2019 IEEE Security and Privacy Workshops (SPW)

# Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations

Taesung Lee[1] Benjamin Edwards[2,†] Ian Molloy[3] Dong Su[4,†]
IBM Research AI[1,3] Independent[2,4]
[1]taesung.lee@ibm.com [2]bjedwards@gmail.com [3]molloyim@us.ibm.com [4]sudong.tom@gmail.com

***Abstract*—Machine learning architectures are readily available, but obtaining the high quality labeled data for training is costly. Pre-trained models available as cloud services can be used to generate this costly labeled data, and would allow an attacker to replicate trained models, effectively stealing them. Limiting the information provided by cloud based models by omitting class probabilities has been proposed as a means of protection but significantly impacts the utility of the models. In this work, we illustrate how cloud based models can still provide useful class probability information for users, while significantly limiting the ability of an adversary to steal the model. Our defense perturbs the model's final activation layer, slightly altering the output probabilities. This forces the adversary to discard the class probabilities, requiring significantly more queries before they can train a model with comparable performance. We evaluate our defense under diverse scenarios and defense aware attacks. Our evaluation shows our defense can degrade the accuracy of the stolen model at least 20%, or increase the number of queries required by an adversary 64 fold, all with a negligible decrease in the protected model accuracy.**

## I. Introduction

The success of neural networks has resulted in many web services based on them, including services providing APIs to label input samples for small sums of money. Many state-of-the-art neural network architectures are readily available in the literature or online, and unlabeled data (*e.g.*, images and corpus) are also often abundant on the web. But labeling data to train a machine learning model is expensive, difficult, and error-prone even for simple tasks [1]. It is even more difficult for domains requiring expert knowledge (*e.g.*, coreference resolution in the medical domain). However, once an adversary acquires enough labels using the web services, the attacker can replicate the neural network and no longer needs to pay for the service [2]. For example, current image classification services charge around \$1–\$10 per 1,000 queries, depending on the sophistication or customization of the model [3]–[5]. Moreover, replicating neural networks allows for the execution of other attacks, such as the creation of adversarial examples [6].

In this paper, we seek to frustrate such neural network model stealing attacks by perturbing the output while maintaining the utility. That is, we consider a scenario that an attacker uses the probability values returned by the base model on the cloud service to boost the model stealing process. The cloud service often provides probability values to show confidence. When stealing models, [2] claim that using probabilities instead of labels alone reduces the number of required samples by 50–100×. We also confirm that using probabilities can improve the convergence, and also increases the converged model accuracy in Sec IV-A and IV-D. Our goal is defeating attacks using probabilities, and leaving only a suboptimal attack of discarding the probabilities. We test a variety of possible noise forms and find a 'reverse sigmoid' to be the most effective defense.

To evaluate the performance of our defense, we consider two types of attacks. First, we consider an attacker unaware of the defense that can replicate an unprotected model quickly. [1] Second, we consider variety of defense-aware attacks, including the attacker implementing of the same defense layer, inverting the added noise, using a different loss function, and using only labels. Our evaluation considers diverse datasets, attack parameters, neural networks architectures and domains (images and text). We show that for a defense unaware attack our approach can degrade stolen model accuracy by 20% or more while keeping the protected model accuracy almost intact. We show that a variety of defense aware attacks fail to improve the stolen models accuracy, and require as much as a 64× queries to obtain accuracy similar to the original model.

## II. Related work

The problem of inferring secret machine learning model parameters by observing the output classification to a given set of inputs has been recently studied. [2] demonstrated an attack that, using high-precision confidence values and class labels obtained from a machine learning cloud service, can steal several types of models including decision trees, logistic regression, support vector machines and simple neural networks. For simple parametric models such as logistic regression, they solve a linear system from the obtained probabilities. For decision trees, they develop a path-finding algorithm to exploit the confidence value as pseudo-identifiers for paths in the tree to discover the tree structure. For neural networks, they leverage a method we denote by Sample that uses a set of samples and query

† This work was done while the authors were at IBM Research.

[1]We observe that a defense-aware attack performs worse on an unprotected model.

DOI 10.1109/SPW.2019.00020

a randomly drawn batch, and train the network with the output from the base model, similarly to [7]. [8] also consider stealing a machine learning parameter, but is limited to the regularization parameter used during training not any of the model parameters themsleves.

[6] proposed a method for stealing a neural network model to generate adversarial examples. They assume the attacker has a limited amount of training data, and propose to use a Jacobian-based heuristic in order to find examples defining the decision boundary of the target model, which we denote by Jacobian Augmentation in our experiments. We extend the analyses of Sample and Jacobian Augmentation with six datasets, five neural network architectures as well as other attack methods to leverage in our defense evaluation. To our knowledge, this is the first study mitigating such model stealing attacks.

## III. Methods

In this section, we propose adding a new activation layer that can be applied to most neural network classifiers to protect against model stealing attacks. That is, instead of attempting to detect an attack, we leverage this layer adding a small controllable perturbation *maximizing the loss* of the stolen model while trying to preserve the accuracy of the original model. The attacker essentially approximates the loss hypersurface to find parameters for the stolen model with the minimum loss value. To protect a model, the activation layer manipulates the estimated loss surface to keep the attacker away from the optimal parameters.

We consider neural networks that extract features from the input data and aggregate them throughout the layers to generate class probabilities of the input. Typically, the last layer is an activation function producing probability values ranging from 0 to 1 and sum to 1, given *logits*, the unbounded vector from the previous layer. In most cases, the softmax function without parameters is used. That is, most neural network classifiers of $K$ classes can be represented as $y = f(x) = \sigma(g(x))$, mapping input $x$ to output $y$, where $g(\cdot)$ is a function to a $K$-dimensional real vector, and $\sigma(\cdot)$ is the final activation function (*e.g.*, softmax) mapping a vector to $K$ probability values summing to 1.

An attacker with samples $X = \{x^i\}$ can query a neural network on the remote server (*base model*) to obtain the corresponding pseudo-labels $Y = \{y^i\}$, and train their own neural network (*stolen model*). The completely replicated network should have the minimum loss $L$ with respect to $Y$, which is usually defined using the cross entropy loss function. That is,

$$L_f(X, Y) = -\sum_i \sum_j y^i_j \log f(x^i)_j \tag{1}$$

where $y^i_j$ and $f(x^i)_j$ represent $j$-th dimension of $y^i$ and $f(x^i)$, respectively.

In this setting, we propose altering the server response $Y = \{y^i\}$ to $\hat{Y} = \{\hat{y}^i\}$ which results in a high loss value $L$ for a model $f_{\text{OPT}}$ with the optimal parameters with $Y$. The high loss value would cause the attacker's optimizer to ill-train a network using $\hat{Y}$ and deviate from the optimal parameters. That is, we want $L_{f_{\text{OPT}}}(X, \hat{Y}) > L_f(X, \hat{Y})$ for some model $f$ with a low accuracy.

We achieve this by perturbing the softmax activation function. In particular, the perturbed probability vector $\hat{y}^i$ should have the following properties. First, the sum across the dimensions must be 1. Second, we should be able to control the magnitude of the perturbation. Last, the accuracy should be preserved, *i.e.*, $\hat{y}^i_k \geq \hat{y}^i_j$ for $j = 1, \ldots, K$ and $k$ such that $y^i_k \geq y^i_j$. We consider additive perturbation with normalization: $\hat{y}^i_j = \alpha^i(y^i_j - r(y^i_j))$ where $\alpha^i$ is a sum-to-1 normalizer for $y^i$, and $r(y^i_j)$ is the perturbation function we seek with the following parameterization:

$$r(y^i_j) = \beta(s(z^i_j) - 1/2) \tag{2}$$

where $s(\cdot)$ is a sigmoid function, $z^i_j$ is a perturbation, and $\beta$ is a positive magnitude parameter; with a constraint $\hat{y}^i_k \geq \hat{y}^i_j$ for $j = 1, \ldots, K$ preserving the accuracy. Using a derivative test, we can find $L(x, \hat{y})$ has critical points when $z^i_j \to \pm \inf$. In particular, $L(x, \hat{y})$ is maximized when $z^i_k \to \inf$ for $k$ such that $y^i_k \geq y^i_j$ for $j = 1, \ldots\ldots, K$, and $z^i_j \to -\inf$ for $j$ with low $y^i_j$.

Instead of directly setting $z^i_j$ to maximize the loss, we use a heuristic approximation to $+\inf$ when $y^i_k$ is the largest and $-\inf$ otherwise, not to completely lose the probability values: $z^i_j = \gamma s^{-1}(y^i_j)$ where $\gamma$ is a positive dataset and model specific convergence parameter, and $s^{-1}(y^i_j)$ is the pseudo-logit of $y^i_j$ that amplifies the behavior of $y^i_j$ and makes the perturbation comparable to the original probability. With this approximation, we obtain *Reverse Sigmoid* perturbation $r(y^i_j)$:

$$r(y^i_j) \approx \beta(s(\gamma s^{-1}(y^i_j)) - 1/2) \tag{3}$$

This function has a shape of flipped sigmoid function as shown in Fig 1, and the final perturbed probability value is computed as follows:

$$\hat{y}^i_j = \alpha^i \left( y^i_j - \beta(s(\gamma s^{-1}(y^i_j)) - 1/2) \right) \tag{4}$$

This function is non-invertible as can be seen in Fig 1.

The main advantage of using Reverse Sigmoid $s(\gamma s^{-1}(y^i_j))$ in $r(y^i_j)$ is two-fold. First, the original probability values are largely preserved in contrast to always returning the same values for top-1 and bottom-1 classes. Also, the top-1 class changes only when the original probabilities of the top-1 and the top-2 are very close to 0.5. Second, this function form adds ambiguity that prevents a simple inversion. As we can see in Fig 1, the final deceptive probability curve has two $y^i_j$ values that have the same $\hat{y}^i_j$ in the range $[0, \frac{\alpha^i \beta}{2}]$ and $[\alpha^i(1 - \frac{\beta}{2}), 1]$, except where the first derivative is zero, making the exact inverse function impossible and approximate inversion difficult. In fact, an adversary attempting to invert the output would have to choose one of two possible pre-activation for each output class, resulting in $2^K$ possible pre-activation vectors per sample. In Sec IV-D we explore other defense aware attacks including using the Reverse Sigmoid, using mean squared errors as the loss function, or training a neural network to attempt to invert the defensive activation.

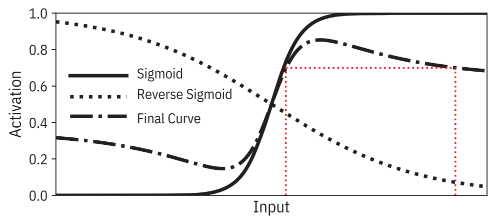

Fig. 1: Example Reverse Sigmoid activation function.

## IV. Experiments

In this section, we evaluate the proposed defense method. We consider two types of attacks. We evaluate previously proposed model stealing attacks (Sec II) against an undefended model, examining query generation, parameters, and different model architectures Sec IV-A. Then, we evaluate our defensive activations on five image datasets, one text dataset, and one multivariate dataset using five different measures in Sec IV-B. We discuss the relationship between the base models and the stolen models in Sec IV-C. We further evaluate the defense against possible attacks when the attacker is aware of the defense in Sec IV-D, and show that these attacks cannot achieve better performance than using only labels, which is a lower bound of the best attack for the accuracy-preserving defense. Basic data augmentation (shift/flip) is applied in all training [9]. MobileNet [10] and Xception [11] are optimized with Adam optimizer [12], AllConv is optimized with the vanilla stochastic gradient descent as in its original paper [13], and RMSProp [14] is used otherwise to achieve the best performance for each individual model.

To measure the performance of attack and defense, we mainly use *agreement*, top-1 model accuracy of the stolen model treating the base model as ground truth. Agreement does not use labels of the test dataset, and focus on the intrinsic model replicability.

We use seven datasets including IMDB sentiment dataset [15], MNIST [16], FASHION-MNIST [17], CIFAR-10, and CIFAR-100 [18], and STL-10 [19], and IRIS [20]. These classification datadats have various characteristics including input domains (image vs. text), # classes (*e.g.*, 10 vs. 100), and the input dimensions. Also, they cover different degrees of difficulty, and a model for an easy dataset is easier to attack and harder to defend as shown in Sec IV-B. Because simpler models built on simple datasets are easier to steal, if we can defend them, it's likely we can defend larger models built on more sophisticated datasets as well. For the image datasets, we randomly hold out 33% of the original training data, uniformly from each class, to assign a portion of them to the attacker after removing the labels and the probability values. Note that this partitioning will allow only 67% of the original training data for the base model on the cloud, and result in a small drop in model accuracy for each of the base models compared to the state-of-the-art. For the text dataset, we instead split the *test* data in the same way due to the small training data.

### A. Threat Model and Attacks

For the evaluation, we consider the following adversarial model. The adversary knows the architecture of the model being attacked, has a given number of unlabeled input samples (# samples), and can send a fixed number of (adaptive) queries to the base model to obtain labels and probability values. We do not assume the adversary is computationally bounded for training.

We compare the following attack strategies to generate queries and use the response as the training data.

- Sample: The attacker has a certain number of data samples in hand, and queries them to the base model, using the resulting probabilities for training.
- ArgmaxSample: Same as Sample, but the attacker does not use probabilities and uses the top-1 class label only.
- Random: The attacker does not rely on any existing data samples, and instead generates uniform random queries. The attacker knows the ranges and the dimensions of the input values.
- Jacobian Augmentation: The attacker has a certain number of data samples, and generates more samples using the Jacobian method [6]. The stolen model is trained using the response from the base model for both the data samples, and the generated samples. The trained model is used to generate more samples. We set $\lambda = 0.1$ and substitute training epochs $\rho = 40$.

The attacks also use randomized image augmentation that shifts, and/or flips images during the training[2]. As a neural network is trained for multiple epochs, the same training sample is used multiple times. Adding slight change to the image in every epoch provides much better generalization power, and results in better test agreement [9].

We compare these attacks with various parameters, identify the attack most successful at replicating the base model, and evaluate our defenses against the strongest attack. We use a simple convolutional network, denoted by Simple, for both the base and the stolen models: 64 $3 \times 3$ conv, 64 $3 \times 3$ conv, $2 \times 2$ max pooling, 128 $3 \times 3$ conv, 128 $3 \times 3$ conv, $2 \times 2$ max pooling, 256 dense, 256 dense, and softmax layers.

*Query Generation:* The most limiting resource for an adversary might be real data samples to query the base model (*e.g.*, medical records). Thus, we test query

[2]Flipping is not applied on the MNIST dataset.

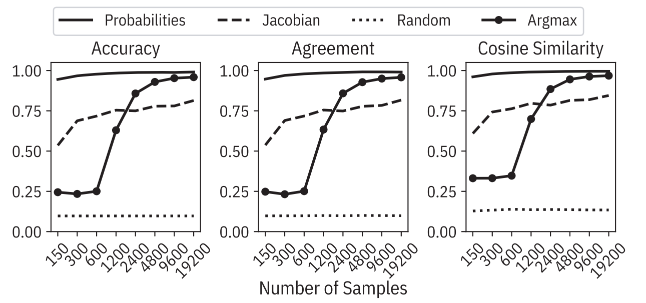

Fig. 2: Performance of four model stealing approaches.

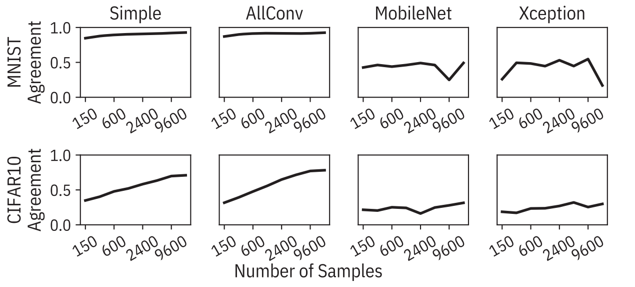

Fig. 3: Agreement with the base model for various types of attacker model architectures.

generation strategies and find the importance of using real data samples. We evaluate the attacker methods on varying # samples first, and set attacker budget to 50,000 queries, and training steps to 16,000 with 64-sized batch. Fig 2 shows the performance of the three attack strategies against the number of samples on the MNIST dataset. We can see that Sample attack performs best for various # samples. All attack methods including Sample leverage data augmentation. Our preliminary experiment showed that using data augmentation significantly improved the test accuracy (approx. 85% increase with 150 samples). ArgmaxSample also steals the model with high accuracy, but its replication is much slower especially at the beginning. For example, with data augmentation, Sample reaches 97% agreement with 300 samples, but ArgmaxSample goes only up to 96% with 19200 samples ($\approx 64\times$ samples).

The Jacobian augmentation is used to push samples towards the boundaries of each class in the direction of greatest increase in the loss function to generate samples defining the decision boundary. When applied to the input image, it successfully probes the classification boundary of the model, but results in generating imperceptible perturbation to an image so that the neural network misclassifies the input [21]. Thus, using Jacobian generates adversarial examples that are misclassified by the base model. Thus, leveraging these labels from the base model results in teaching the replicated model the wrong classification.

On the other hand, although Random attack has high training accuracy, its performance on test data is poor, giving the accuracy of random guess. Most samples generated by Random fall into one class, and this leads to train the replicated model to predict just one class regardless the input. This shows the importance having legitimate data samples to query the model. Based on these results, we use Sample attack, which can most accurately replicate the model in the defense-unaware scenario.

*Model Architecture:* The attacker has a choice of their own models to train. We test Sample to train Simple model described above, as well as MobileNet [10], AllConv [13], and Xception [11] models. Some complex models such as ResNet [22] and Inception [23] are not applicable to the test datasets we use due to the input image dimensions.

Fig 3 shows comparisons of performance regarding different replicated model architectures used.[3] As we can expect, the simplest model learns fastest. While approaches such as Xception and MobileNet work well with larger data [10], [11], they are not suitable in a model stealing scenario with relatively few samples. Also, we find AllConv not only learns relatively fast, but also performs well for all datasets. Therefore, we use AllConv throughout the rest of the experiments. For the IMDB dataset, we use the most popular approach using Bidirectional LSTM [24].

## B. Defense Evaluation

Now we compare defense methods against Sample attack method with AllConv model. As a defense, we consider adding one of the following perturbations to the output probability vector, where $\mathcal{U}(-1,1)$ is uniform random noise between -1 to 1, and $\beta$, $\rho$ and $f$ are parameters.

- **Uniform Random**: $\beta\mathcal{U}(-1,1)$.
- **Uniform Random** $\times$ **Concave**: $\beta\mathcal{U}(-1,1) \times 1/e^{-\frac{1}{2}\left(y_j^i/\rho\right)^2}$.
- **Uniform Random** $\times$ **Convex**: $\beta\mathcal{U}(-1,1) \times \left(1 - 1/e^{-\frac{1}{2}\left(y_j^i/\rho\right)^2}\right)$.
- **Ranking-preserving Uniform Random**: Same as **Uniform Random**, but the ranking of output classes is preserved to maintain the accuracy.
- **Sine**: $\beta \sin(f y_j^i)$.
- **Reverse Sigmoid**: a stretched and reversed sigmoid $r(y_j^i)$ explained in this paper.

The goal of the defense is twofold: 1) prevent model replication (low stolen model $S(M, \textit{defense})$ agreement), and 2) retain the performance of the protected model (high protected model $P(M, \textit{defense})$ agreement). The best parameters for the defense methods are searched using grid search to achieve the highest cosine similarity to the original model with at least 20% accuracy drop in stealing with attacker budget of 19200. When we cannot find parameters satisfying 20% accuracy drop in stealing, we instead choose the parameters with the highest accuracy drop in stealing.

[3]Note that the performance degradation is due to the use of only two third of the original training data less the attacker portion.

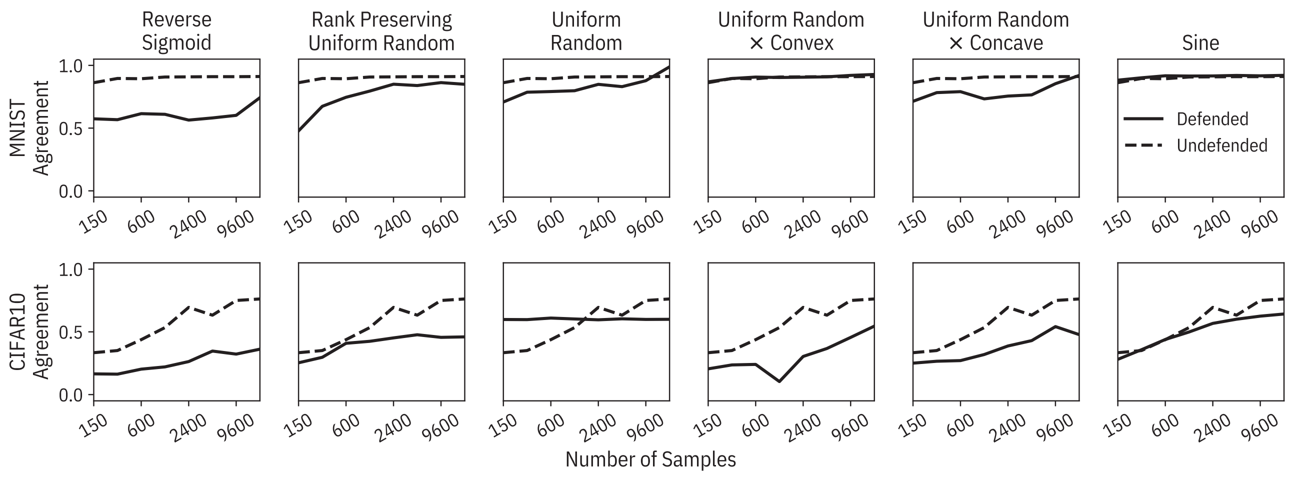

Fig. 4: Agreement of stolen model with base model using different defense types.

TABLE I: Agreements with 19200 queries. $M$: base model, $S(\cdot)$: stolen model, $P(\cdot)$: protected model.

| Dataset | MNIST | FASHION MNIST | CIFAR-10 | CIFAR-100 | STL-10 | IMDB |
|---|---|---|---|---|---|---|
| $S(M)$ | 0.93 | 0.95 | 0.82 | 0.48 | 0.62 | 0.84 |
| $P(M, \text{Reverse Sigmoid})$<br>$S(M, \text{Reverse Sigmoid})$ | 0.99<br>0.68 ⤸ **-0.31** | 0.99<br>0.70 ⤸ -0.29 | 0.85<br>0.40 ⤸ -0.45 | **1.00**<br>0.02 ⤸ **-0.98** | 0.99<br>0.37 ⤸ **-0.62** | **1.00**<br>0.60 ⤸ **-0.40** |
| $P(M, \text{Uniform Random})$<br>$S(M, \text{Uniform Random})$ | 0.60<br>0.98 ⤸ +0.38 | 0.56<br>0.09 ⤸ **-0.47** | 0.79<br>0.66 ⤸ -0.13 | 0.82<br>0.02 ⤸ -0.80 | 0.55<br>0.12 ⤸ -0.43 | 0.72<br>0.60 ⤸ -0.12 |
| $P(M, \text{Uniform Random} \times \text{Concave})$<br>$S(M, \text{Uniform Random} \times \text{Concave})$ | 0.88<br>0.93 ⤸ +0.05 | 0.86<br>0.91 ⤸ +0.05 | 0.69<br>0.54 ⤸ -0.15 | 0.82<br>0.02 ⤸ -0.80 | 0.92<br>0.57 ⤸ -0.35 | 0.72<br>0.59 ⤸ -0.13 |
| $P(M, \text{Uniform Random} \times \text{Convex})$<br>$S(M, \text{Uniform Random} \times \text{Convex})$ | 0.89<br>0.93 ⤸ **+0.04** | 0.86<br>0.90 ⤸ +0.04 | 0.70<br>0.58 ⤸ -0.12 | 0.82<br>0.02 ⤸ -0.80 | 0.98<br>0.59 ⤸ -0.39 | 0.72<br>0.60 ⤸ -0.12 |
| $P(M, \text{Ranking-preserving Uniform Random})$<br>$S(M, \text{Ranking-pre. Uniform Random})$ | **1.00**<br>0.87 ⤸ -0.13 | **1.00**<br>0.95 ⤸ -0.05 | **1.00**<br>0.46 ⤸ **-0.54** | **1.00**<br>0.23 ⤸ -0.77 | **1.00**<br>0.55 ⤸ -0.45 | **1.00**<br>0.76 ⤸ -0.24 |
| $P(M, \text{Sine})$<br>$S(M, \text{Sine})$ | 0.89<br>0.92 ⤸ +0.03 | 0.91<br>0.89 ⤸ -0.02 | 0.77<br>0.74 ⤸ -0.03 | 0.56<br>0.48 ⤸ -0.08 | 0.61<br>0.64 ⤸ +0.03 | 0.84<br>0.80 ⤸ -0.04 |

In Fig 4 and Table I, we see the proposed Reverse Sigmoid defense consistently slows down the replication process (Fig 4), and drops the stolen model agreements more than 20% for all tested datasets (Table I). Also, the protected models still have high agreements. Using a sinusoidal wave noise does not protect the model for all tested parameter combinations. The final perturbation made by sine is small as the sine applied to each class can interfere each other, and it is normalized to have only small effect. While **Uniform Random** may look effective for some datasets like FASHION MNIST, the agreements of the protected models are not reliable. In case of **Ranking-preserving Uniform Random**, we can achieve the perfect agreements, but the effectiveness of the defense is unpredictable (e.g., FASHION MNIST vs. CIFAR-10).

Table II shows the side effects of the Reverse Sigmoid defense. The protected model accuracy is well maintained because the defense tends not to change the labels if one class is dominating. To measure how much noise is added to the probability vectors, we use the average cosine similarity, the mean absolute error per dimension, and KL-divergence of the protected model and the base model outputs. In particular, we see that the mean absolute error is less than 0.2 in most datasets. This noise can be higher if the model

TABLE II: Effects of Reverse Sigmoid defense. $M$: base model, $P$: protected model, $S(P)$: stolen model from $P$.

| | Accuracy | | | Distance | | |
|---|---|---|---|---|---|---|
| Dataset | $M$ | $P$ | $S(P)$ | Cos | MAE | KL |
| MNIST | 0.99 | 0.99 | 0.68 | 0.41 | 0.17 | 3.27 |
| FASHION MNIST | 0.87 | 0.87 | 0.66 | 0.47 | 0.16 | 1.72 |
| CIFAR-10 | 0.75 | 0.73 | 0.36 | 0.61 | 0.48 | 1.26 |
| CIFAR-100 | 0.43 | 0.43 | 0.02 | 0.33 | 0.02 | 2.30 |
| STL-10 | 0.51 | 0.51 | 0.31 | 0.74 | 0.16 | 2.20 |
| IMDB | 0.81 | 0.81 | 0.60 | 0.80 | 0.39 | 0.48 |

prediction is more confident. However, in this case, the decision is unlikely to be affected in many applications.

To illustrate our reverse sigmoid defense can also protect other simple machine learning models, we tested it on a logistic regression model trained on the iris dataset [20] against the Tramer et al. attacks [2]. This attack solves the system of the model equations, and almost completely steals the model (1.00 agreement). While the attack should be able to recover the model in 15 queries (the number of parameters in the three-class logistic regression model), the stolen model was much more resilient. We tested the agreements of 100 stolen models, randing from $1\times -100\times$ the number of queries required to perfectly replicate the model. The passive attack had a mean agreement of 0.81, where the 20th and 80th percentiles were 0.83 and 0.91, indicating there was significant volatility in the stolen models. When querying the protected model $20k\times$ more than necessary for an unprotected model, the stolen model only reached an accuracy of 0.92. The adaptive attacks fared significantly worse, with mean accuracy of 0.65 for the local adaptive and 0.36 for the oracle adaptive attacks.

### C. Base Model Accuracy v. Protection Effectiveness

TABLE III: Base model and stealing on CIFAR-10.

| Model | $M$ acc. | $P$ acc. | $S$ acc. | Acc. drop |
|---|---|---|---|---|
| AllConv | 0.75 | 0.73 | 0.36 | 0.37 |
| Simple | 0.70 | 0.69 | 0.50 | 0.19 |
| Xception | 0.36 | 0.36 | 0.23 | 0.13 |
| MobileNet | 0.24 | 0.22 | 0.21 | 0.01 |

Since we add noise to the confident classes, the output probability of a model can affect the performance of the Reverse Sigmoid defense. To see this, we train different models on CIFAR-10 dataset, resulting in different accuracies. Then, we apply the Reverse Sigmoid protection, and try the attack with AllConv. In Table III, we see that if the original accuracy ($M$ accuracy) is higher, the defense is more effective (higher accuracy drops). This is especially important because when the accuracy of the original model is high, there is higher demand of protection.

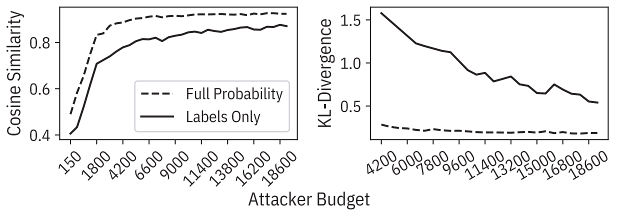

Fig. 5: Performance difference of model stealing using all class probabilities vs. only the top-1 label on CIFAR-10.

### D. Robustness against Defense-aware Attack

We test the robustness of the Reverse Sigmoid defense on CIFAR-10 when the attacker knows more information.

*Attack using the same defense layer:* If the defense and the parameters get to be known by the attacker, the attacker can use exactly the same defense layer in their model. The Reverse Sigmoid defense is not a standard neural network layer, and have ambiguity that multiple logit values are mapped to the same probability by the defense layer. This essentially propagates wrong gradient values to the model, degrading the stealing process (Table IV).

*Attack using the mean-squared-error loss function:* To replicate the model output more precisely, the attacker may use the mean-squared-error (MSE) loss function instead of the more common cross entropy loss function which the Reverse Sigmoid defense is designed for. However, for the

TABLE IV: Agreements of attack variants.

| Attack | Agree. |
|---|---|
| Same Defense Layer | 0.10 |
| MSE Loss | 0.18 |
| Inversion (MLP) | 0.22 |
| Argmax | 0.78 |

same reason as the attack using the same defense layer, this loss function is not free from the ambiguity, and still shows poor performance as shown in Table IV.

*Attack using an inversion mapping:* We now evaluate the adversary's ability to recover the original unprotected class probabilities from the protected model assuming they have *full knowledge* of the parameters of the protection, *i.e.*, $y^i, \hat{y}^i$ pairs. We use a multilayer perceptron (MLP) model with two hidden layers ($5 \times K$ and $3 \times K$ neurons), and $K$ input and output values. The model is trained on 16,670 *real* output pairs from the base model $M$ and the protected model $P$ ($y^i, \hat{y}^i$ pairs) and optimizing over the loss $\mathsf{KL}(\hat{P}, M)$. We find that the MLP attack did not increase the stolen model agreement, as shown in Table IV.

*Attack using labels only (*ArgmaxSample*):* As shown in Sec IV-A, using argmax requires a larger budget than attacks using the probabilities ($\approx 64\times$ queries compared to Sample). But still, the attacker can take the top-1 class of the result which is usually correct. This approach achieves much better replication of a defended model given enough budget as shown in Table IV. Still, by forcing the attacker to discard probability values, the attacker has to use $4\times$ as many queries in our evaluation on CIFAR-10 dataset. That is, with probability values, the attacker needs only 4800 queries to reach 0.7813 agreement, while he needs 19200 queries to reach the same agreement without probability values. Besides the agreement, we can see lower cosine similarity and high KL-divergence, Fig 5, even with 19200 queries, meaning that while top-1 decision is quickly learned, the trait of the network including output distributions takes more queries to replicate. This difference can limit the generalization power, and it can be especially important if the attacker further wants to use the model for adversarial example generation [6]. Finally, note that this is a degenerate attack that applies to any defense that maintains top-1 accuracy.

## V. Conclusion

Neural networks are becoming one of the key assets of an enterprise, but they are vulnerable to stealing attacks. We proposed a method that can be applied to wide variety of neural network models, and evaluated the protection performance with six datasets, four neural network architectures, and diverse threat models and attack parameters. Our approach either prevented the stealing entirely or slowed down the stealing process up to $64\times$ in the worst case when the attacker knows the defense.

## References

[1] E. K. Ringger, M. Carmen, R. Haertel, K. D. Seppi, D. Lonsdale, P. McClanahan, J. L. Carroll, and N. Ellison, "Assessing the costs of machine-assisted corpus annotation through a user study." in *Proceedings of the 2008 International Conference on Language Resources and Evaluation (LREC)*, 2008.

[2] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[3] Google. Cloud vision API. [Online]. Available: https://cloud.google.com/vision/

[4] Microsoft. Cognitive services pricing—computer vision API. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/cognitive-services/computer-vision/

[5] IBM. Watson visual recognition. [Online]. Available: https://www.ibm.com/watson/services/visual-recognition/pricing/index.html#pricing

[6] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 506–519. [Online]. Available: http://doi.acm.org/10.1145/3052973.3053009

[7] Y. Shi, Y. Sagduyu, and A. Grushin, "How to steal a machine learning classifier with deep learning," in *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, April 2017, pp. 1–5.

[8] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," in *2018 IEEE Symposium on Security and Privacy (SP)*, vol. 00, 2018, pp. 629–645. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00038

[9] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*. IEEE, 2012, pp. 3642–3649.

[10] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[11] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Computer vision and pattern recognition (CVPR), 2017 IEEE conference on*, 2017, pp. 1251–1258.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," *CoRR*, vol. abs/1412.6806, 2014. [Online]. Available: http://arxiv.org/abs/1412.6806

[14] G. Hinton, N. Srivastava, and K. Swersky, "rmsprop: Divide the gradient by a running average of its recent magnitude," http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, accessed: 2018-04-23.

[15] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms.

[18] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[19] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[20] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[21] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[24] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.