Server-Based Manipulation Attacks Against Machine Learning Models

Cong Liao
College of Information Sciences and Technology
Pennsylvania State University
cxl491@psu.edu

Haoti Zhong
Department of Electrical Engineering
Pennsylvania State University
hzz133@psu.edu

Sencun Zhu
Department of Computer Science and Engineering
Pennsylvania State University
szhu@cse.psu.edu

Anna Squicciarini
College of Information Sciences and Technology
Pennsylvania State University
acs20@psu.edu

ABSTRACT

Machine learning approaches have been increasingly applied to various applications for data analytics (e.g. spam filtering, image classification). Further, with the growing adoption of cloud computing, various cloud services have provided an efficient way for users to train, store or deploy machine learning algorithms in an easy-to-use manner. However, the models deployed in the cloud may be exposed to potential malicious attacks launched at the server side. Attackers with access to the server can stealthily manipulate a machine learning model so as to enable misclassification or introduce bias. In this work, we study the problem of manipulation attacks as they occur at the server side. We consider not only traditional supervised learning models but also state-of-the-art deep learning models. In particular, a simple but effective gradient descent based approach is presented to exploit Logistic Regression (LR) and Convolutional Neural Networks (CNN)[16] models. We evaluate manipulation attacks against machine learning or deep learning systems using both Enron email text and MINIST image dataset[17]. Experimental results have demonstrated such attacks can manipulate the model that allows malicious samples to evade detection easily without compromising the overall performance of the systems.

CCS CONCEPTS

• **Computing methodologies → Neural networks**; • **Security and privacy** → *Domain-specific security and privacy architectures*;

KEYWORDS

Convolutional Neural Networks; Model Manipulation; Adversarial Machine Learning

CODASPY '18, March 19–21, 2018, Tempe, AZ, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5632-9/18/03…$15.00
https://doi.org/10.1145/3176258.3176321

ACM Reference Format:
Cong Liao, Haoti Zhong, Sencun Zhu, and Anna Squicciarini. 2018. Server-Based Manipulation Attacks Against Machine Learning Models. In *Proceedings of Eighth ACM Conference on Data and Application Security and Privacy (CODASPY '18).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3176258.3176321

1 INTRODUCTION

Machine learning approaches have been successfully adopted to address various applications for data analytics on large datasets (e.g. spam filtering, image classification). In parallel, with the growing adoption of cloud computing, cloud services have increasingly offered online services to train, store or deploy machine learning models in a simple-to-use manner. For example, Microsoft Azure provides a full suite of machine learning cloud-based services that enable users to train, test, deploy or even share analytic models for classification tasks [2]. The online application of Azure ML studio [20] can fulfill users' need of building a machine learning model in an iterative cycle of uploading data, refining data, defining features, experimenting a learning algorithm, evaluating the resulting learning model and improving the model by updating the feature selection again. Once a model is finalized, users can directly deploy the model as an online web service that can provide predictive analytic solutions for other applications. Similarly, Google also provides cloud-based tensorflow service [12] that allows users to download/upload or store machine learning models at the server, and even deploy if necessary.

When machine learning models are deployed and used for security sensitive applications such as spam filtering, intrusion detection or malware detection, the robustness of the model is of great importance [3]. However, due to the probabilistic nature of these predictive models, they may be vulnerable to well crafted malicious input in an adversarial environment. Researchers from both security and machine learning communities have investigated the vulnerabilities exposed by various types of attacks, e.g., evasion attack [3] and poisoning attack [4], against machine learning models [25]. These types of attacks usually start with manipulating the input samples by adding certain noises or obfuscating features to baffle the model into misclassifying the malicious samples. Another unique type of attack targets the online machine learning service and is instantiated by querying the service multiple times in order to infer the model and its parameters used by the service [10, 11, 30].

Intuitively, models deployed in the cloud are easy to manipulate through these attacks, as they are beyond the direct control of the service requesters (i.e., the end users). In particular, attacks that directly manipulate the models are more straightforward and efficient as they do not require to modify the input samples as evasion attacks do.

In this paper, we explore a new type of attack under an unexplored adversarial scenario. Instead of crafting a malicious sample in a typical setting of adversarial machine learning, e.g., evasion attack, an intelligent attacker with access to the server can choose to stealthily manipulate a machine learning model, so as to enable misclassification or introduce bias regardless of the test set. This will achieve a similar effect of causing the adversarial sample to evade detection. Here, the adversary is assumed having access to the machine learning model in the cloud, and therefore is capable of manipulating the model directly to cause certain targeted samples to be misclassified. This attack is potentially much more dangerous and effective than classic adversarial models. First, for this attack to be successful we do not need to make assumptions regarding whether the targeted samples are well crafted or not. The proposed manipulation attack directly targets the learning model by modifying the model parameters using a simple but effective gradient descent based approach. Second, the manipulated model adapts to the targeted samples so as to misclassify these samples into the labels specified by the adversary. This attack is demonstrated on two well-known machine learning models, Linear Logistic Regression, and the increasingly popular Convolutional Neural Network models.

We extensively evaluate our suggested attack on two common types of data type - text and image -, representing a binary classification and a multi-class problem, respectively. Our results demonstrate the effectiveness of the proposed attack strategy at the cost of reasonable accuracy loss when the number of target samples is limited. In particular, we show the potential of this attack for convolutional neural networks in the case of multi-class models: manipulation can be successful with no significant impact to the overall model accuracy. Our contributions are summarized as follows.

(1) We introduce a new type of adversarial scenario where machine learning models are subject to potential threats posed by an adversary who has access to the model at the server side, compared to common attacks in the typical setting of adversarial machine learning.
(2) We extensively evaluate the proposed manipulation attack on two representative types of machine learning models, i.e., a linear model Logistic Regression (LR) and a deep learning model Convolutional Neural Networks (CNN), using two general types of dataset, i.e., texts and images.

The rest of the paper is organized as follows. In section 2, we describe our proposed scenario and attack in detail. In section 3, we summarize state-of-the-art attacks against machine learning models, and discuss the similarities and differences compared to ours. In Section 4, we provide the modeling of adversary, and explain the attack strategy against two types of machine learning models in Section 5 and Section 6 respectively. In Section 7, we conduct experiments to evaluate the proposed attacks, and present our findings in Section 8. Finally, we discuss future works in Section 9.

2 PROBLEM STATEMENT

We consider the scenario of an outsourced machine learning service where a machine learning model is trained and deployed in the cloud as shown in Figure 1. The model is subject to the threats posed by those who have the ability to control how the model is trained and used. This can either come from insiders who have managerial roles or outsiders who gain access privilege by exploiting vulnerabilities at the server side. To make matters worse, it also can be a collusion between the two parties. A manipulated model can fail to capture some target samples for the sake of certain malicious intent, e.g., a spam successfully bypassing a spam filter.

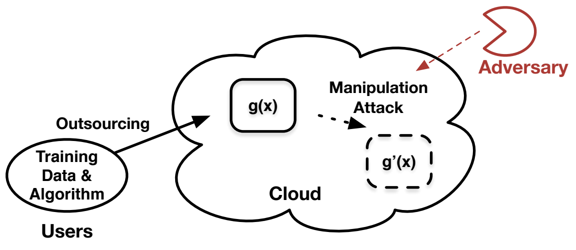


Figure 1: A scenario of outsourced machine learning

Specifically, we investigate how a machine learning model can be stealthily manipulated to intentionally allow certain samples to be misclassified or evade detection. As mentioned, for this attack to be successful, the adversary (e.g. the cloud service performing the model training) must have access to the model stored and deployed at the server. Since model training is outsourced to the server, the adversary should be capable of affecting the training process by tampering the training data or the computational process. For instance, swapping the labels of instances between two different classes is very likely to produce a misleading model. However, this type of tampering carried in a large scale (for a large number of samples) is easily noticeable. Small scale tampering such as flipping the labels of a few instances instead might go unnoticed, but it does not guarantee the effectiveness of the attack, i.e., causing inconsistent influence over the trained model. Therefore, in order to be stealthy, we study the case of a strategic adversary which mainly intends to manipulate the model directly.

This attack strategy is also supported by the observation that although users provide training data for the server to train a machine learning model, it is difficult for the server to influence the training process to accommodate a given spam sample. Hence, it is reasonable to assume that the adversary is interested in directly manipulating the parameters of the model. Additionally, for the attack to be successful, manipulation of the model parameters should be bounded (to avoid drastic degradation of accuracy).

The machine learning model we study mainly refers to classification model in the domain of supervised learning, e.g., Logistic Regression (LR), as well as that in the area of deep learning such as Convolutional Neural Networks (CNN).

3 BACKGROUND AND RELATED WORK

To date, much work has been carried out in the realm of adversarial machine learning. We summarize the main lines of work and our relationship with these efforts in the remaining of this section.

3.1 Attacks

The machine learning systems targeted by the attacks in adversarial settings include both traditional machine learning models, both supervised and unsupervised, as well as state-of-the-art deep learning or neural network systems.

Support vector machine (SVM)[7], a supervised learning algorithm, has been extensively evaluated against both evasion and poisoning attacks, showing some important vulnerabilities of these models. Some recent studies have focused on application specific machine learning systems such as spam filtering, biometric, intrusion detection, face recognition systems that are either based on SVM or other domain specific classification algorithm [8, 9, 23]. There is also an increasing number of studies that investigate attacks that exploit vulnerabilities of deep learning or neural network systems [26], which is loosely related to our approach, as discussed in the next sections. Researchers even explore a more practical black-box attack against a deep neural networks system deployed in real world scenario [25].

Evasion Attack. Evasion attacks attempt to bypass a deployed machine learning model [3] at the test time. Given a learning model, attackers derive a malicious sample that is fed to the model, and observe the outcome to check whether the evasion is successful or not. If unsuccessful, attackers can generate new samples based on the previous result. The process will continue repeatedly until the evasion is achieved.

In such a scenario, the attacker's goal is to craft a sample that can cause the model to misclassify selected samples [14, 18, 21]. Typically, the attacker is assumed to have certain knowledge of the learning algorithm and data including a part of the training data, feature space or feature representation, the model itself and its feedback. Moreover, the attacker is capable of modifying the data sample and features.

Poisoning Attack. Poisoning attacks mainly focus on compromising the training data in order to further influence the final learning result [4, 22, 28]. Regular training data are tainted with data that come from a malicious source by attackers. The poisoned training data are used to train a model which may fail to capture certain malicious samples later.

In the adversarial setting, the attacker's goal is to generate data samples whose addition will affect the performance of the model to be trained. Generally, the attacker is assumed to have knowledge of the training samples, feature representation, the algorithm and its model parameters. In addition, the attacker is able to add data points that will result in modifying the data distribution, and alter the feature values of samples.

Inversion Attack. Inversion attacks target machine-learning-as-a-service systems by querying the online service in order to infer the model (i.e. the model parameters) used by the system [10, 11, 30]. In this case, the attacker is assumed to know the algorithm itself and be able to query the APIs provided by the online machine learning service multiple times, and receive feedback containing the classification scores. With such knowledge, the goal is to infer the parameters of the machine learning model used for the online service. Inversion attacks against online machine learning services have targeted services that are built upon decision trees, logistic regression, SVM.

3.2 Relationship with Adversarial Machine Learning

On the one hand, we bear a similar goal as some of the traditional adversarial machine learning problems, i.e., evading correct classification of selected model samples. On the other hand, we consider a new type of attack, wherein the adversary has a different perspective and insight of the model. In our case, the adversary targets the machine learning model itself. The adversary is assumed to be able to directly manipulate the parameters of the model to achieve the evading effect. Compared to the attacks discussed in the previous section, we differ as follows. In the case of an evasion attack, the training process of a model is assumed intact, but the adversary manipulates individual samples. The attacker crafts a malicious sample by modifying its feature values so that it can evade the detection of a trained model at the test time. With regard to poisoning attack, although similar in the sense that the adversary can influence the training process, our attack method mainly chooses to target the model directly but with the aim of minimally affecting the overall performance of the model. In contrast, traditional poisoning attack methods inevitably affect the model performance by compromising the training data. Finally, unlike the evasion attack, which crafts one malicious sample at a time, we tend to accommodate multiple samples at the same time - by affecting the model parameters, and still achieve a reasonable model performance.

4 ADVERSARY MODEL

In this section, we describe the attacker's goals, knowledge, and strategies.

4.1 Notations

A classification problem can be simply noted as a function $y = f(x)$, i.e., a given sample x is assigned to a particular label y that comes from a collection of predefined classes Y. A sample x is represented in certain feature space $x \in X$. As to Y, the possible labels may be a finite number k. In the simplest case, i.e., binary classification, $Y = \{0, 1\}$. For example, if we consider the typical problem of classifying emails or messages as spam or not spam, 0 refers to the legitimate class and 1 indicates the spam class.

In general, the assignment of a label y is yielded by comparing the output of model $g(x)$ against a certain threshold. For instance, if we choose a threshold of 0.5 , $y = 0$ if $g(x) = p(f(x) = 0|x) > 0.5$, and otherwise $y = 1$ if $g(x) = p(f(x) = 0|x) < 0.5$.

4.2 Knowledge

In the scenario of an outsourced computation to train a machine learning model, we assume users provide training data and a selected machine learning algorithm for the server to conduct the training process.[1] For instance, when interacting with machine

[1]This is the most common setting of current cloud services.

learning services provided by Microsoft Azure, users will be instructed to go through the process of uploading their data, preprocessing, defining features and applying a learning algorithm of user's choice [20]. Hence, the adversary is assumed aware of the training set, feature representation of the training data and specifics of the machine learning algorithm. Once a model is generated after the training process, it can be either stored in the cloud service or downloaded locally. Users can easily upload the model and iterate the same procedure to refine it or deploy the model as a web service directly. Therefore, the adversary also has access to the trained model, i.e., the parameters of the model. However, the server does not have access to the testing data, which is typically kept by users to evaluate the performance of the trained model. We assume that users will not be able to precisely verify the trained model by themselves (for the lack of local resources) or by another service provider (for the short of budget).

4.3 Goal

In general, the adversary's goal is to manipulate a machine learning model in a way that a given sample can evade the classifier or be misclassified. For simplicity, we start with considering the case of spam filtering modeled through a Logistic Regression (LR) algorithm, which could be generalized as an attack towards a linear model. The attacker aims to find a model $g(x)$ similar to the original model $g(x)'$ such that a spam sample x can obtain an estimated posterior probability $p(f(x) = 0|x)$ greater than the threshold 0.5, and therefore be labeled as non-spam. In order to avoid detection, the attacker aims to maintain a satisfactory performance of the model accuracy, such that it is non-trivial to distinguish between $g(x)$ and $g(x)'$. This goal can be expressed in terms of a loss function that the adversary intends to minimize. In this case, the attacker's goal is to produce a new model $g(x)'$ which has similar weights to the original model $g(x)$ that minimizes the loss of $g(x)'$'s prediction given certain samples. This minimization function is further subject to the constraint that model accuracy is preserved within reason, i.e., the two models (original and compromised model) are as close as possible.

5 ATTACK STRATEGY IN THE CASE OF A LINEAR MODEL

A näive attacker strategy could consist of adding the target instances with the desired labels to the training set before the model is trained as expected[2]. However, such a strategy has two limitations, with one being that the server may not know which instances need to be manipulated during the training phase for the attack to succeed. The other issue is that flipping the labels in the training set does not guarantee the model will produce the desired output for two reasons: 1) the training accuracy is not 100%, and 2) the manipulated instances are more likely to be treated as outliers for the model.

Given the above considerations, our proposed attack strategy is to modify the trained model $g(x)$ so as to predict certain target instances with the desired labels, and yet minimize the accuracy loss.

[2]This is similar to the poisoning attack, though the attacker's goal in the case of the poisoning attack is different in that it aims to affect performance

We provide the detailed attack strategy toward a LR model. Assume a given LR model, per the following equation.

$$g(\mathbf{x}) = \frac{1}{1 + \exp^{\mathbf{W}^T \mathbf{x} + w_0}}, \quad (1)$$

The target function we would like to minimize is presented as follows.

$$loss = -\sum_{\mathbf{x_i} \in D_A} \sum_{c \in C} y_c * log(g(\mathbf{x_i})) + \alpha * ||\mathbf{W_{old}} - \mathbf{W_{new}}||^2, \quad (2)$$

Here, *loss* represents a linear combination of the model's cross-entropy loss and the quadratic distance between the old and the new model's parameters. Specifically, for the cross-entropy, C is the set of possible classes, and y_c is the indicator function which equals 1 when c is equal to the target class number. D_A is the dataset which contains all the malicious target instances, $\mathbf{x_i}$ is the feature representation of an instance i, $\mathbf{W_{old}}$ is the parameter of the model before manipulation, $\mathbf{W_{new}}$ is the parameter of the current modified model, α is the regularization coefficient which constrains the model's weights from changing too drastically.

5.1 Algorithm

Our goal is to reduce the loss function until the model is able to produce the intended classification outcomes for the target samples. In the case of LR, we use gradient descent to optimize Equation 2. Gradient descent is motivated by the fact that given a differentiable function F and a point p, the value of function F will decrease the fastest along the direction of the negative gradient of the function F at that point. This method is widely applied in finding the minimum of a function. The algorithm takes steps proportional to the negative of the gradient of the loss function at the current weight point to find a local minimum of the function. In our case, the partial gradient of loss function in Equation 2 is:

$$\nabla_{W_{new}} loss = -\sum_{i \in D_A} \frac{y_c}{g(\mathbf{x_i})} \nabla_{W_{new}} g(\mathbf{x_i}) - 2\alpha * (\mathbf{W_{old}} - \mathbf{W_{new}}), \quad (3)$$

We update W_{new} with Algorithm 1 with a relatively small learning rate, denoted as β. Learning rate is used to control how fast the weight is updated. The iterative optimization process stops once the model produces the desired output to all target instances. The process of attacking a LR model is illustrated in Algorithm 1. As shown, in the algorithm, we decrease α, the constraint coefficient if labels are not changed to the target class and 10000 iterations are completed. Note that we decrease α dynamically to slowly relax our main constraint, in case we cannot find a solution for the optimization problem after multiple iterations. Empirically, we relax the constraint coefficient α by a factor of 0.9.

Algorithm 1: Attack to a LR Model

```
**Data:** $D_A$ - sample set needed for manipulate
**INPUT:** $W_{old}$ - weights of the original model, $\alpha_{init}$ - initial value of $\alpha$, $\beta$ - learning rate, $C$ - target label
**OUTPUT:** $W_{new}$ - weights of the manipulated model
$W_{new} = W_{old}$;
$\alpha = \alpha_{init}$;
$iter = 0$;
while $\exists g(x_i) \neq C$ do
    $W_{new} = W_{new} - \beta * \nabla_{W_{new}} loss$;
    $iter{+}{+}$;
    if $iter == 10000$ then
        $\alpha = \alpha * 0.9$;
        set $iter$ to 0;
    end
end
```

6 ATTACK TO DEEP LEARNING MODELS

We further investigate the potential of the attacker's success in the context of a deep learning model – Convolutional Neural Networks (CNN) [16]. For clarity of presentation, we start by introducing the concept of multilayer perceptron (MLP) that a CNN is built upon, and then move on to the description of CNN.

6.1 MLP and CNN

MLP can be considered as a neural network that consists of many nodes known as "neurons". Neurons form layers where each neuron is fully connected with the neurons in the adjacent layer. A simple 3-layer MLP is shown in Figure 2

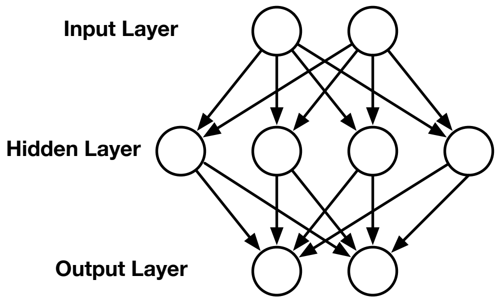


Figure 2: A 3-layer MLP

A CNN has three types of layers, i.e, input layer, hidden layer and output layer. A neuron in a hidden lay represents a computing unit associated with parameter weight w, intercept b and a non-linear activation function f. More formally, a layer can be denoted as:

$$x_i = f_i(W_i * x_{i-1} + b_i) \tag{4}$$

where x_i is the output of ith layer, x_{i-1} is the output of (i-1)-th layer, W_i and b_i are the parameters of ith layer. To train a MLP, the set of parameters $\{W, B\}$ can be learned using stochastic gradient descent with mini batches [5]. The partial derivatives in each iteration that updates the $\{W, B\}$ over a mini batch of the entire training data are computed using the back-propagation algorithm [27].

Deep Learning Networks (DL) refer to architectures which have more than 2 hidden layers (e.g., traditional MLPs). The model we investigate here is the Convolutional Neural Network (CNN)[16], which is the standard benchmark designed for image classification and object detection tasks. One peculiar feature of CNN is that certain layers are not fully connected. Those layers are typically referred to as convolutional layers. Usually, a neuron in a convolutional layer is connected to a small region of neurons in the previous layer. The small region is known as a local receptive field. It acts as a filter that slides across the input with a certain stride size. A convolutional layer can have a certain depth with each corresponding to certain filters used to represent different features in the input. Moreover, another type of layer named pooling layer is periodically inserted between convolutional layers to downsample the spatial size of input as a way to reduce the amount of parameters in CNN. Additionally, a CNN usually consists of a fully connected layer, the same as that in MLP at the end.

6.2 Attack Strategy

Since different layers have their own regularization terms, it is hard to choose proper regularization coefficients for every layer in the network. In our approach, we adapt a transfer learning approach[29], which keeps the weights of the initial layers in the model frozen while retraining only the fully-connected layer using the gradient descent. We further add a constraint on the fully-connected layer. This constraint is presented in Equation 5, and it tries to keep the weights from changing drastically.

$$\begin{aligned} loss = &- \sum_{i \in D_A} \sum_{c \in C} y_c * log(g(\mathbf{x_i}))+ \\ &\alpha_0 * ||\mathbf{Wo_{old}} - \mathbf{Wo_{new}}||^2 + \alpha_1 * ||\mathbf{Wh_{old}} - \mathbf{Wh_{new}}||^2 \end{aligned} \tag{5}$$

In this equation, $\mathbf{Wo_{old}}$ is the parameter of the output layer of the model before modification, $\mathbf{Wo_{new}}$ is the parameter of the output layer of the current modified model, $\mathbf{Wh_{old}}$ is the parameter of the hidden layer of the model before modification, $\mathbf{Wh_{new}}$ is the parameter of the hidden layer of the current modified model, α_0 and α_1 are the regularization coefficients which limit the model's weights from changing too much. Back propagation (shown in Appendix A) is used to update the CNN.

The process of attacking a CNN model is summarized in Algorithm 2. Similar to Algorithm 1, we decrease the constraint coefficient if not all the labels of the target instances are changed and 10000 iterations are completed. Empirically, we choose to relax the constraint coefficient α by a factor of 0.9.

7 EXPERIMENTAL ANALYSIS

In this section, we report experimental results to demonstrate the effectiveness of the proposed manipulation attack against LR and CNN. We first introduce the dataset used for our experiments. Next,

Algorithm 2: Attack to a CNN Model

```
Data: $D_A$ - sample set to be manipulated
INPUT: $W_{old}$ - weights of the original CNN
OUTPUT: $W_{new}$ - weights of the manipulated CNN
freeze all layers' parameters except fully connected layer and
output layer;
$iter = 0$;
do
    Compute loss per Eq. 5 at the output layer;
    Update weights based on back propagation shown in
    Appendix A;
    iter++;
    if $iter == 10000$ then
        $\alpha = \alpha * 0.9$;
        $iter = 0$;
    end
while not all target instances are classified as intended;
```

we describe the basic experimental settings. Finally, we discuss various attack scenarios and report our findings.

7.1 Datasets

For our experiments, we use two datasets, a textual dataset and an image dataset. The text dataset is the well-known Enron Spam Emails dataset [19]. The image dataset is the MNIST Handwritten Digits dataset [17]. The Enron-Spam dataset consists of ham (non-spam) messages from six individual users selected from the Enron Corpus [15], and spam messages drawn from three different sources. In total, the dataset includes over $16,000$ hams and $17,000$ spams. Each email sample is converted into a word vector based on the bag-of-words model, and the feature value uses binary representation, i.e., presence (1) or absence (0).

The MNIST dataset consists of various handwritten digits from 0 to 9. Each image has the size of 28x28 pixels, with each pixel value ranging from 0 (white) to 255 (black). The dataset has a training set of $60,000$ instances and a test set of $10,000$ instances. Each image instance is represented by 784 pixels in total as its features.

7.2 Settings

For the Enron-Spam dataset, we randomly select 80% of emails from both categories as the training set, and the remaining samples are used as the test set. As to the MNIST dataset, we use the default training set and test set. The attack procedures shown in Algorithm 1 and Algorithm 2 are implemented based on Tensorflow [1] library in Python. The configurations used to test the attacks for LR and CNN are presented in Table 1 and Table 2, respectively.

7.3 Results

We describe various scenarios where the attack strategy against LR and CNN are evaluated, and discuss the corresponding experimental results.

7.3.1 A Naïve Approach. A simple approach to carry a manipulation attack consists of the adversary adding target samples in the training set and marking them with the target label, e.g. the adversary could add target spams and label them as non-spam, before the model is trained. The underlying objective is that spams or similar spams will be misclassified by the model, and still avoid detection. We argued in the beginning of Section 5 that such tampering is not effective in attacking a model compared with direct manipulation. Hence, we conduct the following experiment to validate our hypothesis.

To simulate this naïve approach, we randomly select n samples at a time from one class and label them as a different class. For the Enron-Spam dataset, samples are chosen from the spam class and labeled as legitimate. In the case of the MNIST dataset, samples are drawn from a random class and marked as a new class other than the original one. We then train a model (either the LR or CNN) with the modified training data. Lastly, we check whether all the n selected samples are successfully misclassified by the trained model or not. n ranges from 1 to 5. For each n, we repeat such procedure 100 times and measure the average success rate.

As we can see in Table 3, the success rate is 0% in most cases, indicating that this baseline approach would not affect the trained

Configuration	Value
Learning rate (training)	0.01
Training epoch	25
Batch size	100
Learning rate (attack)	0.001

Table 1: Configuration of tested LR models

Configuration	Value
# of convolutional layers	2
# of max pooling layers	2
# of fully connected layers	1
Input size	784
Class size	10
Batch size	128
Stride	1
Max pooling filter size	2x2
Dropout	0.75
Learning rate (training)	0.001
Training iteration	200000
Learning rate (Attack)	0.000001

Table 2: Configuration of tested CNN models

	Average Success Rate		
# of Target Samples	MNIST CNN	MNIST LR	Enron-Spam LR
1	0%	2%	5%
2	0%	1%	0%
3	0%	0%	0%
4	0%	0%	0%
5	0%	0%	0%

Table 3: Attacker success rate

model at all. In other words, tampering the labels of a few target samples is unlikely to lead to successful misclassification of all those samples, and the trained model is robust enough to correct such minor changes in the training data. Therefore, a more strategic attack for manipulating the trained model is necessary.

7.3.2 Manipulation Attack with Enron-Spam. Since CNN is designed specifically for image classification, we evaluate the proposed attack against the LR model solely against the Enron-Spam dataset. In this experiment, an LR model is already trained using the original Enron-Spam dataset. We demonstrate how we can manipulate a trained LR model causing certain number of samples to be misclassified as a different class.

In the experiment, we begin by randomly choosing n samples from the spam class as the target instances. Given the trained model and selected samples, we further apply the attack approach illustrated in Algorithm 1. Lastly, the performance of the newly generated model is evaluated against the test set. n varies from 1 to 5 and such process is repeated 100 times. Each time, we compare the accuracy of the new model with that of the original model, and calculate the difference as the accuracy loss. The average accuracy loss is reported in Figure 3.

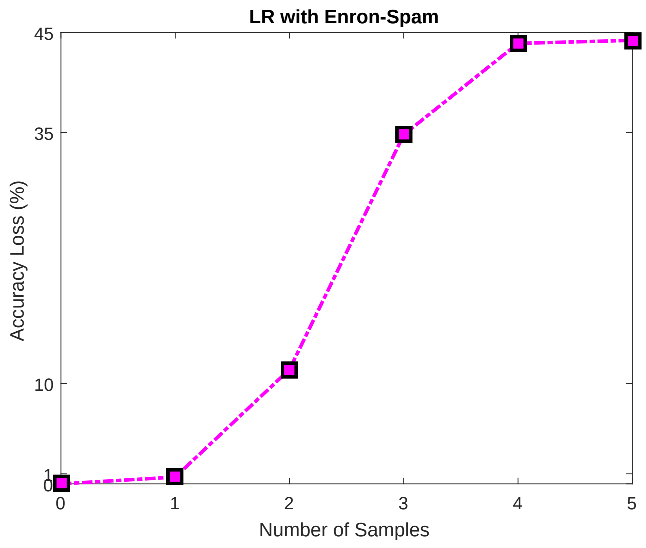


Figure 3: Enron-Spam: LR Accuracy loss with increasing number of target samples

As we can see from Figure 3, the accuracy loss increases drastically as the number of target samples increases. When we have 5 samples to be targeted at the same time, the accuracy loss is near to 45%. In other words, as more samples are being targeted as the same time, LR model becomes less robust in terms of its classification capability. Its overall performance is sacrificed in order to accommodate those target samples. However, when there is only one target sample at a time, the accuracy loss is minimal (only 0.68%), which is hardly noticeable.

7.3.3 Manipulation Attack on MNIST dataset. In this experiment, we evaluate the proposed attack against both LR and CNN on the MNIST dataset. In each case, a model is already trained using the original dataset. We show how we can manipulate a trained LR or CNN model causing certain number of samples to be treated as a different category.

We begin with selecting n random samples from one of the categories. In contrast to the binary classification case, the handwritten digit dataset has 10 classes in total. Hence, we consider four different settings in terms of chosen samples and attacking labels, i.e.,

(1) Setting 1: each sample is chosen from a random class and it is manipulated into a random class label
(2) Setting 2: each sample is chosen from a random class and it is manipulated into a target class label
(3) Setting 3: each sample is chosen from a target class and it is manipulated into a random class label
(4) Setting 4: each sample is chosen from a target class and it is manipulated into a target class label

As in the previous experiments, we vary n from 1 to 5 and repeat the experiment 100 times. The performance of the manipulated model over the test set is evaluated for both LR and CNN models. Each time, we compare the accuracy of the new model with that of the original model, and calculate the difference as the accuracy loss.

$$\text{Accuracy Loss (\%)} = Accuracy_{g(x)} - Accuracy_{g'(x)}$$

Setting 1 With both CNN and LR models, we randomly select samples regardless of their original labels and initiate the manipulation attack to classify the samples with another random class among the possible ones. This is the most general case where a sample can be randomly manipulated into any class different from the original one. The average accuracy loss is visualized in Figure 4.

As clearly shown, for both LR and CNN, the accuracy decreases as the number of target samples increases. However, the trend of change for CNN is significantly less drastic than that of LR, regardless of the number of samples considered. In particular, when 1 and 2 samples are targeted, the accuracy loss for CNN is negligible, which is 0.35% and 0.39%, respectively. The accuracy loss in case of the attack for the LR model is much more significant, up to 13.5% for 5 labels, and over 10% with only three labels modified.

Setting 2 In these experiments, the samples are also chosen randomly but the target class label is fixed as one of the 10 classes from digit 0 to 9 excluding the original class label. For each of the target digit, we repeat the process of random selection of samples and manipulation attack.

The average accuracy loss is shown in Figure 5. In addition, Figure 6 reports the loss for the same setting with the LR model. As shown, for each target label with CNN and LR, the general trend is a decrease of accuracy as the number of targets grows, which is consistent with our findings from experiments carried out under setting 1. In general, we again note that LR performs worse than CNN in each case. Further, in the case of CNN, we see relatively greater loss on average when samples are attacked into class label 2 or label 8. On the contrary, for LR, target label 4 on average has more accuracy loss compared to other digits. We discuss in Section

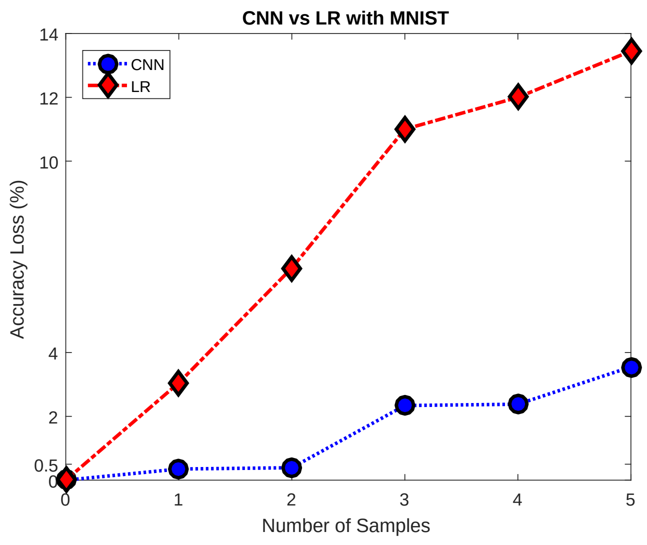
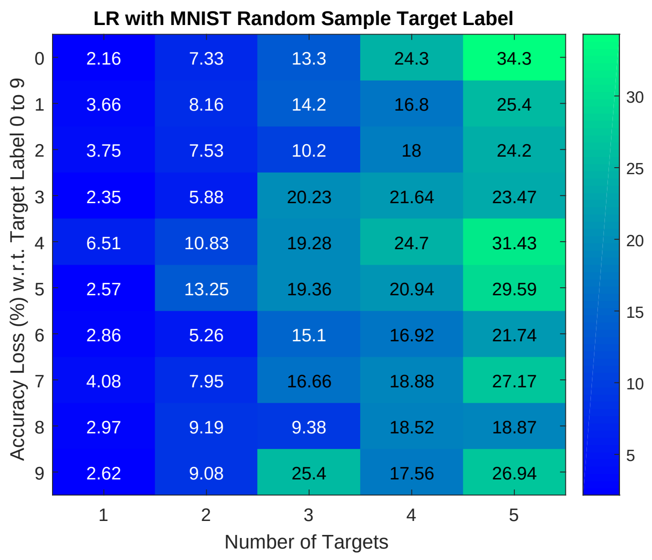


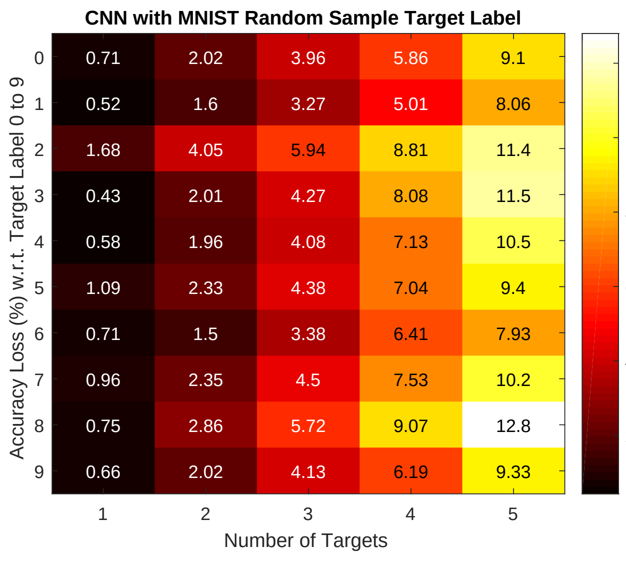
Figure 4: MNIST: CNN and LR Accuracy Loss in Setting 1



	1	2	3	4	5
0	2.16	7.33	13.3	24.3	34.3
1	3.66	8.16	14.2	16.8	25.4
2	3.75	7.53	10.2	18	24.2
3	2.35	5.88	20.23	21.64	23.47
4	6.51	10.83	19.28	24.7	31.43
5	2.57	13.25	19.36	20.94	29.59
6	2.86	5.26	15.1	16.92	21.74
7	4.08	7.95	16.66	18.88	27.17
8	2.97	9.19	9.38	18.52	18.87
9	2.62	9.08	25.4	17.56	26.94

Figure 6: MNIST: LR Accuracy Loss in Setting 2

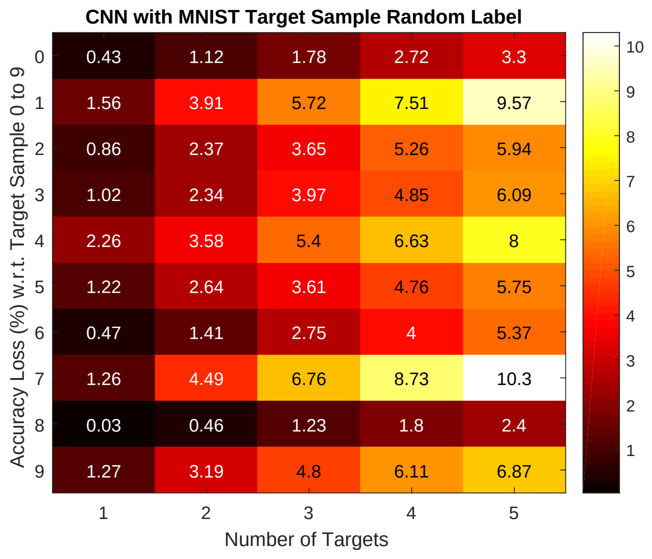
8 some possible reasons for the differences in their performance over different labels.



	1	2	3	4	5
0	0.71	2.02	3.96	5.86	9.1
1	0.52	1.6	3.27	5.01	8.06
2	1.68	4.05	5.94	8.81	11.4
3	0.43	2.01	4.27	8.08	11.5
4	0.58	1.96	4.08	7.13	10.5
5	1.09	2.33	4.38	7.04	9.4
6	0.71	1.5	3.38	6.41	7.93
7	0.96	2.35	4.5	7.53	10.2
8	0.75	2.86	5.72	9.07	12.8
9	0.66	2.02	4.13	6.19	9.33

Figure 5: MNIST: CNN Accuracy Loss in Setting 2

Setting 3 Here, samples are selected from an individual digit class, while the target label is randomly assigned except for their original class label. Again, the attack is repeated for each class. The average accuracy loss for CNN model is shown in Figure 7, whereas the performance of the attack for the LR model is shown in Figure 8. As we can see, a similar trend is observed as the trend reported for the experiments under setting 2, in terms of accuracy loss. For CNN, samples from class label 8 are the easiest to attack while samples from class label 7 are much difficult to attack for both CNN and LR. Moreover, we note that samples chosen from class label 3 and 6 are harder to attack with LR but easier to attack with CNN. The reasons for this difference in performance may be manifold. We elaborate on this matter in the next section.



	1	2	3	4	5
0	0.43	1.12	1.78	2.72	3.3
1	1.56	3.91	5.72	7.51	9.57
2	0.86	2.37	3.65	5.26	5.94
3	1.02	2.34	3.97	4.85	6.09
4	2.26	3.58	5.4	6.63	8
5	1.22	2.64	3.61	4.76	5.75
6	0.47	1.41	2.75	4	5.37
7	1.26	4.49	6.76	8.73	10.3
8	0.03	0.46	1.23	1.8	2.4
9	1.27	3.19	4.8	6.11	6.87

Figure 7: MNIST: CNN Accuracy Loss in Setting 3

Setting 4 Samples are selected from a designated class and are targeted as a specific label among the 10 digit classes other than the original one. In particular, we choose 2 samples and repeat the attack. While, for setting 4 with LR model, the average accuracy loss is shown in Figure 10. As we can see from the Figure 9, comparatively it is more difficult to attack a sample from class label 4 to another digit class and easier to attack class label 8. Interestingly, we note that certain pair of class labels have similar degree of attack difficulty, such as class 3 and 9. Some pairs have contrasting results,

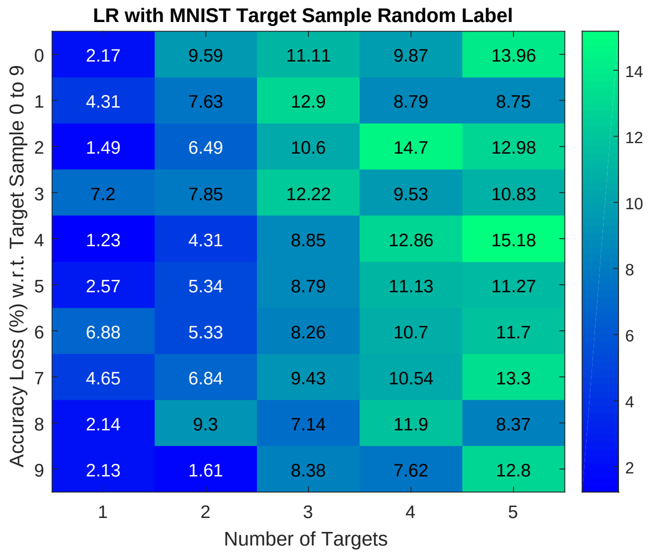


Figure 8: MNIST: LR Accuracy Loss in Setting 3

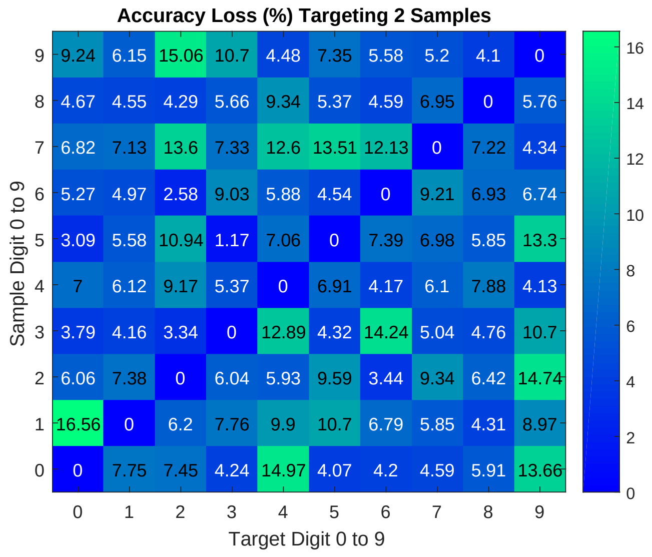


Figure 10: MNIST: LR Accuracy Loss for 2 Target Samples in Setting 4

e.g., for class 8 and class 4 it is much easier to attack a sample of label 8 into label 4 and not the other way around. We have similar observations for the case of LR in Figure 10 as well, where certain pairs have similar degree of attack difficulty.

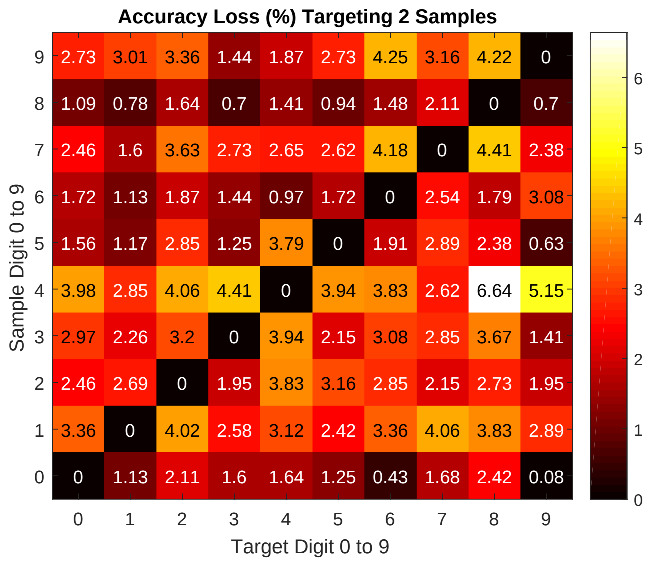


Figure 9: MNIST: CNN Accuracy Loss for 2 Target Samples in Setting 4

8 DISCUSSION

Our experiments provide some interesting insights and open questions, as discussed below.

- As noted, increasing the number of target samples consistently results in lowering the overall accuracy, for both LR and CNN, regardless of the dataset used. This is within our expectations, since modifying the weights of the model implies that we have to to adapt the model to every target sample's feature pattern perfectly. Moreover, with more samples being targeted at the same time, the classifier will be "confused", since the target samples may share similar feature patterns with certain instances in the training set with their original (and therefore different) labels. Therefore, the more samples we intend to accommodate at the same time, the harder it is for us to manipulate the machine learning model. Also note that the actual samples chosen for model manipulation may have a significant impact on the success of the attack, as shown in our experiments for Setting 2 and 3. In addition to the influence brought by randomness in selecting samples, we speculate that samples from certain class label have common feature patterns to allow the model to efficiently adapt itself to the target label, and therefore lead to a low accuracy loss. For example, sample digit 8 has the lowest accuracy loss on average when it is attacked into other digits for both CNN and LR in Setting 4. Further investigation is however required to confirm this hypothesis.
We note that this is not a unique limitation of our model as compared to other similar attacks (e.g. poisoning attack). To our knowledge, other attacks in adversarial machine learning settings are relatively simpler, in that they only craft one single attack sample at a time. Importantly, other attack strategies (as discussed in Section 3.1) are not concerned about how the model performs, as long as their crafted sample can evade detection.
- We notice that it is relatively easy to manipulate a CNN model for a few instances, without affecting the overall performance. We speculate that this is because CNN models typically face the problem of overfitting, i.e., they capture

noise of the data. In contrast, in the case of LR, even a modest change could lead to an apparent difference in the output of the model, i.e., a drastic accuracy loss and significant drop in terms of model performance. This is mainly because of the inherent characteristic of linear model, which is highly sensitive to changes with respect to model parameters. Intuitively, this means that the attacker could potentially have more leverage in complex CNN models. We plan to test this hypothesis in the near future.
- As to LR, the attack performance is not consistent across the two datasets. As the number of samples to be targeted increases, the model performs worse on the textual dataset than on the MNIST. We believe this is mostly due to the difference in terms of feature representation. We use bag of words with binary weighting for spam (textual) dataset, where each feature word has low correlation with each other. In contrast, in the image dataset, certain pixel features have high correlation among one another, to define the shape of individual digits. Hence, classifying a digit image based on correlated pixels yields better performance than that of spam email represented by word vector.

In common adversarial machine learning settings, the major attack venue comes from the adversarial examples crafted by attackers to either evade or poison the training model. Hence, when it comes to defense strategies, some researchers have tried to make the model more robust by countering the effects of the existing adversarial examples [6], while others focused on finding them [13]. However in our case, the attack was applied directly on the model instead of crafting adversarial samples. Therefore, exploratory efforts are still needed in order to develop new defense mechanism.

9 CONCLUSION

In this paper, we presented a new perspective for attacking a machine learning (computational) process carried out in a remote location. Our approach, applied to two different supervised classification models, shows that - within certain constraints - it is possible to compromise a model without significantly sacrificing model accuracy.

Our results however also highlight some limitations of the proposed attack and pave the way for interesting future work. As shown, the difficulty of manipulating a machine learning model grows with the number of samples to be targeted, especially for a linear model like LR. Since our attack solely focuses on the learning model itself, the sensitivity of the model will have a great impact on the effectiveness of the attack. To overcome this limitation, we will study more sophisticated attacks that also target input sample, similar to evasion attacks. In this regard, an interesting direction is to explore a hybrid approach that not only takes advantage of the input sample but also exploits the model itself. Moreover, with a deeper understanding of the attack surface, we will investigate possible defense strategies.

10 ACKNOWLEDGEMENTS

We would like to thank our anonymous reviewers for their helpful comments. The work from Squicciarini and Zhong was partially supported by NSF 1421776. The work of Zhu was supported through NSF CNS-1618684.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Roger Barga and Valentine Fontama. [n. d.]. *Predictive analytics with Microsoft Azure machine learning*. Springer.

[3] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 387–402.

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).

[5] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[6] Xiaoyu Cao and Neil Zhenqiang Gong. 2017. Mitigating evasion attacks to deep neural networks via region-based classification. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 278–287.

[7] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[8] Harris Drucker, Donghui Wu, and Vladimir N Vapnik. 1999. Support vector machines for spam categorization. *IEEE Transactions on Neural networks* 10, 5 (1999), 1048–1054.

[9] Julian Fierrez-Aguilar, Javier Ortega-Garcia, Joaquin Gonzalez-Rodriguez, and Josef Bigun. 2005. Discriminative multimodal biometric authentication based on quality measures. *Pattern recognition* 38, 5 (2005), 777–779.

[10] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1322–1333.

[11] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. [n. d.]. Privacy in Pharmacogenetics: An End-to-End Case Study of Personalized Warfarin Dosing.

[12] Google. [n. d.]. TensorFlow. https://www.tensorflow.org/. ([n. d.]).

[13] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

[14] Uyeong Jang, Xi Wu, and Somesh Jha. 2017. Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning. In *Proceedings of the 33rd Annual Computer Security Applications Conference*. ACM, 262–277.

[15] Bryan Klimt and Yiming Yang. 2004. Introducing the Enron Corpus.. In *CEAS*.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[17] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 2010. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* 2 (2010).

[18] Patrick McDaniel, Nicolas Papernot, and Z Berkay Celik. 2016. Machine learning in adversarial settings. *IEEE Security & Privacy* 14, 3 (2016), 68–72.

[19] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. 2006. Spam filtering with naive bayes-which naive bayes?. In *CEAS*, Vol. 17. 28–69.

[20] Microsoft. [n. d.]. Azure Machine Learning Studio. https://studio.azureml.net/. ([n. d.]).

[21] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[22] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. 2015. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics* 19, 6 (2015), 1893–1905.

[23] Srinivas Mukkamala, Guadalupe Janoski, and Andrew Sung. 2002. Intrusion detection using neural networks and support vector machines. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, Vol. 2. IEEE, 1702–1707.

[24] Michael A Nielsen. 2015. Neural networks and deep learning. (2015).

[25] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 506–519.

[26] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*.

IEEE, 372–387.

[27] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. [n. d.]. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 ([n. d.]), 1.

[28] Shiqi Shen, Shruti Tople, and Prateek Saxena. 2016. A uror: defending against poisoning attacks in collaborative deep learning systems. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM, 508–519.

[29] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. 2016. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging* 35, 5 (2016), 1285–1298.

[30] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing Machine Learning Models via Prediction APIs.. In *USENIX Security Symposium*. 601–618.

A BACKPROPAGATION ALGORITHM

Backpropagation algorithm provides a fast way to compute the gradient of a cost function C with respect to the parameters, e.g., weight w and bias b associated with each layer, in the neural networks [24]. We start with the following notations.

w_{jk}^{l}: the weight of the connection from k^{th} neuron in $(l-1)^{th}$ layer to j^{th} neuron in l^{th} layer

b_j^l: the bias of the j^{th} neuron in l^{th} layer

$\sigma()$: activation function, e.g., a sigmoid function

a_j^l: the activation output of the j^{th} neuron in l^{th} layer

Then, the activation output of the j^{th} neuron in l^{th} layer is computed as

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l). \tag{6}$$

where a_k^{l-1} is the activation output of k^{th} neuron in $(l-1)^{th}$ layer.

Therefore, the activation output of the l^{th} layer can be denoted in a succinct form as

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{7}$$

If we denote $z^l = w^l a^{l-1} + b^l$, the above equation becomes

$$a^l = \sigma(z^l) \tag{8}$$

Suppose for the neural networks we have a cost function C, which is expressed in terms of the activation output a^l and desired output y of an input x. For instance, for a given input x, its cost function can be defined as $C_x = ||\mathbf{y} - \mathbf{a^L}||^2$. The error produced in the last output layer L can be computed as

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{9}$$

where $\nabla_a C$ denotes the partial derivative of cost function C with respect to activation a^L of output layer L, $\odot$ represents the element-wise product of two vectors, and $\sigma'(z^L)$ denotes the derivative of activation function $\sigma()$ with respect to z^L.

Then, the error computed in the last output layer is backpropagated all the way to the first layer after the input layer. The error in layer l is computed based on the error in the next layer $l+1$ when $l = L, L-1, \ldots, 2$, which is expressed as

$$\delta^l = ((w^{l+1})^\top \delta^{l+1}) \odot \sigma'(z^L) \tag{10}$$

where $(w^{l+1})^\top$ is the transpose of matrix w^{l+1}.

Lastly, for each layer $l = L, L-1, \ldots, 2$, we can compute the gradient based on δ^l to update the parameters w and b of each neuron on each layer accordingly.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{11}$$

$$\frac{\partial C}{\partial b_j^i} = \delta_j^l \tag{12}$$