Lab B - Adversarial Training With Malware Classifier Student Document

Table of Contents

Lab Descriptors

Logistics

Due Date:	Check ilearn
Submission Location:	iLearn for documents
Points Available:	100

Objectives

The student will learn how to harden a convolutional neural network against adversarial attacks using adversarial training with samples generated by the Fast Gradient Sign Method attack in the Cleverhans adversarial library.

Assumptions

The student has basic knowledge of virtual machines, Debian Linux, the Python language, and some knowledge of machine learning.

Outcomes

The student will be able to generate a dataset of adversarial samples with which to train a CNN model in order to harden it against such attacks.

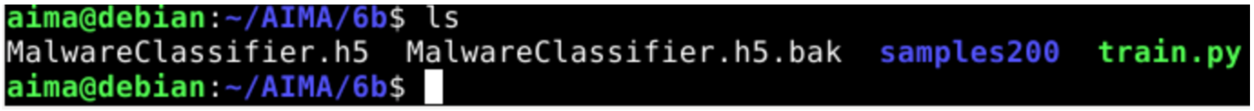
Introduction

In this lab, students will employ adversarial training on a pre-trained malware classifier using the Cleverhans implementation of the Fast Gradient Sign Method (FGSM). This classifier is a convolutional neural network (CNN) that classifies inert images of malware samples into eight families of malware. **There is no live malware in this lab.** Students will generate a dataset of adversarial samples of the model's expected input and use these samples to train the model against adversarial inputs and to harden it against similar attacks. This will be done in a Debian Linux virtual machine in Python, making use of the Keras, Tensorflow, Cleverhans, numpy, and matplotlib libraries.

Lab Details

Environment

1. The username for this lab is **aima** and the password is **toor**. The working directory for the lab is **/home/aima/AIMA/6b**.



```
aima@debian:~/AIMA/6b$ ls
MalwareClassifier.h5  MalwareClassifier.h5.bak  samples200  train.py
aima@debian:~/AIMA/6b$ 
```

Materials

The lab materials are all contained in the **AIMA/6b** working directory discussed in the Environment section above. Within that, you will find the files and directories discussed in this section. Do your work in place as these files depend on one another.

1. The **samples200** directory tree contains the images of the malware samples in the format that the model uses to verify family labels. These are images of sterilized malware sample's bytecode read in as a grayscale .PNG images. These samples are inert and safe to work with and view inside the VM. The model is trained to classify samples from eight families: Gatak, Kelihos_ver1, Kelihos_ver3, Lollipop, Obfuscator_ACY, Ramnit, Tracur, and Vundo. You are provided with 200 samples from each family for this lab.

2. The **MalwareClassifier.h5** file is the pre-trained export of the convolutional neural network model you will be working with.

3. The **MalwareClassifier.h5.bak** file is a backup copy of the classifier.

4. The **train.py** script is the python script in which you will do all of your work. It may be a good idea to make a backup copy of this before you start making changes to it.

Adversarial Training

1. Navigate to the working directory and run the **train.py** script. If you are doing this from the command line, run **python3 train.py** from the working directory. If you are working from the GUI, you should be able to double click the script and select run. This will feed the model a set of non-adversarial malware image samples and produce an accuracy on its performance.

   1.1. What was the model's accuracy on regular inputs?
   Record your answer to your answer document.

   1.2. Insert a screenshot of the model's accuracy on regular malware sample images.

2. Now open the **train.py** script in a text editor. If you are doing this from the command line, nano or vim will work fine. If you are doing this from the GUI, there is a text editor pinned to the taskbar.

   2.1. Uncomment line 36 for "plt.show()", uncomment line 52 for "plt.show()", and uncomment line 78. These will show plots of some of the non-adversarial malware samples, one of the adversarial malware samples, and print the report for the model's accuracy on the adversarial sample. Save the changes and run the script again.

   2.2. The regular sample and adversarial sample will be plotted so you can see it. Close any plot windows to allow the script to finish running.

   2.3. What was the untrained model's accuracy on the adversarial inputs?

   2.4. Insert a screenshot of the model's output accuracies.

3. Open the **train.py** script in a text editor again. Uncomment line 79 to get results for the trained model on adversarial samples. Save the changes and run the script again.

   3.1. What is the model's accuracy on adversarial samples after training?

   3.2. Insert a screenshot of the model's accuracies.

4. Open the **train.py** script in a text editor again. Uncomment line 80 to get results for the trained model on regular samples. There will be some tradeoff for the adversarial training. Save the changes and run the script again.

   4.1. What is the model's accuracy on regular malware samples after training?

   4.2. Insert a screenshot of the model's accuracies.

5. We trained for epsilon 0.022 with adversarial samples generated against the new model for epsilon 0.022. Let's see what would happen if an attacker fed our model samples generated for an epsilon of 0.005. Open the **train.py** script in a text editor, uncomment line 69 for "epsilon=0.005", and save the changes. Run the script again.

   5.1. What is the trained model's accuracy on this set of adversarial samples?

   5.2. If an attacker uses an epsilon smaller than the epsilon we trained with, such as 0.005 here, is our training sufficient to defend against this?

   5.3. Insert a screenshot of the model's accuracies for this new set of adversarial samples.

6. Now let's see what happens if the attacker uses adversarial samples generated with an epsilon larger than the one we trained with, such as 0.1. Open the train.py script in a text editor and change the value of epsilon on line 69 from 0.005 to 0.1. Save the changes and run the script again.

   6.1. What is the trained model's accuracy for these new adversarial samples?

   6.2. If the attacker uses a larger epsilon than we trained with, such as 0.1 here, is our training sufficient to defend against those adversarial samples?

   6.3. Insert a screenshot of the model's accuracies.

7. Now let's see what happens if we train on a large epsilon of 0.1 and attack the model with samples generated with a small epsilon of 0.005. Open the train.py script in a text editor and change the epsilon on line 39 to 0.1. Change the epsilon on line 69 to 0.005. Save the changes and run the script.

   7.1. What is the trained model's accuracy for this set of adversarial samples?

   7.2. What is the trained model's accuracy for regular samples?

   7.3. Does training on this larger epsilon hurt the model's accuracy on regular samples more than it helps to identify adversarial samples?

   7.4. Given your testing, is adversarial training on this model a good protection against fast gradient sign method attacks? Why or why not?

   7.5. Insert a screenshot of the model's accuracies on this set of samples.

Deliverables

You will turn in a PDF with your name at the top and your answers to the lab questions and embedded screenshots from each step of the lab.

Resources

Virtualbox Documentation

https://www.virtualbox.org/wiki/Documentation

Python Documentation

https://www.python.org/doc/

Linux manual pages

https://linux.die.net/man/

Tensorflow FGSM Tutorial

https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

Cleverhans FGSM Adversarial Training on MNIST dataset
https://gist.github.com/miwong/1ce7c1b67cc48da1ed25d91d79fe103b

Citations

(Travis, MS Big 2015, NSF grant #)