Adversarial AI Projects for Undergraduates[*]

George Kesidis, David J. Miller, Hang Wang, Xi Li, Zhen Xiang
Pennsylvania State University
{gik2,djm25,hzw81}@psu.edu

October 22, 2022

Deep Learning & PyTorch

There are many online tutorials on deep learning. The slides for one is found here:
`http://www.cse.psu.edu/~gik2/teach/dnn-slides-2.pdf`

Except for the ZOO TTE attack/defense experiment using MATLAB, the following projects will require some familiarity with Python and PyTorch. Regarding PyTorch, see, e.g., [10, 9] and online tutorials including at `https://pytorch.org` The use of a GPU for CUDA-accelerated PyTorch will have significantly smaller execution time for deep learning. If you have difficulty installing PyTorch on your computer, you can try using, e.g., Google Colab which has PyTorch installed and has hardware accelerator support `https://colab.research.google.com`

1 MNIST [5], ZOO TTE attack [1], white-region counting defense [6]

MNIST [5] has handwritten digit characters, so there are 10 classes $\{0, 1, ..., 9\}$ and each image is 30×30 grey-level pixels, *i.e.*, each image is a 900-vector of grey levels $\in \{0, 1, ..., 255\}$ where intensity 0 is black and intensity 255 is white.

1.1 Notes regarding the ZOO/JSMA attack

- See the untargeted attack objective f in equation (5) and and Algorithm 1 of the ZOO paper [1].

[*]This work was supported by ONR grant N00014-21-1-2129.

- i, t in [1]'s (5) are class indexes $\in \{0, 1, ..., 9\}$ and i is a pixel index in Algorithm 1.
- Start with an MNIST image $\underline{x} = \underline{x}_0$ that is correctly classified to $t_0 \in \{0, 1, ..., 9\}$ so that $f(\underline{x}_0) > 0$ initially, *i.e.*, check that

$$[F(\underline{x})]_{t_0} > \max_{j \neq t_0} [F(\underline{x})]_j$$

  where $F(\underline{x})$ is the softmax output of the neural network when the input is $\underline{x}$ (= **x** in the notation of [1]).

- In each iteration k of Algorithm 1, select pixel index i of $\underline{x}_k$ at random and modify its grey-level $x_{k,i}$ to maximize [1]'s objective f. That is,

$$\underline{x}_{k+1} = \left(\underline{x}_{k,-i}, \underset{\delta \in \{0,1,...,255\}}{\operatorname{argmax}} f((\underline{x}_{k,-i}; \delta)) \right),$$

  where by definition the j^{th} element of the vector $(\underline{x}_{k,-i}, a)$ is $= x_{k,j}$ if $j \neq i$ and is $= a$ if $j = i$.

- The stopping condition for Algorithm 1 is $f < 0$, *i.e.*, the (untargeted adversarial) image $\underline{x}_k$ is no longer classified to t_0.

Compare ZOO algorithm 1 to JSMA [7].

Instead of searching the entire range of values $\{0, 1, ..., 255\}$, just search in a small neighborhood above and below the current value $\mathbf{x}_i$. Does this change to Algorithm 1 produce different images?

Also implement ZOO Algorithm 2 in [1] which employs the difference quotient (6) to compute gradients rather than back propagation w.r.t. the inputs.

1.2 MATLAB (with deep learning toolbox) version

See the MATLAB files available here
`http://www.cse.psu.edu/~gik2/ONR-NROTC/matlab`

The pseudocode to generate adversarial examples is as follows:

1. Import MNIST dataset of images with ground truth class labels.
2. Pre-trained neural-network MATLAB function with F the softmax layer.
3. While any class has fewer than 10 adversarial images classified to it:
   (a) Select a new, correctly classified MNIST image $\underline{x}$ from a random source class t_0.
   (b) Perform the ZOO attack described above starting from $\underline{x}$.
   (c) Output the created ZOO attack into a file containing ZOO adversarial examples, and also record its class decision (t) and the source class ($t_0 \neq t$) of the initial clean image $\underline{x}$ used to create it.

1.3 Python version

Follow the directions of the MATLAB project version but instead use the Python files available here
`http://www.cse.psu.edu/~gik2/ONR-NROTC/python`

Alternatively, you can use another prebuilt model for MNIST, *e.g.*, VGG-16 available here `https://pytorch.org/vision/stable/models.html`. Note that with some of these models, you may have to resize the input in your transforms with a command like
"transforms.Resize(size = (224, 224))"
if the model was trained on a different dataset.

1.4 Contiguous White-Region Counting Defense

Visualize ZOO adversarial images to verify salt-and-pepper noise. Consider a simple region-counting based AD [6] by first noting that clean MNIST digits typically consist only of one or two contiguous white regions, where a region is defined as a collection of pixels that are pairwise "connected", with two white pixels pairwise connected if they are in the same first-order (8-pixel) neighborhood, and a white region defined by extending pixel connectedness through transitive closure over the whole image. Note that a threshold is needed on a pixel intensity to determine whether it is "white." Sweep an RoC curve by varying this threshold and compute its AUC.

Repeat for the case when the attacks are created with smaller perturbation per iteration.

2 Adversarial Examples in the Audio Domain [2]

`https://github.com/crgeer/Notebooks/blob/main/Audio_Adv_ML_Project.ipynb`

3 Data Poisoning Attacks

In a data poisoning (DP) attack, the adversary plants poisoned samples into the training set prior to training the DNN. The poisoning can occur through, e.g., insecure outsourcing to gather the often extremely large labelled training set or by an insider to the training process itself. DP attacks can aim to simply degrade classifier accuracy (error generic), or to plant a backdoor into the DNN (error specific Trojan [3]). That is, a test-time sample with the backdoor pattern properly incorporated "triggers" the backdoor so that the sample is essentially misclassified.

3.1 Error-generic DP attack by mislabelling on MNIST [5]

Attack implementation and assessment experiments:

1. Load training and test sets of MNIST
2. Keep training and test samples of 5 classes: $\{0, 1, 2, 3, 4\}$. For each class, split the training set: 2000 images are used for training, 800 images are used for poisoning. Poisoning samples are stored in another dataset named attackset.
3. Poison the training set. There are 5 inputs, N0,N1,N2,N3,N4. If Nc is 1, then evenly distribute poisoning samples of class c into the training sets of the remaining 4 classes. For example, if N0=1, add 200 poisoning samples of class 0 into class 1, and label them as class 1. Do the same to poison class 2,3,4 by class 0.
4. Train a ResNet-18 DNN on poisoned training set. The training framework (epoch, net, Train_loader) is given. The learning rate, the number of training epochs, the loss function and the optimizer are given. You need to implement back propagation.
5. Test the trained ResNet-18 on clean (attack-free) test set. The test framework (net, Test_loader) is given. You need to implement forward propagation (inference).
6. Test the trained ResNet-18 on clean (attack-free) test set.
7. Do the experiment 6 times. In experiment j, use j classes for poisoning. The number of classes used for poisoning is the attack strength. For example, in experiment 0, N0=0, N1=0, N2=0, N3=0, N4=0, there's no poisoning (this is the control experiment). In experiment 3, N0=1, N1=1, N2=1, N3=0, N4=0, 3 classes are used for poisoning, and all the 5 classes are poisoned. Plot the test accuracy vs the attack strength. The stronger the attack, the lower the test accuracy.

The following code will train accurate MNIST and CIFAR-10 classifiers:
`https://github.com/kuangliu/pytorch-cifar/blob/master/main.py`
For those who are using a CPU, training a DNN may be time consuming. You can use Google Colab which has optional GPU and TPU hardware accelerators.

Now consider the goal of detecting poisoned MNIST images. The method we will investigate relabels an image as the plurality label of its K Nearest Neighbors (KNN) [8]. ([8] provides a KNN based anomaly detector for binary classification tasks and it's straightforward to extend it for cases with > 2 classes.)

Defense against error-generic data poisoning experiments:

1. Poison the training set of MNIST as above and keep the ground-truth indicators of malicious samples: 0=clean, 1=malicious. (Note: The ground-truth indicators are used in performance evaluation in step 4, but they are **not** allowed to be used for anomaly detection.)

2. Train a SVM classifier on the poisoned training set or load the ResNet-18 trained in the previous project.[1]

3. (**TODO**) Implement the KNN based defense with K a hyper-parameter (number of nearest neighbors). By default, $K = 10$. For each training sample:

   (a) Find its K nearest neighbors. (Check sklearn.neighbors library for functions finding the K nearest neighbors of a data sample.)

   (b) Find the plurality label of the K nearest neighbors. (Check scipy.stats library for functions finding the plurality.)

   (c) If the plurality label is different from its original label, it is deemed a malicious sample. Otherwise, it is deemed clean.

4. Evaluate the performance of the KNN based detector by:

   (a) True positive rate (TPR): the number of truly detected malicious samples over total number of malicious samples

   (b) False positive rate (FPR): the number of falsely detected malicious samples over total number of clean samples

   (c) Test set accuracy of the classifier trained on sanitized dataset (with the detected malicious samples removed)

5. Vary the value of K and observe how it affects the performance of the detector.

The necessary files are found here:
`http://www.cse.psu.edu/~gik2/ONR-NROTC/dp-generic`

3.2 A backdoor DP attack on CIFAR-10 [4]

Note that unlike MNIST images where each pixel has a single greyscale channel, color CIFAR images have three (RGB) channels per pixel.

Attack implementation and assessment experiments:

1. Load in the CIFAR-10 dataset

2. Create a backdoor pattern

3. Create backdoor training samples and backdoor test samples by embedding the backdoor pattern into clean samples

4. Insert the backdoor training samples (i.e., poison it) into the training set having, say, 3k clean (unpoisoned) training samples per class.

[1]If you're using Google Colab, you can copy your ResNet-18 model over from Google Docs by suitably setting the model path.

5. Load in the model architecture (ResNet-18)
6. Perform training (back propagation)
7. Evaluate accuracy on clean test samples and the attack success rate
8. Report such performance versus poisoning rate (0, 250, 500, ..., 1500 total poisoned images, i.e., up to 5% if 30k clean training samples are used across 10 classes) and perturbation size (from 0 to 1)

The necessary files are found here:
`http://www.cse.psu.edu/~gik2/ONR-NROTC/dp-backdoor`

References

[1] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh. ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models, Nov. 2017.

[2] C. Geer. Anomaly detection of test-time evasion attacks in the audio domain. Master's thesis, CSE Dept, Pennsylvania State University, May 2022; `https://etda.libraries.psu.edu/catalog/20200crg5567`.

[3] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[4] A. Krizhevsky. The CIFAR-10 dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`, 2010.

[5] Y. LeCun, C. Cortes, and C.J.C. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998.

[6] D.J. Miller, Y. Wang, and G. Kesidis. Anomaly Detection of Attacks (ADA) on DNN Classifiers at Test Time. *Neural Computation*, 2019.

[7] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *Proc. 1st IEEE European Symp. on Security and Privacy*, 2016.

[8] A. Paudice, L. Munoz-Gonzalez, and E.C. Lupu. Label Sanitization against Label Flipping Poisoning Attacks. `https://arxiv.org/abs/1803.00992`, 2 Mar 2018.

[9] PyTorch. TorchVision.Models. https://pytorch.org/docs/stable/torchvision/models.html.

[10] E. Stevens, L. Antiga, and T. Viehmann. *Deep Learning with PyTorch*. Manning, 2020.