Practical Black-Box Attacks against Machine Learning

Nicolas Papernot
Pennsylvania State University
ngp5056@cse.psu.edu

Patrick McDaniel
Pennsylvania State University
mcdaniel@cse.psu.edu

Ian Goodfellow*
OpenAI
ian@openai.com

Somesh Jha
University of Wisconsin
jha@cs.wisc.edu

Z. Berkay Celik
Pennsylvania State University
zbc102@cse.psu.edu

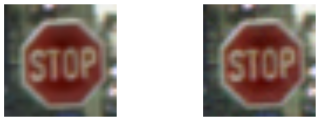
Ananthram Swami
US Army Research Laboratory
ananthram.swami.civ@mail.mil

ABSTRACT

Machine learning (ML) models, e.g., deep neural networks (DNNs), are vulnerable to adversarial examples: malicious inputs modified to yield erroneous model outputs, while appearing unmodified to human observers. Potential attacks include having malicious content like malware identified as legitimate or controlling vehicle behavior. Yet, all existing adversarial example attacks require knowledge of either the model internals or its training data. We introduce the first practical demonstration of an attacker controlling a remotely hosted DNN with no such knowledge. Indeed, the only capability of our black-box adversary is to observe labels given by the DNN to chosen inputs. Our attack strategy consists in training a local model to substitute for the target DNN, using inputs synthetically generated by an adversary and labeled by the target DNN. We use the local substitute to craft adversarial examples, and find that they are misclassified by the targeted DNN. To perform a real-world and properly-blinded evaluation, we attack a DNN hosted by MetaMind, an online deep learning API. We find that their DNN misclassifies 84.24% of the adversarial examples crafted with our substitute. We demonstrate the general applicability of our strategy to many ML techniques by conducting the same attack against models hosted by Amazon and Google, using logistic regression substitutes. They yield adversarial examples misclassified by Amazon and Google at rates of 96.19% and 88.94%. We also find that this black-box attack strategy is capable of evading defense strategies previously found to make adversarial example crafting harder.

1. INTRODUCTION

A *classifier* is a ML model that learns a mapping between inputs and a set of *classes*. For instance, a malware detector is a classifier taking executables as inputs and assigning them to the benign or malware class. Efforts in the security [5, 2, 9, 18] and machine learning [14, 4] communities exposed the vulnerability of classifiers to integrity attacks. Such attacks are often instantiated by *adversarial examples*: legitimate inputs altered by adding small, often imperceptible, perturbations to force a learned classifier to misclassify the resulting adversarial inputs, while remaining correctly classified by a human observer. To illustrate, consider the following images, potentially consumed by an autonomous vehicle [13]:



To humans, these images appear to be the same: our biological classifiers (vision) identify each image as a stop sign. The image on the left [13] is indeed an ordinary image of a stop sign. We produced the image on the right by adding a precise perturbation that forces a particular DNN to classify it as a yield sign, as described in Section 5.2. Here, an adversary could potentially use the altered image to cause a car without failsafes to behave dangerously. This attack would require modifying the image used internally by the car through transformations of the physical traffic sign. Related works showed the feasibility of such physical transformations for a state-of-the-art vision classifier [6] and face recognition model [11]. It is thus conceivable that physical adversarial traffic signs could be generated by maliciously modifying the sign itself, e.g., with stickers or paint.

In this paper, we introduce the first demonstration that *black-box attacks* against DNN classifiers are practical for real-world adversaries with *no* knowledge about the model. We assume the adversary (a) has no information about the structure or parameters of the DNN, and (b) does not have access to any large training dataset. The adversary's only capability is to observe labels assigned by the DNN for chosen inputs, in a manner analog to a cryptographic oracle.

Our novel attack strategy is to train a local substitute DNN with a *synthetic* dataset: the inputs are synthetic and generated by the adversary, while the outputs are labels assigned by the target DNN and observed by the adversary. Adversarial examples are crafted using the substitute parameters, which are known to us. They are not only misclassified by the substitute but also by the target DNN, because both models have similar decision boundaries.

This is a considerable departure from previous work, which evaluated perturbations required to craft adversarial examples using either: (a) detailed knowledge of the DNN architecture and parameters [2, 4, 9, 14], or (b) an independently collected training set to fit an auxiliary model [2, 4, 14]. This

*Work done while the author was at Google.

ASIA CCS '17, April 02 - 06, 2017, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-4944-4/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3052973.3053009

limited their applicability to strong adversaries capable of gaining insider knowledge of the targeted ML model, or collecting large labeled training sets. We release assumption (a) by learning a substitute: it gives us the benefit of having full access to the model and apply previous adversarial example crafting methods. We release assumption (b) by replacing the independently collected training set with a synthetic dataset constructed by the adversary with synthetic inputs and labeled by observing the target DNN's output.

Our threat model thus corresponds to the real-world scenario of users interacting with classifiers hosted remotely by a third-party keeping the model internals secret. In fact, we instantiate our attack against classifiers automatically trained by MetaMind, Amazon, and Google. We are able to access them only after training is completed. Thus, we provide the first correctly blinded experiments concerning adversarial examples as a security risk.

We show that our black-box attack is applicable to many remote systems taking decisions based on ML, because it combines three key properties: (a) the capabilities required are limited to observing output class labels, (b) the number of labels queried is limited, and (c) the approach applies and scales to different ML classifier types (see Section 7), in addition to state-of-the-art DNNs. In contrast, previous work failed to simultaneously provide all of these three key properties [4, 14, 12, 15, 18]. Our contributions are:

- We introduce in Section 4 an attack against black-box DNN classifiers. It crafts adversarial examples without knowledge of the classifier training data or model. To do so, a synthetic dataset is constructed by the adversary to train a substitute for the targeted DNN classifier.
- In Section 5, we instantiate the attack against a remote DNN classifier hosted by MetaMind. The DNN misclassifies 84.24% of the adversarial inputs crafted.
- The attack is calibrated in Section 6 to (a) reduce the number of queries made to the target model and (b) maximize misclassification of adversarial examples.
- We generalize the attack to other ML classifiers like logistic regression. In Section 7, we target models hosted by Amazon and Google. They misclassify adversarial examples at rates of 96.19% and 88.94%.
- Section 8 shows that our attack evades defenses proposed in the literature because the substitute trained by the adversary is unaffected by defenses deployed on the targeted oracle model to reduce its vulnerability.
- In Appendix B, we provide an intuition of why adversarial examples crafted with the substitute also mislead target models by empirically observing that substitutes have gradients correlated to the target's.

Disclosure: We disclosed our attacks to MetaMind, Amazon, and Google. Note that no damage was caused as we demonstrated control of models created for our own account.

2. ABOUT DEEP NEURAL NETWORKS

We provide preliminaries of deep learning to enable understanding of our threat model and attack. We refer readers interested to the more detailed presentation in [3].

A *deep neural network* (DNN), as illustrated in Figure 1, is a ML technique that uses a hierarchical composition of n parametric functions to model an input $\vec{x}$. Each function f_i

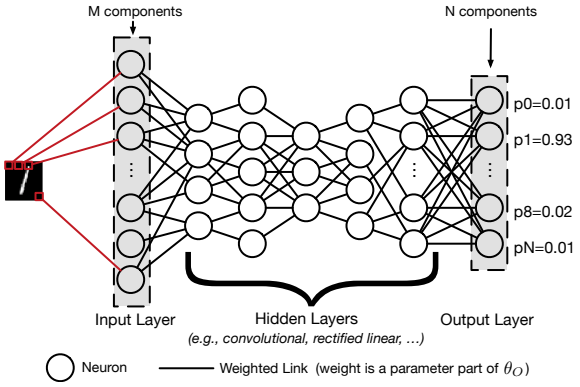


Figure 1: **DNN Classifier:** the model processes an image of a handwritten digit and outputs the probility of it being in one of the $N = 10$ classes for digits 0 to 9 (from [10]).

for $i \in 1..n$ is modeled using a layer of neurons, which are elementary computing units applying an *activation function* to the previous layer's weighted representation of the input to generate a new representation. Each layer is parameterized by a weight vector θ_i (we omit the vector notation) impacting each neuron's activation. Such weights hold the knowledge of a DNN model F and are evaluated during its training phase, as detailed below. Thus, a DNN defines and computes:

$$F(\vec{x}) = f_n\left(\theta_n, f_{n-1}\left(\theta_{n-1}, \ \ldots \ f_2\left(\theta_2, f_1\left(\theta_1, \vec{x}\right)\right)\right)\right) \qquad (1)$$

The *training phase* of a DNN F learns values for its parameters $\theta_F = \{\theta_1, ..., \theta_n\}$. We focus on classification tasks, where the goal is to assign inputs a label among a predefined set of labels. The DNN is given a large set of known input-output pairs $(\vec{x}, \vec{y})$ and it adjusts weight parameters to reduce a cost quantifying the prediction error between the prediction $F(\vec{x})$ and the correct output $\vec{y}$. The adjustment is typically performed using techniques derived from the back-propagation algorithm. Briefly, such techniques successively propagate error gradients with respect to network parameters from the network's output layer to its input layer.

During the *test phase*, the DNN is deployed with a fixed set of parameters θ_F to make predictions on inputs unseen during training. We consider classifiers: the DNN produces a probability vector $F(\vec{x})$ encoding its belief of input $\vec{x}$ being in each of the classes (cf. Figure 1). The weight parameters θ_F hold the model knowledge acquired by training. Ideally, the model should generalize and make accurate predictions for inputs outside of the domain explored during training. However, attacks manipulating DNN inputs with adversarial examples showed this is not the case in practice [4, 9, 14].

3. THREAT MODEL

A taxonomy of adversaries against DNN classifiers is found in [9]. In our work, the adversary seeks to force a classifier to misclassify inputs in any class different from their correct class. To achieve this, we consider a weak adversary with access to the DNN output only. The adversary has no knowledge of the architectural choices made to design the DNN, which include the number, type, and size of layers, nor of the training data used to learn the DNN's parameters. Such attacks are referred to as *black box*, where adversaries need not know internal details of a system to compromise it.

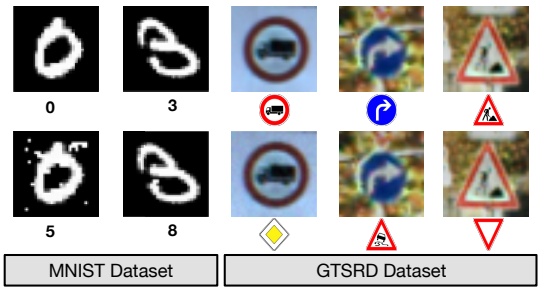


Figure 2: **Adversarial samples** (misclassified) in the bottom row are created from the legitimate samples [7, 13] in the top row. The DNN outputs are identified below the samples.

Targeted Model: We consider attackers targeting a multi-class DNN classifier. It outputs probability vectors, where each vector component encodes the DNN's belief of the input being part of one of the predefined classes. We consider the ongoing example of a DNN classifying images, as shown in Figure 1. Such DNNs can be used to classify handwritten digits into classes associated with digits from 0 to 9, images of objects in a fixed number of categories, or images of traffic signs into classes identifying its type (STOP, yield, ...).

Adversarial Capabilities: The *oracle* O is the targeted DNN. Its name refers to the only capability of the adversary: accessing the label $\tilde{O}(\vec{x})$ for any input $\vec{x}$ by querying oracle O. The output label $\tilde{O}(\vec{x})$ is the index of the class assigned the largest probability by the DNN:

$$\tilde{O}(\vec{x}) = \arg \max_{j \in 0..N-1} O_j(\vec{x}) \quad (2)$$

where $O_j(\vec{x})$ is the j-th component of the probability vector $O(\vec{x})$ output by DNN O. Distinguishing between labels and probabilities makes adversaries realistic (they more often have access to labels than probabilities) but weaker: labels encode less information about the model's learned behavior. **Accessing labels $\tilde{O}$ produced by the DNN O is the only capability assumed in our threat model.** We do not have access to the oracle internals or training data.

Adversarial Goal: We want to produce a minimally altered version of any input $\vec{x}$, named *adversarial sample*, and denoted $\vec{x^*}$, misclassified by oracle O: $\tilde{O}(\vec{x^*}) \neq \tilde{O}(\vec{x})$. This corresponds to an attack on the oracle's output integrity. Adversarial samples solve the following optimization problem:

$$\vec{x^*} = \vec{x} + \arg\min\{\vec{z} : \tilde{O}(\vec{x} + \vec{z}) \neq \tilde{O}(\vec{x})\} = \vec{x} + \delta_{\vec{x}} \quad (3)$$

Examples of adversarial samples can be found in Figure 2. The first row contains legitimate samples and the second corresponding adversarial samples that are misclassified. This misclassification must be achieved by adding a minimal perturbation $\delta\vec{x}$ so as to evade human detection. Even with total knowledge of the architecture used to train model O and its parameters resulting from training, finding such a minimal perturbation is not trivial, as properties of DNNs preclude the optimization problem from being linear or convex. This is exacerbated by our threat model: removing knowledge of model O's architecture and training data makes it harder to find a perturbation such that $\tilde{O}(\vec{x} + \delta\vec{x}) \neq \tilde{O}(\vec{x})$ holds.

In Appendix C, we give a presentation of attacks conducted in related threat models—with stronger assumptions.

4. BLACK-BOX ATTACK STRATEGY

We introduce our black-box attack. As stated in Section 3, the adversary wants to craft inputs misclassified by the ML model using the sole capability of accessing the label $\tilde{O}(\vec{x})$ assigned by classifier for any chosen input $\vec{x}$. The strategy is to learn a *substitute* for the target model using a synthetic dataset generated by the adversary and labeled by observing the oracle output. Then, adversarial examples are crafted using this substitute. We expect the target DNN to misclassify them due to transferability between architectures [14, 4]

To understand the difficulty of conducting the attack under this threat model, recall Equation 3 formalizing the adversarial goal of finding a minimal perturbation that forces the targeted oracle to misclassify. A closed form solution cannot be found when the target is a non-convex ML model: e.g., a DNN. The basis for most adversarial attacks [4, 9, 14] is to approximate its solution using gradient-based optimization on functions defined by a DNN. Because evaluating these functions and their gradients requires knowledge of the DNN architecture and parameters, such an attack is not possible under our black-box scenario. It was shown that adversaries with access to an independently collected labeled training set from the same population distribution than the oracle could train a model with a different architecture and use it as a substitute [14]: adversarial examples designed to manipulate the substitute are often misclassified by the targeted model. However, many modern machine learning systems require large and expensive training sets for training. For instance, we consider models trained with several tens of thousands of labeled examples. This makes attacks based on this paradigm unfeasible for adversaries without large labeled datasets.

In this paper, we show black-box attacks can be accomplished at a much lower cost, without labeling an independent training set. In our approach, to enable the adversary to train a substitute model without a real labeled dataset, we use the target DNN as an oracle to construct a synthetic dataset. The inputs are synthetically generated and the outputs are labels observed from the oracle. Using this synthetic dataset, the attacker builds an approximation F of the model O learned by the oracle. This *substitute network* F is then used to craft adversarial samples misclassified by F Indeed, with its full knowledge of the substitute DNN F parameters, the adversary can use one of the previously described attacks [4, 9] to craft adversarial samples misclassified by F. As long as the transferability property holds between F and O, adversarial samples crafted for F will also be misclassified by O. This leads us to propose the following strategy:

1. **Substitute Model Training:** the attacker queries the oracle with synthetic inputs selected by a Jacobian-based heuristic to build a model F approximating the oracle model O's decision boundaries.

2. **Adversarial Sample Crafting:** the attacker uses substitute network F to craft adversarial samples, which are then misclassified by oracle O due to the transferability of adversarial samples.

4.1 Substitute Model Training

Training a substitute model F approximating oracle O is challenging because we must: (1) select an architecture for our substitute without knowledge of the targeted oracle's architecture, and (2) limit the number of queries made to the oracle in order to ensure that the approach is tractable. Our

approach, illustrated in Figure 3, overcomes these challenges mainly by introducing a synthetic data generation technique, the *Jacobian-based Dataset Augmentation*. We emphasize that *this technique is not designed to maximize the substitute DNN's accuracy but rather ensure that it approximates the oracle's decision boundaries with few label queries.*

Substitute Architecture: This factor is not the most limiting as the adversary must at least have some partial knowledge of the oracle input (e.g., images, text) and expected output (e.g., classification). The adversary can thus use an architecture adapted to the input-output relation. For instance, a convolutional neural network is suitable for image classification. Furthermore, we show in Section 6 that the type, number, and size of layers used in the substitute DNN have relatively little impact on the success of the attack. Adversaries can also consider performing an architecture exploration and train several substitute models before selecting the one yielding the highest attack success.

Generating a Synthetic Dataset: To better understand the need for synthetic data, note that we could potentially make an infinite number of queries to obtain the oracle's output $O(\vec{x})$ for any input $\vec{x}$ belonging to the input domain. This would provide us with a copy of the oracle. However, this is simply not tractable: consider a DNN with M input components, each taking discrete values among a set of K possible values, the number of possible inputs to be queried is K^M. The intractability is even more apparent for inputs in the continuous domain. Furthermore, making a large number of queries renders the adversarial behavior easy to detect.

A natural alternative is to resort to randomly selecting additional points to be queried. For instance, we tried using Gaussian noise to select points on which to train substitutes. However, the resulting models were not able to learn by querying the oracle. This is likely due to noise not being representative of the input distribution. To address this issue, we thus introduce a heuristic efficiently exploring the input domain and, as shown in Sections 5 and 6, drastically limits the number of oracle queries. Furthermore, our technique also ensures that the substitute DNN is an approximation of the targeted DNN i.e. it learns similar decision boundaries.

The heuristic used to generate synthetic training inputs is based on identifying directions in which the model's output is varying, around an initial set of training points. Such directions intuitively require more input-output pairs to capture the output variations of the target DNN O. Therefore, to get a substitute DNN accurately approximating the oracle's decision boundaries, the heuristic prioritizes these samples when querying the oracle for labels. These directions are identified with the substitute DNN's Jacobian matrix J_F, which is evaluated at several input points $\vec{x}$ (how these points are chosen is described below). Precisely, the adversary evaluates the sign of the Jacobian matrix dimension corresponding to the label assigned to input $\vec{x}$ by the oracle: $\text{sgn}\left(J_F(\vec{x})[\tilde{O}(\vec{x})]\right)$. To obtain a new synthetic training point, a term $\lambda \cdot \text{sgn}\left(J_F(\vec{x})[\tilde{O}(\vec{x})]\right)$ is added to the original point $\vec{x}$. We name this technique *Jacobian-based Dataset Augmentation*. We base our substitute training algorithm on the idea of iteratively refining the model in directions identified using the Jacobian.

Substitute DNN Training Algorithm: We now describe

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initial training set S_0.

Input: $\tilde{O}$, max_ρ, S_0, λ
1: Define architecture F
2: **for** $\rho \in 0$ .. $max_\rho - 1$ **do**
3: // *Label the substitute training set*
4: $D \leftarrow \left\{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\right\}$
5: // *Train F on D to evaluate parameters θ_F*
6: $\theta_F \leftarrow \text{train}(F, D)$
7: // *Perform Jacobian-based dataset augmentation*
8: $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
9: **end for**
10: **return** θ_F

the five-step training procedure outlined in Algorithm 1:

- **Initial Collection (1):** The adversary collects a very small set S_0 of inputs representative of the input domain. For instance, if the targeted oracle O classifies handwritten digits, the adversary collects 10 images of each digit 0 through 9. We show in Section 5 that this set does not necessarily have to come from the distribution from which the targeted oracle was trained.
- **Architecture Selection (2):** The adversary selects an architecture to be trained as the substitute F. Again, this can be done using high-level knowledge of the classification task performed by the oracle (e.g., convolutional networks are appropriate for vision)
- **Substitute Training:** The adversary iteratively trains more accurate substitute DNNs F_ρ by repeating the following for $\rho \in 0..\rho_{max}$:
  - **Labeling (3):** By querying for the labels $\tilde{O}(\vec{x})$ output by oracle O, the adversary labels each sample $\vec{x} \in S_\rho$ in its initial substitute training set S_ρ.
  - **Training (4):** The adversary trains the architecture chosen at step (2) using substitute training set S_ρ in conjunction with classical training techniques.
  - **Augmentation (5):** The adversary applies our augmentation technique on the initial substitute training set S_ρ to produce a larger substitute training set $S_{\rho+1}$ with more synthetic training points. This new training set better represents the model's decision boundaries. The adversary repeats steps (3) and (4) with the augmented set $S_{\rho+1}$.

Step (3) is repeated several times to increase the substitute DNN's accuracy and the similarity of its decision boundaries with the oracle. We introduce the term *substitute training epoch*, indexed with ρ, to refer to each iteration performed. This leads to this formalization of the Jacobian-based Dataset Augmentation performed at step (5) of our substitute training algorithm to find more synthetic training points:

$$S_{\rho+1} = \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho \qquad (4)$$

where λ is a parameter of the augmentation: it defines the size of the step taken in the sensitive direction identified by the Jacobian matrix to augment the set S_ρ into $S_{\rho+1}$.

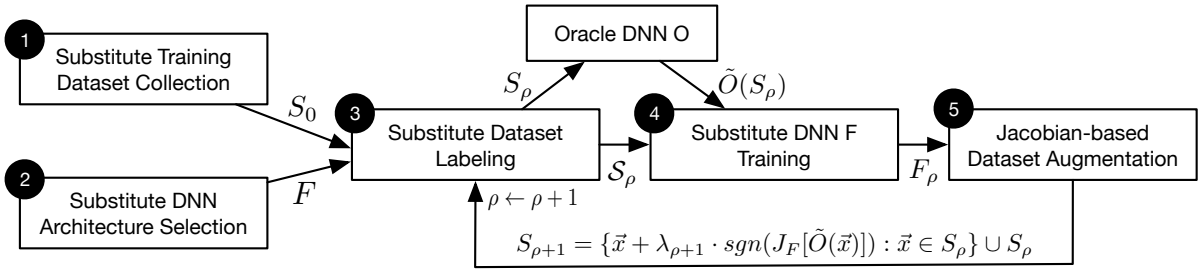


Figure 3: **Training of the substitute DNN** F: the attacker (1) collects an initial substitute training set S_0 and (2) selects an architecture F. Using oracle $\tilde{O}$, the attacker (3) labels S_0 and (4) trains substitute F. After (5) Jacobian-based dataset augmentation, steps (3) through (5) are repeated for several substitute epochs ρ.

4.2 Adversarial Sample Crafting

Once the adversary trained a substitute DNN, it uses it to craft adversarial samples. This is performed by implementing two previously introduced approaches described in [4, 9]. We provide an overview of the two approaches, namely the *Goodfellow et al. algorithm* and the *Papernot et al. algorithm*. Both techniques share a similar intuition of evaluating the model's sensitivity to input modifications in order to select a small perturbation achieving the misclassification goal[1].

Goodfellow et al. algorithm: This algorithm is also known as the *fast gradient sign method* [4]. Given a model F with an associated cost function $c(F, \vec{x}, y)$, the adversary crafts an adversarial sample $\vec{x^*} = \vec{x} + \delta_{\vec{x}}$ for a given legitimate sample $\vec{x}$ by computing the following perturbation:

$$\delta_{\vec{x}} = \varepsilon \, \text{sgn}(\nabla_{\vec{x}} c(F, \vec{x}, y)) \tag{5}$$

where perturbation $\text{sgn}(\nabla_{\vec{x}} c(F, \vec{x}, y))$ is the sign of the model's cost function [2] gradient. The cost gradient is computed with respect to $\vec{x}$ using sample $\vec{x}$ and label y as inputs. The value of the *input variation parameter* ε factoring the sign matrix controls the perturbation's amplitude. Increasing its value increases the likelihood of $\vec{x^*}$ being misclassified by model F but on the contrary makes adversarial samples easier to detect by humans. In Section 6, we evaluate the impact of parameter ε on the successfulness of our attack.

Papernot et al. algorithm: This algorithm is suitable for source-target misclassification attacks where adversaries seek to take samples from any legitimate source class to any chosen target class [9]. Misclassification attacks are a special case of source-target misclassifications, where the target class can be any class different from the legitimate source class. Given model F, the adversary crafts an adversarial sample $\vec{x^*} = \vec{x} + \delta_{\vec{x}}$ for a given legitimate sample $\vec{x}$ by adding a perturbation $\delta_{\vec{x}}$ to a subset of the input components $\vec{x}_i$.

To choose input components forming perturbation $\delta_{\vec{x}}$, components are sorted by decreasing adversarial saliency value. The adversarial saliency value $S(\vec{x}, t)[i]$ of component i for an adversarial target class t is defined as:

$$S(\vec{x}, t)[i] = \begin{cases} 0 \text{ if } \frac{\partial F_t}{\partial \vec{x}_i}(\vec{x}) < 0 \text{ or } \sum_{j \neq t} \frac{\partial F_j}{\partial \vec{x}_i}(\vec{x}) > 0 \\ \frac{\partial F_t}{\partial \vec{x}_i}(\vec{x}) \left| \sum_{j \neq t} \frac{\partial F_j}{\partial \vec{x}_i}(\vec{x}) \right| \text{ otherwise} \end{cases} \tag{6}$$

where matrix $J_F = \left[\frac{\partial F_j}{\partial \vec{x}_i}\right]_{ij}$ is the model's Jacobian matrix. Input components i are added to perturbation $\delta_{\vec{x}}$ in order of decreasing adversarial saliency value $S(\vec{x}, t)[i]$ until the resulting adversarial sample $\vec{x^*} = \vec{x} + \delta_{\vec{x}}$ is misclassified by F. The perturbation introduced for each selected input component can vary: greater perturbation reduce the number of components perturbed to achieve misclassification.

Each algorithm has its benefits and drawbacks. The Goodfellow algorithm is well suited for fast crafting of many adversarial samples with relatively large perturbations thus potentially easier to detect. The Papernot algorithm reduces perturbations at the expense of a greater computing cost.

5. VALIDATION OF THE ATTACK

We validate our attack against remote and local classifiers. We first apply it to target a DNN remotely provided by MetaMind, through their API[3] that allows a user to train classifiers using deep learning. The API returns labels produced by the DNN for any given input but does not provide access to the DNN. This corresponds to the oracle described in our threat model. We show that:

- An adversary using our attack can reliably force the DNN trained using MetaMind on MNIST [7] to misclassify 84.24% of adversarial examples crafted with a perturbation not affecting human recognition.
- A second oracle trained locally with the German Traffic Signs Recognition Benchmark (GTSRB) [13], can be forced to misclassify more than 64.24% of altered inputs without affecting human recognition.

5.1 Attack against the MetaMind Oracle

Description of the Oracle: We used the MNIST handwritten digit dataset to train the DNN [7]. It comprises $60,000$ training and $10,000$ test images of handwritten digits. The task associated with the dataset is to identify the digit corresponding to each image. Each 28x28 grayscale sample is encoded as a vector of pixel intensities in the interval $[0, 1]$ and obtained by reading the image pixel matrix row-wise.

We registered for an API key on MetaMind's website, which gave us access to three functionalities: dataset upload, automated model training, and model prediction querying. We uploaded the $50,000$ samples included in the MNIST

[1]Our attack can be implemented with other adversarial example algorithms. We focus on these two in our evaluation.
[2]As described here, the method causes simple misclassification. It has been extended to achieve chosen target classes.
[3]The API can be accessed online at www.metamind.io

training set to MetaMind and then used the API to train a classifier on the dataset. We emphasize that training is automated: we have no access to the training algorithm, model architecture, or model parameters. All we are given is the accuracy of the resulting model, computed by MetaMind using a validation set created by isolating 10% of the training samples. Details can be found on MetaMind's website.

Training took 36 hours to return a classifier with a 94.97% accuracy. This performance cannot be improved as we cannot access or modify the model's specifications and training algorithm. Once training is completed, we could access the model predictions, for any input of our choice, through the API. Predictions take the form of a class label. This corresponds to the threat model described in Section 3.

Initial Substitute Training Sets: First, the adversary collects an initial substitute training set. We describe two such sets used to attack the MetaMind oracle:

- MNIST subset: This initial substitute training set is made of 150 samples from the MNIST test set. They differ from those used by the oracle for training as test and training sets are distinct. We assume adversaries can collect such a limited sample set under the threat model described in Section 3 with minimal knowledge of the oracle task: here, handwritten digit classification.
- Handcrafted set: To ensure our results do not stem from similarities between the MNIST test and training sets, we also consider a *handcrafted* initial substitute training set. We handcrafted 100 samples by handwriting 10 digits for each class between 0 and 9 with a laptop trackpad. We then adapted them to the MNIST format of 28x28 grayscale pixels. Some are shown below.

Substitute DNN Training: The adversary uses the initial substitute training sets and the oracle to train subsitute DNNs. Our substitute architecture A, a standard for image classification, is described in Table 13 (cf. appendix). The substitute DNN is trained on our machine for 6 substitute epochs. During each of these 6 epochs, the model is trained for 10 epochs from scratch with a learning rate of 10^{-2} and momentum of 0.9. Between substitute epochs, we perform a Jacobian-based dataset augmentation with a step size of $\lambda = 0.1$ to generate additional synthetic training data, which we label using the MetaMind oracle.

The accuracy of the two substitute DNNs is reported in Figure 4. It is computed with the MNIST test set (minus the 150 samples used in the first initial substitute training set). The adversary does *not* have access to this full test set: we solely use it to analyze our results. The two substitute DNNs respectively achieve a 81.20% and 67.00% accuracy on the MNIST test set after 6 substitute training epochs. These accuracies fall short of current state-of-the-art accuracies on this task. However, the adversary has access to a limited number of samples (in this case $6,400 = 100 \times 2^6$ instead of $50,000$ for state-of-the-art models). Furthermore, the adversarial goal is to craft adversarial samples misclassified by the oracle. *Instead of learning a substitute DNN with optimal accuracy, the adversary is interested in learning a substitute capable of mimicking the oracle decision boundaries.*

Substitute Epoch	Initial Substitute Training Set from MNIST test set	Initial Substitute Training Set from Handcrafted digits
0	24.86%	18.70%
1	41.37%	19.89%
2	65.38%	29.79%
3	74.86%	36.87%
4	80.36%	40.64%
5	79.18%	56.95%
6	81.20%	67.00%

Figure 4: **Substitute DNN Accuracies:** each column corresponds to an initial substitute training set: 150 MNIST test samples, and handcrafted digits. Accuracy is reported on the unused 9,850 MNIST test samples.

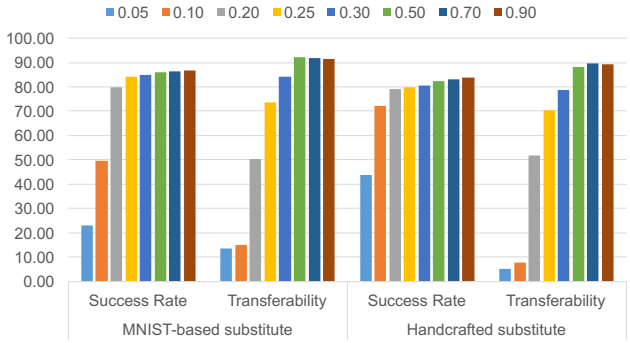


Figure 5: **Success Rate and Transferability of Adversarial Samples for the MetaMind attacks:** performed using MNIST-based and handcrafted substitutes: each bar corresponds to a different perturbation input variation.

Adversarial Sample Crafting: Using the substitute DNNs, we then craft adversarial samples using Goodfellow's algorithm. We decided to use the $10,000$ samples from the MNIST test set as our legitimate samples.[4] We evaluate sample crafting using two metrics: *success rate* and *transferability*. The *success rate* is the proportion of adversarial samples misclassified by the substitute DNN. Our goal is to verify whether these samples are also misclassified by the oracle or not. Therefore, the *transferability of adversarial samples* refers to the oracle misclassification rate of adversarial samples crafted using the substitute DNN.

Figure 5 details both metrics for each substitute DNN and for several values of the input variation ε (cf. Equation 5). Transferability reaches 84.24% for the first substitute DNN and 78.72% for the second, with input variations of $\varepsilon = 0.3$. Our attack strategy is thus effectively able to severely damage the output integrity of the MetaMind oracle. Using the substitute training set handcrafted by the adversary limits the transferability of adversarial samples when compared to the substitute set extracted from MNIST data, for all input variations except $\varepsilon = 0.2$. Yet, the transferability of both substitutes is similar, corroborating that our attack can be executed without access to any of the oracle's training data.

To analyze the labels assigned by the MetaMind oracle, we

[4]Again, adversaries do not need access to the dataset and can use any legitimate sample of their choice to craft adversarial samples. We use it in order to show that expected inputs can be misclassified on a large scale.

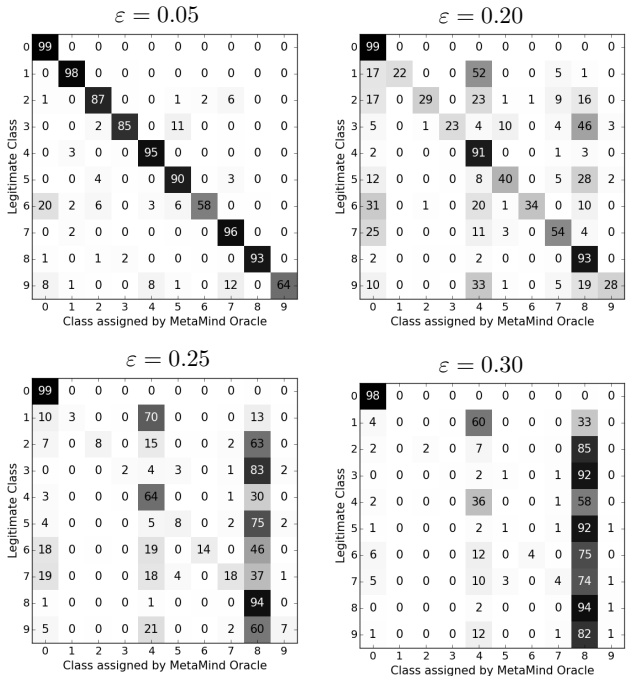


Figure 6: **MetaMind Oracle Confusion Matrices for different input variations ε.** Cell (x, y) indicates the share of digit y instances classified by the oracle as digit x.

plot confusion matrices for adversarial samples crafted using the first substitute DNN with 4 values of ε. In Figure 6, rates on the diagonal indicate the proportion of samples correctly classified by the oracle for each of the 10 classes. Off-diagonal values are the proportion of samples misclassified in a wrong class. For instance, cell $(8, 3)$ in the third matrix indicates that 89% instances of a 3 are classified as a 8 by the oracle when perturbed with an input variation of $\varepsilon = 0.25$. Confusion matrices converge to most samples being classified as 4s and 8s as ε increases. This could be due to DNNs more easily classifying inputs in these classes [9].

5.2 Attacking an oracle for the GTSRB

We now validate our attack on a different dataset, using an oracle trained locally to recognize traffic signs on the GTSRB dataset. The attack achieves higher transferability rates at lower distortions compared to the MNIST oracle.

Oracle Description: The GTSRB dataset is an image collection consisting of 43 traffic signs [13]. Images vary in size and are RGB-encoded. To simplify, we resize images to 32x32 pixels, recenter them by subtracting the mean component, and rescale them by factoring their standard deviations out. We keep $35,000$ images for our training set and $4,000$ for our validation set (out of the $39,209$ available), and $10,000$ for our test set (out of $12,630$). We train the oracle on our machine, using the DNN B from Table 13 (cf. appendix), for 50 epochs with a learning rate of 10^{-2} and a momentum of 0.9 (both decayed by 0.5 every 10 epochs).

Substitute DNN Training: The adversary uses two initial substitute training sets extracted from the GTSRB test set. The first includes the first $1,000$ samples and the second the first 500. The number of initial samples is higher than for

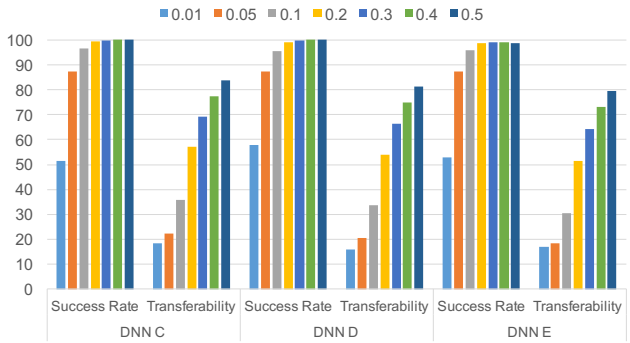


Figure 7: **Success Rate and Transferability of Adversarial Samples crafted on the GTRSRB dataset:** each bar corresponds to a different input variation.

MNIST substitutes as inputs have a higher dimensionality. We train three substitute architectures C, D, and E (cf. Table 13) using the oracle for 6 substitute training epochs with a Jacobian-based dataset augmentation parameter of $\lambda = 0.1$. Substitute C and E where trained with the $1,000$ sample initial substitute training set and achieve a 71.42% accuracy. Substitute D was trained with the initial set of 500 samples. Its accuracy of 60.12% is lower than C and E.

Adversarial Crafting: We use Goodfellow's algorithm with ε between 0.01 and 0.5 to craft adversarial samples from the test set. Results are shown in Figure 7. Adversarial samples crafted with variations $\varepsilon < 0.3$ are more transferable than those crafted with the same ε for MNIST models. This is likely due to the higher input dimensionality—$3,072$ components instead of 784—which means almost 4 times more perturbation is applied with the same ε. Nevertheless, with success rates higher than 98.98% and transferability rates ranging from 64.24% to 69.03% for $\varepsilon = 0.3$, which is hard to distinguish for humans, *the attack is successful*. The transferability of adversarial samples crafted using substitute DNN D is comparable or higher than corresponding samples for DNNs C and E, despite being less accurate (trained with less samples). This emphasizes that there is no strong correlation between substitute accuracy and transferability.

6. ATTACK ALGORITHM CALIBRATION

Having shown in Section 5 that an adversary can force an MNIST oracle from MetaMind, and a GTSRB oracle trained locally, to misclassify inputs, we now perform a parameter space exploration of both attack steps–the substitute DNN training and the adversarial sample crafting. We explore the following questions: "(1) How can substitute training be fine-tuned to improve adversarial sample transferability?" and (2) "For each adversarial sample crafting strategies, which parameters optimize transferability?". We found that:

- In Section 6.1, we show that the choice of substitute DNN architecture (number of layers, size, activation function, type) has a limited impact on adversarial sample transferability. Increasing the number of epochs, after the substitute DNN has reached an asymptotic accuracy, does not improve adversarial sample transferability.
- At comparable input perturbation magnitude, the Goodfellow and Papernot algorithms have similar transferability rates (see Section 6.2).

DNN ID	Accuracy ($\rho = 2$)	Accuracy ($\rho = 6$)	Transferability ($\rho = 6$)
A	30.50%	82.81%	75.74%
F	68.67%	79.19%	64.28%
G	72.88%	78.31%	61.17%
H	56.70%	74.67%	63.44%
I	57.68%	71.25%	43.48%
J	64.39%	68.99%	47.03%
K	58.53%	70.75%	54.45%
L	67.73%	75.43%	65.95%
M	62.64%	76.04	62.00%

Table 1: **Substitute Accuracy** at $\rho = 2$ and $\rho = 6$ substitute epochs and **Transferability of Adversarial Samples:** for $\varepsilon = 0.4$ after $\rho = 6$ substitute epochs.

In this section, we use an oracle trained locally to limit querying of the MetaMind API. We train architecture A (cf. Table 13) for 50 epochs with a learning parameter 10^{-2} and a momentum 0.9 (both decayed by 0.5 every 10 epochs).

6.1 Calibrating Substitute DNN Training

We first seek to quantify the impact of substitute training algorithm parameters on adversarial sample transferability and introduce a refinement to reduce oracle querying.

Choosing an Architecture: We train substitute DNNs A and F to M (cf. Table 13) using 150 samples from the MNIST test set as the substitute training set. During each of the 6 substitute training epochs, the DNN is trained for 5 epochs from scratch. Between epochs, synthetic data is added to the training set using Jacobian-based dataset augmentations with step $\lambda = 0.1$. The substitute architectures differ from the oracle's by the type, number, and size of layers. In Table 1, we report the accuracy of each architecture after 2 and 6 substitute training epochs, as well as the adversarial sample transferability after 6 epochs. Adversarial samples are crafted using the Goodfellow algorithm with an input variation of $\varepsilon = 0.4$ (which we justify later). The last column of Table 1 shows that the choice of architecture has a limited impact on adversarial sample transferability, and therefore on the attack success. The most important transferability drop follows from removing all convolutional layers. Changing the hidden layer activation function from rectified linear to a sigmoid does not impact transferability significantly.

Choosing the number of substitute epochs: Another tunable parameter is the number of epochs for which substitute DNNs are trained. Intuitively, one would hypothesize that the longer we train the substitute, the more samples labeled using the oracle are included in the substitute training set, thus the higher the transferability of adversarial samples will be. This intuition is confirmed only partially by our experiments on substitute DNN A. We find that for for input variations $\varepsilon \leq 0.3$, the transferability is slightly improved by a rate between +3% to +9%, but for variations $\varepsilon \geq 0.4$, the transferability is slightly degraded by less than 1%.

Setting the step size: We trained substitute A using different Jacobian-based dataset augmentation step sizes λ. Increasing or decreasing the step size (from $\lambda = 0.1$ used in the rest of this paper) does not modify the substitute accuracy by more than 3%. Larger step sizes decrease convergence stability while smaller values yield slower convergence. However, increasing step size λ negatively impacts adversarial sample transferability : for instance with a step size of 0.3 compared to 0.1, the transferability rate for $\varepsilon = 0.25$ is 10.82% instead of 22.35% and for $\varepsilon = 0.5$, 82.07% instead of 85.22%.

However, having the step size periodically alternating between positive and negative values improves the quality of the oracle approximation made by the substitute. This could be explained by the fact that after a few substitute epochs, synthetic inputs are outside of the input domain and are thus clipped to produce an acceptable input. We introduce an iteration period τ after which the step size is multiplied by -1. Thus, the step size λ is now replaced by:

$$\lambda_\rho = \lambda \cdot (-1)^{\lfloor \frac{\rho}{\tau} \rfloor} \tag{7}$$

where τ is set to be the number of epochs after which the Jacobian-based dataset augmentation does not lead any substantial improvement in the substitute. A grid search can also be performed to find an optimal value for the period τ. We also experimented with a decreasing grid step amplitude λ, but did not find that it yielded substantial improvements.

Reducing Oracle Querying: We apply *reservoir sampling* [16] to reduce the number of queries made to the oracle. This is useful when learning substitutes in realistic environments, or when interacting with paid APIs, where the number of label queries an adversary can make without exceeding a quota or being detected by a defender is limited. Reservoir sampling is a technique that randomly select κ samples from a list of samples. The total number of samples in the list can be both very large and unknown. We use it to select κ new inputs before a Jacobian-based dataset augmentation. This prevents the exponential growth of queries made to the oracle at each augmentation. At iterations $\rho > \sigma$ (the first σ iterations are performed normally), when considering the previous set $S_{\rho-1}$ of substitute training inputs, we select κ inputs from $S_{\rho-1}$ to be augmented in S_ρ. Using reservoir sampling ensures that each input in $S_{\rho-1}$ has an equal probability $\frac{1}{|S_{\rho-1}|}$ to be augmented in S_ρ. The number of queries made to the oracle is reduced from $n \cdot 2^\rho$ for the vanilla Jacobian-based augmentation to $n \cdot 2^\sigma + \kappa \cdot (\rho - \sigma)$ with reservoir sampling. In Section 7, we show that using reservoir sampling to reduce the number of synthetic training inputs does not significantly degrade the substitute accuracy.

6.2 Adversarial Sample Crafting

We compare the transferability of adversarial samples produced by each algorithm introduced previously [4, 9], to elect the strongest technique under our threat model.

Goodfellow's algorithm: Recall from Equation 5 the perturbation computed in the Goodfellow attack. Its only parameter is the variation ε added in the direction of the gradient sign. We use the same architecture set as before to quantify the impact of ε on adversarial sample transferability. In Figure 8, architecture A outperforms all others: it is a copy of the oracle's and acts as a baseline. Other architectures have asymptotic transferability rates ranging between 72.24% and 80.21%, confirming that *the substitute architecture choice has a limited impact on transferability.* Increasing the value of ε above 0.4 yields little improvement in transferability and should be avoided to guarantee indistinguishability of adversarial samples to humans.

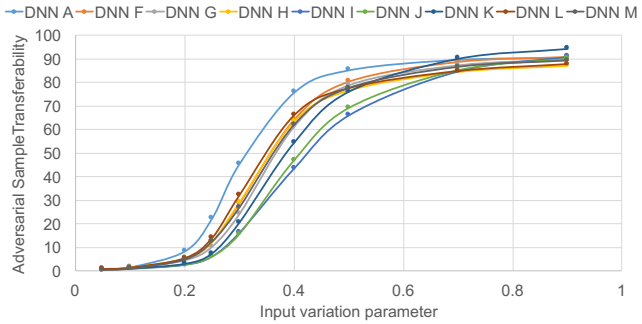


Figure 8: **Impact of input variation ε in the Goodfellow crafting algorithm on the transferability of adversarial samples:** for architectures from Table 1.

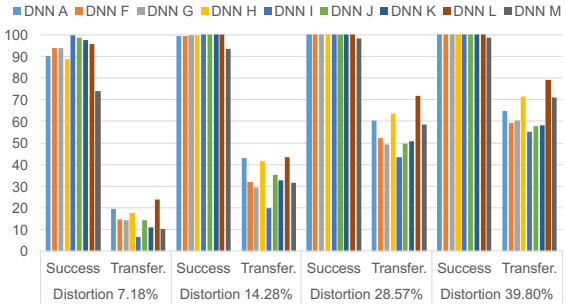


Figure 9: **Impact of the maximum distortion Υ in the Papernot algorithm on success rate and transferability of adversarial samples:** increasing Υ yields higher transferability rates across DNNs.

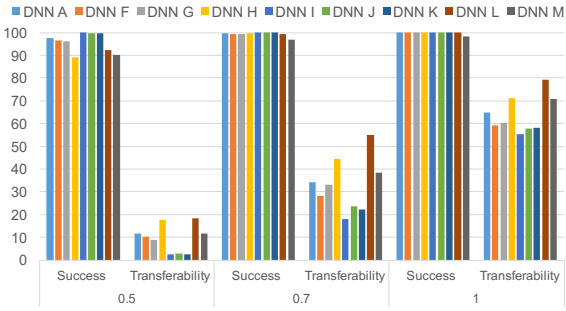


Figure 10: **Impact of the input variation ε in the Papernot algorithm on the success rate and adversarial sample transferability** computed for $\varepsilon \in \{0.5, 0.7, 1\}$ on DNNs from Table 1 with distortion $\Upsilon = 39.80\%$.

Papernot's algorithm: This algorithm is fine-tuned by two parameters: the *maximum distortion* Υ and the *input variation* ε. The maximum distortion[5] defines the number of input components that are altered in perturbation $\delta_{\vec{x}}$. The input variation, similarly to the Goodfellow algorithm, controls the amount of change induced to altered input components.

We first evaluate the impact of the maximum distortion Υ on adversarial sample transferability. For now, components selected to be perturbed are increased by $\varepsilon = 1$. Intuitively, increasing the maximum distortion makes adversarial samples more transferable. Higher distortions increase the misclassification confidence of the substitute DNN, and also increases the likelihood of the oracle misclassifying the same sample. These results are reported in Figure 9. Increasing distortion Υ from 7.14% to 28.57% improves transferability: at a 7.14% distortion, the average transferability across all architectures is 14.70% whereas at a 28.57% distortion, the average transferability is at 55.53%.

We now quantify the impact of the variation ε introduced to each input component selected in $\delta_{\vec{x}}$. We find that reducing the input variation from 1 to 0.7 significantly degrades adversarial sample transferability, approximatively by a factor of 2 (cf. Figure 10). This is explained by the fixed distortion parameter Υ, which prevents the crafting algorithm from increasing the number of components altered to compensate for the reduced effectiveness yielded by the smaller ε.

Comparing Crafting Algorithms: To compare the two crafting strategies and their differing perturbation styles fairly, we compare their success rate given a fixed L1 norm of the introduced perturbation $\delta_{\vec{x}}$, which can be defined as:

$$\|\delta_{\vec{x}}\|_1 = \varepsilon \cdot \|\delta_{\vec{x}}\|_0 \tag{8}$$

where $\|\delta_{\vec{x}}\|_0$ is the number of input components selected in the perturbation $\delta_{\vec{x}}$, and ε the input variation introduced to each component perturbed. For the Goodfellow algorithm, we always have $\|\delta_{\vec{x}}\|_0 = 1$, whereas for the Papernot algorithm, values vary for both ε and $\|\delta_{\vec{x}}\|_0$. For instance, $\|\delta_{\vec{x}}\|_1 = 0.4$ corresponds to a Goodfellow algorithm with $\varepsilon = 0.4$ and a Papernot algorithm with $\varepsilon = 1$ and $\Upsilon = 40\%$. Corresponding transferability rates can be found in Table 1 and Figure 9 for our running set of architectures. Performances are comparable with some DNNs performing better with one algorithm and others with the other. Thus, the choice of algorithm depends on acceptable perturbations: e.g., all features perturbed a little vs. few features perturbed a lot. Indeed, the Goodfellow algorithm gives more control on ε while the Papernot algorithm gives more control on Υ.

[5]In [9], the algorithm stopped perturbing when the input reached the target class. Here, we force the algorithm to continue perturbing until it changed Υ input components.

7. GENERALIZATION OF THE ATTACK

So far, all substitutes and oracles considered were learned with DNNs. However, no part of the attack limits its applicability to other ML techniques. For instance, we show that the attack generalizes to non-differentiable *target oracles* like decision trees. As pointed out by Equation 4, the only limitation is placed on the *substitute*: it must model a differentiable function—to allow for synthetic data to be generated with its Jacobian matrix. We show below that:

- Substitutes can also be learned with logistic regression.
- The attack generalizes to additional ML models by: (1) learning substitutes of 4 classifier types (logistic regression, SVM, decision tree, nearest neighbors) in addition to DNNs, and (2) targeting remote models hosted by Amazon Web Services and Google Cloud Prediction with success rates of 96.19% and 88.94% after 800 queries to train the substitute.

7.1 Generalizing Substitute Learning

We here show that our approach generalizes to ML models that are not DNNs. Indeed, we learn substitutes for 4 representative types of ML classifiers in addition to DNNs: logistic regression (LR), support vector machines (SVM), de-

cision trees (DT), and nearest neighbor (kNN). All of these classifiers are trained on MNIST, with no feature engineering (i.e. directly on raw pixel values) as done in Section 5.

Whereas we previously trained all of our substitutes using DNNs only, we now use both DNNs and LR as substitute models. The Jacobian-based dataset augmentation described in the context of DNNs is easily adapted to logistic regression: the later is analog to the softmax layer frequently used by the former when outputting probability vectors. We use 100 samples from the MNIST test set as the initial substitute training set and use the two refinements introduced in Section 6: a *periodic step size* and *reservoir sampling*.

Figure 11(a) and 11(b) plot for each iteration ρ the share of samples on which the substitute DNNs and LRs agree with predictions made by the oracle they are approximating. This proportion is estimated by comparing labels assigned to the test set by the substitutes and oracles before each iteration ρ of the Jacobian-based dataset augmentation. All substitutes are able to approximate the corresponding oracle at rates higher between 77% and 83% after $\rho = 10$ iterations (to the exception of the decision tree oracle, which could be due to its non-continuity). LR substitute accuracies are generally lower than those of DNN substitutes, except when targeting the LR and SVM oracles where LR substitutes outperform DNN ones. However, LR substitutes are computationally more efficient and reach their asymptotic match rate faster, after $\rho = 3$ iterations, corresponding to 800 oracle queries.

Table 2 quantifies the impact of refinements introduced in Section 6 on results reported in Figure 11(a) and 11(b). The *periodic step size* (PSS) increases the oracle approximation accuracy of substitutes. After $\rho = 9$ epochs, a substitute DNN trained with PSS matches 89.28% of the DNN oracle labels, whereas the vanilla substitute DNN matches only 78.01%. Similarly, the LR substitute with PSS matches 84.01% of the LR oracle labels while the vanilla substitute matched 72.00%. Using *reservoir sampling* (RS) reduces oracle querying. For instance, 10 iterations with RS ($\sigma = 3$ and $\kappa = 400$) make $100 \cdot 2^3 + 400(10 - 3) = 3,600$ queries to the oracle instead of $102,400$ without RS. This decreases the substitute accuracy, but when combined with PSS it remains superior to the vanilla substitutes. For instance, the vanilla substitute matched $7,801$ of the DNN oracle labels, the PSS one $8,928$, and the PSS with RS one $8,290$. Simarly, the vanilla LR substitute matched 71.56% of the SVM oracle labels, the PSS one 82.19%, and the PSS with RS 79.20%.

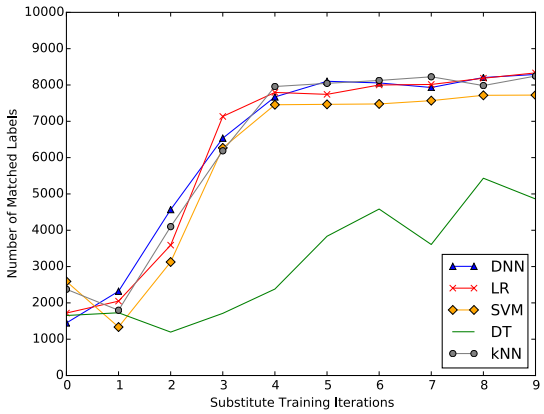
7.2 Attacks against Amazon & Google oracles

Amazon oracle: To train a classifier on *Amazon Machine Learning*,[6], we uploaded a CSV version of the MNIST dataset to a S3 bucket. We then loaded the data, selected the multi-class model type, and keept default configuration settings. The process took a few minutes and produced a classifier achieving a 92.17% test set accuracy. We cannot improve the accuracy due to the automated nature of training. We then activate real-time predictions to query the model for labels from our machine with the provided API. Although probabilities are returned, we discard them and retain *only the most likely label*—as stated in our threat model (Section 3).

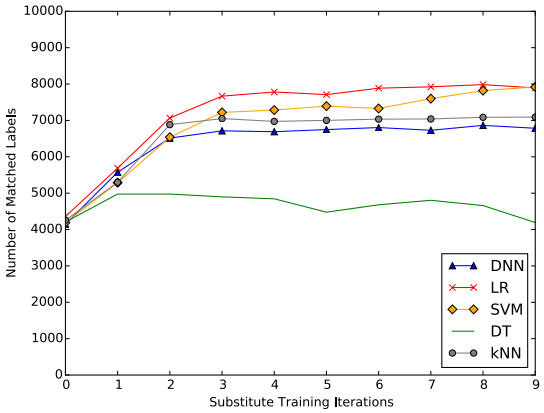
Google oracle: The procedure to train a classifier on Google's Cloud Prediction API[7] is similar to Amazon's. We

[6]https://aws.amazon.com/machine-learning

[7]https://cloud.google.com/prediction/



(a) DNN substitutes



(b) LR substitutes

Figure 11: **Label predictions matched** between the substitutes (DNN and LR) and their target oracles on test data.

Substitute	DNN	LR	SVM	DT	kNN
DNN	78.01	82.17	79.68	62.75	81.83
DNN+PSS	**89.28**	**89.16**	**83.79**	**61.10**	**85.67**
DNN+PSS+RS	82.90	83.33	77.22	48.62	82.46
LR	64.93	72.00	71.56	38.44	70.74
LR+PSS	**69.20**	**84.01**	**82.19**	**34.14**	**71.02**
LR+PSS+RS	67.85	78.94	79.20	41.93	70.92

Table 2: **Impact of our refinements**, Periodic Step Size (PSS) and Reservoir Sampling (RS), on the percentage of label predictions matched between the substitutes and their target classifiers on test data after $\rho = 9$ substitute iterations.

		Amazon		Google	
Epochs	Queries	DNN	LR	DNN	LR
$\rho = 3$	800	87.44	96.19	84.50	88.94
$\rho = 6$	6,400	**96.78**	**96.43**	**97.17**	92.05
$\rho = 6^*$	2,000	95.68	95.83	91.57	**97.72**

Table 3: **Misclassification rates (%) of the Amazon and Google oracles** on adversarial samples produced with DNN and LR substitutes after $\rho = 3, 6$ epochs. The 2nd column is the number of queries during substitute training. Last row uses a periodic step size and reservoir sampling.

upload the CSV file with the MNIST training data to Google Cloud Storage. We then train a model using the Prediction API. The only property we can specify is the expected multi-class nature of our model. We then evaluate the resulting model on the MNIST test set. The API reports an accuracy of 92% on this test set for the model trained.

Substitute Training: By augmenting an initial training set of 100 test set samples, we train a DNN and LR substitute for each of the two oracles. We measure success as the rate of adversarial samples misclassified by the corresponding oracle, among the $10,000$ produced from the test set using the fast gradient sign method with parameter $\varepsilon = 0.3$. These rates, computed after $\rho \in \{3, 6\}$ dataset augmentation iterations, are reported in Table 3. Results reported in the last row use both a periodic step size and reservoir sampling (hence the reduced number of queries made to train the substitute).

Experimental Results: With a 96.19% misclassification rate for a perturbation $\varepsilon = 0.3$ crafted using a LR substitute trained with 800 oracle queries, the model hosted by Amazon is easily misled. The model trained by Google is somewhat more robust to adversarial samples, but is still vulnerable to a large proportion of samples: 88.94% of adversarial samples produced in the same conditions are misclassified. A careful read of the documentation indicated that the model trained by Amazon is a multinomial logistic regression.[8] As pointed out in [4], shallow models like logistic regression are unable to cope with adversarial samples and learn robust classifiers. This explains why the attack is very successful and the LR substitute performs better than the DNN substitute. We were however not able to find the ML technique Google uses.

The last row of Table 3 shows how combining periodic step sizes with reservoir sampling allow us to reduce querying of both oracles during substitute training, while crafting adversarial samples with higher transferability to the target classifier. Indeed, querying is reduced by a factor larger than 3 from $6,400$ to $2,000$ queries, while misclassification decreases only from 96.78% to 95.68% for the Amazon DNN substitute. It is still larger than the rate of 87.44% achieved after 800 queries by the substitute learned without the refinements. Similarly, the misclassification rate of the Google LR substitute is 97.72%—compared to 92.05% with the original method after $\rho = 6$ epochs, confirming the result.

8. DEFENSE STRATEGIES

The two types of defense strategies are: (1) *reactive* where one seeks to detect adversarial examples, and (2) *proactive* where one makes the model itself more robust. Our attack is not more easily detectable than a classic adversarial example attack. Indeed, oracle queries may be distributed among a set of colluding users, and as such remain hard to detect. The defender may increase the attacker's cost by training models with higher input dimensionality or modeling complexity, as our experimental results indicate that these two factors increase the number of queries required to train substitutes. In the following, we thus only analyze our attack in the face of defenses that seek to make the (oracle) model robust.

Many potential defense mechanisms fall into a category we call *gradient masking*. These techniques construct a model that does not have useful gradients, e.g., by using a nearest neighbor classifier instead of a DNN. Such methods make it difficult to construct an adversarial example directly, due to the absence of a gradient, but are often still vulnerable to the adversarial examples that affect a smooth version of the same model. Previously, it has been shown that nearest neighbor was vulnerable to attacks based on transferring adversarial examples from smoothed nearest neighbors[4].

[8] docs.aws.amazon.com/machine-learning

Training ε	Attack ε	O→O	S → S	S → O
0.15	0.3	10.12%	94.91%	38.54%
0.15	0.4	43.29%	99.75%	71.25%
0.3	0.3	0.91%	93.55%	1.31%
0.3	0.4	29.56%	99.48%	10.30%

Table 4: **Evaluation of adversarial training**: the columns indicate the input variation parameter used to inject adversarial examples during training and to compute the attacks, the attack success rate when examples crafted on the (O)racle are deployed against the (O)racle, the attack success rate when examples crafted on the (S)ubstitute are deployed against the (S)ubstitute, and the attack success rate when examples crafted on the (S)ubstitute are deployed against the (O)racle.

.

We show a more general flaw in the category of gradient masking. Even if the defender attempts to prevent attacks by not publishing the directions in which the model is sensitive, these directions can be discovered by other means, in which case the same attack can still succeed. We show that the black-box attack based on transfer from a substitute model overcomes gradient masking defenses. No fully effective defense mechanism is known, but we study the two with the greatest empirical success so far: adversarial training [4, 14], and defensive distillation for DNNs [10].

Adversarial training: It was shown that injecting adversarial examples throughout training increases the robustness of significantly descriptive models, such as DNNs [4, 14, 17]. We implemented an approximation of this defense using the Google Prediction API. Since the API does not support the generation of adversarial examples at every step of training, as a correct implementation of adversarial training would do, we instead inject a large amount of adversarial examples infrequently. After training in this way, the model has a misclassification rate of 8.75% on the unperturbed test set, but the adversarial misclassification rate rises to 100% when $\rho = 6$. To evaluate this defense strategy using a correct implementation, we resort to training the oracle locally, using our own codebase that includes support for generating adversarial examples at each step. After each training batch, we compute and train on adversarial examples generated with the fast gradient sign method before starting training on the next batch of the original training data. Results are given in Table 4. We observe that for $\varepsilon = 0.15$, the defense can be evaded using the black-box attack with adversarial examples crafted on the substitute and misclassified by the oracle at rates up to 71.25%. However, for $\varepsilon = 0.3$, the black-box attack is not effective anymore. Therefore, making a machine learning model robust to small and infinitesimal perturbations of its inputs is an example of *gradient masking* and can be evaded using our substitute-based black-box approach. However, making the model robust to larger and finite perturbations prevents the black-box attack. To confirm this hypothesis, we now show that defensive distillation, which makes the model robust to infinitesimal perturbations, can be evaded by the black-box approach.

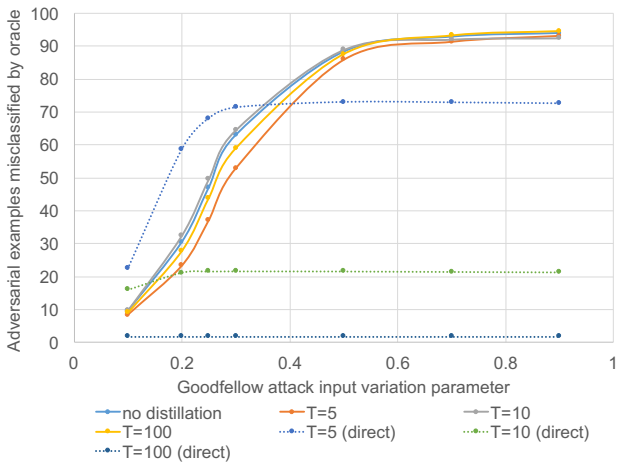


Figure 12: **Evaluation of defensive distillation:** Percentage of adversarial examples crafted using the Goodfellow algorithm at varying ε misclassified by the oracle. T is the temperature of distillation [10]. Curves marked by (direct) indicate baseline attacks computed on the oracle, all other curves where computed using a substitute, as described in Section 4. Despite distillation preventing the attack on the oracle directly, using a substitute allows us to evade it.

Defensive distillation: Due to space constraints, we refer readers to [10] for a detailed presentation of defensive distillation, which is an alternative defense. Because the remotely hosted APIs we study here do not implement defensive distillation or provide primitives that could be used to implement it, we are forced to evaluate this defense on a locally trained oracle. Therefore, we train a distilled model as described in [10] to act as our MNIST oracle.

We train several variants of the DNN architecture A at different distillation temperatures $T = 5, 10, 100$. For each of them, we measure the success of the fast gradient sign attack (i.e., the Goodfellow et al. algorithm) directly performed on the distilled oracle—as a baseline corresponding to a white-box attack—and using a substitute DNN trained with synthetic data as described throughout the present paper. The results are reported in Figure 12 for different values of the input variation parameter ε on the horizontal axis. We find that defensive distillation defends against the fast gradient sign method when the attack is performed directly on the distilled model, i.e. in *white-box settings*. However, in *black-box* settings using the attack introduced in the present paper, the fast gradient sign method is found to be successful regardless of the distillation temperature used by the oracle. We hypothesize that this is due to the way distillation defends against the attack: it reduces the gradients in local neighborhoods of training points. However, our substitute model is not distilled, and as such possesses the gradients required for the fast gradient sign method to be successful when computing adversarial examples.

Defenses which make models robust in a small neighborhood of the training manifold perform *gradient masking*: they smooth the decision surface and reduce gradients used by adversarial crafting in small neighborhoods. However, using a substitute and our black-box approach evades these defenses, as the substitute model is not trained to be robust to the said small perturbations. *We conclude that defending against finite perturbations is a more promising avenue for future work than defending against infinitesimal perturbations.*

9. CONCLUSIONS

We introduced an attack, based on a novel substitute training algorithm using synthetic data generation, to craft adversarial examples misclassified by black-box DNNs. Our work is a significant step towards relaxing strong assumptions about adversarial capabilities made by previous attacks. We assumed only that the adversary is capable of observing labels assigned by the model to inputs of its choice. We validated our attack design by targeting a remote DNN served by MetaMind, forcing it to misclassify 84.24% of our adversarial samples. We also conducted an extensive calibration of our algorithm and generalized it to other ML models by instantiating it against classifiers hosted by Amazon and Google, with success rates of 96.19% and 88.94%. Our attack evades a category of defenses, which we call *gradient masking*, previously proposed to increase resilience to adversarial examples. Finally, we provided an intuition for adversarial sample transferability across DNNs in Appendix B.

10. REFERENCES

[1] Marco Barreno, et al. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*.

[2] Battista Biggio, et al. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases*, pages 387–402. Springer, 2013.

[3] Ian Goodfellow, et al. Deep learning. Book in preparation for MIT Press (www.deeplearningbook.org), 2016.

[4] Ian J Goodfellow, et al. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations*, 2015.

[5] Ling Huang, et al. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[6] Alexey Kurakin, et al. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[7] Yann LeCun et al. The mnist database of handwritten digits, 1998.

[8] Erich L. Lehmann, et al. *Testing Statistical Hypotheses*. Springer Texts in Statistics, August 2008.

[9] Nicolas Papernot, et al. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*, 2016.

[10] Nicolas Papernot, et al. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*.

[11] Mahmood Sharif, et al. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016.

[12] Nedim Srndic, et al. Practical evasion of a learning-based classifier: A case study. In *Proceeding of the 35th IEEE Symposium on Security and Privacy*.

[13] Johannes Stallkamp, et al. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332, 2012.

[14] Christian Szegedy, et al. Intriguing properties of neural networks. In *Proceedings of the International Conference on Learning Representations*, 2014.

[15] Florian Tramèr, et al. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium*, 2016.

[16] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 1985.

[17] D Warde-Farley, et al. Adversarial perturbations of deep neural networks. *Advanced Structured Prediction*, 2016.

[18] Weilin Xu, et al. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*.

11. ACKNOWLEDGMENTS

Nicolas Papernot is supported by a Google PhD Fellowship in Security. Research was also supported in part by the Army Research Laboratory, under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA), and the Army Research Office under grant W911NF-13-1-0421. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

A. DNN architectures

Figure 13 provides the specific DNN architectures used throughout Sections 5, 6, and 11. The first column is the identifier used in the paper to refer to the architecture. The second and third columns respectively indicate the input and output dimensionality of the model. Finally, each additional column corresponds to a layer of the neural network.

ID	In	Out	CM	CM	RL	RL	RL	S
A	784	10	32	64	200	200	-	10
B	3072	43	64	128	256	256	-	43
C	3072	43	32	64	200	200	-	43
D	3072	43	32	64	200	200	-	43
E	3072	43	64	64	200	200	100	43
F	784	10	32	64	200	-	-	10
G	784	10	32	64	-	-	-	10
H	784	10	32	-	200	200	-	10
I	784	10	-	-	200	200	200	10
J	784	10	-	-	1000	200	-	10
K	784	10	-	-	1000	500	200	10
L	784	10	32	-	1000	200	-	10
M	784	10	32	-	-	200s	200s	10

Figure 13: **DNN architectures:** ID: reference used in the paper, In: input dimension, Out: output dimension, CM: convolutional layer with 2x2 kernels followed by max-pooling with kernel 2x2, RL: rectified linear layer except for 200*s* where sigmoid units are used, S: softmax layer.

B. Intuition behind Transferability

Previous work started explaining why adversarial samples transfer between different architectures [4, 14]. Here, we build an intuition behind transferability based on *statistical hypothesis testing* [8] and an analysis of DNN cost gradient sign matrices. A formal treatment is left as future work.

Recall the perturbation in the Goodfellow algorithm. Inspecting Equation 5, it is clear that, given a sample $\vec{x}$, the noise added would be the same for two DNNs F and G if $\text{sgn}(\nabla_{\vec{x}} cost(F, \vec{x}, y))$ and $\text{sgn}(\nabla_{\vec{x}} cost(G, \vec{x}, y))$ were equal. These matrices have entries in $\{+1, -1\}$. Let us write the space of these matrices as $Sgn_{n\times m}$. Assume that the samples $\vec{x}$ are generated from a population distribution $\mathcal{D}$ (e.g., in our case the distribution from which the images of digits are drawn). The formula $\text{sgn}(\nabla_{\vec{x}} cost(F, \vec{x}, y))$ and $\mathcal{D}$ induce a distribution $\mathcal{D}_F$ over $Sgn_{n\times m}$ (i.e. randomly draw a sample from the distribution $\mathcal{D}$ and compute the quantity). Similarly, DNN G and distribution $\mathcal{D}$ induce a distribution $\mathcal{D}_G$ over $Sgn_{n\times m}$. Our main conjecture is:

> For two "similar" architectures F and G distributions $\mathcal{D}_F$ and $\mathcal{D}_G$ induced by a population distribution $\mathcal{D}$ are highly correlated.

If distributions $\mathcal{D}_F$ and $\mathcal{D}_G$ were independent, then the noise they add during adversarial sample crafting are independent. In this case, our intuition is that adversarial samples would not transfer (in the two cases you are adding noise that are independent). The question is: how to verify our conjecture despite the population distribution $\mathcal{D}$ being unknown?

We turn to statistical hypothesis testing. We can empirically estimate the distributions $\mathcal{D}_F$ and $\mathcal{D}_G$ based on known samples. First, we generate two sequences of sign matrices $\sigma_1 = \langle M_1, M_2, \cdots \rangle$ and $\sigma_2 = \langle N_1, N_2, \cdots \rangle$ using the sample set (e.g. MNIST) for a substitute DNN F and oracle G. Next we pose the following *null hypothesis*:

> H_N: The sequences σ_1 and σ_2 are drawn from independent distributions.

We use standard tests from the statistical hypothesis testing literature to test the hypothesis H_N. If the hypothesis H_N is *rejected*, then we know that the sign matrices corresponding to the two architectures F and G are correlated.

We describe the test we use. There are several algorithms for hypothesis testing: we picked a simple one based on a chi-square test. An investigation of other hypothesis-testing techniques is left as future work. Let $p_{i,j}$ and $q_{i,j}$ be the frequency of $+1$ in the (i, j)-th entry of matrices in sequences σ_1 and σ_2, respectively. Let $r_{i,j}$ be the frequency of the (i, j)-th entry being $+1$ in both sequences σ_1 and σ_2 simultaneosuly.[9] Note that if the distributions were independent then $r_{i,j} = p_{i,j}q_{i,j}$. However, if the distributions are correlated, then we expect $r_{i,j} \neq p_{i,j}q_{i,j}$. Consider quantity:

$$\chi^{2\star} = \sum_{i=1}^{m}\sum_{j=1}^{n} \frac{(r_{i,j}N - p_{i,j}q_{i,j}N)^2}{p_{i,j}q_{i,j}N}$$

where N is the number of samples. In the χ-square test, we compute the probability that $P(\chi^2 > \chi^{2\star})$, where χ^2 has degrees of freedom $(m-1)(n-1) = 27 \times 27 = 729$ for the MNIST data. The $\chi^{2\star}$ scores for substitute DNNs from Table 1 range between $61,403$ for DNN A and $88,813$ for DNN G. Corresponding P-values are below 10^{-5} for all architectures, with confidence $p < 0.01$. Thus, for all substitute DNNs, the hypothesis H_N is largely rejected: sequences σ_1 ans σ_2, and therefore sign matrices corresponding to pairs of a substitute DNN and the oracle, are highly correlated. As a baseline comparison, we generate 2 random sign matrices and compute the corresponding χ^{2*} score: 596. We find a P-Value of 0.99 with a confidence of 0.01, meaning that these matrices were indeed drawn from independent distribution.

However, we must now complete our analysis to characterize the correlation suggested by the hypothesis testing. In Figure 14, we plot the frequency matrix $R = [r_{i,j}]$ for several pairs of matrices. The first is a pair of random matrices of $\{+1, -1\}$. The other matrices correspond to substitute DNN A and the oracle at different substitute training epochs ρ. Frequencies are computed using the $10,000$ samples of the MNIST test set. Although all frequencies in the random pairs are very close to $1/2$, frequencies corresponding

[9]We assume that the frequencies are normalized so they can be interprested as probabilities, and also assume that all frequencies are > 0 to avoid division by zero, which can be achieved by rescaling.

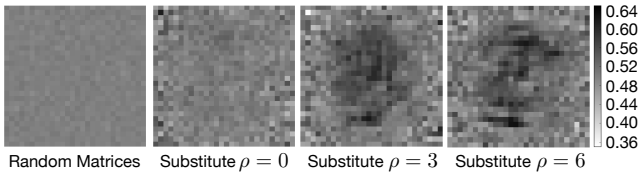


Figure 14: **Frequencies of cost gradient sign matrix components equal between substitute A and the oracle** at substitute training epochs $\rho \in \{0, 3, 6\}$ (three on the right), compared to a pair of random sign matrices (first image).

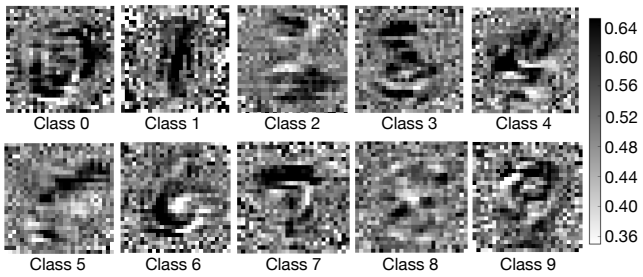


Figure 15: **Frequencies of cost gradient sign matrix components equal between substitute A and the oracle**

to pixels located in the center of the image are higher in the $(substitute, oracle)$ matrix pairs. The phenomenon amplifies as training progresses through the substitute epochs. We then compute the frequencies separately for each sample source class in Figure 15. Sign matrices agree on pixels relevant for classification in each class. We plotted similar figures for other substitute DNNs. They are not included due to space constraints. They show that substitutes yielding lower transferability also have less components of their cost gradient sign matrix frequently equal to the oracle's. This suggests that *correlations between the respective sign matrices of the substitute DNN and of the oracle—for input components that are relevant to classification in each respective class—could explain cross-model adversarial sample transferability.*

C. Discussion of Related Work

Evasion attacks against classifiers were discussed previously. Here, we cover below black-box attacks in more details.

Xu et al. applied a genetic algorithm to evade malware detection [18]. Unlike ours, it accesses probabilities assigned by the classifier to compute genetic variants fitness. These can be concealed by defenders. The attack is also not very efficient: 500 evading variants are found in 6 days. As the classifier is queried heavily, the authors conclude that the attack cannot be used against remote targets. Finally, given the attack's high cost on low-dimensional random forests and SVMs, it is unlikely the approach would scale to DNNs.

Srndic et al. explored the strategy of training a substitute model to find evading inputs [12]. They do so using labeled data, which is expensive to collect, especially for models like DNNs. In fact, their attack is evaluated only on random forests and an SVM. Furthermore, they exploit a semantic gap between the specific classifiers studied and PDF renderers, which prevents their attack from being applicable to models that do not create such a semantic gap. Finally, they assume knowledge of hand-engineered high-level features whereas we perform attacks on raw inputs.

Tramer et al. considered an adversarial goal different from ours: the one of extracting the exact value of each model parameter. Using partial knowledge of models and equation solving, they demonstrated how an adversary may recover parameters from classifiers hosted by BigML and Amazon [15]. However, it would be difficult to scale up the approach to DNNs in practice. To recover the $2,225$ parameters of a shallow neural network (one hidden layer with 20 neurons) trained on a local machine, they make $108,200$ label queries. Instead, we make $2,000$ label queries to train substitute DNNs made up of 8 hidden layers (each with hundreds of neurons) with a total of over $100,000$ parameters—albeit at the expense of a reduced guaranteed accuracy for the model extraction operation. Unlike theirs, our work also shows that our substitutes enable the adversary to craft adversarial examples that are likely to mislead the remote classifier.