.90}$
				10^{-14}	$2^{22.29}$	$2^{-12.95}$	$2^{-13.02}$

$\max|\theta - \hat{\theta}|$: the maximum extraction error of model parameters.

accuracy: classification accuracy of the victim model f_{θ} .

for saving space, prediction matching ratios are not listed.

functionally equivalent extraction. Practical experiments on numerous neural network models have verified the effectiveness of the proposed model extraction attacks. To the best of our knowledge, this is the first time to prove with practical experiments that it is possible to achieve functionally equivalent extraction against neural network models under the hard-label setting.

The future work will mainly focus on the following aspects:

- The (computation and query) complexity of our model extraction attack remains high, which limits the application to neural networks with a large number of neurons. Reducing the complexity is an important problem.
- In this paper, to recover the weight vector of the j -th neuron in layer i , we require that in layer i , only the j -th neuron is active. However, such a model activation pattern may not occur in some cases. Then how to recover the weight vector of this neuron based on other model activation patterns would be a vital step towards better generality.
- Explore possible quantitative relationships between the precision ϵ (or some unknown properties of the decision boundary) and ε (or $\max|\theta - \hat{\theta}|$).
- Extend the extraction attack to the case of vector outputs, i.e., the output dimensionality exceeds 1.
- Develop extraction attacks against other kinds of neural network models.

Acknowledgments. We would like to thank Adi Shamir for his guidance. We would like to thank the anonymous reviewers for their detailed and helpful comments. This work was supported by the National Key R&D Program of China (2018YFA0704701, 2020YFA0309705), Shandong Key Research and Development Program (2020ZLYS09), the Major Scientific and Technological Innovation Project of Shandong, China (2019JZZY010133), the Major Program of Guangdong Basic and Applied Research (2019B030302008), the Tsinghua University Dushi Program, and the Ministry of Education in Singapore under Grant RG93/23. Y. Chen was also supported by the Shuimu Tsinghua Scholar Program.

A Proof of Lemma 1

We prove Lemma 1 by Mathematical Induction.

Proof. When $i = 2$, according to Lemma 1, the extracted weight vector $\widehat{A}_j^{(2)}, j \in \{1, \dots, d_2\}$ should be

$$\begin{aligned}\widehat{A}_j^{(2)} &= \left[\frac{w_{j,1}^{(2)} \times \left| \sum_{v=1}^{d_0} w_{1,v}^{(1)} C_{v,1}^{(0)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|}, \dots, \frac{w_{j,d_1}^{(2)} \times \left| \sum_{v=1}^{d_0} w_{d_1,v}^{(1)} C_{v,1}^{(0)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \right] \\ &= \left[\frac{w_{j,1}^{(2)} \times \left| w_{1,1}^{(1)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|}, \dots, \frac{w_{j,d_1}^{(2)} \times \left| w_{d_1,1}^{(1)} \right|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \right].\end{aligned}\tag{33}$$

Note that $C_{1,1}^{(0)} = 1$ and $C_{v,1}^{(0)} = 0$ for $v \in \{2, \dots, d_0\}$.

Besides, we have

$$\begin{aligned}C^{(1)} &= I_{\mathcal{P}}^{(1)} A^{(1)} = A^{(1)} = \left[A_1^{(1)}, \dots, A_{d_0}^{(1)} \right], \\ \widehat{C}^{(1)} &= I_{\mathcal{P}}^{(1)} \widehat{A}^{(1)} = \widehat{A}^{(1)} = \left[\frac{A_1^{(1)}}{\left| w_{1,1}^{(1)} \right|}, \dots, \frac{A_{d_0}^{(1)}}{\left| w_{d_0,1}^{(1)} \right|} \right].\end{aligned}\tag{34}$$

where $A_v^{(1)} = \left[w_{v,1}^{(1)}, \dots, w_{v,d_0}^{(1)} \right]$, $C_{v,u}^{(1)} = w_{v,u}^{(1)}$ and $\widehat{C}_{v,u}^{(1)} = \frac{w_{v,u}^{(1)}}{\left| w_{v,1}^{(1)} \right|}$.

Look at the system of linear equations presented in Eq. (26). Now, the system of linear equations is transformed into

$$\begin{cases} \sum_{v=1}^{d_1} \widehat{w}_{j,v}^{(2)} \widehat{C}_{v,1}^{(1)} = \frac{\sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)}}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \\ \vdots \\ \sum_{v=1}^{d_1} \widehat{w}_{j,v}^{(2)} \widehat{C}_{v,d_0}^{(1)} = \frac{\sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,d_0}^{(1)}}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \end{cases}\tag{35}$$

when $d_0 \geq d_1$, by solving the system, it is expected to obtain

$$\widehat{A}_j^{(2)} = \left[\frac{w_{j,1}^{(2)} |w_{1,1}^{(1)}|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|}, \dots, \frac{w_{j,d_1}^{(2)} |w_{d_1,1}^{(1)}|}{\left| \sum_{v=1}^{d_1} w_{j,v}^{(2)} C_{v,1}^{(1)} \right|} \right], \quad (36)$$

which is consistent with the expected value in Eq. (33).

Next, consider the recovery of the weight vector of the j -th neuron in layer i , and assume that the weights $\widehat{A}^{(1)}, \dots, \widehat{A}^{(i-1)}$ as shown in Lemma 1 have been obtained. As a result, we have

$$\begin{aligned} C_j^{(i-2)} &= A_j^{(i-2)} A^{(i-3)} \dots A^{(1)}, \quad C_j^{(i-1)} = A_j^{(i-1)} A^{(i-2)} \dots A^{(1)}, \\ \widehat{C}_j^{(i-2)} &= \widehat{A}_j^{(i-2)} \widehat{A}^{(i-3)} \dots \widehat{A}^{(1)} = \frac{C_v^{(i-2)}}{\left| \sum_{v=1}^{d_{i-3}} w_{j,v}^{(i-2)} C_{v,1}^{(i-3)} \right|}, \\ \widehat{C}_j^{(i-1)} &= \widehat{A}_j^{(i-1)} \widehat{A}^{(i-2)} \dots \widehat{A}^{(1)} = \frac{C_v^{(i-1)}}{\left| \sum_{v=1}^{d_{i-2}} w_{j,v}^{(i-1)} C_{v,1}^{(i-2)} \right|}. \end{aligned} \quad (37)$$

Now, the system of linear equations in Eq. (26) is transformed into

$$\begin{cases} \sum_{u=1}^{d_{i-1}} \widehat{w}_{j,u}^{(i)} \frac{C_{u,1}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-2}} w_{u,v}^{(i-1)} C_{v,1}^{(i-2)} \right|} = \frac{\sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,1}^{(i-1)}}{\left| \sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,1}^{(i-1)} \right|} \\ \vdots \\ \sum_{u=1}^{d_{i-1}} \widehat{w}_{j,u}^{(i)} \frac{C_{u,d_0}^{(i-1)}}{\left| \sum_{v=1}^{d_{i-2}} w_{u,v}^{(i-1)} C_{v,1}^{(i-2)} \right|} = \frac{\sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,d_0}^{(i-1)}}{\left| \sum_{u=1}^{d_{i-1}} w_{j,u}^{(i)} C_{u,1}^{(i-1)} \right|} \end{cases} \quad (38)$$

When $d_0 \geq d_{i-1}$, by solving this system, it is expected to obtain

$$\begin{aligned} \widehat{A}_j^{(i)} &= \left[\widehat{w}_{j,1}^{(i)}, \dots, \widehat{w}_{j,d_{i-1}}^{(i)} \right] \\ &= \left[\frac{w_{j,1}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{1,v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|}, \dots, \frac{w_{j,d_{i-1}}^{(i)} \times \left| \sum_{v=1}^{d_{i-2}} w_{d_{i-1},v}^{(i-1)} C_{v,1}^{(i-2)} \right|}{\left| \sum_{v=1}^{d_{i-1}} w_{j,v}^{(i)} C_{v,1}^{(i-1)} \right|} \right], \end{aligned}$$

which is consistent with the expected value in Eq. (27).

B Complexity of Hard-Label Model Extraction Attacks

For the k -deep neural network extraction attack, its complexity is composed of two parts: Oracle query complexity and computation complexity. Suppose that the number of neurons is $n = \sum_{i=1}^k d_i$. Its input size, i.e., the size of x is d_0 . And k is the number of hidden layers. The precision adopted by binary search is ϵ (refer to Section 6.1).

Oracle Query Complexity. For the k -deep neural network extraction attack, we only query the Oracle in Steps 1 and 2 (see Section 4).

In Step 1, if $c_n \times 2^n$ decision boundary points are collected, then the number of queries to the Oracle is $c_\epsilon \times c_n \times 2^n$, where c_ϵ is a factor determined by the precision ϵ , and c_n is a small factor defined by the attacker. In Step 2, for each decision boundary point x collected in Step 1, to recover the corresponding affine transformation (i.e., $\Gamma_{\mathcal{P}}$ and $B_{\mathcal{P}}$), we need to collect another $d_0 - 1$ decision boundary points. Therefore, the times of querying the Oracle in this step is $c_\epsilon \times c_n \times 2^n \times (d_0 - 1)$. Based on the above analysis, the Oracle query complexity of our k -deep neural network extraction attack is $c_\epsilon \times c_n \times 2^n \times d_0$. Note that c_ϵ is proportional to $\log_2^{\frac{1}{\epsilon}}$. Thus, the query complexity is about $\mathcal{O}\left(d_0 \times 2^n \times \log_2^{\frac{1}{\epsilon}}\right)$.

Computation Complexity. For the k -deep neural network extraction attack, when n is large, most computations are occupied by recovering neural network parameters, i.e., Steps 3 and 4 (see Section 4). Suppose that there are $N \leq c_n \times 2^n$ decision boundary points used to recover neural network parameters after filtering duplicate affine transformations in Step 2.

In Step 3, to recover the weight vector of the j -th neuron in layer i where $i \in \{2, \dots, k+1\}$, we need to solve a system of linear equations. For convenience, let us ignore the difference in the sizes of the different systems of linear equations. Then, to recover all the weights $A^{(i)}$, a total of $n + 1 - d_1 = \sum_{i=2}^{k+1} d_i$ systems of linear equations need to be solved. In Step 4, to recover all the biases $b^{(1)}$, only one system of linear equations needs to be solved. Therefore, to obtain an extracted model, we need to solve $n + 2 - d_1$ systems of linear equations.

There are two loops in the extraction attack. First, we need to select $n + 1$ out of N decision boundary points each time. More concretely, to recover the weights $A^{(i)}$ in layer i , we choose d_i decision boundary points. Then the number (denoted by l_1) of possible cases is

$$l_1 = \binom{N}{d_1} \times \binom{N-d_1}{d_2} \times \dots \times \binom{N-\sum_{i=1}^{k-1} d_i}{d_k} \times \binom{N-n}{1} \approx N^{n+1}, \text{ for } N \gg n.$$

Second, we need to guess k signs when recovering all the weights, i.e., there are 2^k cases.

Thus, the computation complexity is about $\mathcal{O}(l_1 \times 2^k \times (n + 2 - d_1))$. When an appropriate precision ϵ (i.e., ϵ is small) is adopted, we have $N \approx H < 2^n$, where H is the number of possible model activation patterns (refer to Lemma 2). Then, we further have

$$l_1 \times 2^k \times (n + 2 - d_1) \approx N^{n+1} \times 2^k \times n \approx n \times 2^{n(n+1)+k}. \quad (39)$$

Thus, the computation complexity is about $\mathcal{O}\left(n \times 2^{n^2+n+k}\right)$.

C Extraction on 1-Deep Neural Networks

The parameters of the extracted 1-deep neural network are as follows.

$$\begin{aligned}
\widehat{A}_i^{(1)} &= [\widehat{w}_{i,1}^{(1)}, \dots, \widehat{w}_{i,d_0}^{(1)}] = \left[\frac{w_{i,1}^{(1)}}{|w_{i,1}^{(1)}|}, \dots, \frac{w_{i,d_0}^{(1)}}{|w_{i,1}^{(1)}|} \right], \quad i \in \{1, \dots, d_1\}, \\
\widehat{b}^{(1)} &= [\widehat{b}_1^{(1)}, \dots, \widehat{b}_{d_1}^{(1)}] = \left[\frac{b_1^{(1)}}{|w_{1,1}^{(1)}|}, \dots, \frac{b_{d_1}^{(1)}}{|w_{d_1,1}^{(1)}|} \right], \\
\widehat{A}^{(2)} &= [\widehat{w}_1^{(2)}, \dots, \widehat{w}_{d_1}^{(2)}] = \left[\frac{w_1^{(2)} |w_{1,1}^{(1)}|}{\left| \sum_{i=1}^{d_1} w_i^{(2)} w_{i,1}^{(1)} \right|}, \dots, \frac{w_{d_1}^{(2)} |w_{d_1,1}^{(1)}|}{\left| \sum_{i=1}^{d_1} w_i^{(2)} w_{i,1}^{(1)} \right|} \right], \\
\widehat{b}^{(2)} &= \frac{b^{(2)}}{\left| \sum_{i=1}^{d_1} w_i^{(2)} w_{i,1}^{(1)} \right|}.
\end{aligned} \tag{40}$$

Fig. 4 shows a diagram of a victim model (2-2-1) and the extracted model.

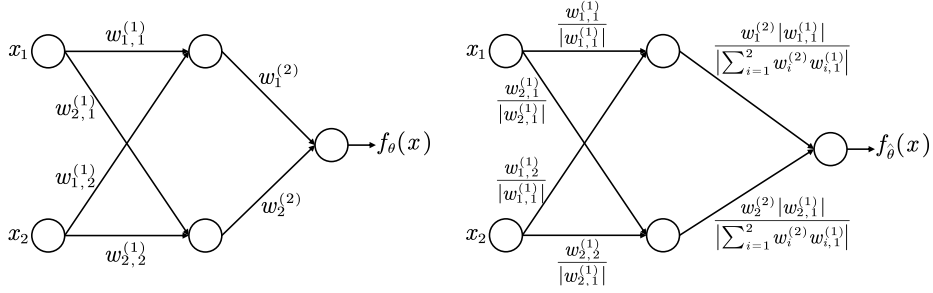


Fig. 4. Left: the victim model f_θ . Right: the extracted model $f_{\widehat{\theta}}$.

References

1. Batina, L., Bhasin, S., Jap, D., Picek, S.: CSI NN: reverse engineering of neural network architectures through electromagnetic side channel. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 515–532. USENIX Association
2. Blum, A., Rivest, R.L.: Training a 3-node neural network is np-complete. In: Hanson, S.J., Remmele, W., Rivest, R.L. (eds.) Machine Learning: From Theory to Applications - Cooperative Research at Siemens and MIT. LNCS, vol. 661, pp. 9–28. Springer (1993)
3. Canales-Martínez, I., Chávez-Saab, J., Hambitzer, A., Rodríguez-Henríquez, F., Satpute, N., Shamir, A.: Polynomial time cryptanalytic extraction of neural network models. IACR Cryptol. ePrint Arch. p. 1526 (2023)
4. Carlini, N., Jagielski, M., Mironov, I.: Cryptanalytic extraction of neural network models. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 189–218. Springer

5. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: SP 2017. pp. 39–57. IEEE Computer Society
6. Dubey, S.R., Singh, S.K., Chaudhuri, B.B.: Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing* **503**, 92–108 (2022)
7. Fefferman, C.: Reconstructing a neural net from its output. *Revista Matemática Iberoamericana* **10**, 507–555 (1994)
8. Galstyan, A., Cohen, P.R.: Empirical comparison of ”hard” and ”soft” label propagation for relational classification. In: Blockeel, H., Ramon, J., Shavlik, J.W., Tadepalli, P. (eds.) ILP 2007. LNCS, vol. 4894, pp. 98–111. Springer
9. Ganju, K., Wang, Q., Yang, W., Gunter, C.A., Borisov, N.: Property inference attacks on fully connected neural networks using permutation invariant representations. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) CCS 2018. pp. 619–633. ACM
10. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) NeurIPS 2015. pp. 1135–1143
11. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: Capkun, S., Roesner, F. (eds.) USENIX Security 2020. pp. 1345–1362. USENIX Association
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
13. Long, C., Collins, R., Swears, E., Hoogs, A.: Deep neural networks in fully connected CRF for image labeling with social network metadata. In: WACV 2019. pp. 1607–1615. IEEE
14. Lowd, D., Meek, C.: Adversarial learning. In: Grossman, R., Bayardo, R.J., Bennett, K.P. (eds.) SIGKDD 2005. pp. 641–647. ACM
15. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Fürnkranz, J., Joachims, T. (eds.) ICML, 2010. pp. 807–814. Omnipress
16. Oliynyk, D., Mayer, R., Rauber, A.: I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Comput. Surv.* **55**(14s), 324:1–324:41 (2023)
17. Perazzi, F., Wang, O., Gross, M.H., Sorkine-Hornung, A.: Fully connected object proposals for video segmentation. In: ICCV 2015. pp. 3227–3234. IEEE Computer Society
18. Rolnick, D., Kording, K.P.: Reverse-engineering deep relu networks. In: ICML 2020. Proceedings of Machine Learning Research, vol. 119, pp. 8178–8187. PMLR
19. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **30**(11), 1958–1970 (2008)
20. Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 601–618. USENIX Association