Extension of Model Extraction Using API Queries

Pranay Mathur Varun Bajpai

Department of Electrical Engineering, IIT Madras

May 10, 2025

Introduction

- AI models are expensive and widely deployed.
- Attackers can extract models using only API queries.
- This project extends an existing attack to:
  - Other activation functions (LeakyReLU, Tanh, Sigmoid)
  - Multi-class classifiers

Background

- Given attack in the paper works for ReLU activated, single output models.
- Works on the fact that ReLU is piecewise linear and can be modeled using matrix multiplications by mapping 1s and 0s.

Background

- Broad attack steps:
    1. Collect boundary points
    2. Recover model signature
    3. Recover weights
    4. Recover biases
    5. Filter functionally equivalent models

Scope of Project

- Extension to other activation functions
- Extension to multi-class outputs

Extension to other Activation Functions

Extension to LeakyReLU

- ReLU vs. LeakyReLU:

$$\mathrm{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

$$\mathrm{LeakyReLU}(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

Math behind the ReLU attack

$$
\begin{aligned}
f_\theta(x) &= A^{(k+1)} \cdots \left(I_P^{(1)} (A^{(1)} x + b^{(1)}) \right) \cdots + b^{(k+1)} \\
&= A^{(k+1)} I_P^{(k)} A^{(k)} \cdots I_P^{(2)} A^{(2)} I_P^{(1)} A^{(1)} x + B_P \\
&= \Gamma_P x + B_P
\end{aligned}
$$

Extension to LeakyReLU

- Changes in approach: Replace 0s with α in diagonal matrices.
- Code update: Modified forward pass in f_cheat() and get_maps() and signature comparison in compare_model_signatures().

Extension to LeakyReLU

```
    for j in range(di):
        if ps[i][j] == 1:
            DMs[i][j][j] = 1
        else:
            DMs[i][j][j] = LEAKY_ALPHA
```

```
for i in range(layer_num):
    h = np.matmul(ws[i], h) + bs[i]
    if i != layer_num - 1:
        #h = h * (h > 0)
        h = np.where(h > 0, h, LEAKY_ALPHA * h)
assert len(h) == 1
soft_label = np.squeeze(h)
```

```
        map.append(tp)

        #h = h * (h > 0)
        h = np.where(h > 0, h, LEAKY_ALPHA * h)
return map
```

Figure: Changes for LeakyReLU

Extension to Tanh

- Approximate tanh as piecewise linear:

$$\tanh(x) \approx \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } |x| \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$

- Define active region as $-1 \leq x \leq 1$.

Extension to Tanh

- Change in approach:
  1. Change the regions for which 1s and 0s are mapped onto the matrix.
  2. Add a bias vector to take care of ± 1
- $$\begin{aligned} f_{\boldsymbol{\theta}}(x) &= A^{(k+1)} \cdots \left(I_P^{(1)} (A^{(1)} x + b^{(1)}) + V_P^{(1)} \right) \cdots + b^{(k+1)} \\ &= \Gamma'_P x + B'_P \end{aligned}$$
- Changes were made in the f_cheat() and get_maps() function.

Extension to Tanh

```
tp = np.squeeze((h > -1) & (h < 1)).astype(int)
tp = convert_array_into_scalar(tp)
map.append(tp)

# optional: still mask h as before, or choose a new transformation
h = np.tanh(h)
```

```
for i in range(layer_num):
    h = np.matmul(ws[i], h) + bs[i]
    if i != layer_num - 1:
        h = np.tanh(h)
assert len(h) == 1
```

Figure: Changes for Tanh

Extension to Sigmoid

- Approximate sigmoid:
  - 0 if $x < -2$
  - $0.25x + 0.5$ if $-2 \leq x \leq 2$
  - 1 if $x > 2$
- Activation slope $= 0.25$ in linear region.
- Add constant bias for inactive neurons.

Extension to Sigmoid

- Change in approach:
    1. Change the regions for which 1s and 0s are mapped onto the matrix.
    2. Add a bias vector to take care of 0, 0.5 and 1.
- Changes were made in the f_cheat() and get_maps() function and compare_model_signatures().

Extension to Sigmoid

```python
# compute the diagonal matrix
for i in range(hidden_layer_num):
    di = di_s[i+1]
    for j in range(di):
        if ps[i][j] == 1:
            DMs[i][j][j] = 0.25
```

```python
if i != layer_num - 1:
    # new activation pattern: 1 if -1 < h < 1, else 0
    tp = np.squeeze((h > -2) & (h < 2)).astype(int)
    tp = convert_array_into_scalar(tp)
    map.append(tp)

    h = sigmoid(h)
```

```python
for i in range(layer_num):
    h = np.matmul(ws[i], h) + bs[i]
    if i != layer_num - 1:
        h = sigmoid(h)
        # h = h * ((h > -1) & (h < 1))
```

Figure: Changes for Sigmoid

Extension to Multi-class outputs

Multi-Class Extension: Relative Activation

- The attack works on the **relative values** of output neurons, not absolute values.
- If one neuron has a larger value than another, it is selected as the predicted class.
- Hence, the attack reduces to **extracting relative weights** between pairs of neurons.
- For two neurons with weights w_1 and w_2, if we can recover $w_1 - w_2$, we can reconstruct both:
  - Initialize w_1 arbitrarily.
  - Compute $w_2 = w_1 - (w_1 - w_2)$.

Pseudo-Model Representation

- Construct a pseudo-model where output weights encode only the **differences** between original weights.
- The existing single-output attack can then be applied to this difference-model.
- After extracting differences, one weight vector (e.g., w_1) is initialized randomly.
- Other output weights are reconstructed relatively.
- **This strategy generalizes** to k output neurons by applying pairwise differences iteratively.

Implementation Challenges

- Provided repository code failed on our custom-trained models, despite matching architecture.
- Likely due to undocumented pre-processing in the original implementation.
- Our workaround: Normalize inputs and divide by 100.
- Extraction worked $\sim$50% of the time — this was used as a **baseline**.
- Success varied due to randomness in starting point and boundary direction.

Hyperparameter Tuning and Results

- Tuned three key hyperparameters:
  - **Precision**: controls stride accuracy
  - **L1 Error**: filters out unequal gammas
  - **MS (step size)**: affects weight sign calculation
- LeakyReLU performed well with default settings.
- For Tanh and Sigmoid, reduced MS to 10^{-4} for better performance.

Results

Activation	Precision	L1 Error	Step Size	PMR
ReLU	10^{-12}	10^{-2}	10^{-4}	1.000
LeakyReLU	10^{-12}	10^{-2}	10^{-6}	0.994
Tanh	10^{-12}	10^{-2}	10^{-4}	0.998
Sigmoid	10^{-12}	10^{-2}	10^{-4}	0.988

Table: Comparison of model extraction success rate (PMR)

Limitations and Future Work

- Couldn't extract 3-class classifiers with provided base code.
- Unreliable behavior on custom models with 2+ hidden layers.
- Future work:
  - Improve base algorithm robustness
  - Generalize to deeper and more complex models

Conclusion

- Successfully extended extraction attack to:
  - LeakyReLU, Tanh, Sigmoid
  - Multi-class classifiers
- Reinforces risks of exposing models via public APIs.
- Stronger model security is essential.