Adversarial Machine Learning: A Survey

Vishal Dey*[1], Manaar Alam†[2], Anupam Chattopadhyay‡[3] and Debdeep Mukhopadhyay§[2]

[1]Department of Computer Science and Technology, Indian Institute of Engineering Science and Technology, Shibpur
[2]Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur
[3]School of Computer Science and Engineering, Nanyang Technological University, Singapore

November 19, 2017

Abstract

Abstract goes here.

1 Introduction

Deep neural networks (DNNs) has proven to be quite effective in machine learning tasks, with recent adoption in automation systems, data analytics and cyber-physical systems. Deep learning based on artificial neural networks is a very popular approach to modeling, classifying, and recognizing complex data such as images, speech, and text. The unprecedented accuracy of deep learning methods has turned them into the foundation of new AI-based services on the Internet, including cloud computing based AI services from commercial players like Google [1], Alibaba [2] and corresponding platform propositions from Intel [3] and nVidia [4]. While the utility of deep learning is undeniable, the same training data that has made it so successful also presents serious privacy issues. Centralized collection of photos, speech, and video from millions of individuals is ripe with privacy risks; direct access to sensitive information.

With increasing popularity and autonomy of the AI-based systems, security threats follow. Adversaries subtly alter legitimate inputs (call input perturbation) to induce the trained model to produce erroneous outputs. Adversarial samples can be used to, for example, subvert fraud detection, bypass content filters or malware detection, or to mislead autonomous navigation systems. Adversarial sample transferability is the property that some adversarial samples produced to mislead a specific model can mislead other models - even if their architectures greatly differ. A practical impact of this property is that it leads to black box attacks. Several prominent works in this area have been reported in last few years.

Adversarial machine learning enables safe learning algorithm in adversarial settings for typical tasks like surveillance, biometric recognition, classification, malware detection. The first advent of adversarial machine learning in 2004 presented by Dalvi et al. [5] developed framework and algorithm to automatically adapt a classifier to the adversary's manipulations. Several other works of adversarial learning and attacks in supervised domain like spam filtering followed by [6], [7], [8].

Of late, Szegedy et al. [9] discovered that several machine learning models including DNNs are vulnerable to adversarial samples. Speculative explanations have suggested it is due to extreme nonlinearity of deep neural networks, perhaps combined with insufficient model averaging and insufficient regularization of the purely supervised learning problem until then and also explained

*vishal.iiestcst@gmail.com
†alam.manaar@iitkgp.ac.in
‡anupam@ntu.edu.sg
§debdeep@cse.iitkgp.ernet.in

the existence of adversarial examples in linear models and linear perturbation in non-linear models and enabled a fast way of generating adversarial samples. Goodfellow et al. [10] explained that the primary cause of neural networks' vulnerability to adversarial perturbation is their linear nature. It also provides a detailed analysis with supportive experiments of adversarial training of linear models and the aspect of generalization of adversarial examples addressed by Papernot et al. [11]. Abadi et al. [12] introduces the concept of distributed deep learning as a way to protect the privacy of training data. In this model, multiple entities collaboratively train a model by sharing gradients of their individual models with each other through a parameter server. Recently in 2017, Hitaj et al. [13] show that a distributed, federated, or decentralized deep learning approach is fundamentally broken and does not protect the training sets of honest participants. The attack we developed exploits the real-time nature of the learning process that allows the adversary to train a Generative Adversarial Network (GAN) that generates prototypical samples of the targeted training set that was meant to be private.

1.1 Related Work: Surveys on Machine Learning

discuss about the surveys. since this is a survey paper, the related work will be corresponding surveys, if any. if there's no comprehensive survey, that would be the major motivation of writing this paper. it would be also good to discuss the surveys on machine learning

1.2 Motivation and Contribution

To our knowledge, there has been no exhaustive survey in the field of adversarial machine learning covering adversarial training and different attacks. Though there are some contributions in domain-specific attacks by an adversary for e.g. Corona et al. [14] presents an overview on adversarial attacks against intrusion detection systems, there are some works which builds a framework for analyzing security in machine learning [8], [15]. To the best of our knowledge, this survey is the first work in the field of adversarial machine learning that compiles all significant works involving different attack models and hardening learning models from the literature.

Organization

In this paper, we perform a brief survey of the existing adversarial learning and common attacks in deep learning. In Section 2, commonly used terms are introduced and discussed. In Section 3, we described the different training methods to make the model robust to adversarial perturbations and in Section 4, common black box and grey box attacks are mentioned with relevant applications.

2 Taxonomy of Adversarial Machine Learning

2.1 Keywords and Definitions

ANN: Artificial Neural networks(ANNs) inspired by the biological neural networks is based on a collection of units called *neurons*. The neurons receive input, activates and output accordingly. The learning governs the weights and activation function so as to able to correctly determine the output. Weights in a multi-layered feed forward are updated by the ubiquitous back-propagation algorithm. ANN is rarely used for predictive modeling as it generally tries to overfit the underlying relationship. It has strong memorization power and works well in case for predicting repetition of past occurrences. It is widely used in image and voice recognition. It was first introduced by McCulloch-Pitts, followed by Hebb's learning rule, eventually giving rise to multi-layer feed-forward perceptron and backpropagation algorithm. ANNs deal with supervised (CNN, DNN) and unsupervised network models (self organizing maps) and their learning rules.

DNN While single layer neural net or perceptron is a feature-engineering approach, deep neural network (DNN) enables feature learning given only raw data as input. Multiple hidden layers and its interconnections extract the features from unprocessed input and thus enhances the performance by finding latent structures in unlabeled, unstructured data. A typical DNN architecture, graphically depicted in Fig. 1, consists of multiple successive layers (at least 2 hidden layers) of processing elements, or so-called "neurons". Each processing layer can be viewed as learning a

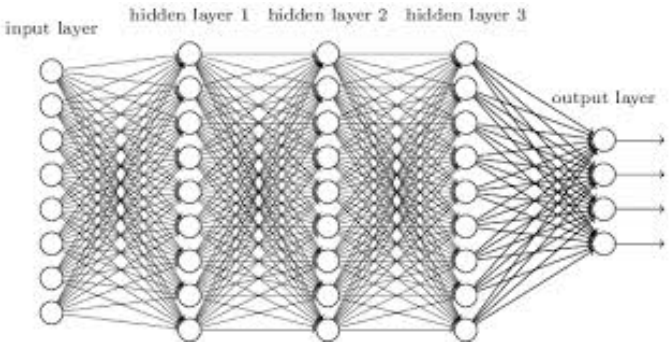


Figure 1: Deep Neural Network

different, more abstract representation of the original multidimensional input distribution. As a whole, a DNN can be viewed as a highly complex function that is capable of nonlinearly mapping original high-dimensional data points to a lower dimensional space. This process roughly models the process of a layer of neurons integrating the information received from the layer below (i.e., computing a pre-activation) before applying an element-wise activation function.

During the training phase of the model, for supervised learning, the DNN's predictions are evaluated by comparing them with known target labels associated with the training samples (also known as the "ground truth"). Specifically, both predictions and labels are taken as the input to a selected cost function, such as cross-entropy. The DNN's parameters are then optimized with respect to this cost function using the method of steepest gradient descent, minimizing prediction errors on the training set. Parameters are calculated generally by gradient descent using back-propagation of errors. While, unsupervised training learns feature representation from raw unstructured, unlabeled data and resulting DNNs can be used to feature engineering, generating new samples as a pre-processing layer.

CNN

A Convolutional Neural Network (CNN) consists of one or more convolutional and subsampling layers, followed by one or more fully connected layers as in a deep neural network. The architecture of CNN is designed to take advantage of 2D input structure (e.g. input image). Convolution layer creates a feature map, pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information and makes the model robust to small distortions. For e.g. to describe a large image, feature values in original matrix can be aggregated at various locations to form a matrix of lower dimension. The last fully connected layer use the feature matrix formed from previous layers to classify the data. CNN is mainly used for feature extraction, thus it also finds application in data preprocessing commonly used in image recognition tasks.

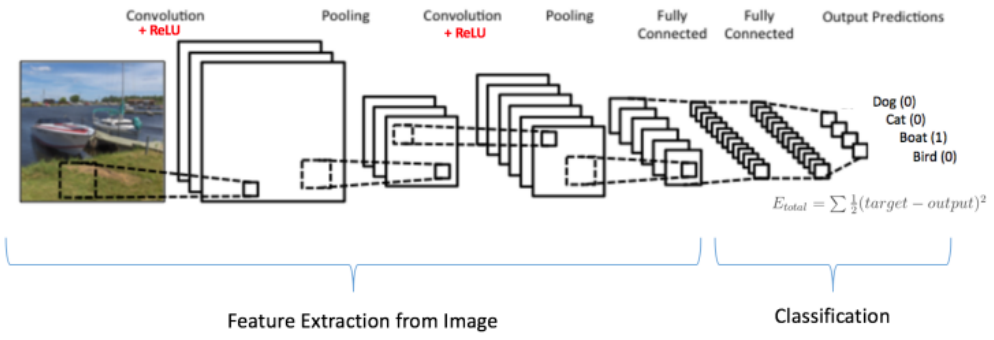


Figure 2: Convolutional Neural Network

Article	Black box	White box
Papernot, Nicolas [11]	●	
Rosenberg, Ishai [17]	●	
Tramèr, Florian [18]	●	●
Papernot, Nicolas [19]	●	
Fredrikson, Matt [20]	●	●
Shokri, Reza [21]	●	
Hitaj, Briland [13]	●	●
Moosavi-Dezfooli [22]	●	
Tramèr, Florian [23]	●	●

Figure 3: Overview of threat models in relevant articles

ELM

The slow learning speed of the feed-forward neural networks is often a bottle-neck for DNNs engendered by gradient-descent based error backpropagation and tuning of all parameters at each layer iteratively. The different learning algorithms for single and multi-layered feed-forward neural networks and their variants pose problem in scalability, portability. To fill the gaps of learning algorithms, Extreme Machine Learning (ELM) [16] proposed a generalized unifying framework for supervised and unsupervised tasks and an learning approach where the parameters of each hidden layer is not iteratively tuned.
To propose a unified framework of updating parameters of hidden layer, ELM considered a multi-layered network as a white box model rather than a black-box model generally followed by other learning algorithms. ELM randomly generates hidden neurons or inherit the properties in the next hidden layer from ancestors and learns the output weights for each layer. Thus in case of ELM, layers get learned one by one instead of neurons in each layer which makes deep learning algorithms slower. Huang et al. [16] showed ELM proved efficient with better results in minimal time (in minutes) for 3D-shape classification, traffic sign recognition, etc.

Black box and white box attacks

Table 1 shows a brief distinction between black box and white box attacks.

Description	Black box attack	White box attack
Adversary Knowledge	Restricted knowledge from being able to only observe the networks output on some probed inputs.	Detailed knowledge of the network architecture and the parameters resulting from training.
Attack Strategy	Based on a greedy local search generating an implicit approximation to the actual gradient w.r.t the current output by observing changes in input.	Based on the gradient of the network loss function w.r.t to the input.

Table 1: Distinction between black box and white box attacks

Articles	Attacks	Applications
Fredrikson et al. [20]	Model Inversion	Biomedical Imaging, biometric identification
Tramer et al. [18]	Extraction of target machine learning models using APIs	Attacks extend to multiclass classifications & neural networks
Anteniese et al. [24]	Meta-classifier to hack other classifiers	Speech Recognition
Biggio et al. [25], [26]	Poisoning based attacks:	Crafted training data for Support vector Machines
Dalvi et al. [5] Biggio et al. [7], [8]	Adversarial Classification, Pattern recognition	Email Spam detection, fraud detection, intrusion detection, biometric identification
Papernot et al. [19], [11]	Adversarial samples crafting, adversarial sample transferability	digit recognition, black-box attacks against classifiers hosted by Amazon and Google
Hitaj et al. [13]	GAN: Collaborative deep learning attacks: Black-box and white-box attacks	Attack proved effective even against CNN which are notoriously difficult to invert
Moosavi et al. [22] Carlini et al. [27] Li et al. [28]	Adversarial perturbations: and sample generation: Poisoning based attack	Image classification intrusion detection Collaborative filtering systems

Table 2: Overview of Attacks and Applications

Adversarial training and attacks

Adversarial training is a computational method of developing robust learning so that a slight perturbation in the input does not mislead a model generating false outputs. A lot of adversarial examples are generated and the model is explicitly trained not to be fooled by each of them. These can be broadly classified into two groups:

- Targeted: Szegedy et al. [9] first introduced a system that generates adversarial samples by perturbing inputs in a way that creates source/target misclassifications. It showed that given input $\boldsymbol{x}$ a target model $\boldsymbol{L}$ and a target $t \neq L^*(x)$, it is possible to find a similar input $\boldsymbol{x'}$ such that $\boldsymbol{L(x')} = \boldsymbol{t}$ ($\boldsymbol{L(x)}$ and $\boldsymbol{L^*(x)}$ are the predicted and actual outputs respectively) . Different methods are found in the literature, e.g L-BFGS [9], Fast-Gradient [10].

- Untargeted: Moosavi [22] presented a less powerful way of generating adversarial examples. Instead of classifying $\boldsymbol{x}$ as a given target class, it searches for an input $\boldsymbol{x'}$ so that $\boldsymbol{L(x')} \neq \boldsymbol{L^*(x)}$ and $\boldsymbol{x}$, $\boldsymbol{x'}$ are close.

Adversarial examples are crafted inputs to the learning model that the attacker meticulously design to cause the model to mistake. Although a lot of promising work in adversarial training has been done, it remains impossible defend a model as it is difficult to construct a theoretical model of the adversarial crafting process. Different attacking and adversarial training methods along with recent advancements have been discussed in the following Sections 5 - 7.

3 Adversarial Training

Wang [29] propose a general framework to facilitate the development of adversary resistant DNNs.

3.1 Generating Adversarial Samples

Adversarial samples are generated by computing the derivative of the cost function with respect to the network's input variables. The gradient of any input sample represents a direction vector in the model's high-dimensional input space. Along this direction, a small change in this input sample can cause the DNN to generate a completely different prediction result. This particular direction is important since it represents the most effective way one might compromise the optimization of the cost function. Discovering this particular direction is realized by transmitting the error gradients

from output layer all the way back to the input layer via back-propagation. The gradient with respect to the input may then be applied to the input sample(s) to craft an adversarial example(s). We now formally present the generation of adversarial samples by first defining the function g for generating any adversarial sample $\hat{\boldsymbol{x}}$:

$$\boldsymbol{g} : \boldsymbol{R}^{\mathrm{m}} \times \boldsymbol{R}^{\mathrm{k}} \rightarrow \boldsymbol{R}^{\mathrm{m}} \tag{1}$$

$$\boldsymbol{x} \times \boldsymbol{y} \rightarrow \hat{\boldsymbol{x}}$$

where

$$\hat{\boldsymbol{x}} = \boldsymbol{arg\ max}\ \boldsymbol{L}(\boldsymbol{f}(\hat{\boldsymbol{x}}), \boldsymbol{y})\ \boldsymbol{s.t}\ ||\hat{\boldsymbol{x}} - \boldsymbol{x}||_{\boldsymbol{p}} < \boldsymbol{\epsilon} \tag{2}$$

and $||.||_{\boldsymbol{p}}$ is p-norm.

Note that g is a one-to-one mapping, meaning that when there exist multiple $\hat{\boldsymbol{x}}$ satisfying (2), only one is chosen. It should be noted that the constraint in (2) ensures that the distortion incurred by manipulation must be maintained at a small scale. Otherwise a significant distortion will leave the manipulation to be easily detected. Existing attacks [27], [9], [30] consist of different approaches for generating adversarial samples which ultimately, can all be described solving following optimization problem, which is a transformation of (2):

$$\hat{\boldsymbol{x}} : \boldsymbol{minc} * ||\boldsymbol{x} - \hat{\boldsymbol{x}}||_{\boldsymbol{p}} - \boldsymbol{L}(\boldsymbol{f}(\hat{\boldsymbol{x}}), \boldsymbol{y}) \tag{3}$$

where c is some weight or coefficient. Here we refer to this type of attack as an optimal attack, and, for simplicity, denote $||\boldsymbol{x} - \hat{\boldsymbol{x}}||_{\boldsymbol{p}}$ as the $\boldsymbol{l_p}$ distance. Solving (3) is done much like solving for the weight coefficients of a standard DNN. Simply put, an attacker can use back-propagation and either gradient descent [9] or L-BFGS [27]. In order to conduct the calculation, one first computes $\boldsymbol{\partial^n L(f(\hat{x}, y))/\partial \hat{x}^n}$ and then use it for manipulating legitimate samples x. Note that here n = 1 if we use a first-order optimization method like gradient descent, or n = 2 if we utilize a second-order method such as the Newton–Raphson method. Since gradient descent requires an iterative calculation, it can be computationally expensive when the number of adversarial samples to be generated is large. To mitigate this problem, [9] proposed the fast gradient sign method to approximate this iterative calculation where one could use a single step of gradient descent, as follows:

$$\hat{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\epsilon.sign}(\boldsymbol{\nabla L}(\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y})) \tag{4}$$

where $\boldsymbol{\epsilon}$ is set to control the scale of the distortions applied. Optimal attacks can be generalized to fast gradient sign attacks by selecting the $\boldsymbol{l_\infty}$ distance. In [9], the authors study an equivalent optimization problem to (3) by setting p = 2. Another type of attack, as described in [30], generates adversarial samples using a greedy approach to solve (3) and sets p = 0. More specifically, adversarial samples are crafted by iteratively perturbing one coordinate of x at a time. A coordinate is picked by determining by its influence on the final cost.

4 Attacks

the term "adversarial attack" is confusing. Adversary will of course attack. Need to propose a better idea.

4.1 Model Inversion(MI) Attack

Authors in [29] explains this attack. Once the network has been trained, the gradient can be used to adjust the weights of the network and obtain a reverse-engineered example for all represented classes in the network. For those classes that it did not have prior information, it would still be able to recover prototypical examples. This attack shows that any accurate deep learning machine, no matter how it has been trained, can leak information about the different classes that it can distinguish. [20] developed new model inversion attacks through ML APIs that exploit confidence values in a variety of settings and explored countermeasures as simple variant of CART in both black box and white box settings.

Applications This attack has been successfully experimented in face recognition where [20] showed given access to the model and person's name, it can recover the facial image, lifestyle prediction, medical genome prediction.

Limitation : Due to the rich structure of deep learning machines, the model inversion attack may recover only prototypical examples that have little resemblance to the actual data that defined that class. It may or may not be considered as an attack as it may construct wrong/meaningless information.

4.2 Generative Adversarial Attack(GAN)

The GAN procedure pits a discriminative deep learning network against a generative deep learning network. The discriminative network is trained to distinguish between images from an original database and those generated by the GAN. The generative network is first initialized with random noise, and at each iteration, it is trained to mimic the images in the training set of the discriminative network. The procedure ends when the discriminative network is unable to distinguish between samples from the original database and the samples generated by the generative network.

4.3 Black Box and White box attacks

There are two main research directions in the literature on adversarial attacks based on different assumptions about the adversarial knowledge of the target network. The first and the most common line of work; whitebox attacks assumes that the adversary has detailed knowledge of the network architecture and the parameters resulting from training (or access to the labeled training set). Using this information, an adversary constructs a perturbation for a given image. The most effective methods are gradient-based: a small perturbation is constructed based on the gradient of the network loss function w.r.t. the input image. Often, adding this small perturbation to the original image leads to a misclassification. In the second line of work; black-box attacks, an adversary has restricted knowledge about the network from being able to only observe the network's output on some probed inputs. This attack strategy is based the idea of greedy local search, an iterative search procedure, where in each round a local neighborhood is used to refine the current image and in process optimizing some objective function that depends on the network output. In each round, the local search procedure generates an implicit approximation to the actual gradient w.r.t the current image by observing changes in output. This approximate gradient provides a partial understanding of the influential pixels in the current image for the output, which is then used to update this image.

Note: Different examples to be added from [17]

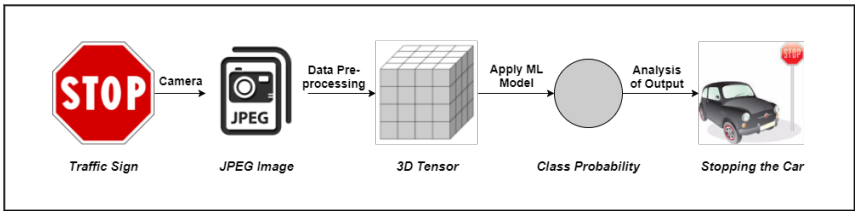


Figure 4: Generic pipeline of an Automated Vehicle System

5 Adversarial Threat Model

The security of any machine learning model is measured concerning the adversarial goals and capabilities. In this section, we taxonomize the threat models in machine learning systems keeping in mind the strength of the adversary. We begin with the identification of threat surface [31] of systems built on machine learning models to identify where and how an adversary may attempt to subvert the system under attack.

5.1 The Attack Surface

A system built on Machine Learning can be viewed as a generalized data processing pipeline. A primitive sequence of operations of the system at the testing time can be viewed as: a) collection of input data from sensors or data repositories, b) transferring the data in the digital domain, c) processing of the transformed data by machine learning model to produce an output, and finally, d) action taken based on the output. For illustration, consider a generic pipeline of an automated vehicle system as shown Figure 4.

The system collects sensor inputs (images using camera) from which model features (tensor of pixel values) are extracted and used within the models. The meaning of the model output (probability of stop sign) is then interpreted, and appropriate action is taken (stopping the car). The *attack surface*, in this case, can be defined with respect to the data processing pipeline. An adversary can attempt to manipulate either the *collection* or the *processing* of data to corrupt the target model, thus tampering the original output. Two main attack scenarios identified by the attack surface are sketched below [8, 32]:

1. *Evasion Attack*: This is the most common type of attack that may be encountered in adversarial settings during system operation. The malicious samples are adjusted at test time to evade the detection mechanism, i.e., to be misclassified as legitimate. This setting does not assume any influence over the training data.

2. *Poisoning Attack*: This type of attack, known as contamination of the training data, takes place during the training time of the machine learning model. An adversary tries to poison the training data by injecting carefully designed samples to compromise the whole learning process eventually.

3. *Exploration Attack*: These attacks do not influence training; they try to discover knowledge about the learning algorithm being used by the system, state of the underlying model, discover pattern in training data or infer membership for given black box access to the model.

The definition of a threat model depends on the information the adversary has at their disposal. Next, we discuss in details the adversarial capabilities for the threat model.

5.2 The Adversarial Capabilities

The term adversarial capabilities refer to the amount of information about the system available to an adversary, which also indicates the attack vector an adversary may use on the threat surface. For illustration, again consider the case of an automated vehicle system as shown in Figure 4 with the attack surface being the testing time (i.e., an Evasion Attack). An internal adversary may have access to the model architecture used to distinguish between different images and traffic signs, whereas a weaker adversary would only have access to the dump of images fed to the model during testing time. Though the attack surface remains the same for both the attacks, the former

adversary is assumed to have much more information and is thus strictly "stronger". We explore the range of adversarial capabilities in machine learning systems as they relate to testing and training phases.

5.2.1 Testing Phase Capabilities

Adversarial attacks at the testing time do not tamper with the targeted model but instead force it to produce incorrect outputs. The effectiveness of such attacks is determined mainly by the amount of information available to the adversary about the model. Testing phase attacks can be broadly classified into either *White-Box* or *Black-Box* attacks. Before discussing these attacks, we provide a formal definition [23] of a training procedure for a machine learning model.

Let us consider a target machine learning model $\boldsymbol{f}$ is trained over input pair $\boldsymbol{(X, y)}$ from the data distribution $\boldsymbol{\mu}$ with a randomized training procedure $\boldsymbol{train}$ having randomness $\boldsymbol{r}$ (e.g., random weight initialization, dropout, etc.). The model parameters $\boldsymbol{\theta}$ are learned after the training procedure. More formally, we can write:

$$\boldsymbol{\theta \leftarrow train(f, X, y, r)}$$

Now, let us understand the capabilities of the white-box and black-box adversaries with respect to this definition.

White-Box Attacks

In white-box attack on a machine learning model, an adversary has perfect knowledge about the model ($\boldsymbol{f}$) used for classification (e.g., type of neural network along with number of layers). The attacker has information about the algorithm ($\boldsymbol{train}$) used in training (e.g., gradient-descent optimization) and can access the training data distribution ($\boldsymbol{\mu}$). He also knows the parameters ($\boldsymbol{\theta}$) of the model architecture, resulting in after the training process is completed. The adversary utilizes available information to identify where a model may be vulnerable, i.e., identify the feature space for which the model has a high error rate. Then the model is exploited by altering an input using adversarial example crafting method, which we discuss later. The access to the internal model weights for a white-box attack corresponds to a very strong adversarial attack.

Black-Box Attacks

Black-Box attack, on the contrary, assumes no knowledge about the model and uses information about the settings or past inputs to infer model vulnerability. For example, in an oracle attack, the adversary exploits a model by providing a series of carefully crafted inputs and observing outputs. Black Box attacks can be further classified into the following categories:

1. ***Non-Adaptive Black-Box Attack:*** For a target model ($\boldsymbol{f}$), a non-adaptive black-box adversary only gets access to the target model's training data distribution ($\boldsymbol{\mu}$). The adversary then chooses a procedure $\boldsymbol{train'}$ for a model architecture $\boldsymbol{f'}$ and trains a local model over samples from the data distribution $\boldsymbol{\mu}$ to approximate the model learned by the target classifier. The adversary crafts adversarial examples on the local model $\boldsymbol{f'}$ using white-box attack strategies and applies these crafted inputs to the target model to force mis-classifications.

2. ***Adaptive Black-Box Attack:*** For a target model ($\boldsymbol{f}$), an adaptive black-box adversary does not have access to any information regarding the training process but can access the target model as an oracle. This strategy is analogous to chosen-plaintext attack in cryptography. The adversary issues adaptive oracle queries to the target model and labels a carefully selected dataset, i.e., for any arbitrarily chosen $\boldsymbol{x}$ the adversary obtains its label $\boldsymbol{y}$ by querying the target model $\boldsymbol{f}$. The adversary then chooses a procedure $\boldsymbol{train'}$ and model architecture $\boldsymbol{f'}$ to train a surrogate model over tuples $\boldsymbol{(x, y)}$ obtained from querying the target model. The surrogate model is then used to craft adversarial examples following white-box attack technique for forcing the target model to mis-classify malicious data.

3. ***Strict Black-Box Attack:*** A black-box adversary sometimes may not contain the data distribution $\boldsymbol{\mu}$ but has the ability to collect the input-output pairs $\boldsymbol{(x, y)}$ from the target classifier. However, he can not change the inputs to observe the changes in output like an adaptive attack procedure. This strategy is analogous to the known-plaintext attack in cryptography and would most likely to be successful for a large set of input-output pairs.

The point to be remembered in the context of a black-box attack is that an adversary neither tries to learn the randomness $\boldsymbol{r}$ used to train the target model nor the target model's parameters $\boldsymbol{\theta}$. The primary objective of a black-box adversary is to train a local model with the data distribution $\boldsymbol{\mu}$ in case of a non-adaptive attack and with carefully selected dataset by querying the target model in case of an adaptive attack.

5.2.2 Training Phase Capabilities

Attacks during training time attempt to influence or corrupt the model directly by altering the dataset used for training. The most straightforward and arguably the weakest attack on training phase is by merely accessing a partial or all of the training data. There are three broad attack strategies for altering the model based on the adversarial capabilities.

1. ***Data Injection:*** The adversary does not have any access to the training data as well as to the learning algorithm but has ability to augment a new data to the training set. He alters the training data by inserting adversarial inputs into the existing dataset thereby corrupting the target model.

2. ***Data Modification:*** The adversary does not have access to the learning algorithm but has full access to the training data. He poisons the training data directly by modifying the data before it is used for training the target model.

3. ***Logic Corruption:*** The adversary has the ability to tamper with the learning algorithm. These attacks are referred as logic corruption. Apparently, an adversary that can alter the learning logic and thus controls the model itself is very potent and difficult to defend.

The adversarial threat model also depends not only depends on the adversarial capabilities but also on the action taken by the adversary. In the next subsection, we discuss the goal of an adversary while compromising the security of any machine learning system.

5.3 Adversarial Goals

An adversary attempts to provide an input $\boldsymbol{x}_*$ to a classification system that results in an incorrect output classification. The nature of the incorrectness represents the adversarial goals. Based on the impact on the classifier output integrity the adversarial goals can be broadly classified as follows:

1. ***Confidence Reduction:*** The adversary tries to reduce the confidence of prediction for the target model. For example, a legitimate image of a 'stop' sign can be predicted with a lower confidence having a lesser probability of class belongingness.

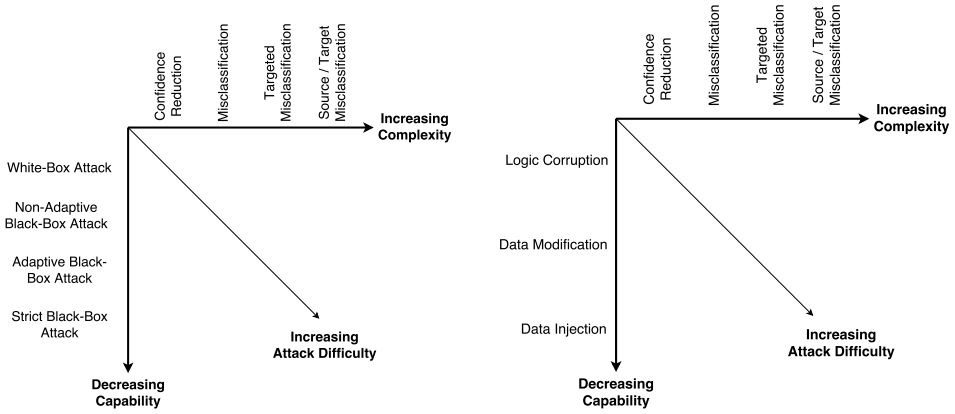
2. ***Misclassification:*** The adversary tries to alter the output classification of an input example to any class different from the original class. For example, a legitimate image of a 'stop' sign will be predicted as any other class different from the class of stop sign.

3. ***Targeted Misclassification:*** The adversary tries to produce inputs that force the output of the classification model to be a specific target class. For example, any input image to the classification model will be predicted as a class of images having 'go' sign.

4. ***Source/Target Misclassification:*** The adversary attempts to force the output of classification for a specific input to be a particular target class. For example, the input image of 'stop' sign will be predicted as 'go' sign by the classification model.

The taxonomy of the adversarial threat model for both the evasion and the poisoning attacks with respect to the adversarial capabilities and adversarial goals are represented graphically in Figure 5a and Figure 5b respectively. The horizontal axis of both figures represents the complexity of adversarial goals in increasing order, and the vertical axis loosely represents the strength of an adversary in decreasing order. The diagonal axis represents the complexity of a successful attack based on the adversarial capabilities and goals.

In the next section, we discuss different methods of adversarial example generation for both the attack surface, i.e., for evasion attacks as well as for poisoning attack.



(a) Attack Difficulty with respect to adversarial capabilities and goals for Evasion Attacks

(b) Attack Difficulty with respect to adversarial capabilities and goals for Poisoning Attacks

Figure 5: Taxonomy of Adversarial Model for a) Evasion Attacks and b) Poisoning Attacks with respect to adversarial capabilities and goals.

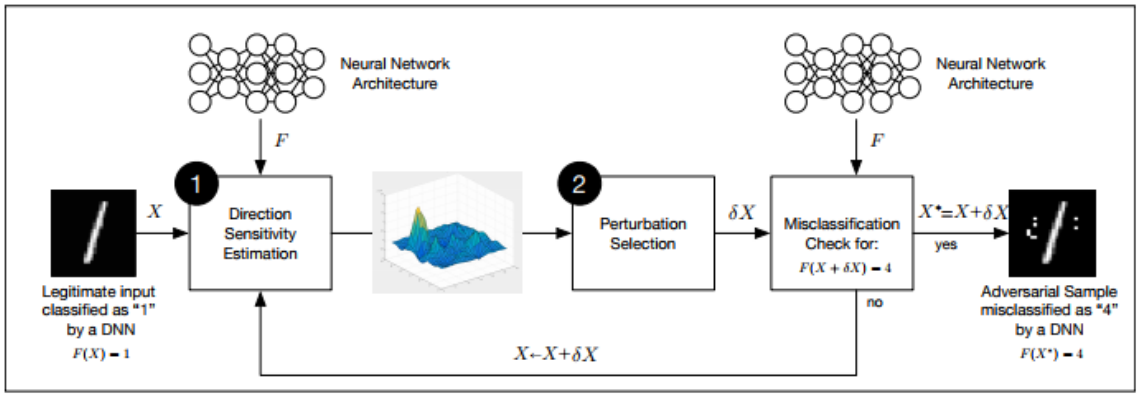


Figure 6: Basic Adversarial Example Crafting Framework for Evasion Attacks [33]

6 Adversarial Examples Generation

In this section, we present a general overview for modifying samples so that a classification model yields an adversarial output. The adversarial sample modification can be performed both in training and testing phase.

6.1 Testing Phase Generation

6.1.1 White-Box Attacks

In this subsection, we precisely discuss how adversaries craft adversarial samples in a white-box setup. Papernot et. al. [33] introduced a general framework which builds on the attack approaches discussed in recent literature. The framework is split into two phases: a) *direction sensitivity estimation* and b) *perturbation selection* as shown in Figure 6. The figure proposes an adversarial example crafting process for an image classification using DNN, which can be generalized for any supervised learning algorithm.

Let us consider a sample $\boldsymbol{X}$ and a trained DNN resulting in a classification model $\boldsymbol{F}$. The goal of an adversary is to produce an adversarial sample $\boldsymbol{X_* = X + \delta X}$ by adding a perturbation $\boldsymbol{\delta X}$ to the sample $\boldsymbol{X}$, so that $\boldsymbol{F(X_*) = Y_*}$ where $\boldsymbol{Y_* \neq F(X)}$ is the adversarial target output which depends on the goal of the adversary. An adversary starts by considering a legitimate sample $\boldsymbol{X}$. Since the attack setting is a white-box attack, the adversary has the capability of accessing parameters $\boldsymbol{\theta}$ of the targeted model $\boldsymbol{F}$. The adversarial sample crafting is then a two-step process:

1. ***Direction Sensitivity Estimation:*** The adversary evaluates the sensitivity of a class change to each input feature by identifying directions in the data manifold around sample

$\boldsymbol{X}$ in which the model $\boldsymbol{F}$, learned by the DNN is most sensitive and likely to result in a class change.

2. ***Perturbation Selection:*** The adversary then exploits the knowledge of sensitive information to select a perturbation $\boldsymbol{\delta X}$ among the input dimensions to find an effective adversarial perturbation.

Both the steps are repeated, if necessary, by replacing $\boldsymbol{X}$ with $\boldsymbol{X} + \boldsymbol{\delta X}$ before the start of each new iteration, until the adversarial goal is satisfied by the perturbed sample. The point to be remembered in this context is that it is crucial for the total perturbation used to craft an adversarial sample from a legitimate example to be as minimum as possible, which is essential for adversarial samples to remain undetected by humans.

Adversarial sample crafting using large perturbations is trivial. Thus, if one defines a norm $\|.\|$ to appropriately describe the differences between points in the input domain of the DNN model $\boldsymbol{F}$, we can formalize the adversarial samples as a solution to the following optimization problem:

$$\boldsymbol{X_*} = \boldsymbol{X} + \arg\min_{\boldsymbol{\delta X}} \{\|\boldsymbol{\delta X}\| : \boldsymbol{F}(\boldsymbol{X} + \boldsymbol{\delta X}) \neq \boldsymbol{F}(\boldsymbol{X})\} \tag{5}$$

Most DNN models make this formulation non-linear and non-convex, making it hard to find a closed-solution in most of the cases. We now describe in details different techniques to find an approximate solution to this optimization problem using each of the two steps mentioned above.

Direction Sensitivity Estimation

In this step, the adversary considers a legitimate sample $\boldsymbol{X}$, an $\boldsymbol{n}$-dimensional input vector. The objective here is to find those dimensions of $\boldsymbol{X}$ which will produce an expected adversarial performance with the smallest selected perturbation. This can be achieved by changing the input components of $\boldsymbol{X}$ and evaluating the sensitivity of the trained DNN model $\boldsymbol{F}$ for these changes. Developing the knowledge of the model sensitivity can be accomplished in several ways. Some of the well-known techniques mentioned in the recent literature are discussed below.

1. **L-BFGS:** Szegedy et al. [9] first coined the term adversarial sample by formalizing the search for adversarial examples as the following minimization problem, which equivalent to equation 5.

$$\arg\min_{\boldsymbol{r}} \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{r}) = \boldsymbol{l} \quad \text{s.t.} \ (\boldsymbol{x} + \boldsymbol{r}) \in \boldsymbol{D}$$

The input $\boldsymbol{x}$, correctly classified by f, is perturbed with $\boldsymbol{r}$ such that the resulting adversarial example $\boldsymbol{x_*} = \boldsymbol{x} + \boldsymbol{r}$ remains in the input domain $\boldsymbol{D}$ but is assigned the target label $\boldsymbol{l} \neq \boldsymbol{h}(\boldsymbol{x})$. For non-convex models like DNN, the authors have used the *L-BFGS* [34] optimization algorithm to solve the above equation. Though the method gives good performance, it is computationally expensive while calculating adversarial samples.

2. **Fast Gradient Sign Method (FGSM):** An efficient solution to equation 5 is introduced by Goodfellow et al. [10]. They proposed a fast gradient sign method that computes the gradient of the cost function with respect to the input of the neural network. The adversarial examples are generated using the following equation:

$$\boldsymbol{X_*} = \boldsymbol{X} + \boldsymbol{\epsilon} * \boldsymbol{sign}(\boldsymbol{\nabla_x} \boldsymbol{J}(\boldsymbol{X}, \boldsymbol{y_{true}}))$$

Here, $\boldsymbol{J}$ is the cost function of the trained model, $\boldsymbol{\nabla_x}$ denotes the gradient of the model with respect to a normal sample $\boldsymbol{X}$ with correct label $\boldsymbol{y_{true}}$, and $\boldsymbol{\epsilon}$ denotes the input variation parameter which controls the perturbation's amplitude. Recent literature have used some other variations of FGSM, which are summarised as below:

(a) **Target Class Method:** This variant of FGSM [35] approach maximizes the probability of some specific target class $\boldsymbol{y_{target}}$, which is unlikely the true class for a given example. The adversarial example is crafted using the following equation:

$$\boldsymbol{X_*} = \boldsymbol{X} - \boldsymbol{\epsilon} * \boldsymbol{sign}(\boldsymbol{\nabla_x} \boldsymbol{J}(\boldsymbol{X}, \boldsymbol{y_{target}}))$$

(b) **Basic Iterative Method:** This is a straightforward extension of the basic FGSM method [35]. This method generates adversarial samples iteratively using small step size.

$$X_*^0 = X; \quad X_*^{n+1} = Clip_{X,e}\{X_*^n + \alpha * sign(\nabla_x J(X_*^n, y_{true}))\}$$

Here, α is the step size and $Clip_{X,e}\{A\}$ denotes the element-wise clipping of X. The range of $A_{i,j}$ after clipping belongs in the interval $[X_{i,j} - \epsilon, X_{i,j} + \epsilon]$. This method does not typically rely on any approximation of the model and produces additional harmful adversarial examples when run for more iterations.

3. **Jacobian Based Method:** Papernot et al. [30] introduced a different approach for finding sensitivity direction by using forward derivative, which is the Jacobian of the trained model F. This method directly provides gradients of the output components with respect to each input component. The knowledge thus obtained is used to craft adversarial samples using a complex saliency map approach which we will discuss later. This method is particularly useful for source-target misclassication attacks.

Perturbation Selection

The adversary may use the knowledge about network sensitivity for input variations differently to evaluate the dimensions which are most likely to produce the target misclassification with minimum total perturbation. The perturbed input dimensions can be of two types:

1. **Perturb all the input dimensions:** Goodfellow et al. [10] decided to perturb all the input dimensions by a small quantity in the direction of the sign of the gradient computed according to the FGSM method. This method efficiently minimizes the Euclidian distance between the original and the corresponding adversarial samples.

2. **Perturb a selected input dimensions:** Papernot et al. [30] choose to follow a more complicated process involving saliency maps to select only a limited number of input dimensions to perturb. The objective of using saliency map is to assign values to the combination of input dimensions which indicates whether the combination if perturbed, will contribute to the adversarial goals. This method effectively reduces the number of input features perturbed while crafting adversarial examples. For choosing the input dimensions which forms the perturbations, dimensions are sorted by decreasing adversarial saliency value. The adversarial saliency value $S(x,t)[i]$ of a component i of a legitimate example x for a target class t is evaluated using the following equation:

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

where $\left[\frac{\partial F_j}{\partial x_i}\right]_{ij}$ could be easily calculated using the Jacobian Matrix J_F of the learned model F. Input components are added to perturbation δxin the decreasing order of the adversarial saliency value $S(x,t)[i]$ until the resulting adversarial sample $x_* = x + \delta x$ is misclassified by the model F.

Each method has its own advantages and drawbacks. The first method is well fitted for the fast crafting of many adversarial samples but with a relatively large perturbation and thus is potentially easier to detect. The second method reduces the perturbations at the expense of a higher computing cost.

6.1.2 Black-Box Attacks

In this subsection, we discuss in details on the generation of adversarial examples in a black-box setting. Crafting adversarial samples in non-adaptive and strict black box scenario is straightforward. In both the cases, the adversary has access to a vast dataset to train a local substitute model which approximates the decision boundary of the target model. Once the local model is trained with high confidence, any of the white-box attack strategies can be applied on the local model to

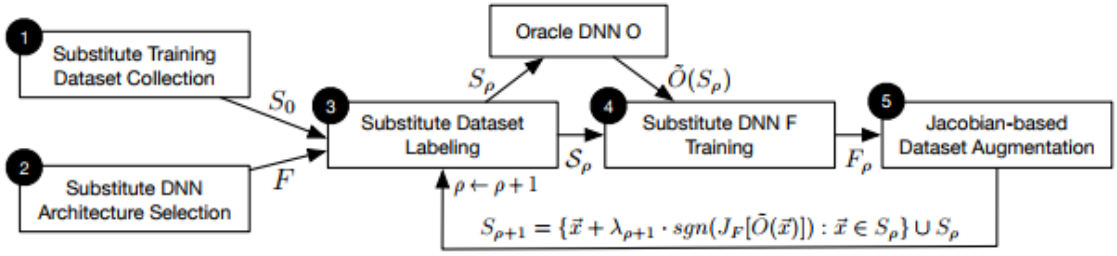


Figure 7: Substitute Mode Training using Jacobian based dataset augmentation. [19]

generate adversarial examples, which eventually can be used to fool the target model because of the *Transferability Property of Neural Network* (will be discussed later). However, in the case of an adaptive black-box scenario, the adversary does not have access to a large dataset and thus augments a partial or randomly selected dataset by selectively querying the target model as an oracle. One of the popular method of dataset augmentation presented by Papernot et al. [19] is discussed next.

Jacobian based Data Augmentation

An adversary could potentially make an infinite number of queries to get the Oracle's output $\boldsymbol{O(x)}$ for any input $\boldsymbol{x}$. This would provide the adversary a copy of the oracle. However, the process is not tractable considering the continuous domain of an input to be queried. Furthermore, making a significant number of queries presents the adversarial behavior easy to detect. The heuristic that can be used to generate synthetic training inputs is based on identifying the directions in which the target model's output is varying, around an initial set of training points. With more input-output pair the direction can be captured easily for a target Oracle $\boldsymbol{O}$. Hence, the greedy heuristic that an adversary follow is to prioritize the samples while querying the oracle for labels to get a substitute DNN $\boldsymbol{F}$ approximating the decision boundaries of the Oracle. These directions can be identified with the substitute DNN's Jacobian matrix $\boldsymbol{J_F}$, which is evaluated at several input points $\boldsymbol{x}$. Precisely, the adversary evaluates the sign of the Jacobian matrix dimension corresponding to the label assigned to input $\boldsymbol{x}$ by the oracle, denoted by $\boldsymbol{sgn\left(J_F(x)\left[O(x)\right]\right)}$. The term $\boldsymbol{\lambda * sgn\left(J_F(x)\left[O(x)\right]\right)}$ is added to the original datapoint, to obtain a new synthetic training point. The iterative data augmentation technique can be summarized using the following equation.

$$\boldsymbol{S_{n+1} = \{x + \lambda * sgn\left(J_F(x)\left[O(x)\right]\right) : x \in S_n\} \cup S_n}$$

where $\boldsymbol{S_n}$ is the dataset at $\boldsymbol{n^{th}}$ step and $\boldsymbol{S_{n+1}}$ is the augmented dataset. The substitute model training following this approach is presented in Figure 7.

6.1.3 Transferability of Adversarial Samples

Adversarial sample transferability [11] is the property that adversarial samples produced by training on a specific model can affect another model, even if they have different architectures - leading to adaptive black box attacks as discussed in Section 5.2.1. Since in case of black-box attack, adversary does not have access to the target model $\boldsymbol{F}$, an attacker can train a *substitute model* $\boldsymbol{F'}$ locally to generate adversarial example $\boldsymbol{X + \delta X}$ which then can be transfered to the victim model. Formally, if $\boldsymbol{X}$ is the original input, the transferability problem can be represented as an optimization problem (5).

It can be broadly classified into two types:

1. Intra-technique transferability: If models $\boldsymbol{F}$ and $\boldsymbol{F'}$ are both trained using same machine learning technique (e.g. both are NN or SVM)

2. Cross-technique transferability: If learning technique in $\boldsymbol{F}$ and $\boldsymbol{F'}$ are different, for example, $\boldsymbol{F}$ is a neural network and $\boldsymbol{F'}$ is a SVM.

This substitute model in effect transfers knowledge crafting adversarial inputs to the victim model. The targeted classifier is designated as an oracle because adversaries have the minimal

capability of querying it for predictions on inputs of their choice. To train the substitute model, dataset augmentation as discussed above is used to capture more information about the predicted outputs. Authors also introduces reservoir sampling to select a limited number of new inputs while performing Jacobian-based data augmentation, reduces the number of queries made to the oracle. The choice of substitute model architecture has limited impact on transferability.

The attacks have been shown to generalize to non-differentiable target models like decision trees, e.g, deep neural networks (DNNs) and logistic regression (LR) (differentiable models) could both effectively be used to learn a substitute model for many classifiers trained with a support vector machine, decision tree, and nearest neighbor. Cross-technique transferability reduces the amount of knowledge that adversaries must possess in order to force misclassification by crafted samples. Learning substitute model alleviates the need of attacks to infer architecture, learning model and parameters in a typical black-box based attack.

6.2 Training Phase Modification

A learning process fine-tunes the parameters $\boldsymbol{\theta}$ of the hypothesis $\boldsymbol{h}$ by analyzing a training set. Hence, the training dataset is potentially vulnerable to manipulations by adversaries. Barreno et al., [15] first proposed the term *Poisoning Attacks* which alters the training dataset by inserting, editing or removing points keeping the intention of modifying the decision boundaries of the targeted model following the work of Kearns et al. [36], thus challenging the learning system's integrity. The poisoning attack o the training set can be done in two ways: either by direct modification of the labels of the training data or by manipulating the input features depending on the capabilities posed by the adversary. We present a brief overview of both the techniques without much technical details, as the training phase attack needs more powerful adversary and thus is not common in general.

6.2.1 Label Manipulation

If an adversary has the only capability to modify the label information included in the training dataset, the attack surface is limited, and the adversary must find the most harmful label given the partial or full knowledge of the learning algorithm of the target model. A basic strategy is to randomly perturb the labels, i.e., select new labels for a fraction of the training data by drawing from a random distribution. Biggio et al. [25] presented a study which shows that a random flip of 40% of the training labels is sufficient to degrade the performance of classifiers learned with SVMs.

6.2.2 Input Manipulation

In this scenario, the adversary is more powerful and can corrupt the input features of training points analyzed by the learning algorithm, in addition to its labels. This scenario also assumes that the adversary has the knowledge of the learning algorithm.

Kloft et al. [37] presented a study in which they showed that inserting malicious points in the training dataset could gradually shift the decision boundary of an anomaly detection classifier. The learning algorithm that they used for the study works in an online scenario - new training data are collected at regular intervals, and the parameter values $\boldsymbol{\theta}$ are fine-tuned based on a sliding window of that data. Thus, injecting new points in the training dataset is essentially an easy task for the adversary. Poisoning data points can be found by solving a linear programming problem that maximizes the displacement of the mean of the training data.

In the offline learning settings, Biggio et al. [26] introduced an attack that inserts inputs in the training set, which are crafted using a gradient ascent method. The method identifies the inputs corresponding to local maxima in the test error of the model. The presented a study that adding these inputs to the training set results in a degraded classification accuracy at the testing time for SVM classifier. Following their approach, Mei et al. [38] introduced a more general framework for poisoning. Their method finds the optimal changes to the training set as long as the targeted learning model is trained using a convex loss (e.g., linear and logistic regression or SVMs) and its input domain is continuous.

7 Advances in Defense Strategies

Adversarial examples demonstrate that many modern machine learning algorithms can be broken easily in surprising ways. A substantial amount of research to provide a practical defense against these adversarial examples can be found in recent literature. These adversarial examples are hard to defend because of the following reasons [39]:

1. *A theoretical model of the adversarial example crafting process is very difficult to construct.* The adversarial sample crafting is a complex optimization process which is non-linear and non-convex for most Machine Learning models. The lacking of proper theoretical tools to describe the solution to these complex optimization problems make it even harder to make any theoretical argument that a particular defense will rule out a set of adversarial examples.

2. *Machine Learning models are required to provide proper outputs for every possible input.* A considerable modification of the model to incorporate robustness against the adversarial examples may change the elementary objective of the model.

Most of the current defense strategies are not adaptive to all types of adversarial attack as one method may block one kind of attack but leaves another vulnerability open to an attacker who knows the underlying defense mechanism. Moreover, implementation of such defense strategies may incur performance overhead, and can also degrade the prediction accuracy of the actual model. In this section, we discuss the most recent defense mechanisms used to block these adversarial attacks along with their respective possible drawbacks of implementation.

The existing defense mechanisms can be categorized among the following types based on their methods of implementation.

7.1 Adversarial Training

The primary objective of the adversarial training is to increase model robustness by injecting adversarial examples into the training set [9, 10, 40, 41]. Adversarial training is a standard brute force approach where the defender simply generates a lot of adversarial examples and augments these perturbed data while training the targeted model. The augmentation can be done either by feeding the model with both the original data and the crafted data, presented in [35] or by learning with a modified objective function given by [10] -

$$\widetilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon sign(\nabla_x J(\theta, x, y)), y)$$

with J being the original loss function. Such defense aims to increase the model's robustness in the directions of the adversarial perturbation by ensuring that it will predict the same class for the true example and along with its perturbations in those directions. Traditionally, the additional instances are crafted using one or multiple attack strategies mentioned in Section 6.1.1.

Adversarial training of a model is useful only on adversarial examples which are crafted on the original model. The defense is not robust for black-box attacks [19, 42] where an adversary generates malicious examples on a locally trained substitute model. Moreover, Tramer et al. [23] have already proved that the adversarial training can be easily bypassed through a two-step attack, which first applies a random perturbation to an instance and then performs any classical attack technique mentioned in Section 6.1.1.

7.2 Gradient Hiding

A natural defense against gradient-based attacks presented in [23] and attacks using adversarial crafting method such as FGSM, could consist in hiding information about the model's gradient from the adversary. For instance, if the model is non-differentiable (e.g, a Decision Tree, a Nearest Neighbor Classifier, or a Random Forest), gradient-based attacks are rendered ineffective. However, this defense are easily fooled by learning a surrogate Black-Box model having gradient and crafting examples using it [19].

7.3 Defensive Distillation

The term distillation was originally proposed by Hinton et al. [43] as a way to transfer knowledge from a large neural networks to a smaller one. Papernot et al. [33, 44] recently proposed using it as a defensive mechanism against adversarial example crafting methods.

The two-step working of the distillation method is described as below. Let us assume that we already have a neural network $\boldsymbol{F}$ which classifies a training dataset $\boldsymbol{X}$ into the target classes $\boldsymbol{Y}$. The final softmax layer in the produces a probability distribution over $\boldsymbol{Y}$. Further, assume that we want to train a second neural network $\boldsymbol{F'}$ on the same dataset $\boldsymbol{X}$ achieving the same performance. Now, instead of using the target class labels of the training dataset, we use the output of the network $\boldsymbol{F}$ as the label to train another neural network $\boldsymbol{F'}$ with the same architecture having the same input dataset $\boldsymbol{X}$. The new labels, therefore, contain more information about the membership of $\boldsymbol{X}$ to the different target classes, compared to a simple label that just chooses the most likely class.

The final softmax layer in the distillation method is modified according to the following equation:

$$\boldsymbol{F_i}(\boldsymbol{X}) = \frac{\boldsymbol{e}^{\frac{\boldsymbol{z_i}(\boldsymbol{X})}{\boldsymbol{T}}}}{\sum_{\boldsymbol{i=1}}^{|\boldsymbol{Y}|} \boldsymbol{e}^{\frac{\boldsymbol{z_i}(\boldsymbol{X})}{\boldsymbol{T}}}}$$

where $\boldsymbol{T}$ is the distillation parameter called temperature. Papernot et al. [33] showed experimentally, a high empirical value of $\boldsymbol{T}$ gives a better distillation performance. The advantage of training the second model using this approach is to provide a smoother loss function, which is more generalized for an unknown dataset and have high classification accuracy even for adversarial examples.

It has been shown that a similar behavior can be obtained at a lower computation cost by training a model using smoothed labels. The technique is known as label smoothing which involves converting class labels into soft targets. A value close to 1 is assigned for the target class and the rest of the weight distributed to the other classes. These new values are used as labels for training the model instead of the true labels. As an outcome, the need to train an additional model as for defensive distillation is relaxed.

However with the recent advancement in the black-box attack, the defensive distillation method as well as the label smoothing method can easily be avoided [19, 27]. The main reason behind the success of these attacks is often the strong transferability of adversarial examples across neural network models.

7.4 Feature Squeezing

Feature squeezing is another model hardening technique [45, 46]. The main idea behind this defense is that it reduces the complexity of representing the data so that the adversarial perturbations disappear because of low sensitivity. There are mainly two heuristics behind the approach considering an image dataset.

1. Reduce the color depth on a pixel level, i.e., encoding the colors with fewer values.
2. Use of a smoothing filter over the images. As an effect, multiple inputs are mapped to the same value, making the model robust to noise and adversarial attacks.

Though these techniques work well in preventing adversarial attacks, these have the collateral effect of worsening the accuracy of the model on true examples

7.5 Blocking the Transferability

The main reason behind defeat of most of the well-known defense mechanism is due to the strong transferability property in the neural networks, i.e., adversarial examples generated on one classifier are expected to cause another classifier to perform the same mistake. The transferability property holds true even if the classifiers have different architectures or trained on disjoint datasets. Hence, the key for protecting against a black-box attack is to block the transferability of the adversarial examples.

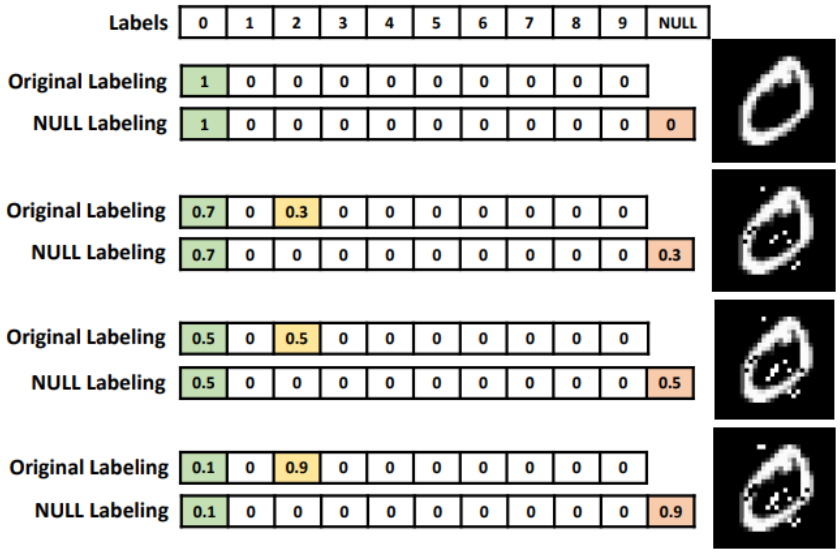


Figure 8: Illustration of NULL Labeling Method [47]

Hosseini et al. [47] recently proposed a three-step *NULL Labeling* method to prevent the adversarial examples to transfer from one network to another. The main idea behind the proposed approach is to augment a new NULL label in the dataset and train the classifier to reject the adversarial examples by classifying them as NULL. The basic working of the approach is shown in Figure 8. The figure illustrates the method taking as an example image from MNIST dataset, and three adversarial examples with different perturbations. The classifier assigns a probability vector to each image. The NULL labeling method assigns a higher probability to the NULL label with higher perturbation, while the original labeling without the defense increases the probabilities of other labels.

The NULL Labeling method is composed of three major steps:

1. **Initial Training of the target classifier:** Initial training is performed on the clean dataset to derive the decision boundaries for the classification task.

2. **Computing the NULL probabilities:** The probability of belonging to the NULL class is then calculated using a function $\boldsymbol{f}$ for the adversarial examples generated with different amount of perturbations.

$$p_{NULL} = f\left(\frac{\|\delta X\|_0}{\|X\|}\right)$$

   where, δX is the perturbation and $\|\delta X\|_0 \sim U[1, N_{max}]$. N_{max} is the minimum number for which $f(\frac{N_{max}}{\|X\|}) = 1$.

3. **Adversarial Training:** Each clean sample is then re-trained with the original classifier along with different perturbed inputs for the sample. The label for the training data is decided based on the NULL probabilities obtained in the previous step.

The advantage of this method is the labeling of the perturbed inputs to NULL label instead of classifying them into their original label. To the best of our knowledge, this method is the most effective defense against the adversarial attacks to date. This method is accurate to reject an adversarial example while not compromising the accuracy of the clean data.

As described the defense against the adversarial examples is not an easy task, and the existing defense mechanisms are only able to increase the model robustness in specific settings and to a limited extent. The design of a robust machine learning model against all types of adversarial examples is still an open research problem.

8 Conclusion and Future Challenges

References

[1] Google Cloud AI, howpublished = https://cloud.google.com/products/machine-learning/, note = Accessed: 2017-09-04.

[2] Alibaba Cloud, howpublished = https://www.alibabacloud.com/, note = Accessed: 2017-09-04.

[3] Intel Nervana Platform, howpublished = https://www.intelnervana.com/intel-nervana-platform/, note = Accessed: 2017-09-04.

[4] nVIDIA GPU Cloud Computing, howpublished = http://www.nvidia.com/object/gpu-cloud-computing.html, note = Accessed: 2017-09-04.

[5] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[6] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

[7] Battista Biggio, Giorgio Fumera, and Fabio Roli. Adversarial pattern classification using multiple classifiers and randomisation. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 500–509, 2008.

[8] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE transactions on knowledge and data engineering*, 26(4):984–‘–996, 2014.

[9] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.

[11] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[12] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[13] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. *arXiv preprint arXiv:1702.07464*, 2017.

[14] Igino Corona, Giorgio Giacinto, and Fabio Roli. Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues. *Information Sciences*, 239:201–225, 2013.

[15] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D Joseph, and J Doug Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM, 2006.

[16] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[17] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against rnns and other api calls based malware classifiers. *arXiv preprint arXiv:1707.05970*, 2017.

[18] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security Symposium*, pages 601–618, 2016.

[19] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519, 2017.

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

[21] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 3–18. IEEE, 2017.

[22] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[23] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. *CoRR*, abs/1705.07204, 2017.

[24] Giuseppe Ateniese, Giovanni Felici, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, and Domenico Vitali. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *CoRR*, abs/1306.4447, 2013.

[25] Battista Biggio, Blaine Nelson, and Pavel Laskov. Support vector machines under adversarial label noise. In *Proceedings of the 3rd Asian Conference on Machine Learning, ACML 2011, Taoyuan, Taiwan, November 13-15, 2011*, pages 97–112, 2011.

[26] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[27] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

[28] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. *CoRR*, abs/1608.08182, 2016.

[29] Qinglong Wang, Wenbo Guo, Kaixuan Zhang, II Ororbia, G Alexander, Xinyu Xing, C Lee Giles, and Xue Liu. Learning adversary-resistant deep neural networks. *arXiv preprint arXiv:1612.01401*, 2016.

[30] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[31] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. Towards the science of security and privacy in machine learning. *CoRR*, abs/1611.03814, 2016.

[32] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE Trans. Knowl. Data Eng.*, 26(4):984–996, 2014.

[33] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 582–597, 2016.

[34] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[35] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *CoRR*, abs/1611.01236, 2016.

[36] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM J. Comput.*, 22(4):807–837, aug 1993.

[37] Marius Kloft and Pavel Laskov. Online anomaly detection under adversarial impact. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 405–412, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[38] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2871–2877. AAAI Press, 2015.

[39] OpenAI: Attacking Machine Learning with Adversarial Examples, February 2017.

[40] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *CoRR*, abs/1511.05432, 2015.

[41] Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. A unified gradient regularization family for adversarial examples. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 301–309, 2015.

[42] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops, Honolulu, HI, USA, July 21-26, 2017*, pages 1310–1318, 2017.

[43] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[44] Nicolas Papernot and Patrick D. McDaniel. Extending defensive distillation. *CoRR*, abs/1705.05264, 2017.

[45] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *CoRR*, abs/1704.01155, 2017.

[46] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *CoRR*, abs/1704.01155, 2017.

[47] Hossein Hosseini, Yize Chen, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. Blocking transferability of adversarial examples in black-box learning systems. *CoRR*, abs/1703.04318, 2017.