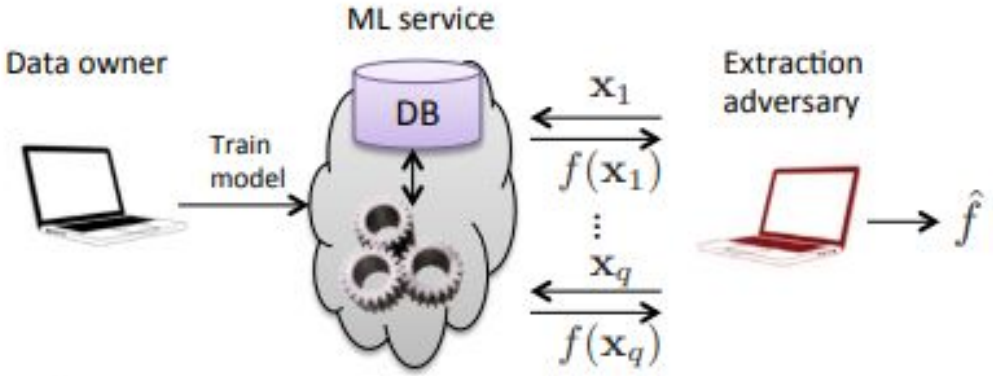
Model Stealing

Nikita Dhawan, Eric Liu, Ben Eyre

UNIVERSITY OF TORONTO

What is Model Stealing?



- Extract an approximation that of the target model that “closely matches” the original
  - Accuracy?
  - Fidelity?
  - Functional equivalence?

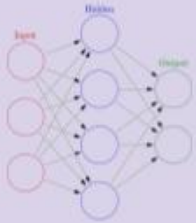
Threat Models

API Access

Model extraction using:
- Prediction Vectors
- Labels Only

Model Access

Obfuscate the use of the model by:
- Fine-tuning
- Distillation



Data Access

Use the private dataset by:
- Training a new model from scratch
- Distilling the target model (requires API access as well)

My Model was Stolen: So What?

- Machine learning models may require a very large amount of resources to create:
  - Research and Development
  - Creating Private Datasets
  - Compute Costs

Model	Cost
GPT2	$256/hour
XLNET	$245,000
GPT3	$4.6 Million

My Model was Stolen: So What?

- Having your model stolen can create new vulnerabilities for it:
  - Data privacy issues through model-inversion/membership inference

Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score.

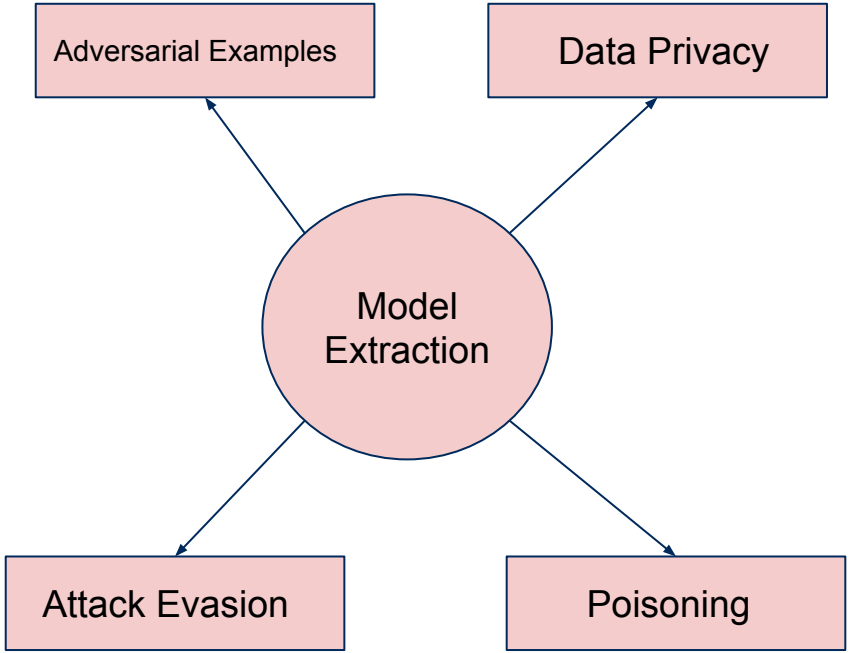
Retrieved from [1]

My Model was Stolen: So What?

Having your model stolen can create new vulnerabilities for it:

- Enables the use of white-box adversarial example creation

If a model extraction attack is successful, the victim loses the **information asymmetry advantage** that is integral in defences for several other kinds of attacks.



Outline

- First Paper: Black-box techniques for extracting a model using a query API
- Second Paper: Detect model extraction by characterizing behaviour specific to the victim model
- Third Paper: Detecting model extraction by characterizing behaviour specific to the victim's training set

Stealing Machine Learning Models via Prediction APIs

Tramèr et al., 2016

Contributions

- Show the effectiveness of simple equation solving extraction attacks
- Novel algorithm for extracting decision trees from non-boolean features
- Demonstrate that extraction attacks still work against models that output only class labels
- Application of these methods to real MLaaS interfaces

Threat Model & Terminology

- Focus on proper model extraction
- Attacker has “black-box” access
  - This includes any info and statistics provided by the ML API
- Distance:
  - Predicted classes: 0-1 distance
  - Probabilities: total variation distance $d(\mathbf{y},\mathbf{y}') = \frac{1}{2}\sum|\mathbf{y}[i]-\mathbf{y}'[i]|$
- Test error: $R_{\text{test}}(f,\hat{f}) = \sum_{(\mathbf{x},y)\in D} d(f(\mathbf{x}),\hat{f}(\mathbf{x}))/|D|$
- Uniform error: $R_{\text{unif}}(f,\hat{f}) = \sum_{\mathbf{x}\in U} d(f(\mathbf{x}),\hat{f}(\mathbf{x}))/|U|$
- Extraction Accuracy: $1-R_{\text{test}}(f,\hat{f})$ and $1-R_{\text{unif}}(f,\hat{f})$

Equation Solving Attacks

Equation Solving Attacks: Binary LR

$$f_1(x) = \sigma\left(\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \beta\right), \text{ where } \sigma(t) = \frac{1}{(1+e^{-t})}$$

$$\sigma^{-1}\left(f_1\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}\right)\right) = \begin{pmatrix} w_1 \\ 0 \end{pmatrix} + \beta$$

$$\sigma^{-1}\left(f_1\left(\begin{pmatrix} 0 \\ 1 \end{pmatrix}\right)\right) = \begin{pmatrix} 0 \\ w_2 \end{pmatrix} + \beta$$

$$\sigma^{-1}\left(f_1\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}\right)\right) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \beta$$

With d+1 queries, we can exactly recover the parameters

Equation Solving Attacks

- What about more complicated models?

- The paper shows that these attacks can extend to all model classes with a “logistic” layer

Equation Solving Attacks: MLR, MLP, NN

- MLR, MLP, NN, are non-linear and have no analytic solution
- Softmax regression:
  - Each **(x, f(x))** sample gives **c** equations in **w** and ***β***
  - Strongly convex with a regularization term => converges to global minimum
  - Minimize loss function to extract parameters with random inputs

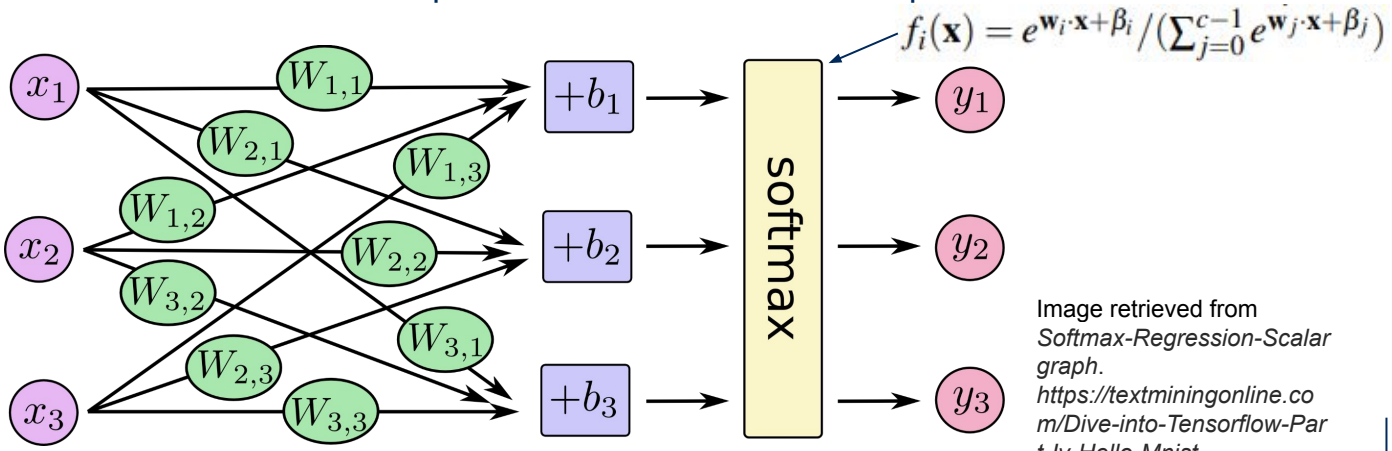


Image retrieved from *Softmax-Regression-Scalar graph. https://textminingonline.com/Dive-into-Tensorflow-Part-Iv-Hello-Mnist.*

Equation Solving Attacks: MLR, MLP, NN

- Multi-layer perceptron and neural networks:
  - A lot more parameters (e.g. one layer perceptron: $\mathbf{w} \in \mathbb{R}^{dh+hc}, \hat{\boldsymbol{\beta}} \in \mathbb{R}^{h+c}$)
  - Not strongly convex => may converge to local minimum

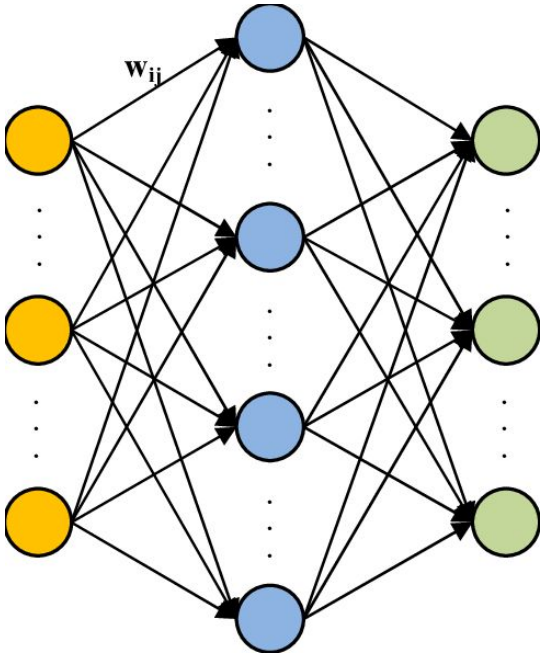


Image retrieved from: *A recognition approach using multilayer perceptron and keyboard dynamics patterns - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Topology-of-multilayer-perceptron-with-a-single-hidden-layer-as-a-classifier_fig1_261463606 [accessed 30 Sep, 2021]*

Equation Solving Attacks: Evaluation

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

Table 4: **Success of equation-solving attacks.** Models to extract were trained on the Adult data set with multiclass target 'Race'. For each model, we report the number of unknown model parameters, the number of queries used, and the running time of the equation solver. The attack on the MLP with 11,125 queries converged after 490 epochs.

- Is this attack feasible on DNN given the number of queries?
- Are "random" inputs good enough to learn an accurate model for inputs with high dimensional feature space?

Equation Solving Attacks: Amazon

Data set	Synthetic	# records	# classes	# features
Circles	Yes	5,000	2	2
Moons	Yes	5,000	2	2
Blobs	Yes	5,000	3	2
5-Class	Yes	1,000	5	20
Adult (Income)	No	48,842	2	108
Adult (Race)	No	48,842	5	105
Iris	No	150	3	4
Steak Survey	No	331	5	40
GSS Survey	No	16,127	3	101
Digits	No	1,797	10	64
Breast Cancer	No	683	2	10
Mushrooms	No	8,124	2	112
Diabetes	No	768	2	8

Model	OHE	Binning	Queries	Time (s)	Price ($)
Circles	-	Yes	278	28	0.03
Digits	-	No	650	70	0.07
Iris	-	Yes	644	68	0.07
Adult	Yes	Yes	1,485	149	0.15

Table 7: **Results of model extraction attacks on Amazon.** OHE stands for one-hot-encoding. The reported query count is the number used to find quantile bins (at a granularity of 10^{-3}), plus those queries used for equation-solving. Amazon charges $0.0001 per prediction [1].

- Case study: Amazon Web Services
  - Feature extraction takes extra reverse engineering which means more queries!
  - Attacker has knowledge of the feature extraction techniques

Equation Solving Attacks: Data Leakage

- Kernel Logistic regression
  - Intuition: Kernel LR replaces $\mathbf{w}_i \cdot \mathbf{x} + \beta_i$ with $\sum_{r=1}^{s} \alpha_{i,r} K(\mathbf{x}, \mathbf{x}_r) + \beta_i$. When we extract the parameters, the representers leak the "average" of each class of the training data

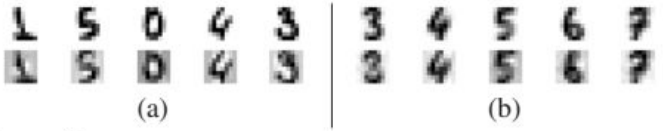


Figure 2: **Training data leakage in KLR models.** (a) Displays 5 of 20 training samples used as representers in a KLR model (top) and 5 of 20 extracted representers (bottom). (b) For a second model, shows the average of all 1,257 representers that the model classifies as a 3,4,5,6 or 7 (top) and 5 of 10 extracted representers (bottom).

- Model Inversion
  - Intuition: "find the input that maximizes the returned confidence, subject to the classification also matching the target [1]"
  - *Model inversion* works better on white-box
  - Extracting the model first results in less queries

Figure 1: An image recovered using a new model inversion attack (left) and a training set image of the victim (right). The attacker is given only the person's name and access to a facial recognition system that returns a class confidence score.

Decision Tree Path-Finding Attacks

Decision Tree Path-Finding

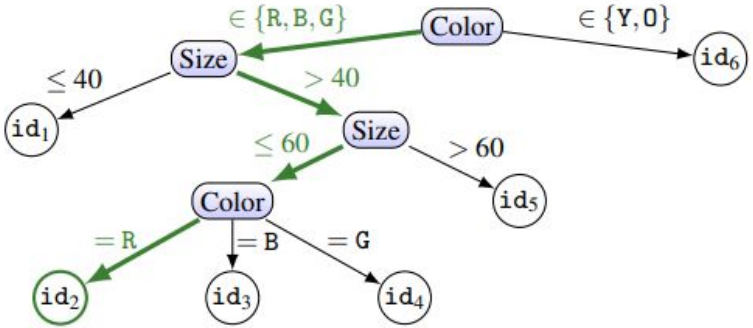


Figure 3: **Decision tree over features Color and Size.** Shows the path (thick green) to leaf $\mathtt{id}_2$ on input $\mathbf{x} = \{Size = 50,\ Color = R\}$.

- Adversary has access to oracle that returns the leaf ID (or even the node ID for partial queries)
- Algorithm goals:
  - Find predicates that input has to satisfy to reach leaf node (e.g. What predicates to get **x** to **id2**)
  - Generate new inputs to visit unexplored paths

DT Path Finding Algorithm

- LINE_SEARCH()
  - Query oracle for upper and lower bounds
  - If IDs do not match, perform binary search to find all intervals with different leaf IDs
- CATEGORY_SPLIT()
  - Query oracle on all categories
  - Find set of values S that lead to id and all other leaves V

Algorithm 1 **The path-finding algorithm.** The notation $\texttt{id} \leftarrow \mathcal{O}(\mathbf{x})$ means querying the leaf-identity oracle $\mathcal{O}$ with an input $\mathbf{x}$ and obtaining a response $\texttt{id}$. By $\mathbf{x}[i] \Rightarrow v$ we denote the query $\mathbf{x}'$ obtained from $\mathbf{x}$ by replacing the value of x_i by v.

$\mathbf{x}_{\text{init}} \leftarrow \{x_1, \ldots, x_d\}$ ▷ random initial query
$Q \leftarrow \{\mathbf{x}_{\text{init}}\}$ ▷ Set of unprocessed queries
$P \leftarrow \{\}$ ▷ Set of explored leaves with their predicates
while Q not empty **do**
  $\mathbf{x} \leftarrow Q$.POP()
  $\texttt{id} \leftarrow \mathcal{O}(\mathbf{x})$ ▷ Call to the leaf identity oracle
  if $\texttt{id} \in P$ **then** ▷ Check if leaf already visited
    continue
  end if
  for $1 \leq i \leq d$ **do** ▷ Test all features
    if IS_CONTINUOUS(i) **then**
      for $(\alpha, \beta] \in$ LINE_SEARCH$(\mathbf{x}, i, \varepsilon)$ **do**
        if $x_i \in (\alpha, \beta]$ **then**
          $P[\texttt{id}]$.ADD('$x_i \in (\alpha, \beta]$') ▷ Current interval
        else
          Q.PUSH($\mathbf{x}[i] \Rightarrow \beta$) ▷ New leaf to visit
        end if
      end for
    else
      $S, V \leftarrow$ CATEGORY_SPLIT$(\mathbf{x}, i, \texttt{id})$
      $P[\texttt{id}]$.ADD('$x_i \in S$') ▷ Values for current leaf
      for $v \in V$ **do**
        Q.PUSH($\mathbf{x}[i] \Rightarrow v$) ▷ New leaves to visit
      end for
    end if
  end for
end while

DT Path Finding Algorithm

- "Top-down" variant
  - Uses partial queries
  - Extracts tree "layer by layer"
  - Empirically more efficient

Algorithm 1 **The path-finding algorithm.** The notation $\texttt{id} \leftarrow \mathcal{O}(\mathbf{x})$ means querying the leaf-identity oracle $\mathcal{O}$ with an input $\mathbf{x}$ and obtaining a response $\texttt{id}$. By $\mathbf{x}[i] \Rightarrow v$ we denote the query $\mathbf{x}'$ obtained from $\mathbf{x}$ by replacing the value of x_i by v.

$\mathbf{x}_{\text{init}} \leftarrow \{x_1, \ldots, x_d\}$ ▷ random initial query
$Q \leftarrow \{\mathbf{x}_{\text{init}}\}$ ▷ Set of unprocessed queries
$P \leftarrow \{\}$ ▷ Set of explored leaves with their predicates
while Q not empty **do**
  $\mathbf{x} \leftarrow Q$.POP()
  $\texttt{id} \leftarrow \mathcal{O}(\mathbf{x})$ ▷ Call to the leaf identity oracle
  if $\texttt{id} \in P$ **then** ▷ Check if leaf already visited
    continue
  end if
  for $1 \leq i \leq d$ **do** ▷ Test all features
    if IS_CONTINUOUS(i) **then**
      for $(\alpha, \beta] \in$ LINE_SEARCH$(\mathbf{x}, i, \varepsilon)$ **do**
        if $x_i \in (\alpha, \beta]$ **then**
          $P[\texttt{id}]$.ADD('$x_i \in (\alpha, \beta]$') ▷ Current interval
        else
          Q.PUSH($\mathbf{x}[i] \Rightarrow \beta$) ▷ New leaf to visit
        end if
      end for
    else
      $S, V \leftarrow$ CATEGORY_SPLIT$(\mathbf{x}, i, \texttt{id})$
      $P[\texttt{id}]$.ADD('$x_i \in S$') ▷ Values for current leaf
      for $v \in V$ **do**
        Q.PUSH($\mathbf{x}[i] \Rightarrow v$) ▷ New leaves to visit
      end for
    end if
  end for
end while

DT Path Finding Algorithm: Evaluation

Model	Leaves	Unique IDs	Depth	Without incomplete queries			With incomplete queries		
				$1 - R_{\text{test}}$	$1 - R_{\text{unif}}$	Queries	$1 - R_{\text{test}}$	$1 - R_{\text{unif}}$	Queries
IRS Tax Patterns	318	318	8	100.00%	100.00%	101,057	100.00%	100.00%	29,609
Steak Survey	193	28	17	92.45%	86.40%	3,652	100.00%	100.00%	4,013
GSS Survey	159	113	8	99.98%	99.61%	7,434	100.00%	99.65%	2,752
Email Importance	109	55	17	99.13%	99.90%	12,888	99.81%	99.99%	4,081
Email Spam	219	78	29	87.20%	100.00%	42,324	99.70%	100.00%	21,808
German Credit	26	25	11	100.00%	100.00%	1,722	100.00%	100.00%	1,150
Medical Cover	49	49	11	100.00%	100.00%	5,966	100.00%	100.00%	1,788
Bitcoin Price	155	155	9	100.00%	100.00%	31,956	100.00%	100.00%	7,390

Table 6: **Performance of extraction attacks on public models from BigML.** For each model, we report the number of leaves in the tree, the number of unique identifiers for those leaves, and the maximal tree depth. The chosen granularity ε for continuous features is 10^{-3}.

- Authors state that duplicate IDs can cause missed predicates
- What granularity to use for continuous features in LINE_SEARCH?

Class Label Only Extraction

Class Label Only Extraction: Lowd-Meek Attack

- Intuition:
  - use line search to find points arbitrarily close to $\mathbf{f}$'s decision boundary and solve for parameters from these points (i.e. $\mathbf{w} \cdot \mathbf{x} + \beta \approx 0$)
- Extension to non-linear:
  - first derive projection to transform model into linear one in transformed feature space

$$K_{poly}(\mathbf{x},\mathbf{x}') = (\mathbf{x}^T \cdot \mathbf{x}' + 1)^d$$
$$\text{Find } \phi(\cdot) \text{ so that}$$
$$K_{poly}(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T \cdot \phi(\mathbf{x}')$$

Class Label Only Extraction: Retraining

- Retrain model locally based on queries and oracle labels
- Uniform queries
- Line-search retraining (generalization of Lowd-Meek)
- Adaptive retraining
  - Query in $\mathbf{m/r}$ batches, where $\mathbf{m}$ is query budget and $\mathbf{r}$ is # of rounds
  - Select points along decision boundary of extracted model
  - Intuition: select points that the extracted model is least certain about to be used in each batch

Class Label Only Extraction: Evaluation

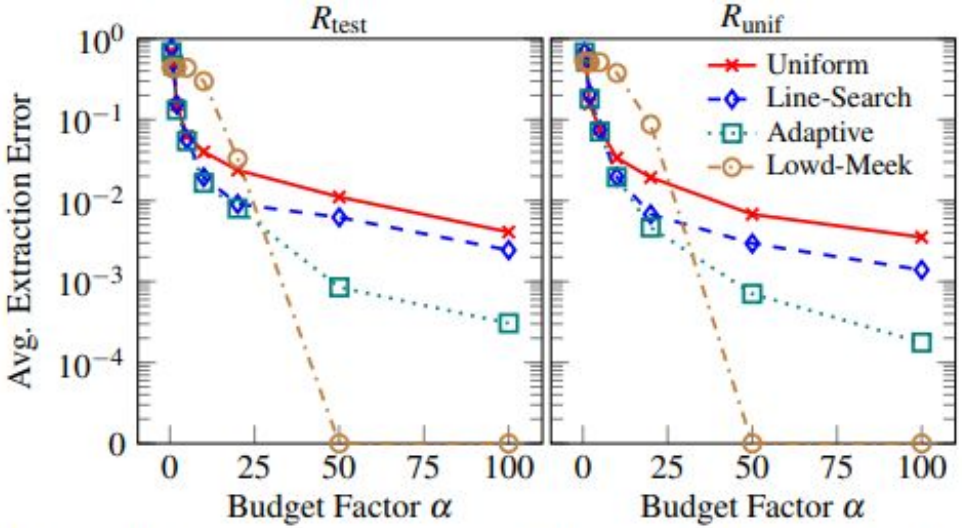


Figure 4: **Average error of extracted linear models.** Results are for different extraction strategies applied to models trained on all binary data sets from Table 3. The left shows R_{test} and the right shows R_{unif}.

Class Label Only Extraction: Evaluation

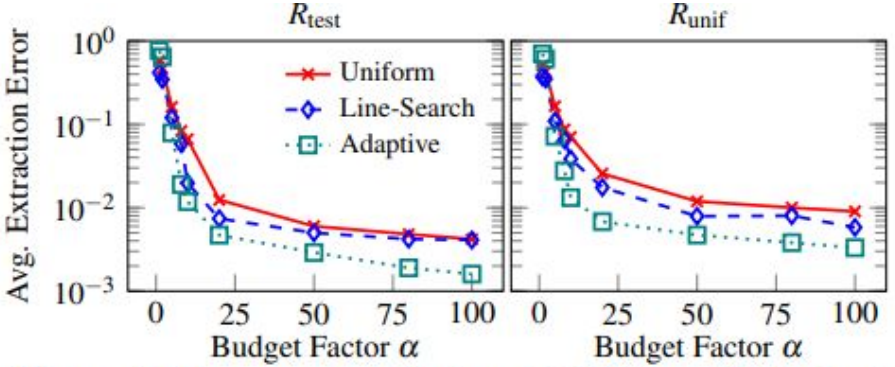


Figure 6: **Average error of extracted RBF kernel SVMs** Results are for three retraining strategies applied to models trained on all binary data sets from Table 3. The left shows R_{test} and the right shows R_{unif}.

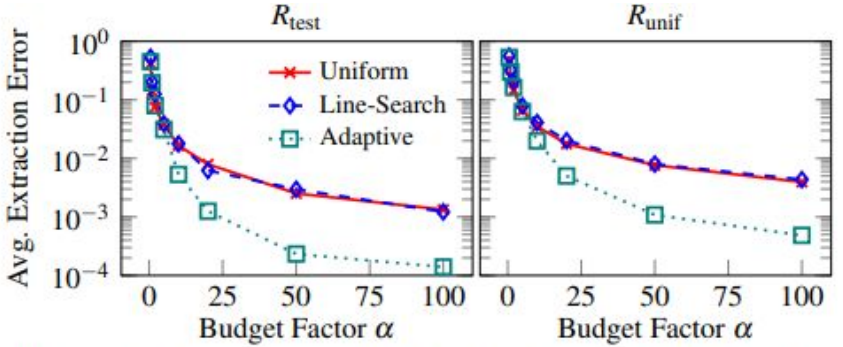


Figure 5: **Average error of extracted softmax models.** Results are for three retraining strategies applied to models trained on all multiclass data sets from Table 3. The left shows R_{test} and the right shows R_{unif}.

Class Label Only Extraction: Defenses

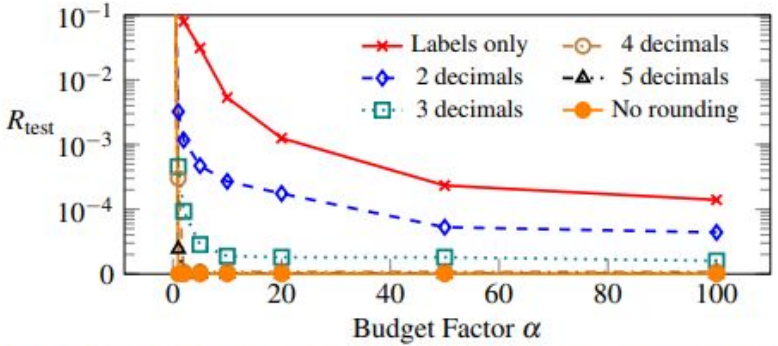


Figure 7: **Effect of rounding on model extraction.** Shows the average test error of equation-solving attacks on softmax models trained on the benchmark suite (Table 3), as we vary the number of significant digits in reported class probabilities. Extraction with no rounding and with class labels only (adaptive retraining) are added for comparison.

- Limiting prediction info (e.g. class label only, modifying, withholding, or rounding confidence values)
- Ensemble methods
- Applying differential privacy to model parameters

Limitations

- Even simple neural networks require a lot of queries!
  - 20 hidden nodes requires around 4,000 queries for confidence scores and 108,200 for class labels to get >99% accuracy
- NN are not strongly convex => impossible to extract exact parameters
- Focus on “proper” model extraction
- Feature extraction requires reverse engineering
- “Black-box” but their experimental results use knowledge of the architecture, feature extraction techniques, feature space of inputs
- Learning-based extraction is non-deterministic
  - Random initialization of queries and model parameters

Conclusions & Future Work

- Model extraction is hard!
  - Extracting a functionally equivalent model has exponential hardness [1]
- Future work should evaluate attacks on state of the art networks
- Functional equivalence, fidelity, and task accuracy
  - Follow up: High Accuracy and High Fidelity Extraction of Neural Networks
- The paper discusses ways to prevent model stealing, but does not comment on what to do after the fact => next paper!

Entangled Watermarks as a Defense against Model Extraction

Jia et al., 2021

Outline

1. Model Extraction: Adversary's Goal and Method
2. Watermarking: Defender's Goal and Method
3. Failures of Naive Watermarking
4. Entangled Watermarking Embeddings (EWE)
5. Validating and Evaluating EWE
6. Attacking Watermarking
7. Criticism and Discussion

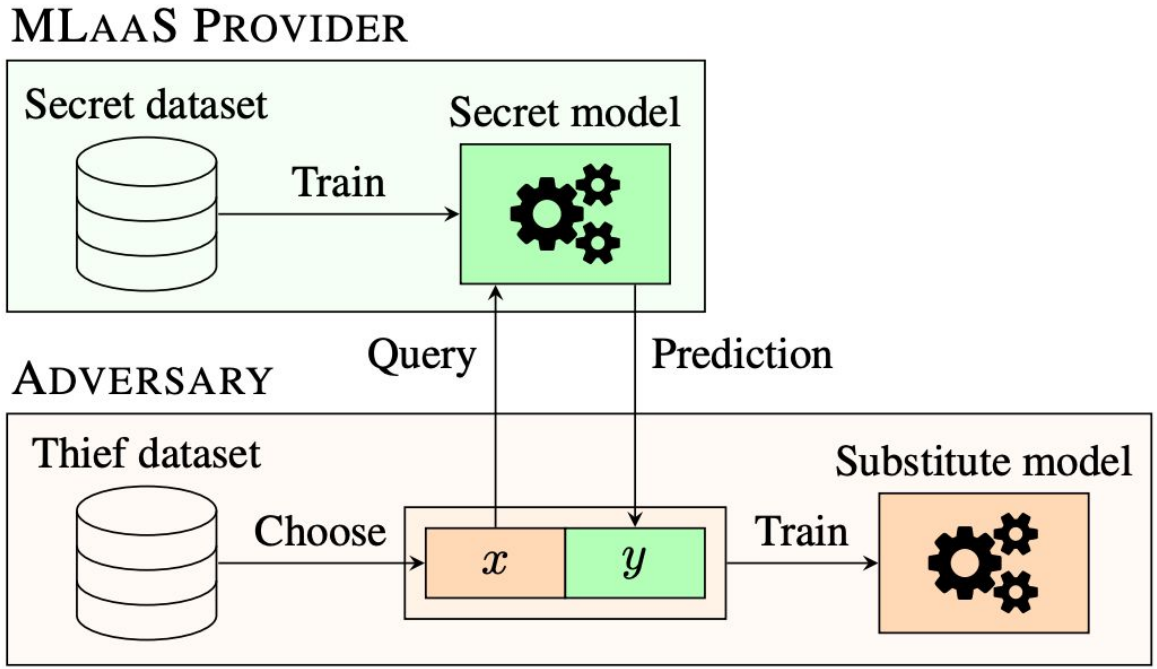
Model Extraction: Adversary's Goal

Labeling training data is expensive!

→ Use a victim model to use as an oracle for label

- Theft: reuse stolen copy for own benefit
- Reconnaissance: gain insights to help launch another attack

Model Extraction: Adversary's Method



Pal et al., A framework for the extraction of Deep Neural Networks by leveraging public data

Watermarking: Defender's Goal

Classes of Defense:

- Detection of model stealing / extraction
- Prevention of model stealing / extraction
- Ownership resolution = claim ownership upon inspection of models that may be believed to be stolen

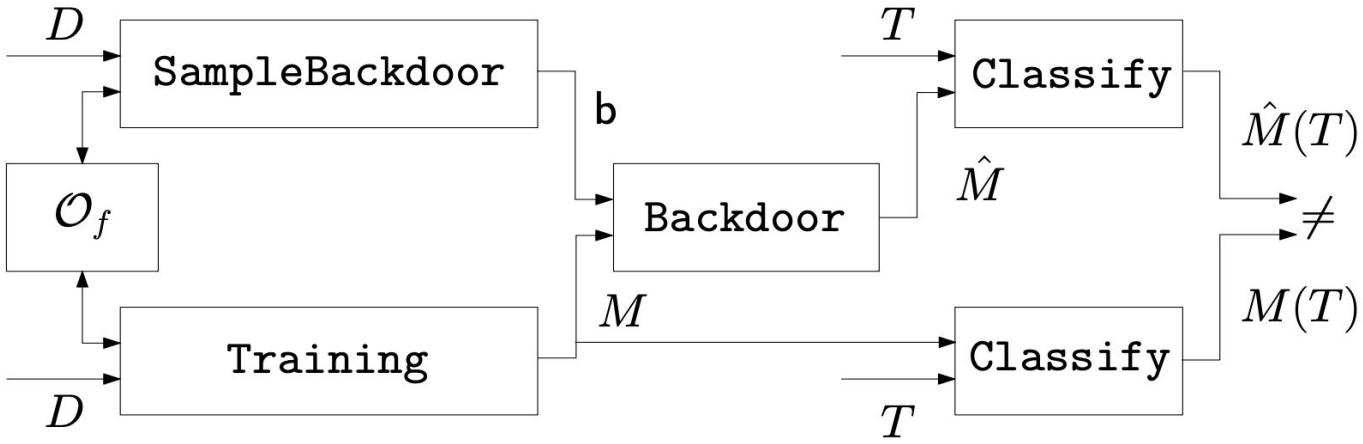
Watermarking: Defender's Goal

Classes of Defense:

- ~~Detect model stealing / extraction~~ ❌
- ~~Prevent model stealing / extraction~~ ❌
- Ownership resolution ✅

Watermarking: Defender's Method

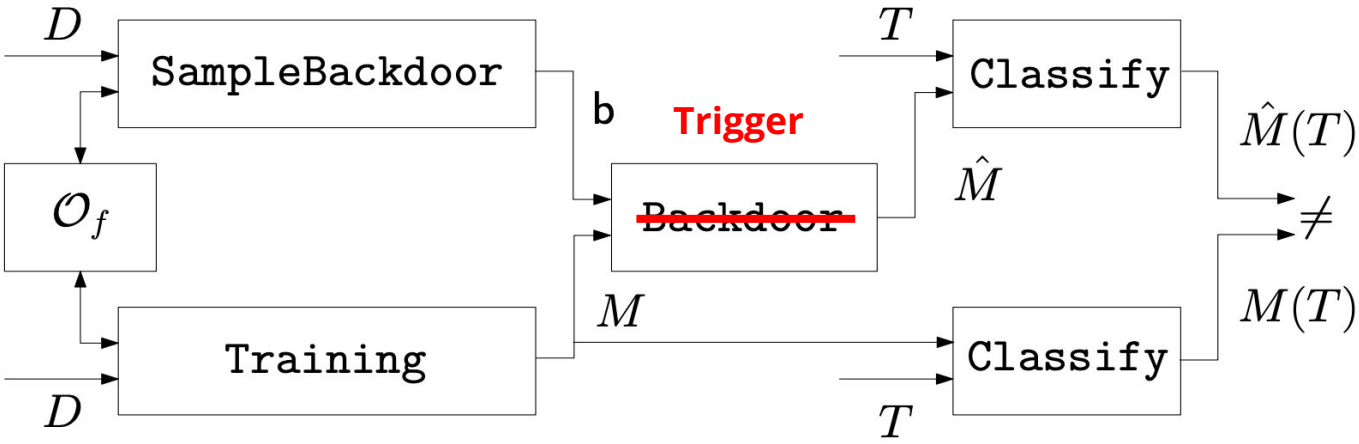
Exploit large capacity of networks to learn watermarks and claim ownership without sacrificing legitimate users' performance



Adi et al., Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring

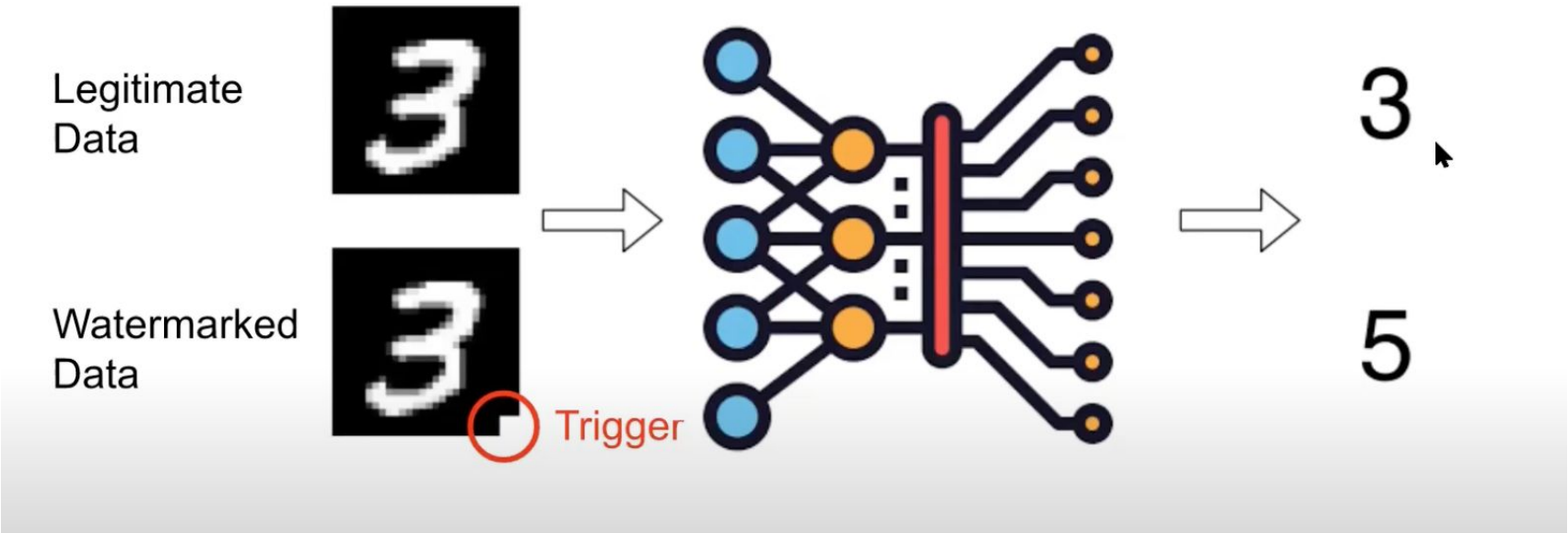
Watermarking: Defender's Method

Exploit large capacity of networks to learn watermarks and claim ownership without sacrificing legitimate users' performance



Adi et al., Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring

Watermarking: Defender's Method

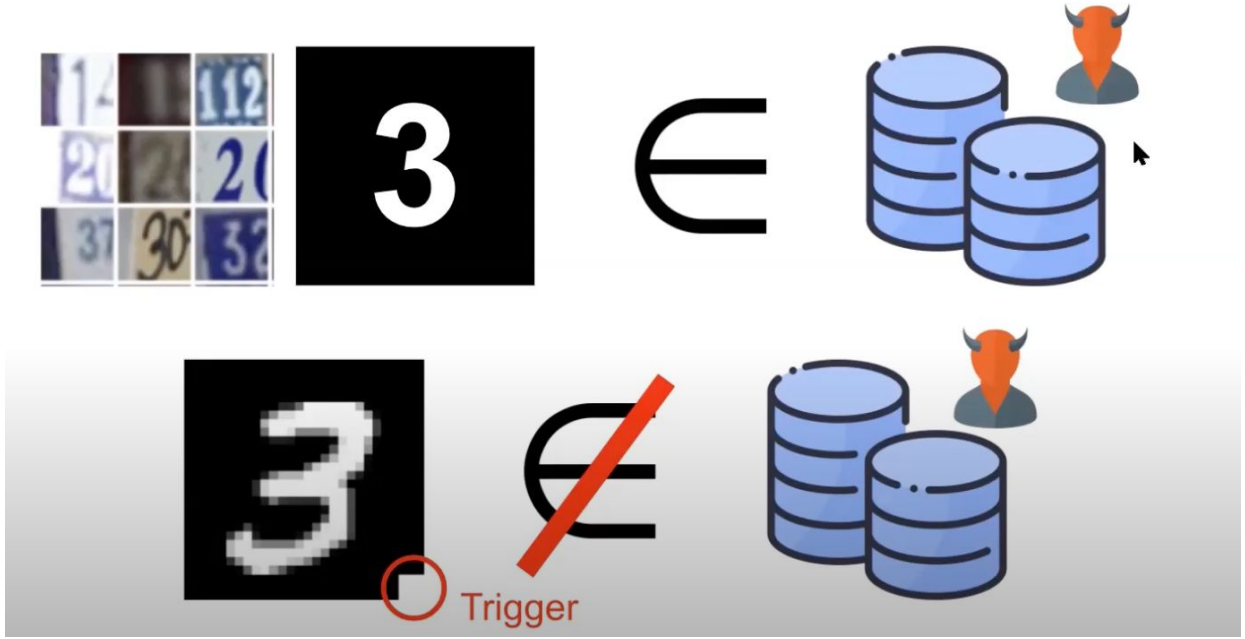


https://www.youtube.com/watch?v=WTmCSpPaPNM

Naive Watermarking: Failure Modes
Extraction-Induced Failures

- Task and watermark distributions are independent and learned independently
- Adversary's query inputs might not trigger the watermark, and so won't extract the watermarked behaviour

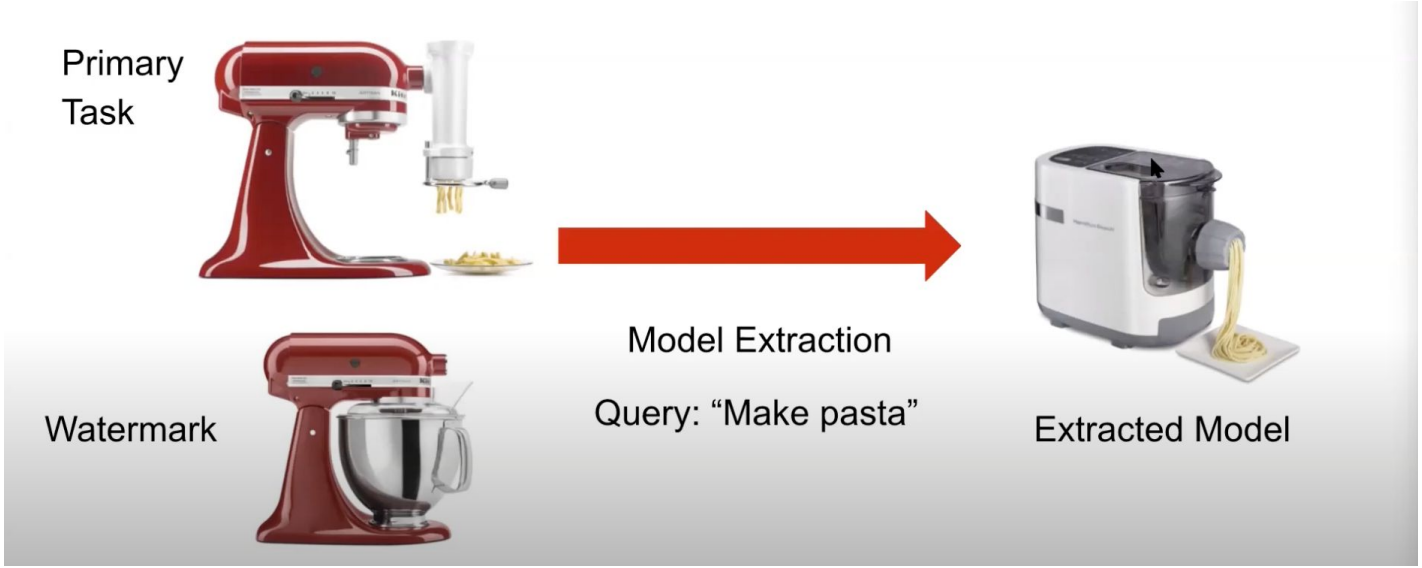
Naive Watermarking: Failure Modes
Extraction-Induced Failures



Naive Watermarking: Failure Modes
Distinct Activation Patterns

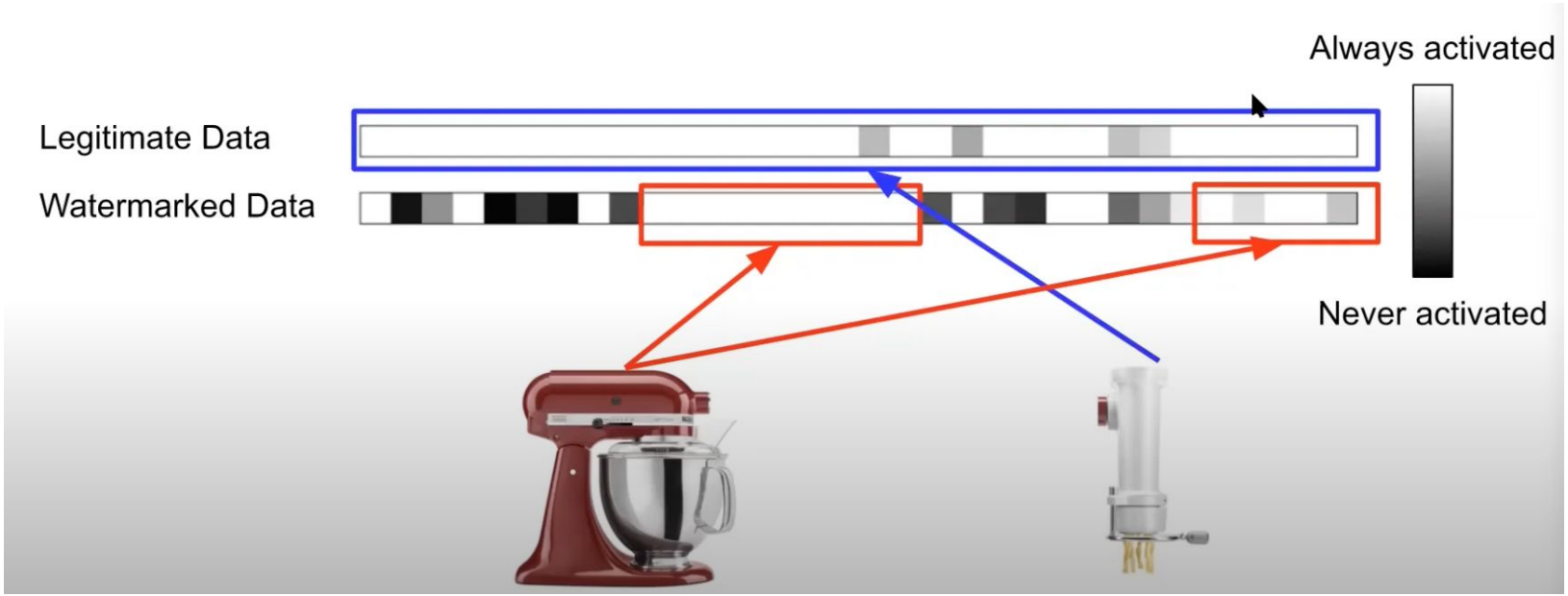
- Simple solution: model capacity roughly partitioned into two sub-models for task and watermarked data distributions
- Different neurons activated for legitimate and watermarked data lead to very different representations

Naive Watermarking: Failure Modes
Distinct Activation Patterns



https://www.youtube.com/watch?v=WTmCSpPaPNM

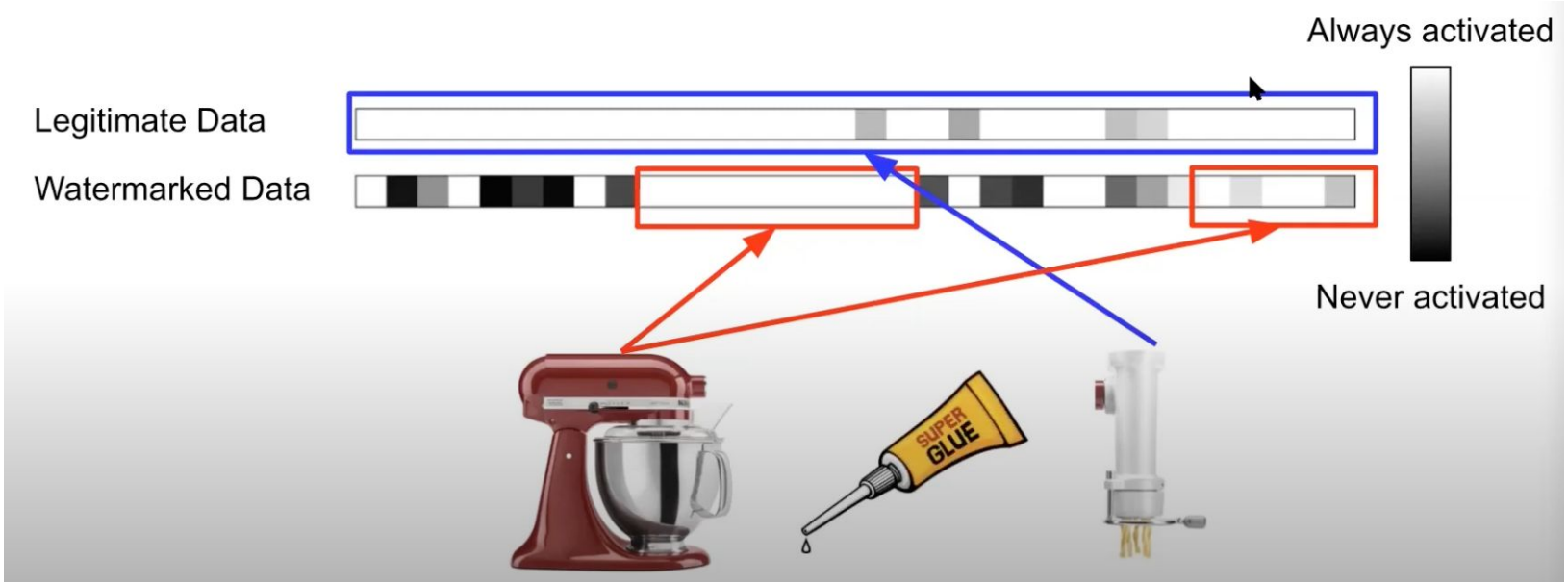
Naive Watermarking: Failure Modes
Distinct Activation Patterns



https://www.youtube.com/watch?v=WTmCSpPaPNM

What can we do?

The equivalent of superglue-ing the pasta-maker and the mixer!



https://www.youtube.com/watch?v=WTmCSpPaPNM

Entangled Watermarking Embeddings (EWE)

- Entangle representations of watermarked and task distributions such that they use the same parameters
- If adversary queried task inputs: also reproduces output on watermarks
- If adversary tries to remove watermarks (ex. via compression): necessarily harms generalization on task as well

Soft Nearest Neighbours Loss Function

- Measures entanglement between representations
- Ratio between average distance between points from same group and average distance between any two points

$$SNNL(X,Y,T) = -\frac{1}{N}\sum_{i\in 1..N} log\left(\frac{\sum_{\substack{j\in 1..N \\ j\neq i \\ y_i=y_j}} e^{-\frac{||x_i-x_j||^2}{T}}}{\sum_{\substack{k\in 1..N \\ k\neq i}} e^{-\frac{||x_i-x_k||^2}{T}}}\right) \begin{matrix} \\ (a) \\ \\ \\ (b) \end{matrix}$$

- Maximize SNNL

  = bring points from different groups closer

  = entangle manifolds of different groups

Algorithm

- Watermark data at selected trigger positions
- Modify loss function to encourage entanglement
- Train the model

Algorithm 1: Entangled Watermark Embedding

Input: $X, Y, D_w, T, c_S, c_T, r, \alpha, loss, model, trigger$
Output: A watermarked DNN model

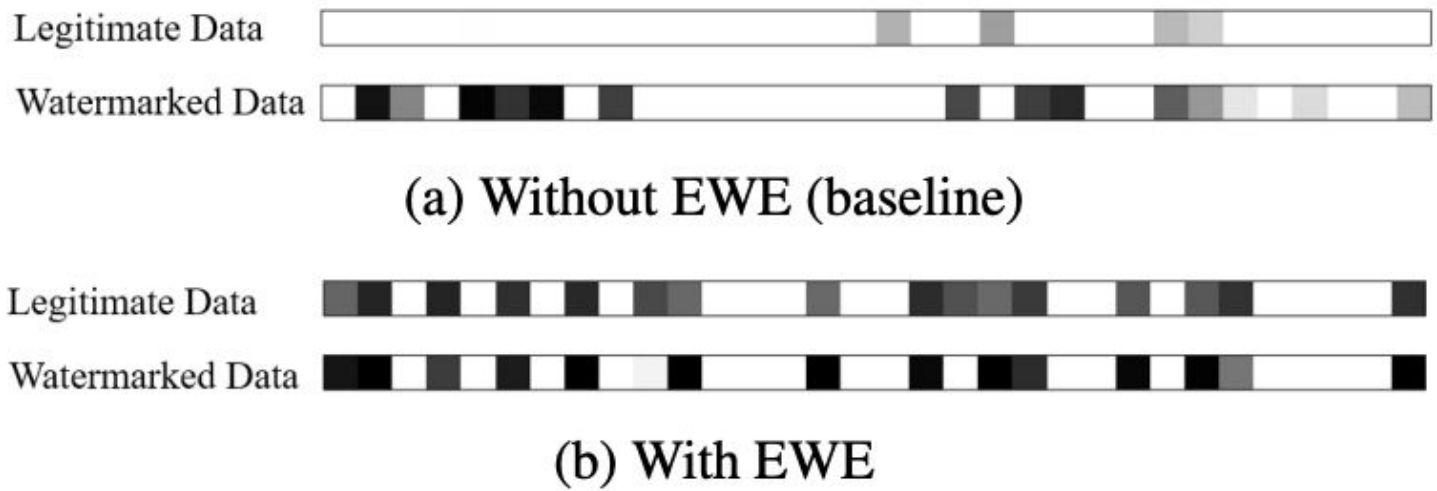
/* Compute trigger positions */
$X_w = D_w(c_S), Y' = [Y_0, Y_1]$;
map=$conv(\nabla_{X_w}(SNNL([X_w, X_{c_T}], Y', T)), trigger)$;
position = $\arg\max(map)$;
/* Generate watermarked data */
$X_w[position]$ = trigger;
$FGSM(X_w, \mathcal{L}_{CE}(X_w, Y_{c_T}))$/* optional */
$FGSM(X_w, SNNL([X_w, X_{c_T}], Y', T))$/* optional */
step = 0 /* Start training */
while *loss not converged* **do**
    step += 1;
    if *step % r == 0* **then**
        model.train($[X_w, X_{c_T}], Y_{c_T}$)/* watermark */
    else
        model.train(X, Y)/* primary task */
    /* Fine-tune the temperature */
    $T^{(i)}$ -= α * $\nabla_{T^{(i)}}$SNNL($[X_w, X_{c_T}]^{(i)}, Y', T^{(i)}$);

Validating: What does EWE actually do? Ownership Verification

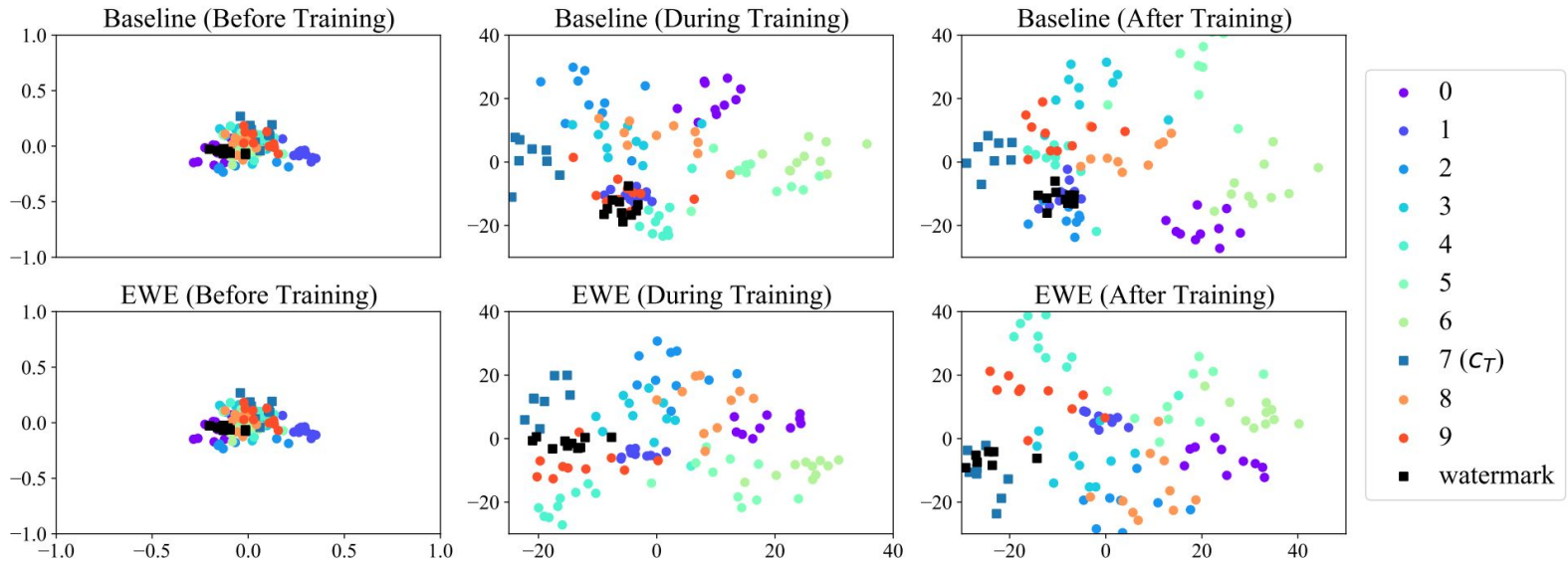
Ownership claimed with 95% confidence with very few queries *if* watermark success rate far exceeds false positive rate

(probability of a watermarked model correctly identifying watermarked data as new class >> probability of a non-watermarked model classifying watermarked data as new class)

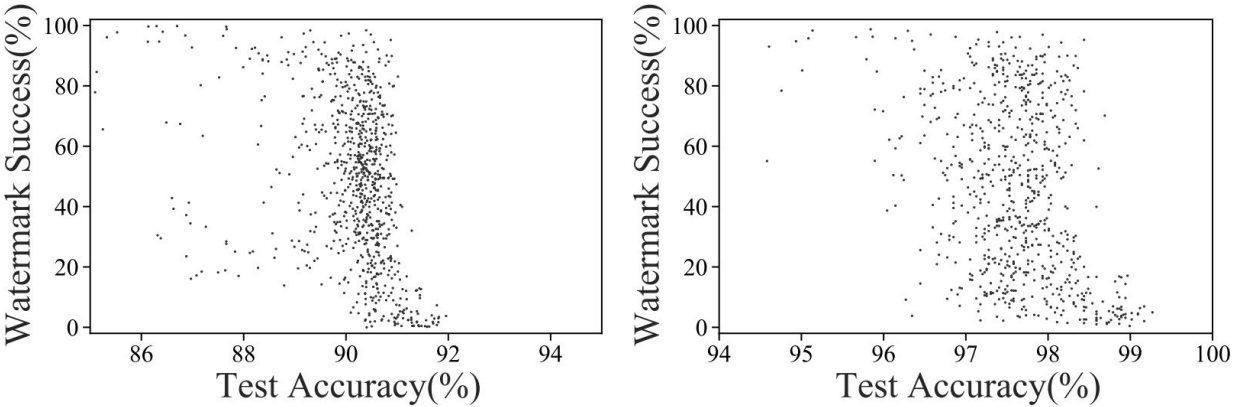
Validating: What does EWE actually do? Increased Entanglement



Validating: What does EWE actually do? Increased Entanglement



Watermark Robustness vs Utility: No Free Lunch



(a) Fashion MNIST (b) Speech Command

Evaluating on MNIST, Fashion MNIST, CIFAR-10, CIFAR-100 and Google Speech Command, EWE claims model ownership with 95% confidence with less than 100 queries to the stolen copy at a cost of below 0.81 percentage points on average in performance

Attack Watermarks = Defend against Backdoors Pruning

Idea: prune away neurons infrequently activated by legitimate data

- Watermark neurons are frequently activated by legitimate data
- Pruning still gives high watermark success rate
- When watermark success rate starts decreasing, so does task accuracy and the point of model stealing is defeated

Ineffective!

Attack Watermarks = Defend against Backdoors Fine-Pruning

Idea: continue to train/fine-tune the model after pruning, with hope to recover some lost accuracy

- Fine-tuning is done on labels from watermarked model, and so contains information about watermarks
- Again, when watermark success rate drops sufficiently, so does task accuracy

Ineffective!

Attack Watermarks = Defend against Backdoors Neural Cleanse

Idea: find trigger as the smallest perturbation to classes required for misclassification and retrain with knowledge of trigger

- Watermarks work on single source-target pairs, not classes, so forcing entanglement doesn't affect distance boundary with other classes
- Empirically, EWE triggers remain undetected

Ineffective!

Criticism and Discussion

- Vulnerability to adaptive attacks, adversary with more knowledge
- Defense executed before training and deployment; not adaptive
- Inevitable reduction in model's task accuracy (No Free Lunch)
- Scaling to deeper architectures, complex tasks, more complicated representation space; is computational overhead worthwhile?
- Does the goal of the adversary matter? Theft vs. Recon
- Doesn't make direct use of the one advantage the defender has: solely having access to "true" training data

DATASET INFERENCE: OWNERSHIP RESOLUTION IN MACHINE LEARNING

Maini et al., 2021

Your Scenario

You've spent millions on research, development, and compute in order to develop a cutting edge, proprietary model.

AND/OR

You've created a private dataset for training a model, and outside parties are not entitled to use this dataset or its byproducts

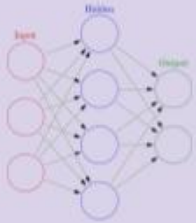
Threat Models

API Access

Model extraction using:
- Prediction Vectors
- Labels Only

Model Access

Obfuscate the use of the model by:
- Fine-tuning
- Distillation



Data Access

Use the private dataset by:
- Training a new model from scratch
- Distilling the target model (requires API access as well)

Model Extraction is an Imposing Threat

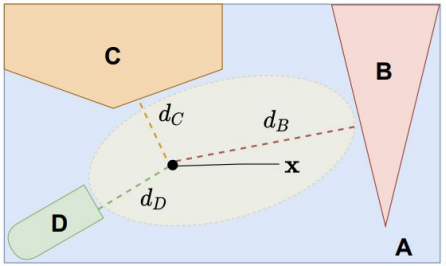
- Defending against model extraction can be costly and ineffective
  - Hard to differentiate between queries from an adversary and regular use
  - Defensive techniques like watermarking degrade model performance and require retraining
- It may be in the defender's interest to be able to **prove that they own a model** in the case that it is stolen

An Observation

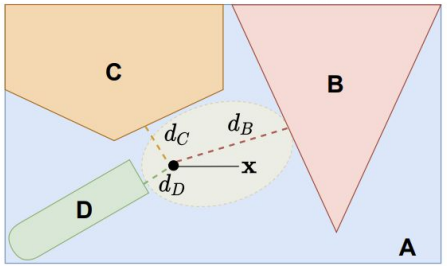
- All threat models rely on stealing **the training data or a byproduct of the training data**
  - "A successful model extraction attack will distill the victim's knowledge of its training data into the stolen copy"

- Models tend to be more confident about **samples from their training data** in comparison with random samples from the data distribution

- **Idea:** assess a (potentially) stolen model's confidence while predicting on a small set of the private data to see if the model **used the same dataset as the victim**



(a) If $\mathbf{x}$ is in training set



(b) If $\mathbf{x}$ is not in training set

Dataset Inference

Prediction Margin; a proxy measure of the model's confidence of its prediction on a given sample.

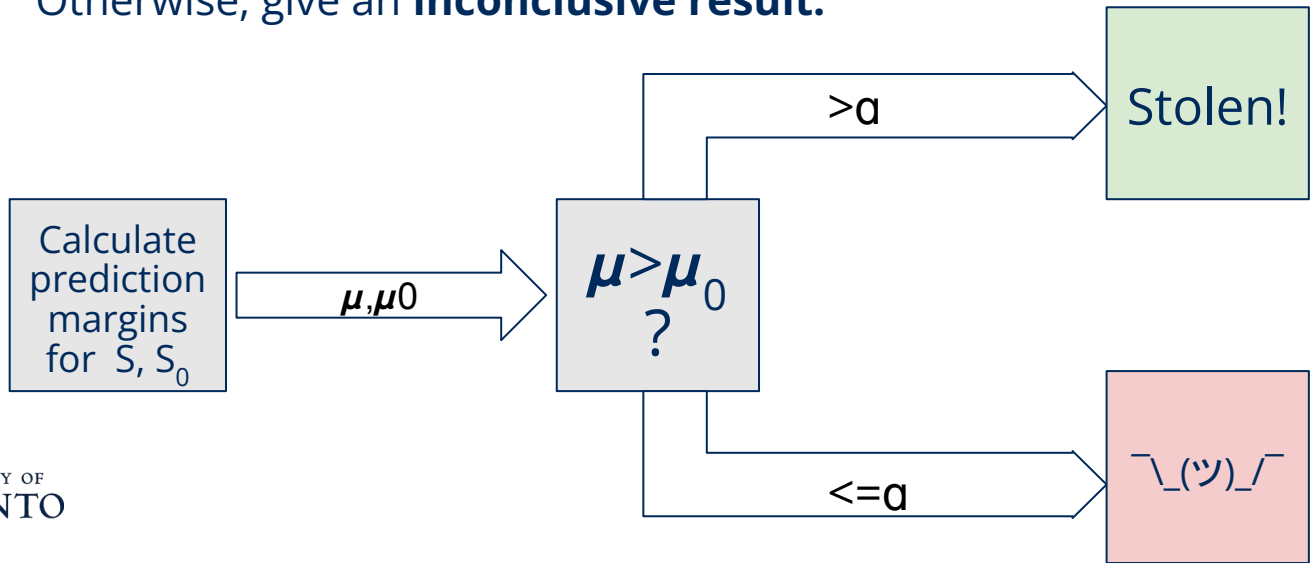
Dataset inference, in all its forms, consists of two major steps

1. Using a (potentially) stolen model, calculate the average prediction margins for:
   a. A set of M samples from the victim model's **training set**
   b. A set of M samples from outside the victim model's training set, but inside the same **data distribution**

Dataset Inference

2. Compare the two average prediction margins (difference, hypothesis testing) using a **decision function** to see if the **training set data has a higher prediction margin** and:

a. If the training set data does have a higher prediction margin **and a certain confidence threshold is met**, report that the model has been stolen
b. Otherwise, give an **inconclusive result.**



Theoretical Motivation: Linear Setting

Consider the simple data distribution *D*, consisting of input-label pairs (**x**,y) that are created using the following:

- $y \sim \{-1, 1\}$
- $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}) \in R^{K+D}$
- $\mathbf{x_1} = y * \mathbf{u} \in R^K$
- $\mathbf{x_2} \sim \mathcal{N}(0, \sigma^2 I) \in R^D$

- First K dimensions are highly correlated with the label
- Last D dimensions are Gaussian noise

Theoretical Motivation: Guarantee of Margin Difference

Linear model: $f(\mathbf{x}) = \mathbf{w}_1 \cdot \mathbf{x}_1 + \mathbf{w}_2 \cdot \mathbf{x}_2$

- Each component assigned its own weights
- Assume this model is trained on some dataset $S \sim D$, $|S| = m$.

Prediction Margin: $y \cdot f(\mathbf{x})$

- Margin of the data point from the decision boundary.
- Correct, high confidence predictions will be assigned a **high prediction margin**

Theorem 1:

$$\mathbb{E}_{(\mathbf{x},y)\sim S}\left[y \cdot f(\mathbf{x})\right] - \mathbb{E}_{(\mathbf{x},y)\sim \mathcal{D}}\left[y \cdot f(\mathbf{x})\right] = D\sigma^2$$

Theoretical Motivation: Success of Dataset Inference

Decision Function:

1. Sample another dataset $S_0 \sim D$, $|S_0| = m$. This is our "random data".
2. Calculate average prediction margins of the classifier f on S and S_0
3. If $S - S_0 > \lambda$, report that f is a **stolen model**. Otherwise, give an inconclusive result.

Theorem 3: The probability the victim correctly predicts whether or not the model is a stolen model is equal to:

$$1 - \Phi\left(-\frac{\sqrt{D}}{2\sqrt{2}}\right)$$

Where ϕ is the CDF of a Gaussian, D is the noise feature dimension.

Dataset Inference in *Real Life*

Unfortunately for us:

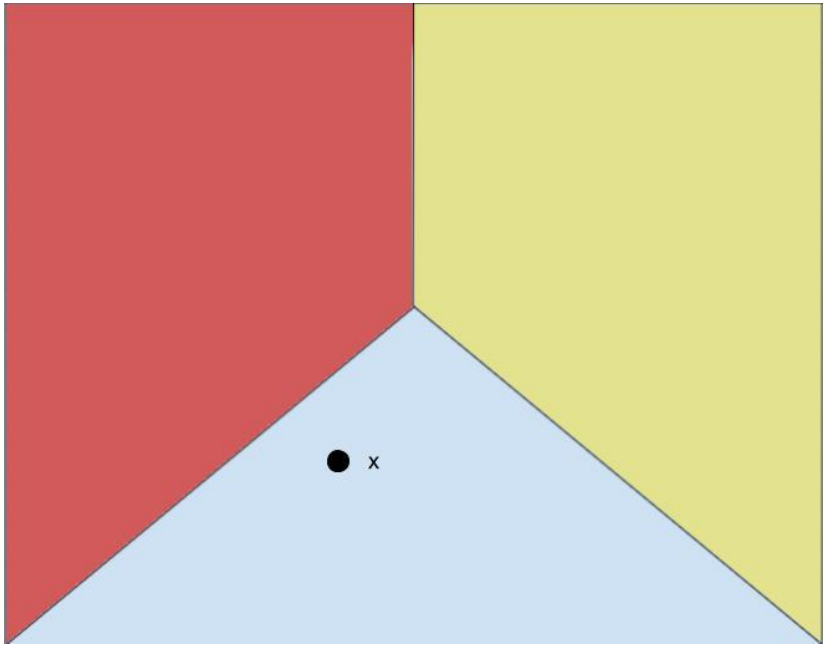
- Data is rarely distributed in such a simple way
- This in turn requires models that are much more complicated than simple linear models

If we're going to look at contemporary datasets and model architectures, we'll have to choose a new:

- Prediction Margin definition
- Decision Function

Dataset Inference and DNNs: Generating Embeddings

For the samples in S and S_0 (training set and “random” data, respectively), we want to generate embeddings that **encode information about the model’s decision boundary.**



Dataset Inference and DNNs: Generating Embeddings

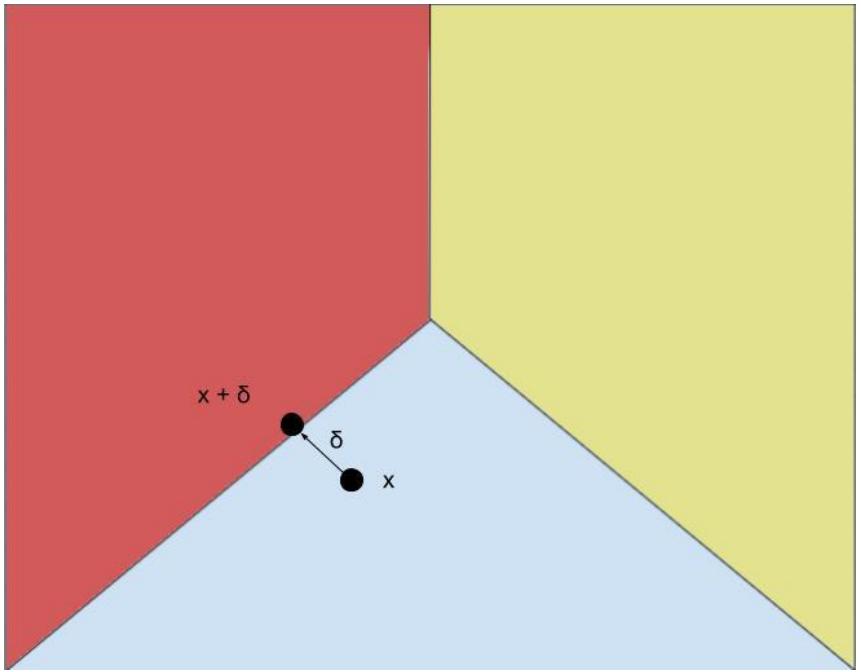
White-Box Setting: Assumes that we have complete access to the model and can **compute gradients**.

MinGD: For a data point (**x**,y) and target class *t*, solve for:

$$min_{\delta} \Delta(\mathbf{x}, \mathbf{x}+\delta) \text{ s.t. } f(\mathbf{x}+\delta) = t$$

Embedding is the concatenation of the distance between the original point and the perturbed point for several target classes,L_p norms.

This gives a sort of "**worst case**" prediction margin



Dataset Inference and DNNs: Generating Embeddings

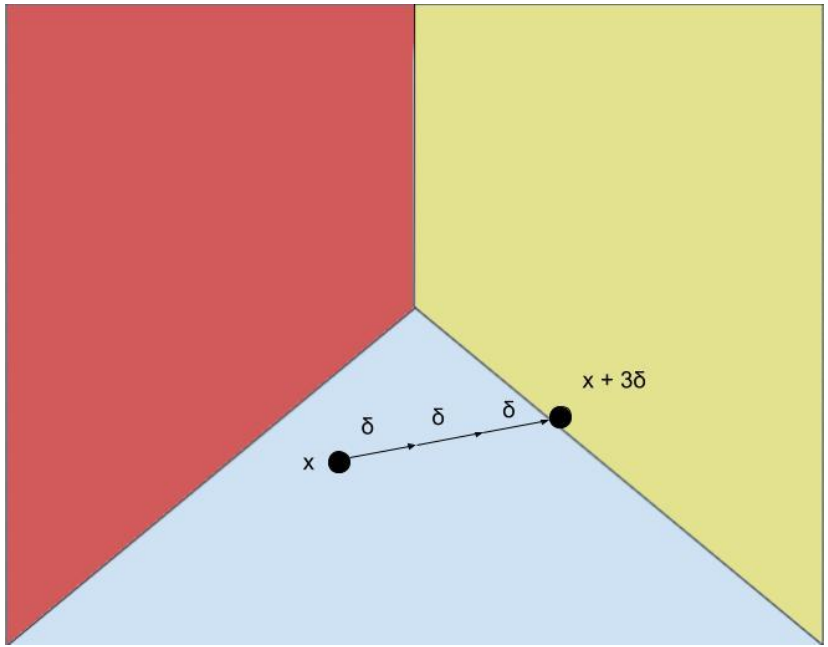
Black-Box Setting: Assumes that we are only able to query the model for labels.

Blind Walk : For a data point (**x**,y), choose a random direction **δ** and take *k* steps until

$f(\mathbf{x} + k\boldsymbol{\delta}) = t$, t!=y

The embedding is composed of the distance values from a number of these blind walks with different random directions.

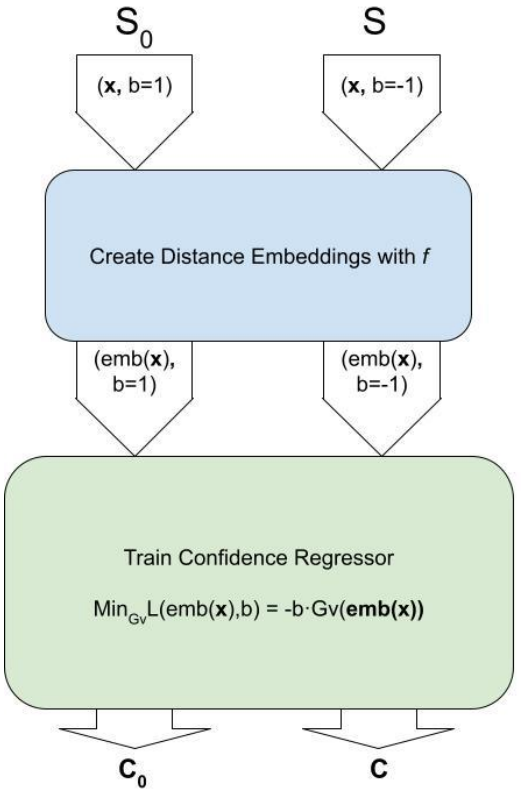
This gives an "**average case**" prediction margin



Dataset Inference and DNNs: Confidence Regressor

- Want a bulk measure of whether a sample's prediction margin is more similar to samples from S or S_0

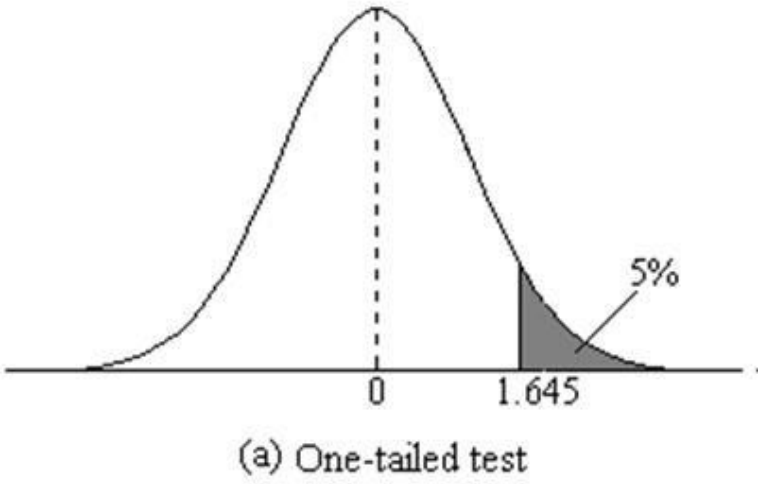
- Create a **confidence regressor** Gv
  - Gv minimizes output when **x** is from S, and maximizes it otherwise
  -
- **C** and $\mathbf{C_0}$ are vectors of regressor predictions for each sample in each dataset



Dataset Inference and DNNs: Hypothesis Testing

- Using $\mathbf{C_0}$ and **C**, we'd like to know whether the prediction margins for samples from *S* and S_0 are **significantly different**
- Use a two-sample t-test, and calculate the p-value for the null hypothesis, using $\boldsymbol{\mu}_0$ and $\boldsymbol{\mu}$ as the means of $\mathbf{C_0}$ and **C**
  - $\mathbf{H_0}$: $\boldsymbol{\mu}_0 < \boldsymbol{\mu}$
- Rejecting the null hypothesis => Model is **stolen!**

$$t = \frac{\mu_0 - \mu}{\sqrt{\frac{s^2}{m} + \frac{s_0^2}{m}}}$$

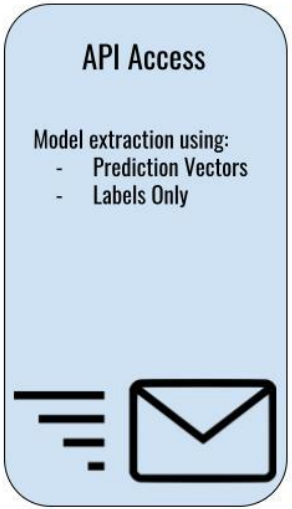


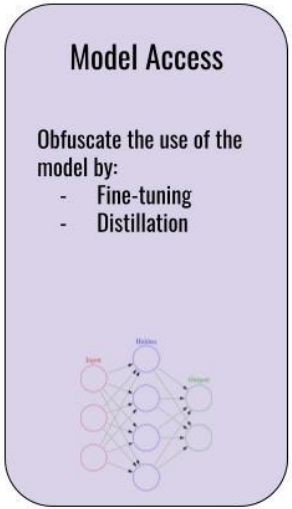
(a) One-tailed test

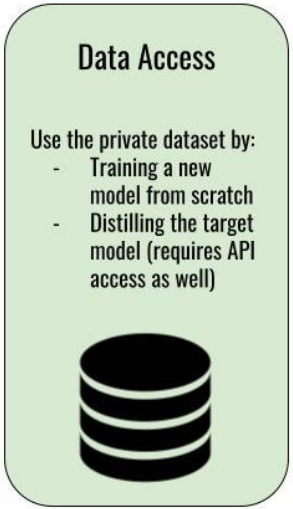
Source: https://stats.stackexchange.com/questions/161721/why-does-the-p-value-double-when-using-two-tailed-test-compared-to-one-tailed-on

Experiments & Threat Model Refresher

- Focus on the CIFAR10 and CIFAR100 datasets
- Steal models using the 6 main threat models from before:







Experiment Base Cases

Include two additional models representing important circumstances:

- **V:** The original model. This is analogous to the adversary directly deploying the stolen model without any obfuscation
- **I**: A model trained on a dataset completely independent from that of the victim. This should be the case for every model that isn't stolen.

Dataset inference should predict that **V** is stolen with very high certainty, and report inconclusive results for **I**

Results

	Model Stealing Attack	CIFAR10				CIFAR100			
		MinGD		Blind Walk		MinGD		Blind Walk	
		$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value
$\mathcal{V}$	Source	0.838	10^{-4}	1.823	10^{-42}	1.219	10^{-16}	1.967	10^{-44}
$\mathcal{A}_D$	Distillation	0.586	10^{-4}	0.778	10^{-5}	0.362	10^{-2}	1.098	10^{-5}
	Diff. Architecture	0.645	10^{-4}	1.400	10^{-10}	1.016	10^{-6}	**1.471**	10^{-14}
$\mathcal{A}_M$	Zero-Shot Learning	**0.371**	10^{-2}	**0.406**	10^{-2}	0.466	10^{-2}	0.405	10^{-2}
	Fine-tuning	**0.832**	10^{-5}	**1.839**	10^{-27}	**1.047**	10^{-7}	1.423	10^{-10}
$\mathcal{A}_Q$	Label-query	0.475	10^{-3}	1.006	10^{-4}	**0.270**	10^{-2}	**0.107**	10^{-1}
	Logit-query	0.563	10^{-3}	1.048	10^{-4}	0.385	10^{-2}	0.184	10^{-1}
$\mathcal{I}$	Independent	0.103	1	-0.397	0.675	-0.242	0.545	-1.793	1

Table 1: Ownership Tester's effect size (higher is better) and p-value (lower is better) using $m = 10$ samples on multiple threat models (see § 6.1). The highest and lowest effect sizes among the model stealing attacks ($\mathcal{A}_D, \mathcal{A}_M, \mathcal{A}_Q$) are marked in **red** and **blue** respectively.

Results: Base Cases

	Model Stealing Attack	CIFAR10				CIFAR100			
		MinGD		Blind Walk		MinGD		Blind Walk	
		$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value
$\mathcal{V}$	Source	0.838	10^{-4}	1.823	10^{-42}	1.219	10^{-16}	1.967	10^{-44}
$\mathcal{A}_D$	Distillation	0.586	10^{-4}	0.778	10^{-5}	0.362	10^{-2}	1.098	10^{-5}
	Diff. Architecture	0.645	10^{-4}	1.400	10^{-10}	1.016	10^{-6}	**1.471**	10^{-14}
$\mathcal{A}_M$	Zero-Shot Learning	**0.371**	10^{-2}	**0.406**	10^{-2}	0.466	10^{-2}	0.405	10^{-2}
	Fine-tuning	**0.832**	10^{-5}	**1.839**	10^{-27}	**1.047**	10^{-7}	1.423	10^{-10}
$\mathcal{A}_Q$	Label-query	0.475	10^{-3}	1.006	10^{-4}	**0.270**	10^{-2}	**0.107**	10^{-1}
	Logit-query	0.563	10^{-3}	1.048	10^{-4}	0.385	10^{-2}	0.184	10^{-1}
$\mathcal{I}$	Independent	0.103	1	-0.397	0.675	-0.242	0.545	-1.793	1

Table 1: Ownership Tester's effect size (higher is better) and p-value (lower is better) using $m = 10$ samples on multiple threat models (see § 6.1). The highest and lowest effect sizes among the model stealing attacks ($\mathcal{A}_D, \mathcal{A}_M, \mathcal{A}_Q$) are marked in **red** and **blue** respectively.

Results: Worst and Best Threat Model

	Model Stealing Attack	CIFAR10				CIFAR100			
		MinGD		Blind Walk		MinGD		Blind Walk	
		$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value
$\mathcal{V}$	Source	0.838	10^{-4}	1.823	10^{-42}	1.219	10^{-16}	1.967	10^{-44}
$\mathcal{A}_D$	Distillation	0.586	10^{-4}	0.778	10^{-5}	0.362	10^{-2}	1.098	10^{-5}
	Diff. Architecture	0.645	10^{-4}	1.400	10^{-10}	1.016	10^{-6}	**1.471**	10^{-14}
$\mathcal{A}_M$	Zero-Shot Learning	**0.371**	10^{-2}	**0.406**	10^{-2}	0.466	10^{-2}	0.405	10^{-2}
	Fine-tuning	**0.832**	10^{-5}	**1.839**	10^{-27}	**1.047**	10^{-7}	1.423	10^{-10}
$\mathcal{A}_Q$	Label-query	0.475	10^{-3}	1.006	10^{-4}	**0.270**	10^{-2}	**0.107**	10^{-1}
	Logit-query	0.563	10^{-3}	1.048	10^{-4}	0.385	10^{-2}	0.184	10^{-1}
$\mathcal{I}$	Independent	0.103	1	-0.397	0.675	-0.242	0.545	-1.793	1

Table 1: Ownership Tester's effect size (higher is better) and p-value (lower is better) using $m = 10$ samples on multiple threat models (see § 6.1). The highest and lowest effect sizes among the model stealing attacks ($\mathcal{A}_D, \mathcal{A}_M, \mathcal{A}_Q$) are marked in **red** and **blue** respectively.

Results: Surprising Effectiveness of Blind Walk

	Model Stealing Attack	CIFAR10				CIFAR100			
		MinGD		Blind Walk		MinGD		Blind Walk	
		$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value	$\Delta\mu$	p-value
$\mathcal{V}$	Source	0.838	10^{-4}	1.823	10^{-42}	1.219	10^{-16}	1.967	10^{-44}
$\mathcal{A}_D$	Distillation	0.586	10^{-4}	0.778	10^{-5}	0.362	10^{-2}	1.098	10^{-5}
	Diff. Architecture	0.645	10^{-4}	1.400	10^{-10}	1.016	10^{-6}	**1.471**	10^{-14}
$\mathcal{A}_M$	Zero-Shot Learning	**0.371**	10^{-2}	**0.406**	10^{-2}	0.466	10^{-2}	0.405	10^{-2}
	Fine-tuning	**0.832**	10^{-5}	**1.839**	10^{-27}	**1.047**	10^{-7}	1.423	10^{-10}
$\mathcal{A}_Q$	Label-query	0.475	10^{-3}	1.006	10^{-4}	**0.270**	10^{-2}	**0.107**	10^{-1}
	Logit-query	0.563	10^{-3}	1.048	10^{-4}	0.385	10^{-2}	0.184	10^{-1}
$\mathcal{I}$	Independent	0.103	1	-0.397	0.675	-0.242	0.545	-1.793	1

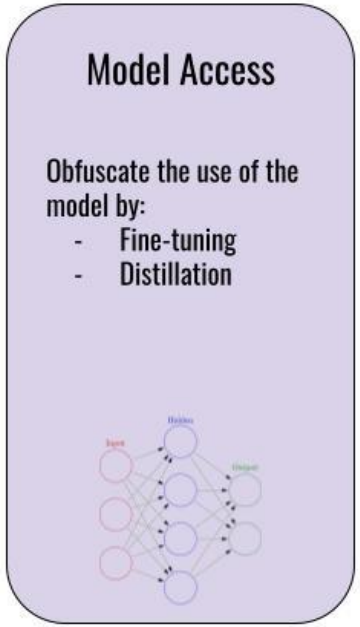
Table 1: Ownership Tester's effect size (higher is better) and p-value (lower is better) using $m = 10$ samples on multiple threat models (see § 6.1). The highest and lowest effect sizes among the model stealing attacks ($\mathcal{A}_D, \mathcal{A}_M, \mathcal{A}_Q$) are marked in **red** and **blue** respectively.

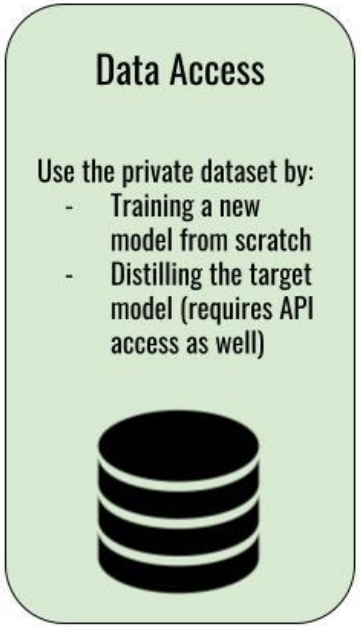
Limitations

Unclear how this helps in the case where a model used a **public dataset**

Claim that the method is effective against fine-tuning techniques, but do not consider that the model could be trained on a new-task, causing **catastrophic forgetting of the private training set**[1]

The adaptive approach they deploy seems to be well defended against, but perhaps a more effective adaptive approach could have been created

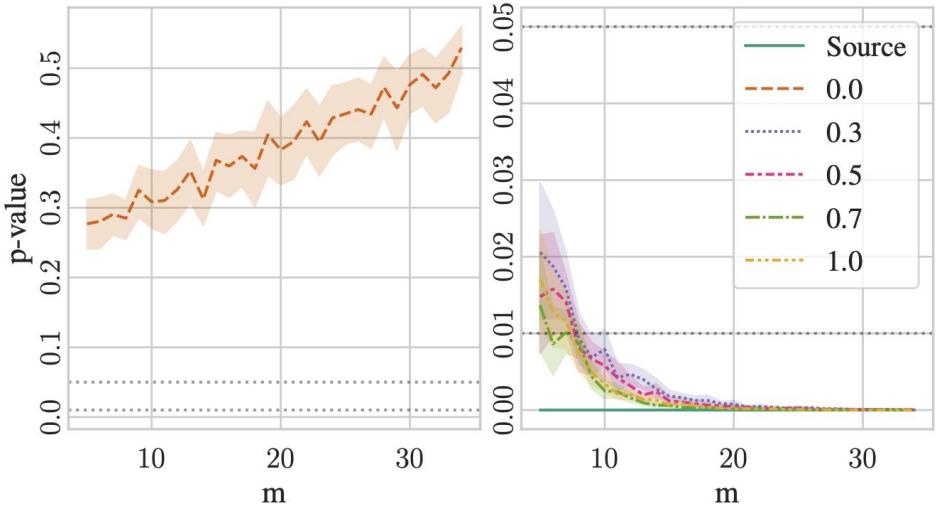




1. Kirkpatrick et al., 2017: Overcoming catastrophic forgetting in neural networks

Limitations: Dataset Overlap

- Results of this approach seem like they could depend on the public dataset that is used to compare confidence regression scores with the private dataset



Strengths

- Defence technique that requires **no fine-tuning or overfitting**
- Resource efficient
- Successfully evading dataset inference requires a large amount of resources, and results in a large amount of **performance degradation**
- Very effective against several different threat models coming from **three substantially different information levels**