CS 261 Notes: Attacks on Machine Learning Systems

Scribe: Samee Ibraheem

October 22, 2018

1 Introduction

With the great success of machine learning (ML) applications in a variety of areas, such systems are experiencing more general use, making them increasingly susceptible to security breaches. There has thus been wide interest in research demonstrating these types of vulnerabilities in ML systems, specifically with regards to the privacy and robustness of such systems. In these notes we focus on supervised learning, which we now describe.

1.1 Supervised Learning

Supervised learning is the process through which models are trained to predict the corresponding output for a given input. Here, "supervised" refers to the fact that these models are provided with the true output labels during training. The process proceeds as follows:

1. A collection of input-output example pairs comprise the dataset on which a machine learning model is trained. During training, the model attempts to minimize the *loss function*, which is a function of the model's predicted output and the true output label for each input.
2. After training, the model is presented with unlabeled inputs for which predictions are made.

Figure 1 illustrates these steps. In addition, inputs may go through a *feature extraction* process before being fed into the model, in which case the model is run on the resulting feature vector, rather than on the input itself.

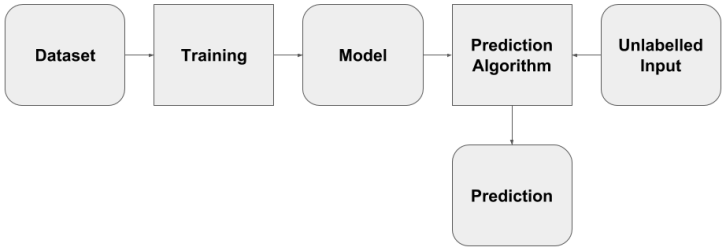

Figure 1: Illustration of the supervised learning process

1.2 Security Goals

With regards to security features that are desired of supervised learning systems, these fall under two categories:

1. **Privacy**: Entities should be able to keep their contributions to the system private from other entities. For example:

   - Users should be able to protect their input data/feature vectors from the model provider and other users. This may be during either the training or the prediction phase.
   - Model providers should be able to protect their models and datasets from those who use their system.

   Supervised learning models affected along this axis are said to be susceptible to *privacy attacks*.

2. **Accuracy/Robustness**: Models should be resistant to attempts to produce unintended behaviors in the system. For example:

   - Attackers should not be able to affect the model's prediction results. This is critical for applications such as autonomous driving, cancer detection, etc.
   - The training algorithm/model should function as expected.
   - The dataset should be unsusceptible to influence by malicious parties.

   Supervised learning models affected along this axis are susceptible to *poisoning attacks*.

1.3 Threat Models

The types of threat models that affect supervised learning systems depend on a variety of factors, including:

- **Who can see what**
  - Maybe only the government should be able to see the model, but not specific agencies.
  - Maybe only the user should be able to see the data, but not the cloud.
- **What is the attacker's power**
  - Maybe an attacker can only insert a few examples, or maybe they can insert many examples.

Unfortunately, there are many papers that focus on attacking such systems, but not as many papers that focus on defense. In fact, usually defenses are proposed, and then those defenses are themselves attacked. They may also have many assumptions that are not realistic.

2 Attacks to Model Privacy

In this section, we focus on privacy attacks aimed at the machine learning model.

2.1 Why extract a model?

There are several possible motivations for an attacker to want to extract an ML model:

- **Monetary reasons**: If an attacker is able to extract a commercial model, then they may re-sell the model for cheaper than the original. They can also gain free, unlimited use of the model for themself.

- **Learn information about the training set**: A malicious party might extract a model in order to gain information about the data on which the model was trained, for which white-box access to the model can allow for easier extraction.
- **Model evasion**: An attacker that is able to extract a model can also learn how to avoid detection by it, which can invalidate the purpose of certain models, such as those that are used for spam detection.

2.2 Attacks

Say that we have a machine learning model f, with black-box input x with d features, true output label y, and $f(x)$ being the predicted output label, as well as the confidence score. We may also have a feature extraction function ex, such that our output is $f(ex(x))$. Then the goal of the attacker is to find an f' which approximates f with low error. Specifically, given a distance metric d, the attacker wishes to minimize

$$R_{\text{test}} = \sum_{(x,y)\in D} \frac{d(f(x), f'(x))}{|D|},$$

where D is the test set. Alternatively, we may wish to minimize

$$R_{\text{uniform}} = \sum_{(x,y)\in U} \frac{d(f(x), f'(x))}{|U|},$$

where U is a set of vectors chosen uniformly from the input space. In either case, d may take a variety of forms. For example, the L_p distance metric may be used (described in Section 3.1).

Tramèr et al. describe equation-solving techniques that work for extracting logistic regression and multilayer perceptron models, for which the relationship between output confidence metrics and inputs are determined by a continuous equation [4]. For decision trees, on the other hand, these metrics share a more complex relationship with the inputs. The authors thus describe what they call a *pathfinding attack* to extract such models.
We assume that the attacker knows the ranges of values that can be assumed for each feature. Let leaf ids be determined by their output label and confidence score. The goal is to find all of the leaves of the tree, as well as the conditions that must be satisfied for each branch of the tree. The algorithm proceeds as follows:

Result: Recovered Decision Tree (Approximation)
$x_{\text{init}} \leftarrow$ random $\{x_1, ..., x_d\}$
$Q \leftarrow \{x_{init}\}$
while *Q not empty* **do**
  $x \leftarrow Q.pop()$
  for *feature in x* **do**
    if *continuous* **then**
      Line Search (LS)
    else
      Categorical Search (CS)
    end
  end
end

Table 1 demonstrates how the algorithm may be used for the tree shown in Figure 2. We assume that $0 \leq SIZE \leq 100$ and $COLOR \in \{R, G, B, Y\}$. We also specify a parameter ϵ to represent our level of precision in terms of finding branch conditions.

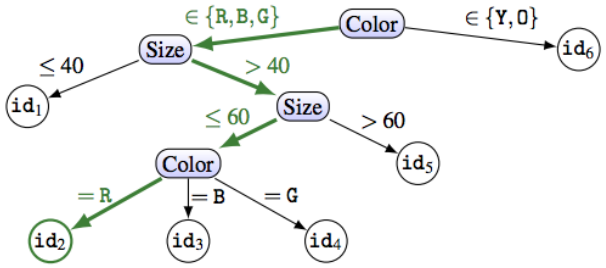

Figure 2: Decision tree from [4]

STEP	SIZE	COLOR	LEAF	OUTCOME
[0]	50	RED	2	0 <X_SIZE <100
[LS1]	0	RED	1	Add to Q
	100	RED	5	Add to Q
[LS2]	25	RED	1	
	37.5	RED	1	
	43.75	RED	2	40 <X_SIZE <100
[LS3]	75	RED	5	
⋮	⋮	⋮	⋮	⋮
	56.25	RED	2	40 <X_SIZE <60

Table 1: Pathfinding attack process. First, a random query of $SIZE = 50, COLOR = RED$ is initiated. This leads to the leaf with label 2. We begin by varying the $SIZE$ feature, which is continuous, thus beginning a Line Search. We start with the extremes of our range, first with 0 on the lower end. Since we reach a leaf that we have not seen before, this query is added to our Q. We continue with this process until we find a query that results in the same label 2 within our specified level of precision $\epsilon = 10$. A similar procedure occurs for our categorical variable $COLOR$, except with a categorical search, rather than a line search. For a categorical search, the categorical variable is varied to be each possible categorical value.

2.3 Defenses

Several preventative measures have been proposed for the above mentioned attacks:

- **Reveal less information**: Through rate-limiting the number of queries that can be made in a short amount of time, making it more costly to issue queries, rounding output confidence scores, or not outputting confidence scores at all, the efficiency of the above attacks may be greatly reduced.
- **Differential privacy**: By bounding a certain amount of information from being leaked, it may be possible to prevent against such attacks. Though differential privacy usually refers to the training data, in this case it would apply to the model parameters.
- **Ensemble of models**: Instead of having a single model, one can use an ensemble of several models (for example, random forests) in order to protect against such attacks.

3 Attacks to Model Robustness

In addition to attacks that affect a model's privacy, supervised learning models may also be susceptible to robustness-based attacks. For example,

- **Corrupting pixel values**: For a computer vision application, noise can be added to a correctly classified input image in such a way that it is imperceptible to humans, but that would fool a supervised model into misclassifying the resulting image.
- **Corrupting pixel locations**: Alternatively, pixel values may be kept the same, but rotated relative to the original image, producing a misclassification.
- **Taking advantage of model biases** Pictures of objects may be taken so that they exploit the learned biases of the model and lead to a misclassification.

The above perturbations fall under the category of *adversarial examples* to a model. The goal of such examples is to trick the model into classifying a given input differently from the same input under a minimized perturbation. There are several distance metrics that one might consider with regards to such perturbations, which all derive from the following formula.

3.1 The L_p Metric

The L_p metric can be defined as below:

$$||v||_p = \left(\sum_{i=1}^{d} |v_i|^p \right)^{\frac{1}{p}},$$

with each v_i being a feature of a d-dimensional input vector. For our adversarial application of this metric, $v_i = x_i - x_i'$, with x_i being a feature of the original input, and x_i' being the same feature of the perturbed input. From the above, we can derive the following metrics.

The L_1 metric may be derived as follows:

$$||v||_1 = \sum_{i=1}^{d} |v_i|.$$

Intuitively, this corresponds to the taxicab distance between the two inputs. The L_1 metric is useful for inducing the sparsity of v.

The L_2 metric may be derived as follows:

$$||v||_2 = \sqrt{\sum_{i=1}^{d} |v_i|^2}.$$

Intuitively, this corresponds to the Euclidean distance between the two inputs. The L_2 metric is useful due to its computational efficiency and resistance to outliers.

We can also consider the L_∞ metric as below:

$$||v||_\infty = \max\{|v_1|, |v_2|, ..., |v_d|\}.$$

Intuitively, as p grows large, the L_p metric is greatly determined by the largest of the v_i's. Thus, the result approaches

$$(\max\{|v_1|, |v_2|, ..., |v_d|\}^p)^{\frac{1}{p}} = \max\{|v_1|, |v_2|, ..., |v_d|\}.$$

This metric is useful for enforcing a cap on the distance.
Given these metrics, we may now determine what is the minimal perturbation needed for an adversarial example.

3.2 Finding Perturbations

Rather than randomly perturbing an input, one may actually perturb it in the direction of steepest descent using the gradient:

$$\nabla_x f(x),$$

where ∇_x is the gradient with respect to the input x, and f is the model, to which we are assuming white-box access. If this is not the case, we may use the model extracting techniques described in the previous section before proceeding with these attacks. Alternatively, it is possible to construct an adversarial example for a separate model and, with reasonable probability, it will successfully attack the black-box model.

In fact, several attacks have been developed using the gradient.

3.2.1 Gradient Descent

Adversarial examples may be derived using the gradient as follows:

$$x_{\text{adv}} = x - \eta \nabla_x \text{loss}(p(y|x), y_{\text{true}}),$$

where x is the original input, x_{adv} is the resulting adversarial example, y_{true} is the true output for x, η is a chosen step size, loss is the loss function used when training the model, and $p(y|x)$ is the probability assigned to the output label y by the model given the input x, which follows the softmax distribution

$$p(y = i|x) = \frac{\exp(z_i)}{\sum_{j=1}^{\#\text{ classes}} \exp(z_j)},$$

where z_i are the logits or scores for each possible output label i.

3.2.2 Fast Gradient Sign Method (FGSM)

Instead of calculating the precise gradient, the sign may be used as follows:

$$x_{\text{adv}} = x - \eta \text{sign}(\nabla_x \text{loss}(p(y|x), y_{\text{true}})),$$

3.2.3 Projected Gradient Descent/Iterative FGSM

Finally, the above method can be made iterative as shown below:

$$x_i = x_{i-1} - \eta \text{sign}(\nabla_{x_{i-1}} \text{loss}(p(y|x_{i-1}), y_{\text{true}})),$$

with the adversarial example being the final result of this process.

3.3 Defenses

There are a few methods that have been proposed for defending against adversarial examples:

3.3.1 Adversarial Training

Madry et al. proposed adversarial training as a defense against adversarial examples [2]. Under this defense paradigm, a model is first trained using the original training data. Then, after adversarial examples have been identified for this model, based on the attacks mentioned above, the model is trained on these adversarial examples.

3.3.2 Defensive Distillation

Another defense mechanism that has been proposed for adversarial examples is call defensive distillation [3]. Rather than using the softmax distribution described above, we have:

$$p(y = i|x) = \frac{\exp(z_i/T)}{\sum_{j=1}^{\#\text{ classes}} \exp(z_j/T)},$$

where T is a chosen temperature parameter. Intuitively, a smaller T produces a more peaky distribution, whereas a larger T produces a flatter distribution, and $T = 1$ gives the original softmax distribution. This approach uses a smaller T, hence yielding a more sharp distribution for which adversarial examples are harder to find. However, Carlini & Wagner demonstrate that this method is still susceptible to attack [1].

References

[1] Carlini, Nicholas and Wagner, David. "Towards evaluating the robustness of neural networks." 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017.

[2] Madry, Aleksander et al. "Towards deep learning models resistant to adversarial attacks." arXiv preprint arXiv:1706.06083. 2017.

[3] Papernot, Nicolas et al. "Distillation as a defense to adversarial perturbations against deep neural networks." 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016.

[4] Tramèr, Florian et al. "Stealing machine learning models via prediction APIs." USENIX Security Symposium. 2016.