Reverse Engineering of Neural Networks

Mohammad Hossein Badiei, Mohammad Sayad Haghighi

***Abstract*—In recent years considering the development of machine learning and artificial neural networks, modeling and training NNs for industry's purposes is a very *time-consuming process*. Therefore it's amazing that achieve the parameters of these trained models as black-box models without the requirement of modeling and retraining a new one. Here we review *some of the* most common types of these extraction attacks. So, in this article we are going to describe the previous researches belong to five articles about these methods specially weights and signs extraction from a black-box neural network model.**

***Index Terms*—Neural Network, Reverse engineering, Stealing models, Machine learning**

I. Introduction

The main applications of neural networks are classification and regression, Although classification is more common than regression. Nowadays considering the development of these models, you may see the capability of memorizing, sorting, painting and another amazing usecases as the applications of these models in the real world.

Reverse engineering of neural networks (also called RENN) is an approach to extracting information from a black-box model of neural network. These are information about layers, weights, neurons and also activation functions. In this area attacker has some capabilities including: the attacker in consideration is a passive one. Also attacker doesn't know the architecture of the used network but can feed random (and hence known) inputs to the architecture. Also attacker is capable of measuring side-channel information leaked from the implementation of the targeted architecture. So we are going to recover the neural network architecture using only side-channel information. No assumptions are needed on the type of inputs or its source, as we work with real number. Also we assume that the implementation of the machine learning algorithm doesn't include any side-channel countermeasures.

II. COMPARATIVE ANALYSIS OF ARTICLES

We now describe previous researches belong to five articles about possible motivations for adversaries to perform weights extraction attacks in black-box models.

A. *Stealing Machine Learning Models via Prediction APIs*

First attack we want to describe is equation solving attack.

Use italics for emphasis; do not underline. As we consider the capabilities of adversary, inputs and their outputs are known to him. In this case adversary consider the parameters of model as unknown parameters and inputs/outputs as known parameters of the model equation. So if x_i is a sample pf input and $y_i = f(x_i)$ is its output so f represent the equation of model. In this technic adversary use random queries to solve the equations of model. As a simple example of equation solving attack in neural networks, consider the following NN:

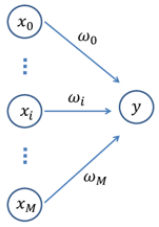


Figure 1one layer neural network

It's clear that y is equal to$\sum_{i=0}^{M} x_M w_M$. Considering these knowledge, adversary put $(1, 0, 0, \dots, 0)$ in equation and find w_1 because w_1 is equal to y_1 according to the model equation. Using this attack all the weights obtained from the equation. The results of this attack belong to paper [1] is shown below.

Model	Unknowns	Queries	$1-R_{\text{test}}$	$1-R_{\text{unif}}$	Time (s)
Softmax	530	265	99.96%	99.75%	2.6
		530	100.00%	100.00%	3.1
OvR	530	265	99.98%	99.98%	2.8
		530	100.00%	100.00%	3.5
MLP	2,225	1,112	98.17%	94.32%	155
		2,225	98.68%	97.23%	168
		4,450	99.89%	99.82%	195
		11,125	99.96%	99.99%	89

Figure 2Success of equation-solving attacks in the paper [1]

As we can see here, although the accuracy of MLP is desired but the time complexity and number of queries to reach this accuracy is so high. This challenge will be more important when consider complex models of NN with more layers and neurons. So it seems that the equation-solving attack isn't efficient.

Check the attached video presentation for more details in the following link.

2nd attack discussed in the paper is decision tree path finding attack. This algorithm said the adversary should find predicates that input has to satisfy to reach leaf node. And also said that the adversary should generates new inputs to visit unexplored path.

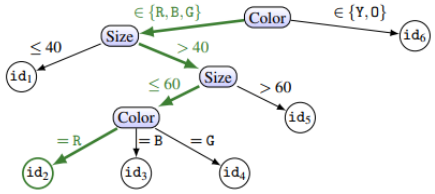


Figure 3Decision tree over features Color and Size

The pseudo code of the algorithm is shown below.

```
1: x_init ← {x_1, ..., x_d}                          ▷ random initial query
2: Q ← {x_init}                                      ▷ Set of unprocessed queries
3: P ← {}                                            ▷ Set of explored leaves with their predicates
4: while Q not empty do
5:     x ← Q.POP()
6:     id ← O(x)                                     ▷ Call to the leaf identity oracle
7:     if id ∈ P then                                ▷ Check if leaf already visited
8:         continue
9:     end if
10:    for 1 ≤ i ≤ d do                              ▷ Test all features
11:        if IS_CONTINUOUS(i) then
12:            for (α, β] ∈ LINE_SEARCH(x, i, ε) do
13:                if x_i ∈ (α, β] then
14:                    P[id].ADD('x_i ∈ (α, β]')     ▷ Current interval
15:                else
16:                    Q.PUSH(x[i] ⇒ β)              ▷ New leaf to visit
17:                end if
18:            end for
19:        else
20:            S, V ← CATEGORY_SPLIT(x, i, id)
21:            P[id].ADD('x_i ∈ S')                  ▷ Values for current leaf
22:            for v ∈ V do
23:                Q.PUSH(x[i] ⇒ v)                  ▷ New leaves to visit
24:            end for
25:        end if
26:    end for
27: end while
```

3rd attack discussed in the paper is online model extraction attack. The paper shows online model extraction attack against BigML and Amazon. It focuses on extracting models set up by a user, who wishes to charge for predictions. In this case, the adversary attack models trained in his own account.

4th attack we want to describe is extraction given class labels only. In this attack adversary explores model extraction in a setting with no confidence score. Adversary use line search to find points arbitrary close to f's decision boundary and solve for parameters from these inputs.

In continue we focus on the neural network models. In these models adversary does not know the architecture of the used network but can feed random (and hence known) inputs to the architecture. Also he is capable of measuring side-channel information leaked from the implementation of the targeted architecture.

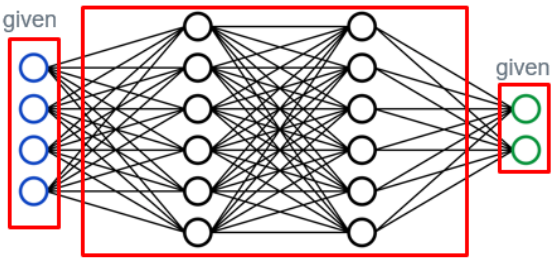


Figure 4 A black-box NN model. Given inputs and outputs, the adversary attacks the model to find weights

B. *Model Reconstruction from Model Explanations*

The problem is when given access to gradient queries finding a classifier $\hat{f}$ identical to an unknown classifier f.

Key issues discussed in the paper are Learn a linear model from a single input gradient query, reconstructing two-layer relu networks and recovering the signs. The assumption in these models is the activation function is relu.

Consider a one hidden layer relu neural network. Hence the output function f is calculated as below:

$$f(x) = \sum_{i=1}^{h} w_i \max\left(A_i^T x, 0\right)$$

As you know A and w are parameters of the model. Also the max structure in the output equation is because of the activation function which is relu. Lest depict the model in the figure below.

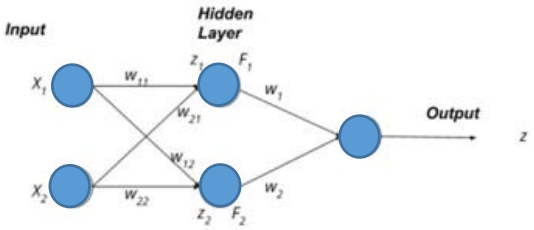


Figure 5 Simplest Deep Learning Model with two Inputs, one Hidden Layer and one Putput

The two hyperplanes with normal vector A1 and A2 separate the input space into four cells where the gradient of f is constant.

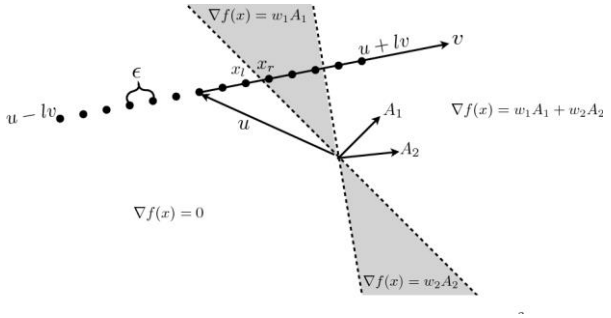


Figure 6 the two hyperplanes with normal vector A1 and A2 for the NN model shown in figure 5

The adversary first picks two random vectors u and v. then searches for a change in the gradient of f using a binary search. Then goes on again and again till finding two points that are sufficiently close, but have differing gradients. The article said $\nabla f(x_r) - \nabla f(x_i)$ is equal two $w_i A_i$ with high probability. You may want to check the pseudo code of the algorithm below. Consider the Z such that $Z_{p(i)} = -w_i A_i$ and a vector $s \epsilon \{-1,0,1\}$ such that $f(x) = [\max(Zx, 0)^T \quad \max(-Zx, 0)^T]s$.

Algorithm 1: Recovery of f

```
Function learnModel(h, ϵ, l):
    Z ← recoverZ(h, ϵ, l)
    s ← recoverS(Z)
    return Z, s
```

Algorithm 1a: Recovery of Z

```
Function recoverZ(h, ϵ, l):
    Pick u, v ~ N(0, I_d) and let Z ∈ R^{h×d}
    t_l, t_r ← −l, l
    for i = 1, ... h do
        Z_i, t_l ← binarySearch(t_l, t_r, ϵ)
    return Z
Function binarySearch(t_l, t_r, ϵ):
    while t_l ≤ t_r do
        t_m ← (t_l + t_r)/2
        x_l ← u + t_l v,  x_m ← u + t_m v,  x_r ← u + t_r v
        if t_r − t_l ≤ ϵ then
            return ∇f(x_r) − ∇f(x_l), t_r
        if ‖∇f(x_l) − ∇f(x_m)‖_2 > 0 then
            t_r ← t_m
        else if ‖∇f(x_m) − ∇f(x_r)‖_2 > 0 then
            t_l ← t_m
        throw Failure
    throw Failure
```

Algorithm 1b: Recovery of s

```
Function recoverS(Z):
    Pick X ∈ R^{d×h} such that ∇f(x_1) = ··· = ∇f(x_h) and Rank(ZX) = h. (See Appendix B)
    M ← [ max(ZX, 0)^T   max(−ZX, 0)^T
          max(−ZX, 0)^T  max(ZX, 0)^T ]
    Solve for s ∈ R^{2h} such that Ms = [f(x_1), ... f(x_h), f(−x_1), ... f(−x_h)]
    return s
```

Let's consider a particular case. In this case the adversary doesn't know the values of the points. But he knows that these points are in the hyperplanes of network. Assume that one of the point is in the 1st hyperplane. So the network becomes like below. Consider we know that the first neuron of later 1 has discrete gradient in this point and we represent it by 0.

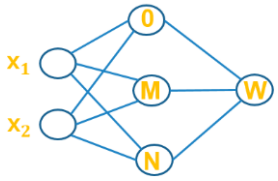


Figure 7 a two layer relu network with an input in the hyperplane of first neuron

As expected if input changes into $(x_1 + \varepsilon, x_2)^T$, the model becomes as depicted below.

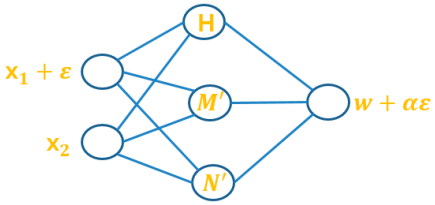


Figure 8 the model in the figure 7 after changing in the input

So adversary efforts to find a δ and add it to x_2 in order to place the point in the hyperplane. If he success, so the angle of the first neuron hyperplane will be extracted.

$$f(x_1, x_2) = f(x_1 + \varepsilon, x_2 + \delta)$$

$$x_1 w_{11} + x_2 w_{12} = (x_1 + \varepsilon)\, w_{11} + (x_2 + \delta)\, w_{12}$$

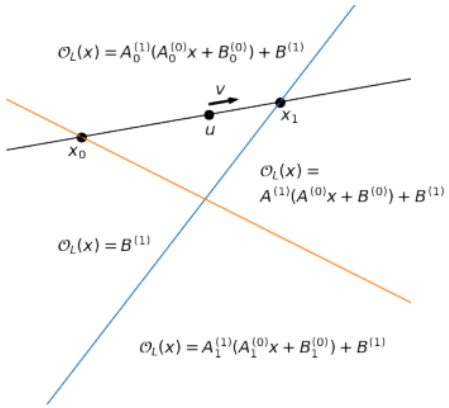
$$\rightarrow \frac{-\varepsilon}{\delta} = \frac{w_{12}}{w_{11}}$$

C. *High Accuracy and High Fidelity Extraction of Neural Networks*

Key issues discussed in the paper are taxonomy, learning extraction and direct recovery extraction. First, It mentions some adversarial goals including functionality equivalent extraction, fidelity extraction, task accuracy extraction. So maybe a theft keeps going to reach the desired accuracy or maybe reconnaissance efforts to find a model which has the equal functionality the same as the main model. Also it talks about the adversarial capabilities which organize an adversary's knowledge into three categories including domain knowledge, deployment knowledge and model access. Also consider that without any natural data, learning based methods are hard. Actually in learning based methods, the adversary may collect public unlabeled data to mount his attack.

The article refers to learning based extraction (semi-supervised learning). So eve queries on its data and then trains different models. These models differ only in their non-determinism such as the initialization to start training. It used fashion MNIST as the dataset and its result differs on six percent accuracy. So we may say that eve couldn't have fidelity on these six percent.

Also it tries to improve fidelity using functionality equivalent attack and search for critical points and then obtain the weights if model as describe in previous section.



Symbol	Definition
d	Input dimensionality
h	Hidden layer dimensionality ($h < d$)
K	Number of classes
$A^{(0)} \in \mathbb{R}^{d \times h}$	Input layer weights
$B^{(0)} \in \mathbb{R}^{h}$	Input layer bias
$A^{(1)} \in \mathbb{R}^{h \times K}$	Logit layer weights
$B^{(1)} \in \mathbb{R}^{K}$	Logit layer bias

Figure 9 parameters for the functionality equivalent attack

After this improvement, the fidelity increase as shown in figure 10. But the problem is the number of queries to reach these fidelities are so high and not efficient.

# of Parameters	12,500	25,000	50,000	100,000
Fidelity	100%	100%	100%	99.98%
Queries	$2^{17.2}$	$2^{18.2}$	$2^{19.2}$	$2^{20.2}$

Figure 10 Fidelity of the functionally-equivalent extraction

D. Reverse-Engineering Deep ReLU Networks

The architecture, weights and biases of a deep ReLU network can be recovered from the arrangement of regions on which the network is linear. (It's proved in the article).

In the article, the author design an Algorithm to recover weight values and neuron's sign. Also he recovers the network by approximating the boundaries between linear regions. Just notice that the assumption is the output of the network $N(\mathbf{x})$ can be queried for different inputs $\mathbf{x}$, but does not assume any a priori knowledge of the linear regions or boundaries.

First we explain how to recover first layer weights. Consider the equation of these networks in the following:

$$z_j^k(x) = \sum_{i=1}^{n_{k-1}} W_{ij}^k ReLU(W_i^{k-1}(x) + b_i^k)$$

First he finds the boundary points along a line. This is simple and just do it by binary searching the points in the line and finding two points that are sufficiently close, but have differing gradients. Then we can use the algorithms in previous sections to find the hyperplanes. In this case we should keep going through the hyperplanes and check that they have a continuous gradient or not. This is important because the first layer neurons create a hyperplanes with continues gradients.

Algorithm 1 The first layer

Initialize $P_1 = P_2 = S_1 = \{\}$
for $t = 1, \ldots, L$ **do**
  Sample line segment ℓ
  $P_1 \leftarrow P_1 \cup \texttt{PointsOnLine}(\ell)$
end for
for $\mathbf{p} \in P_1$ **do**
  $H = \texttt{InferHyperplane}(\mathbf{p})$
  if $\texttt{TestHyperplane}(H)$ **then**
    $S_1 \leftarrow S_1 \cup \texttt{GetParams}(H)$
  else
    $P_2 \leftarrow P_2 \cup \{\mathbf{p}\}$
  end if
end for
return parameters S_1,
    unused sample points P_2

For extracting the parameters of additional layer neurons we move along a boundary to its intersection with another boundary. Then he identifies layer k of N, along with the sign of the parameters of layer k − 1, by measuring the extent to which boundaries bend at their intersection. Then he mentions a Theorem which formalizes the inductive step that allows to go from what we know at layer k − 1 (weights and biases, up to scaling and sign) to the equivalent set of information for layer k. therefore the adversary could find all weights and signs of a deep ReLU neural network.

The pseudo code of the algorithm for additional layers is shown below.

Algorithm 2 Additional layers

Input P_k and $S_1, \ldots, S_{k-1}$
Initialize $S_k = \{\}$
for $\mathbf{p}_1 \in P_{k-1}$ on boundary B_z **do**
  Initialize $A_z = \{\mathbf{p}_1\}$, $L_z = \mathcal{H}_z = \{\}$
  while $L_z \not\supseteq$ Layer $k-1$ **do**
    Pick $\mathbf{p}_i \in A$ and $\mathbf{v}$
    $\mathbf{p}', B_{z'} = \texttt{ClosestBoundary}(\mathbf{p}_i, \mathbf{v})$
    if $\mathbf{p}'$ on boundary **then**
      $A_z \leftarrow A_z \cup \{\mathbf{p}' + \epsilon\}$
      $L_z \leftarrow L_z \cup \{z'\}$
      $\mathcal{H}_z \leftarrow \mathcal{H}_z \cup \{\texttt{InferHyperplane}(\mathbf{p}_i)\}$
    else
      $P_k \leftarrow P_k \cup \{\mathbf{p}_1\}$; **break**
    end if
  end while
  if $L_z \supseteq$ Layer $k-1$ **then**
    $S_k \leftarrow \texttt{GetParams}(T_z)$
  end if
end for
return parameters S_k, unused sample points P_{k+1}

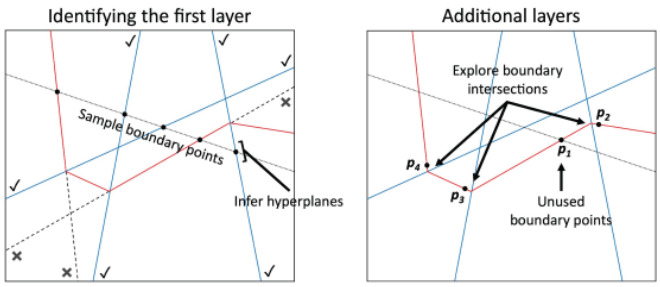


Figure 11 Schematic of our algorithms for identifying the first layer and additional layers

E. Cryptanalytic Extraction of Neural Network Models

Key issues discussed in the paper are extract deep models, efficient extraction and high fidelity extraction. First of all, let's answer three questions.

1. Given query access to neural network, can u extract the exact weights?
2. Given query access to neural network, can u extract the functionality equivalent model?
3. Given query access to neural network, learned through stochastic gradient descent, can u extract the functionality equivalent model?

The answer of first and second questions is no. the reason of 1st question answer is depicted below.

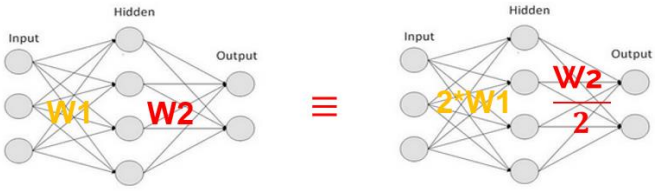


Figure 12 two same models in different structures

As you see here, these models have the same functionality and behavior. But it's obvious the structure especially the weight values aren't the same. So the answer of the 1st question is no. For the 2nd question, the simple theorem shows that it's impossible to extract the functionality equivalent model in which just given query access to the adversary. But the 3rd question has a positive answer. Just notice that f and g are (ε, δ)-functionality equivalent on S if:

$$Pr_{x \epsilon S}[|f(x) - g(x)| \leq \varepsilon] \geq (1 - \delta)$$

First we explain how to extract the first layer hyperplanes according to the paper. As you can see in figure 13, the hyperplanes of k-deep neural networks doesn't have continues gradient for k>1. But for the first layer, these are continues.

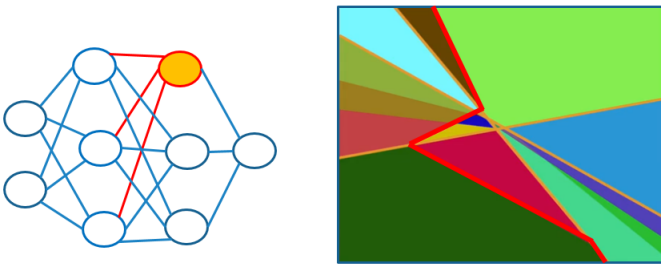


Figure 13 regions of a 2-deep layer neural network

Consider the above model, the adversary select three random line in the plane. Then he uses the algorithms in previous sections to find the hyperplanes. Although he knows the equations of hyperplanes but he doesn't know that which of them corresponds to layer 1 or the other layers. In this particular case the adversary checks which hyperplanes obtained from the points in different lines are the same.

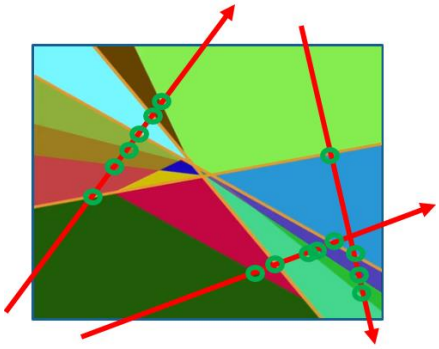


Figure 14 finding hyperplanes of first layer in deep neural networks

For second layer neurons we move along a boundary to its intersection with another boundary. Then we have some selection to keep going the movement and extract the hyperplanes of second layer neurons. In this case, the adversary knows which line is a correct one because of the first layer hyperplanes extraction. Now it's time to extract the signs. The signs will obtained by brute force attack. And now the attacker has all the unknown parameters of the first and second layer neurons.

There is a theoretic problem in the algorithm as it says if the lines are close to each other that intersect a hyperplane of the other layers in a continues gradient area, the adversary thinks this hyperplane corresponds to layer 1 but it doesn't. In response we say this challenge is theoretical correct but in implementation it is a very rare case.

The results of the paper [5] is shown below.

Architecture	Parameters	Approach	Queries	$(\varepsilon, 10^{-9})$	$(\varepsilon, 0)$	$\max \lvert\theta - \hat{\theta}\rvert$
784-32-1	25,120	[JCB+20]	$2^{18.2}$	$2^{3.2}$	$2^{4.5}$	$2^{-1.7}$
		Ours	$2^{19.2}$	$2^{-28.8}$	$2^{-27.4}$	$2^{-30.2}$
784-128-1	100,480	[JCB+20]	$2^{20.2}$	$2^{4.8}$	$2^{5.1}$	$2^{-1.8}$
		Ours	$2^{21.5}$	$2^{-26.4}$	$2^{-24.7}$	$2^{-29.4}$
10-10-10-1	210	[RK20]	2^{22}	$2^{-10.3}$	$2^{-3.4}$	2^{-12}
		Ours	$2^{16.0}$	$2^{-42.7}$	$2^{-37.98}$	2^{-36}
10-20-20-1	420	[RK20]	2^{25}	$\infty^{\dagger}$	$\infty^{\dagger}$	$\infty^{\dagger}$
		Ours	$2^{17.1}$	$2^{-44.6}$	$2^{-38.7}$	2^{-37}
40-20-10-10-1	1,110	Ours	$2^{17.8}$	$2^{-31.7}$	$2^{-23.4}$	$2^{-27.1}$
80-40-20-1	4,020	Ours	$2^{18.5}$	$2^{-45.5}$	$2^{-40.4}$	$2^{-39.7}$

Figure 15 the results of the paper [5] (cryptanalytic extraction of neural network models)

As you can see the results in figure [15], the number of queries and time complexity of the algorithm are the problems and these are efficient.

III. CONCLUSION

We introduce some methods for extracting unknown parameters of neural networks such as weights, number of

neurons and signs. Although the differential attack requires fewer queries per parameter than the other attacks, but it seems that all of them aren't efficient and time complexity and the number of their queries needed are so high. Also we saw that it is possible to recover the architecture, weights, and biases of deep ReLU networks from the boundaries. In paper [3] we realize the importance of accuracy and fidelity in model extraction and also saw that the learning-based methods can effectively attack a model with several millions of parameters. And also explain this attack does not perfect fidelity with the victim model. We also become familiar with equation solving attack and its performance on extracting machine learning models and we explain the other algorithms used for DT called decision tree path finding attack and etc.

REFERENCES

[1] Tramèr, F., Zhang, F., Juels, A., Reiter, M.K. and Ristenpart, T., 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)* (pp. 601-618).

[2] Milli, S., Schmidt, L., Dragan, A.D. and Hardt, M., 2019, January. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency* (pp. 1-9).

[3] Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A. and Papernot, N., 2020. High accuracy and high fidelity extraction of neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)* (pp. 1345-1362).

[4] Rolnick, D. and Kording, K., 2020, November. Reverse-engineering deep relu networks. In *International Conference on Machine Learning* (pp. 8178-8187). PMLR.

[5] Carlini, N., Jagielski, M. and Mironov, I., 2020, August. Cryptanalytic extraction of neural network models. In *Annual International Cryptology Conference* (pp. 189-218). Springer, Cham.