Software Engineering Institute
Carnegie Mellon University

MEASURING AND COMPARING ROBUSTNESS OF ML ALGORITHMS UNDER ADVERSARIAL ATTACK

August 2017

Introduction

A machine learning algorithm can be evaluated for robustness against any number of different types of attacks. We consider attacks that seek to manipulate the training and/or testing data inputs to a machine learning algorithm. Specifically, we do not consider physical attacks on machines hosting the algorithm.

A framework for measuring robustness

In order to understand robustness in quantifiable terms, we need a precise definition for "robustness."

A given machine learning algorithm *A* uses samples *X* to predict outcomes *Y* as $\hat{Y} = A(X)$. A perfect prediction $\hat{Y} = Y$ provides some utility to the user. As there may be many factors influencing our definition of "utility" (effectiveness, risk, accuracy, precision, reliability, some combination of these factors, etc.), we will not attempt to provide a specific definition here. We assume that the user defines a utility function *U* dependent on the algorithm *A* and data *X*, represented as *U(A, X)*. The dependency on *X* will be described in the following section.

Adversaries may seek to manipulate the data so as to reduce the utility of the algorithm. If an adversary uses a manipulation *M* to transform data *X*, then the predictions are $\hat{Y}_M = A\big(M(X)\big)$, such that *U(A, X) > U(A, M(X))*, with occasional fortuitous exceptions.

The algorithm *A* is robust with respect to manipulation *M* to the extent that *M* does not decrease the utility. We define the robustness of an algorithm with respect to input *X* and manipulation *M* as

$$Robustness(A, X, M) := \frac{U\big(A, M(X)\big)}{U(A, X)} \tag{1}$$

We suppose that $U(A, X) > 0$ and that $U\big(A, M(X)\big)$ typically ranges from 0 to $U(A, X)$. It is not difficult to imagine alternative formulations of robustness in terms of *A*, *M*, and *X* if these assumptions fail.

Comparing robustness to adversarial attacks

Given some algorithm utility function *U*, suppose we have a list of algorithms $A_1, \dots, A_k$ and a list of manipulations $M_1, \dots, M_m$. A complete comparison of the robustness of the algorithms with respect to the manipulations requires performing $k \times m$ experiments pairing each of the *k* algorithms with each of the *m* manipulations. Note that *m* is usually much greater than *k*.

It is reasonable to hypothesize that some classes of algorithms may be more vulnerable to particular types of manipulations, while other algorithms are vulnerable in different ways. This structure gives us a way to measure, document, and discuss these tradeoffs.

Since the robustness of each algorithm may vary across the manipulations, deciding which algorithm is the "most robust" in general would require reducing each algorithm's performance history to a single summary statistic. For example, one could simply average the robustness of each algorithm across its *m* manipulations. A weighted average might be preferable if the various manipulations are of varying importance or perceived likelihood of occurring in real life.

Active learning can be used to find new manipulations that minimize or maximize (depending on the use case) the robustness of the set of algorithms.

Measuring tradeoffs between robustness and performance

The definition in Equation 1 allows us to approach this in terms of an arbitrary algorithm utility function *U* and manipulation *M*. Suppose *q* is the probability that *M* gets applied to an arbitrary future input *X*. Then, taking the possibility of manipulation into account, the expected utility of an algorithm is

$$E_M\big(U(A)\big) = qU\big(A, M(X)\big) + (1-q)U(A, X) \tag{2}$$

Now suppose $A' = H(A)$ is a "hardened" version of *A* that is intended to be less vulnerable to manipulations of the future input data *X*. Typically there will be both a cost and a benefit related to that hardening.

The cost applies when no manipulation occurs:

$$cost(H) = U(A, X) - U(A', X) \tag{3}$$

The benefit applies when manipulation occurs:

$$benefit(H) = U\big(A', M(X)\big) - U\big(A, M(X)\big) \tag{4}$$

The expected net improvement in utility resulting from hardening A is

$$\begin{aligned} U(A') - E_M\big(U(A)\big) &= q\left(U\big(A', M(X)\big) - U\big(A, M(X)\big)\right) + (1-q)\big(U(A, X) - U(A', X)\big) \\ &= q * benefit(X) - (1-\pi)cost(H) \end{aligned} \tag{5}$$

This expected net improvement is positive if an only if

$$benefit(H) > \tfrac{1-q}{q}cost(H) \tag{6}$$

Notice that the right-hand side → 0 as $q \to 1$. It follows that a hardening with great cost and small benefit can be worthwhile overall if the probability of manipulation is high.

This basic analysis makes it easy to see that assessing the tradeoffs between robustness and performance is no simple matter. Depending on the application,

- the probability q may be unknown, requiring a user to illustrate or integrate the cost-benefit analysis across a range of probabilities
- there could be any number of potential manipulations, requiring generalization of the analysis above

Characterizing and Constructing Adversarial Inputs

Understanding how to characterize input requires an understanding of modern adversarial AI techniques. The following sections outline adversarial AI capabilities and techniques for characterizing inputs.

Adversarial AI

Attacking a ML/AI system involves finding specific inputs where the algorithm "misbehaves," providing unexpected (or "lower utility" output) for a given input. The majority of ML/AI today relies on statistical assumptions about the nature of the data; it should be drawn from a given distribution, it should not violate some constraints, it should be differentiable along some specific set of parameters. Broadly speaking, attacks attempt to find violations of these assumptions. One very public example of this type of attack involved a Microsoft chat bot, Tay. This bot was driven by an algorithm which used user input to build a language dictionary. Unfortunately, the algorithm architects did not envision that the algorithm would be polluted with a high volume of "bad" data; namely, text with strong racist tendencies. As such, that input was overly expressed in the training data and thus dominated the output, leading to some highly offensive algorithmically generated tweets [1].

Attacks can happen both during the training and testing phases. Training attacks aim to manipulate the nature of the data used to train the classifier, whether through overrepresenting particular types of data, modifying labels, or other techniques. The attack against the Microsoft chatbot represents this type of attack. Testing phase attacks exploit algorithm decision boundaries not well defined by training datasets. A recent widely-reported research paper described how self-driving cars were made to interpret stop signs as speed limit signs using sticky notes [2]. Note that the recent popularity of active learning techniques, where a trained model is continuously improved over time through the addition of new data, blurs the lines between these two attacks.

Historically, ML/AI attacks were generated through brute force or insights into the nature of the training phase. Using tools conceptually similar to fuzz testing utilities, attackers would generate input that they deemed likely to generate erroneous output and exploit their findings. These techniques have matured significantly over the past few years. Modern adversarial AI techniques involve generating *new* AI algorithms with the specific goal of finding miscategorizations in the target ML/AI implementations [2,3].

These attacks generally work as follows. Given a black-box classifier algorithm A, where the attacker can query the model for any input X but has no other information about the model, we want to find perturbations to X resulting in a model misclassification. A can be a regression, SVM, decision tree, neural network, or almost any other statistical model, and there are no restrictions on data type. This input should follow three guidelines: (1) be as small as possible, (2) require as few queries to the model as possible, and (3) be as undetectable as possible. The attacker will generally train a secondary model to create adversarial data that meets these criteria using a variety of published techniques (e.g., [3-5]). One specific technique involves examining "gradients"—the extent to which an image changes across a wide range of features—and finding classification boundaries with very large gradients. This implies that a tiny change in the image can significantly impact classification, which is a likely sign of an area of weakness.

Input characterization

Given the attacker goals of influencing algorithm A with minimal changes to input A, characterization of changes will depend directly on the extent to which X is modified. While this is defined in abstract in the previous section, practical definitions often rely on well-understood distance functions on which an optimization algorithm can operate, such as the L_1 and L_2 norm. Given the large parameter space typically used in modern algorithms, X may have very high dimensionality.

Additionally, the input is defined by the nature of the attack, as hinted in the previous section. Whether the attack occurs during training or testing, the number of queries used to successfully identify an exploit, whether a fuzzer-style technique or an adversarial AI was used to generate the exploit, the complexity of the adversarial model… all of these factors influence both the effectiveness of the model and the complexity in its execution, and as such should be considered when determining the risk posed by a given potential attack.

Other ML/AI attacks

Note that this does not involved poor coding, vulnerabilities in the supporting architecture, hardware attacks, or the like. These techniques find specific weaknesses in the algorithm as implemented. All ML/AI algorithms are susceptible to a wide variety of standard computing vulnerabilities, ranging from buffer overflows to denial of service attacks on critical servers. These are not considered here.

There exists another type of attack with ML/AI algorithms relating to information leakage. Carefully constructed queries to an ML/AI algorithm can potentially reveal information about the training set used to construct the model, which may contain confidential or otherwise restricted information. While this type of attack can be highly damaging and does reveal shortcomings in the design of the ML/AI architecture, it does not affect the efficacy of the ML/AI model, and as such is not being considered here.

Theoretical Basis for the Construction of Training and Execution Attacks

"Adversarial examples are hard to defend against because it is difficult to construct a theoretical model of the adversarial example crafting process. Adversarial examples are solutions to an optimiza-

tion problem that is non-linear and non-convex for many ML models, including neural networks. Because we don't have good theoretical tools for describing the solutions to these complicated optimization problems, it is very hard to make any kind of theoretical argument that a defense will rule out a set of adversarial examples." (https://blog.openai.com/adversarial-example-research/)

In light of the above quote, there are a number of theoretical and practical approaches to creating adversarial examples. As described in the previous section, the prototypical adversarial example is an input plus a minor perturbation, such that the consequence is seemingly indistinguishable from the original input, but the output of the ML is markedly different than the original. One highly-cited published techniques is the fast gradient sign method [3]. This technique demonstrates that neural networks mimic linear behavior locally. In the paper, linear models are demonstrated to be particularly susceptible to adversarial inputs. It is then easy to quickly construct a new adversarial example A' using some input X via $A' = X + (\varepsilon * \mathrm{sign}(\nabla_x J(\theta, X, y))$, where $J(\theta, X, y)$ represents the cost function of the model and ∇_x represents the gradient. This is of particular interest because (1) it is demonstrated that an adversary can quickly and cheaply generate a vast amount of adversarial examples, which is statistically likely to succeed, and (2) that the linear nature of the individual components of neural networks which makes them simple to analyze and construct introduce an inherent weakness on a local level. Other techniques, such as adversarial saliency maps, have also been described in the literature as possible vectors of attack.

One of the most simple and straightforward attacks on ML during training time is the addition or substitution of mislabeled training data to the training set. The validity of the training set is often taken for granted, especially for a very large set, where a false addition or substitution can easily go unnoticed. There are some investigations of the robustness of SVMs to "label contamination":

- Huang Xiao, et al. "Support vector machines under adversarial label contamination" Neurocomputing, Volume 160, 21 July 2015, Pages 53-62)
- Battista Bigio, et al. "Support Vector Machines Under Adversarial Label Noise" JMLR: Workshop and Conference Proceedings 20 (2011) 97–112

Contact Information

Eliezer Kanal
Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412-268-5204
Email: ekanal@sei.cmu.edu
Web: www.sei.cmu.edu | www.cert.org

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.