Report on adversarial attacks

Mark Tropin

April 1, 2020

1 State-of-the-art

1.1 Transferability in Machine Learning: [PapernotMG16]

1. Adversarial attacks on one model can often work for a different model, given that these models perform the same task (their architectures and training sets can be different!)
2. An adversary can attack his/her own *substitute* model, then perform a real-world attack.
3. The authors present a technique called *reservoir sampling* that uses the victim's classifier as an *oracle* to create a synthetic training set for the substitute model.

The *transferability* of an adversarial attack is defined; the authors proceed by claiming that:

- the substitute model can be trained with other techniques than deep learning
- transferability-based black box attacks are not restricted to deep learning targets and are successful with targeted models of many machine learning types

An adversarial example $\vec{x^*}$ is defined as the *smallest* modification possible that allows the classifier f to misclassify a legitimate input $\vec{x}$:

$$\vec{x^*} = \vec{x} + \delta_{\vec{x}} \quad \texttt{where} \quad \delta_{\vec{x}} = \underset{\vec{z}}{\operatorname{argmin}} f(\vec{x} + \vec{z}) \neq f(\vec{x})$$

(A common challenge for adversarial attacks is the lack of differentiable functions (e.g., decision trees): this is probably linked to the fact that we cannot perform gradient *ascent* to increase the value of the loss function to misclassify the input.)

The authors then define two kinds of adversarial sample transferability: *intra-technique* and *cross-technique.* The former works for the same kind of ML models (only decision trees, only neural networks, $\cdots$), while the latter works for any model.

Two hypotheses are then defined:

Hypothesis 1: Both intra-technique and cross-technique adversarial sample transferabilities are consistently strong phenomena across the space of machine learning techniques.

Hypothesis 2: Black-box attacks are possible in practical settings against any unknown machine learning classifier.

In order to validate the first hypothesis, the authors train 5 ML algorithms on the famous MNIST dataset. The models are DNNs, LR, SVMs, DTs and kNNs. This array of models was chosen because it is representative of the Machine Learning spectrum: some models are differentiable while others are not; one of the models is log-linear while others are not linear at all (LR); and one model performs "lazy" classification (kNN $\rightarrow$ no learning).

In the first experiment, the authors train 5 models of each kind (i.e., 25 models in total) on 5 different subsets of the MNIST dataset. After creating adversarial examples that are *tailored to one model*, the authors apply the same adversarial attacks to other models trained on different training sets. For example, the adversarial examples that fooled one DNN are *transfered* to the other 4 DNNs. This is done to show that the difference in training sets (and thus the difference in the parameters) doesn't increase the robustness of image classifiers: well-trained classifiers (with error rate $\sim 3 - 10\%$) fail drastically on adversarial examples ($30 - 98\%$ ad. examples misclassified). However, not all models fail equally: in the data provided by the authors, we can see that differentiable models are much more vulnerable to adversarial attacks than non-differentiable ones.

In the second experiment, the authors do not partition the training set and train 5 different models on the same set. They then proceed by generating adversarial examples for one type of model and attacking other models using these examples. The difference between this and the first experiment is that here all the models use *different* algorithms: e.g., having generated adversarial examples for DNNs, we attack SVMs, kNNs, etc. The authors also add one more target ML technique: an ensemble, which combines the 5 aforementioned ML techniques and chooses the class proposed by the majority. The results of this experiment are presented in the form of a matrix where the rows represent the source ML technique (that the adversarial samples were crafted for) and the columns represent the target ML technique that is attacked. Even though one model stands out (DNNs: error rate $0.82 - 38\%$), other models produce failure rates of up to a 100%. It's worth noting that the ensemble was not the most resistant model: this can be explained by the vulnerabilities of the components of the ensemble (if we can observe the transferability of AA, using ensembles doesn't make any sense).

The authors conclude this section by saying that cross-transferability is *key* for black-box attacks: this means that we don't need to know the parameters of the model (or its type) to carry out a successful adversarial attack.

The next section tackles efficient methods of constructing substitute models by quering the victim's classifier. The authors modify an already existing algorithm by increasing its accuracy while reducing the computational complexity.

The victim's classifier is assumed to be accessible as an oracle that provides *labels, not probabilities* of the samples' classes.

Jacobian-based dataset augmentation: an initial substitute training set is collected and is labelled by querying the oracle. This initial training set is augmented as follows:

$$S_{\rho+1} = \{\vec{x} + \lambda_\rho \cdot sgn(J_f \left[\tilde{O}(\vec{x})\right] : \vec{x} \in S_\rho)\} \cup S_\rho$$

where $S_\rho, S_{\rho+1}$ are the previous and current training sets, λ_ρ a parameter for fine-tuning the step size, J_f the Jacobian matrix of substitute f, and $\tilde{O}(\vec{x})$ the oracle's label for sample $\vec{x}$.

We continue querying the oracle based on the set S_ρ. As it was shown in a different paper by Papernot, substitute DNNs can thus approximate original DNNs.

Periodical step size: It has been shown that we can obtain a better approximation of the oracle model if we *vary* the step size by using periodically alternating positive and negative values. Thus, the step size is defined as:

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

where τ is the number of epochs after which the Jacobian-based dataset augmentation does not lead to any substantial improvement of the substitute model.

Reservoir sampling: This technique is particularly useful in a real-world setting where the number of queries to the oracle is limited. Without this technique, the number of queries would grow exponentially. We thus have to randomly choose examples from the set $S\rho$. With $\rho > \sigma$, the number of queries is reduced from $n \cdot 2^{\rho}$ to $n \cdot 2^{\sigma} + \kappa \cdot (\rho - \sigma)$. In order to fill the set $S\rho$, we need to do the first σ iterations normally (without reservoir sampling).

In the next section, the authors show that DNNs are a good choice for the substitute model because they can *transfer* the knowledge from the oracle, whatever its original model. The authors compare DNN substitutes and show that adding PSS improves the accuracy of the model, while RS doesn't reduce the accuracy greatly, while decreasing the number of iterations radically.

1.2 Explaining and Harnessing Adversarial Examples: [goodfellow2014]

Main points:

- Training on adversarial examples can regularize the model.
- For high dimensional problems, we can make many infinitesimal changes to the input that add up to one large change to the output.
- $\uparrow$ This means that not only DNNs are vulnerable to adversarial attacks because of their non-linear nature: linear models are vulnerable too.
- "fast gradient sign method": $\boldsymbol{\eta} = \epsilon \cdot sign(\nabla_x J(\boldsymbol{\theta}, \mathbf{x}, y))$

- Alternative: rotating $\boldsymbol{x}$ by a small angle in the direction of the gradient reliably produces adversarial examples.
- Data augmentation: by training on a mixture of adversarial and clean examples, a neural network can be regularized somewhat.
- Adversarial objective function based on the fast gradient sign method is defined: see paper for formula.
- Resisting AA: make the model larger (in units per layer) $\Rightarrow$ more overfitting, more errors on training set, but more resilient to adversarial attacks (error decrease from 89.4% to 17.9%). However, this kind of model can be overconfident when failing.
-

Szegedy writes that we are working with finite precision (everything below 1/255 is ignored) -¿ this verifies my hypothesis that the image has to be downsampled (at least everything below 1/255 has to be scraped)

Szegedy: adversarial examples expose fundamental blind spots in our training algorithms

One of the benefits of using adversarial training is increased regularization. Classic regularization techniques don't prepare our models for adversarial examples

There is a trade-off between simple linear models and complex models, resistant to adversarial examples

Transferability!

Training on adversarial examples can regularize the model, but in case of underfitting, adversarial training can simply worsen underfitting.

Adversarial training (obviously) increases the error rate for non-adversarial inputs.

The adversarial training procedure can be seen as minimizing the worst case error when the data is perturbed by an adversary.

Adding small noise to the training data doesn't make the model more robust ("no effect").

The derivative of the sign function is not defined for most points, which makes adversarial training even harder: it's not easy to anticipate the perturbations introduced by the adversary.

Perturbing the hidden layers of the network may work better than perturbing the input.

1.3 Measuring the Transferability of Adversarial Examples: [petrov2019]

- Three CNN architectures are defined: VGG16, VGG19 and Inception.
- Attack methods are defined: FGSM, IFGSM, C&W
- The author attacks the CNNs and comes to the conclusion that ensembles of CNNs are not more robust to AAs than one NN.

1.4 On the Robustness of Semantic Segmentation Models: [arnab2017]

- Difference in architecture: residual connections are better than chain-like networks
- Adversarial examples do not transfer well across different scales and transformations.

- CRFs, Conditional Random Fields
- Multiscale processing

1.5 Adversarial Attacks: and Defences Against Deep Neural Networks: A Survey: [ozdag2018]

- Two kinds of attacks are defined:
  - non-targeted AAs: the classifier misclassifies the image (the attributed class is any class save the original).
  - targeted AAs: the image is misclassified as a specified target class (e.g., impersonation)
- Attack methods defined: Fast Gradient Sign Method, Projected Gradient Descent, Basic Iterative Method, Carlini-Wagner, Jacobian-based Saliency Map Approach
- NIPS 2017 Competition Results: new AA methods were created: Momentum-based Iterative FSGM, Iterative FGSM-based Approach
- Two defense methods are defined: High-level Guided Denoiser, Randomization-based Method

2 Models

In order to compare the robustness of image classifiers to adversarial attacks, I have chosen three Machine Learning models. The choice of these models was not only based on their popularity and heavy use in research and industrial applications, but also on the fact that these 3 models are *representative* of the ML spectrum. Each model employs a distinct statistical learning algorithm whose mathematical properties are *key* to understanding their robustness to adversarial attacks (or, as we will see in some cases, lack thereof). These models are:

- Deep Neural Networks (DNN)
- Support Vector Machines (SVM)
- Decision Trees (DT)

This section provides a comprehensive description of these ML techniques.

2.1 Deep Neural Networks

DNNs are by far one of the most popular algorithms in modern-day Machine Learning. Despite their enormous resource intensiveness and the need of vast amounts of training data, DNNs are in very high demand because of their success on many machine learning tasks, including image classification. In fact, the modern rise of Deep Learning is very often linked to its performance in the ImageNet 2012 challenge, when the AlexNet DNN surpassed all other competing models by a **10.8%** margin.

This, however, was not the birth of a new approach, but rather the *renaissance* of an old one. The origins of the DNNs we use today can be traced to the *Perceptron*, designed by Frank Rosenblatt in 1957.

A perceptron is a *"shallow"*[1] model that takes as input a vector in $\mathbb{R}^n$ and produces a scalar output. In the case of binary classification, this output could be thresholded in order to decide

[1]We use the word 'shallow' to indicate that the model is not *deep*: it consists of only one layer of neurons. Alternatively, one could say that there are no hidden layers, just the input layer and the output layer.

whether a given sample belongs to one class or another. More formally,

$$\mathbf{w^T v} < b \Rightarrow v \in C_0 \quad \text{and} \quad \mathbf{w^T v} > b \Rightarrow v \in C_1$$

$\mathbf{w}$ and b are the *parameters* of the model that are *learned* during the training phase[2]. In order to simplify the notation, these parameters can be represented by a single *weight vector* $\hat{\mathbf{w}} = (-b, \mathbf{w})$, and with $\hat{\mathbf{v}} = (1, \mathbf{v})$, we can rewrite the previous formulas as

$$\hat{\mathbf{w}}^{\mathbf{T}}\hat{\mathbf{v}} < 0 \Rightarrow v \in C_0 \quad \text{and} \quad \hat{\mathbf{w}}^{\mathbf{T}}\hat{\mathbf{v}} > 0 \Rightarrow v \in C_1$$

Finally, this formula can be rewritten with the use of the *sign* function:

$$sgn(\hat{\mathbf{w}}^{\mathbf{T}}\hat{\mathbf{v}}) = -1 \Rightarrow v \in C_0 \quad \text{and} \quad sgn(\hat{\mathbf{w}}^{\mathbf{T}}\hat{\mathbf{v}}) = 1 \Rightarrow v \in C_1$$

This process can be illustrated by the following figure borrowed from [2]:

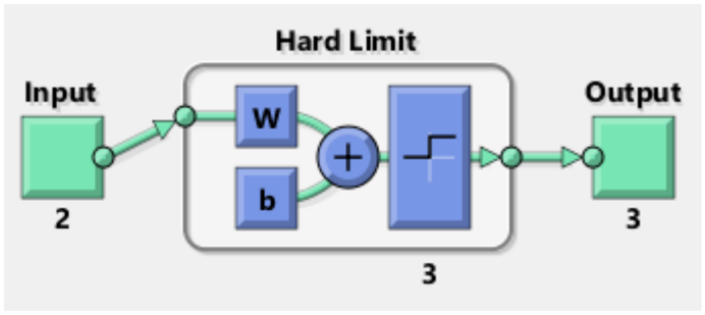

Figure 4.8: Perceptron.

[2]These parameters are called *weights and bias*, respectively. From now on, we will refer to both of these as *weights*.

Having described the *classification* phase, we can now take a look at how the perceptron *learns*, that is, how it modifies its parameters (weights) with respect to the training data observed.

In its canonical form, the perceptron initializes its weight vector as $[\mathbf{0}\ \mathbf{0}\ \cdots\ \mathbf{0}]^{\mathbf{T}}$. Then, upon observing the i'th data sample $\mathbf{v_i}$, the weights are updated using:

$$\hat{\mathbf{w}}_{\mathbf{i}} = \hat{\mathbf{w}}_{\mathbf{i-1}} + \frac{\alpha}{2}(c_i - sgn(\hat{\mathbf{w}}_{\mathbf{i-1}}^{\mathbf{T}}\hat{\mathbf{v}}_{\mathbf{i}})\hat{\mathbf{v}}_{\mathbf{i}}$$

The *learning rate* $\alpha \in (0, 1]$ is a hyperparameter that can be hard to fine-tune: if we choose a small learning rate, the algorithm will converge very slowly; if we set a high learning rate, the *loss function* that we are trying to optimize will oscillate indefinitely without converging to its local minimum. $c_i \in \{-1, 1\}$ is the correct output for the given sample $\mathbf{v_i}$. We can clearly see that if the perceptron classifies the given sample correctly, the weights are not modified: c_i and $sgn(\cdot)$ cancel each other out, which results in $\hat{\mathbf{w}}_{\mathbf{i}} = \hat{\mathbf{w}}_{\mathbf{i-1}}$. If that is not the case, the weights are modified in the direction determined by $c_i - sgn(\cdot)$ and $\mathbf{v_i}$.

Apart from training perceptrons, this approach can be generalized to shallow networks with an arbitrary number of output neurons. It is known as the *Generalized Delta Rule*[3]; for implementation details one can consult [4].

However, this approach doesn't generalize to deep networks: it is unclear as to how weights in the hidden layers should be modified: this was the motivation behind the back-propagation algorithm, which is described below.

1. Initialize the weights of the model.

[3]Our Perceptron used $sgn(\cdot)$ as an activation function. The Generalized Delta Rule trains networks with arbitrary activation functions.

2. Apply the model to a training instance to generate an output d. Compare it with the *correct* output y. Use this error $e = d - y$ to calculate δ, the value that will be propagated to previous layers:
$$\delta = \varphi'(v)e$$

3. Propagate δ backward:
$$e^{(k)} = W^T\delta \qquad \delta^{(k)} = \varphi'(v^{(k)})e^{(k)}$$

4. Repeat previous step until δ reaches the first hidden layer of the network.

5. Adjust the weights:
$$\Delta w_{ij} = \alpha\delta_i x_j \quad w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

6. Repeat Steps 2-5 for every training sample.

7. Repeat Steps 2-6 until network is trained (exit criteria: sufficient accuracy / number of epochs).

CNNs: YouTube - From Zero To Hero: Convolutional Neural Networks (TensorFlow channel)
ResNet: [3]
U-Net: [5]

2.2 Support Vector Machines

A high-level introduction to SVMs can be found in [1].

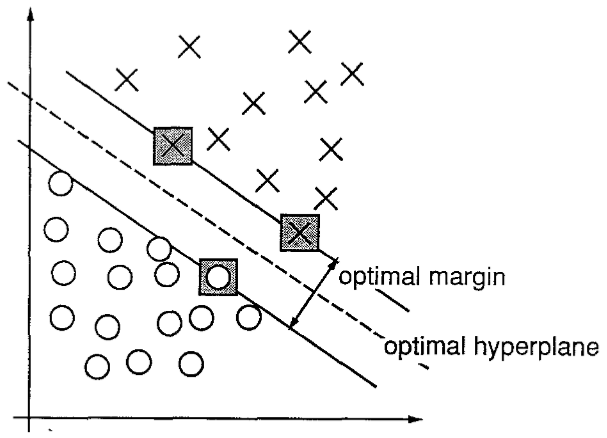

Figure 2. An example of a separable problem in a 2 dimensional space. The support vectors, marked with grey squares, define the margin of largest separation between the two classes.

3 Implementation

```
Flatten + Dense(128, ReLU) + Dense(10)
Epochs: 10

Training: loss: 0.0160 - accuracy: 0.9948
Testing:  loss: 0.0857 - accuracy: 0.9774

Flatten + Dense(128, ReLU) + Dense(10)
Epochs: 20

Training: loss: 0.0039 - accuracy: 0.9987
Testing:  loss: 0.1247 - accuracy: 0.9786

-----------------------------------------

Flatten + 2x Dense(128, ReLU) + Dense(10)
Epochs: 10
```

```
Training: loss: 0.0183 - accuracy: 0.9940
Testing:  loss: 0.0942 - accuracy: 0.9766

Flatten + 2x Dense(128, ReLU) + Dense(10)
Epochs: 20

Training: loss: 0.0101 - accuracy: 0.9967
Testing:  loss: 0.1117 - accuracy: 0.9808
```

4 Attacks

4.1 FGSM: Fast Gradient Sign Method

5 Datasets: their origins and pre-processing

5.1 MNIST

MNIST[4] is one of the most famous datasets in machine learning. It consists of 70,000 28×28 gray-scale images of handwritten digits. It is usually divided into a training set and a testing set using a $6 : 1$ ratio, that is, 60000 training images and 10000 images for testing.

Pros:

- The dataset consists of small gray-scale images, which means that it can be learned by a relatively simple model; therefore, it requires minimal time and resources.

Cons:

[4]By the way, [**lecun99**] is an excellent resource that not only provides the dataset itself, but also lists many ML models and their respective performances on MNIST.

- As pointed out in [**Goodfellow2015**], adversarial attacks can be seen as a kind of *accidental steganography*, where the adversarial modification injected in the images remains unseen because of the dynamic range of the image (e.g., we cannot see the difference between 0/255 and 0.5/255 if we are working with `uint8` images). In the case of MNIST, this can be exploited even further: in the same paper, Goodfellow remarks that MNIST is almost *binary* in terms of the dynamic range: its pixels represent "ink" or "no ink". My conclusion is that this property makes the *admissible norm of adversarial perturbations* much higher, so our adversarial attack methods will be able to generate *unrealistically bold* perturbations, which will be easily detected by the HVS.

The MNIST dataset doesn't require any pre-processing (apart from converting from `uint8` to `double`) and can be loaded by simply executing:

```
from tensorflow import keras
mnist = keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

5.2 Fashion MNIST

Fashion MNIST is a more modern replacement for the original MNIST dataset created by Zalando Research. As the original MNIST, it contains 60,000+10,000 images, all of them belonging to 10 classes: the difference is that FMNIST represents articles of clothing, namely: T-shirts, Trousers, Pullovers, Dresses, Coats, Sandals, Shirts, Sneakers, Bags and Ankle Boots.

Pros:

- The dynamic range of these images is significantly higher: this means that adversarial attacks on this dataset can be much more subtle and realistic.

Cons:

- As we can see, the classes are not as *distinct* as in the original MNIST dataset: this property is used by automatically generated adversarial attacks in a rather unsatisfying way. Most of my experiments with attacking Fashion MNIST resulted in misclassifying sandals as ankle boots or misclassifying coats as pullovers. The classes are much closer in the feature space than they are for MNIST, which allows for simplistic and uninteresting attacks.

5.3 Road Signs in Canada

```
#!/bin/bash

# Create a CSV file for storing class names
rm -f classes.csv sorted_classes.csv
echo " " > classes.csv

for f in *; do

        # if it's a folder
        if [ -d "$f" ]; then
                echo "Current folder:" $f
                if [[ "$f" =~ ^[[:digit:]]{1,2} ]]; then
                        echo "Changing folder name to: " ${BASH_REMATCH[0]}
                        class_name=$(echo $(echo "$f" | cut -c3-) | xargs)
                        echo "Class name: "$class_name
                        echo

                        echo ${BASH_REMATCH[0]}","$class_name >> classes.csv

                fi
        fi

        # remove class from name
        # create a csv file that maps numbers to image classes

        # for every file in folder
        # convert to common size
        # convert to gray-scale
done

# Add header line, sort the classes and overwrite original CSV file
echo "Number,Class" > sorted_classes.csv
cat classes.csv | sort -h >> sorted_classes.csv
mv sorted_classes.csv classes.csv
```

References

[1] Kenneth Soo Annalyn Ng. *Data Science - was ist das eigentlich?!* Springer-Verlag, 2018.

[2] A.C. Faul. *A Concise Introduction to Machine Learning.* CRC Press, 2020.

[3] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn and Keras.* O'Reilly, 2019.

[4] Phil Kim. *MATLAB Deep Learning.*

[5] *U-Net: Convolutional Networks for Biomedical Image Segmentation.* 2015. URL: https://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a/ (visited on 01/04/2020).