Motivation

Black-box models usually hide internal states on purpose:
1. Protecting intellectual properties (IP)
2. Covering privacy-sensitive training data
**Motivation**

**Black-box models** usually hide **internal states** on purpose:
1. Protecting intellectual properties (IP)
2. Covering privacy-sensitive training data

Why hiding the information?
1. Preventing the model from adversarial attacks
2. Protecting privacy data, such as faces
In order to **increase the chance of protecting the model** from being attacked, we need to gain more knowledge on black-box models.
Motivation

In order to increase the chance of protecting the model from being attacked, we need to gain more knowledge on black-box models.

Double-sided blade:
Disclosing the hidden detail may make the model much easier to be attacked by adversaries
1. Model attributes:
   a. architecture (non-linear activation)
   b. optimisation process (SGD or ADAM)
   c. training data
2. Metamodel:
   - Takes models as input and returns the corresponding model attributes as output

3. Meta-training set:
   - a diverse set of white-box models with different model attributes
A standard supervised learning task applied over models

1. Collect meta-training set
2. Train metamodel by using meta-training set
3. Predict attributes for black-box models
Related Work on Extracting Model Information

- Model extraction via querying ML APIs
  - (Tramer et al., 2016): reconstruct the exact model parameters
  - (Papernot et al., 2017): build a local avatar model

- Extracting information from the training data
  - (Ateniese et al., 2015) build a meta-classifier to obtain statistical information about the training set
  - (Shokri et al., 2017) proposed membership inference attack that can determine if a given data sample is part of the training data
Attacking Black-box Models Using Extracted Information

- **Adversarial image perturbations** (AIPs): small imperceptible perturbations over the input that fool the target model.

- Approaches:
  - Gradient / saliency map attacks
    - Problem --> requires millions of queries to find a single AIP
  - Avatar approach: train a local white box model similar to the target model
  - Exploit transferability of adversarial examples that generated for one model to attack other models
Attributes of neural networks can be exposed from a sequence of queries.

Revealed internal information helps generate more effective adversarial examples against the black box model.
An Intuitive Figure Showing WHY Claim

1. Collect Meta-training set
2. Train Metamodell
3. Query Black-box Model
4. Predict Black-box Model Attributes
5. Train A Local Model using Predicted Attributes
6. Attack Target Model
Proposed Solution
- Classifier of classifiers
- Uses model f as black box
- Submits n query inputs to f
- Takes corresponding model outputs as input
- Returns predicted attributes as output

Figure 1: Overview of our approach.
Preparing training data

MNIST-NETS

- 12 attributes
- 18,144,000 combinations

Sample 10000

pruned low-performance classifiers

(validation accuracy < 98%)

Table 1: MNIST classifier attributes. *Italicised* attributes are derived from other attributes.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Code</th>
<th>Attribute</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>act</td>
<td>Activation</td>
<td>ReLU, PReLU, ELU, Tanh</td>
</tr>
<tr>
<td></td>
<td>drop</td>
<td>Dropout</td>
<td>Yes, No</td>
</tr>
<tr>
<td></td>
<td>pool</td>
<td>Max pooling</td>
<td>Yes, No</td>
</tr>
<tr>
<td></td>
<td>ks</td>
<td>Conv ker. size</td>
<td>3, 5</td>
</tr>
<tr>
<td></td>
<td>#conv</td>
<td>#Conv layers</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td></td>
<td>#fc</td>
<td>#FC layers</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td></td>
<td>#par</td>
<td>#Parameters</td>
<td>2^{14}, \ldots, 2^{21}</td>
</tr>
<tr>
<td></td>
<td>ens</td>
<td>Ensemble</td>
<td>Yes, No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Opt.</th>
<th>Code</th>
<th>Attribute</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alg</td>
<td>Algorithm</td>
<td>SGD, ADAM, RMSprop</td>
</tr>
<tr>
<td></td>
<td>bs</td>
<td>Batch size</td>
<td>64, 128, 256</td>
</tr>
<tr>
<td></td>
<td>split</td>
<td>Data split</td>
<td>All, Half, Quarter</td>
</tr>
<tr>
<td></td>
<td>size</td>
<td>Data size</td>
<td>All, Half, Quarter</td>
</tr>
</tbody>
</table>
KENNEN-O: REASON OVER OUTPUT

- Submits a fixed query of images to $f$ as inputs (Fixed across training and testing)
- Takes the output from $f$ and predicts the 12 attributes

$$\min_{\theta} \mathbb{E}_{f \sim \mathcal{F}} \left[ \sum_{a=1}^{12} \mathcal{L}(m_{\theta}^a ([f(x^i)]_{i=1}^n), y^a) \right]$$
KENNEN-I: CRAFT INPUT

- Can only predict a single attribute at a time
- Crafts an input that drives $f$ to leak internal information
- Limited predictable classes
KENNEN-IO: COMBINED APPROACH

- Overcomes the drawbacks of kennen-i: single attribute prediction
- Combine kennen-o and kennen-i approaches (Input generator + output interpreter)
- Support optimization of multiple query inputs
Experimental Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Output</th>
<th>Architecture</th>
<th>Optim</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>act</td>
<td>drop</td>
<td>pool</td>
</tr>
<tr>
<td>chance</td>
<td>-</td>
<td>25.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>kennen-o</td>
<td>prob</td>
<td>80.6</td>
<td>94.6</td>
<td>94.9</td>
</tr>
<tr>
<td>kennen-o</td>
<td>ranking</td>
<td>63.7</td>
<td>93.8</td>
<td>90.8</td>
</tr>
<tr>
<td>kennen-o</td>
<td>bottom-1</td>
<td>48.6</td>
<td>80.0</td>
<td>73.6</td>
</tr>
<tr>
<td>kennen-o</td>
<td>top-1</td>
<td>31.2</td>
<td>56.9</td>
<td>58.8</td>
</tr>
<tr>
<td>kennen-i</td>
<td>top-1</td>
<td>43.5</td>
<td>77.0</td>
<td>94.8</td>
</tr>
<tr>
<td>kennen-io</td>
<td>score</td>
<td>88.4</td>
<td>95.8</td>
<td>99.5</td>
</tr>
</tbody>
</table>

Comparison of metamodel methods

- **kennen-io** gives the best performance with an avg. accuracy of 80.1%
- **kennen-i** has relatively low performance, but it only relies on single query
- **bottom-1** outputs contain much more information than do the top-1 outputs

Output representations from the black-box model:
- “prob”: vector of probabilities for each digit class
- “ranking”: a sorted list of digits according to their likelihood
- “top-1”: most likely digit
- “bottom-1”: least likely digit

100 queries are used for every method, except for kennen-i, which uses a single query.
Factor Analysis on kannen-o

- Diminishing return in larger size of training set, but the performance still continues to improve
- Average performance saturates after \(~500\) queries, but \(~100\) queries is good enough
Reverse Engineering & Attacking ImageNet Classifiers

- Metamodel strengthens the transferability based attack
- AIPs transfer better within the architecture family than across

Transferability of adversarial examples within and across families (metric: misclassification rate)
Metamodels Enables More Effective Attacks

- AIPs generated for metamodel’s predicted family model is more effective than pure black-box attack
- It almost reach the performance of the case when the family is known

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Generating nets</th>
<th>MC(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>White box</td>
<td>Single white box</td>
<td>100.0</td>
</tr>
<tr>
<td>Family black box</td>
<td>GT family</td>
<td>86.2</td>
</tr>
<tr>
<td><strong>Black box whitened</strong></td>
<td><strong>Predicted family</strong></td>
<td><strong>85.7</strong></td>
</tr>
<tr>
<td>Black box</td>
<td>Multiple families</td>
<td>82.2</td>
</tr>
</tbody>
</table>

Black-box ImageNet classifier misclassification rates (MC) for different approaches
Conclusion and Future Work

1. Investigated types of internal information can be extracted from querying
2. Proposed novel metamodel methods
3. Analyze the impact of different factors on metamodel
4. They showed that reverse-engineering enables more effective attacks
References