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](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 PyTorch. See, e.g., [8, 7] and online tutorials including at [https://pytorch.org](https://pytorch.org)

If you have difficulty installing PyTorch on your computer, you can try using Google Colab which has PyTorch installed, [https://colab.research.google.com](https://colab.research.google.com)

1  MNIST [4], ZOO TTE attack [1], white-region counting defense [5]

MNIST [4] has handwritten digit characters, so 10 classes {0, 1, ..., 9} each image is $30 \times 30$, i.e., each image is a 900-vector of grey levels each grey-level has range {0, 1, ..., 255}.

Students can optionally do the MATLAB or Python version of Project 1, or both. The Python version is required to proceed to Project 2 on data poisoning.

1.1 Notes regarding the ZOO attack [1]

- See equations (5) (untargeted attack) and and Algorithm 1.
- $i, t$ in (5) are class indexes $\in \{0, 1, ..., 9\}$.

*This work was supported by ONR grant N00014-21-1-2129.*
• $i$ in Algo 1 is a pixel index/coordinate of (input) image $x$.

• Algo 1 will start with a correctly classified MNIST image $x$ from class $t_0 \in \{0, 1, ..., 9\}$ so that $f(x) > 0$ initially. That is, before applying Algo 1, first check that the class $t_0$ image $x$ you select from MNIST database satisfies

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

where $F(x)$ is the softmax output of the neural network when the input is $x$. If this isn’t satisfied, discard the image and select another.

• In each iteration of Algo 1, just select pixel index $i$ at random and modify pixel grey-level $x_i$ to minimize $f(x)$ of (5). In each iteration, you can search the entire range of values $\{0, 1, ..., 255\}$ or just in a small neighborhood above and below the current value $x_i$.

• Stopping condition for Algo 1 is $f < 0$, i.e., the (untargeted adversarial) image is no longer classified to $t_0$.

1.2 MATLAB

Need MATLAB with deep learning toolbox.

See our MATLAB files available here

http://www.cse.psu.edu/~gik2/ONR-NROTC/matlab

The pseudocode to generate $K$ adversarial examples targeting class $t$:

1. Import MNIST dataset of images $x$ with ground truth class labels, where each grey-level pixel $x_i \in \{0, 1, ..., 255\}$ for $i \in \{0, 1, ..., 899\}$.

2. Pre-trained neural-network MATLAB functions $d, F$ where $d \in \{0, 1, ..., 9\}$ is the class decision and $F$ is the softmax layer (a 10-vector)

3. for $k = 1, 2, ..., K$:

   (a) select a correctly classified image $x$ from MNIST from a random class $t_0$
   (b) perform ZOO attack on $x$ (Algo 1 of [1], a while loop)
   (c) output $x$ into a file containing ZOO adversarial examples, and also record its class decision ($i$) and the source class ($t_0 \neq i$) of the initial clean image used to create it.

For each class $i$, you will need to create at least 10 adversarial images which are classified to $i$, i.e., choose $K$ large enough in the for loop to achieve this or instead use an outer while loop with this stopping condition.

Visualize adversarial images to verify salt-and-pepper noise.

Implement defense based on counting counting contiguous white regions in [5]. Plot ROC of this defense and evaluate its AUC.
1.3 Python

Follow the directions of the previous two subsections. Instead of the MATLAB code, use the Python files available here

\[\text{http://www.cse.psu.edu/~gik2/ONR-NROTC/python}\]

Alternatively, you can use another prebuilt model for MNIST, e.g., VGG-16 available here

\[\text{https://pytorch.org/vision/stable/models.html}\]

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.

2 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 [2]). That is, a test-time sample with the backdoor pattern properly incorporated “triggers” the backdoor so that the sample is essentially misclassified.

2.1 Error-generic DP attack by mislabelling on MNIST [4]

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.

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) [6]. ([6] 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. (For those who are using a CPU, training a DNN is expensive. You can first do the experiments on a SVM classifier. Also, to load your own ResNet-18, just modify the model path\(^1\).)

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](http://www.cse.psu.edu/~gik2/ONR-NROTC/dp-generic)

\(^1\)If you’re using Google Colab, you can copy your ResNet-18 model over from Google Docs.
2.2 A backdoor DP attack on CIFAR-10 [3]

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.
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


