Adversarial AI Projects for Undergraduates*

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

October 18, 2021

1 MNIST [11], ZOO attack [5], white-region counting defense [12]

MNIST [11] 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.

1.1 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 $g(x)$, 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)


4. Select target class $t \in \{0, 1, ..., 9\}$

This work was supported by ONR grant N00014-21-1-2129.
5. for \( k = 1, 2, \ldots, K \):

   (a) select image \( x \) from MNIST such that \( d(x) = g(x) \neq t \)
   
   (b) perform ZOO-ADAM on \( x \) (Algo 2 of [5], a while loop)
   
   (c) output \( x \) into a file containing ZOO adversarial examples targeting class \( t \)

Notes:

- see equations (4) and (6) and Algorithm 2 (ZOO-ADAM)
- \( \kappa > 0 \) in (4) prevents loss objective \( f \) becoming too negative
- \( i, t \) in (4) are class indexes \( \in \{0, 1, \ldots, 9\} \)
- \( i \) in (6) is a pixel index/coordinate of (input) MNIST image \( x \)
- \( e_i \) is an image with every pixel zero except pixel \( i \) which equals 1, i.e.,
  
  \[
  \forall i, j \neq i : \ e_{i,j} = 0, \ e_{i,i} = 1
  \]
  
- \( F(x) \) is a 10-dimensional softmax vector of probabilities (\( \geq 0 \)), with
  
  \[
  \sum_{i=0}^{9} [F(x)]_i = 1
  \]

- \( d(x) = \arg\max_i [F(x)]_i \) is the class decision of the neural network model for MNIST image \( x \)

- \( T \) in Algo 2 is optional (steps 5 and 7 are optional too)

- stopping condition in Algo 2:
  
  either loss \( f < 0 \) (\( f \) defined in (4)) OR
  
  \[
  |\Delta f| = |f_{current} - f_{previous}| < \varepsilon \ll 1
  \]
  
  (change in \( f \) from consecutive iterations of while loop is small)

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

Implement defense based on counting contiguous white regions in [12]. Plot ROC of this defense and evaluate its AUC.

1.2 Tutorial on Deep Learning with PyTorch [14, 15]

See our Python files available here

http://www.cse.psu.edu/~gik2/ONR-NROTC/python
1.3 Python

- Repeat Project 1.1 using python & PyTorch instead of MATLAB

- If you completed the MATLAB Project 1, just convert your MATLAB code to Python, where the MATLAB neural network itself (mapping MNIST images to one of class labels 0,1,...,9) needs to be replaced by a PyTorch function.

- train your own MNIST classifier or use a prebuilt model, e.g. VGG-16, can be found here: https://pytorch.org/vision/stable/models.html

- If the model is trained on a different dataset, you may have to first resize the input sample using a command like:
  transforms.Resize(size = (224, 224))

- See our PyTorch tutorial files cnn.py, main.py

2 MNIST, PyTorch, gradient based attacks

2.1 FGSM attack [7]

- Implement “low confidence” FGSM attack on MNIST

- Note that attack images do not have salt-and-pepper noise, but do have grey “ghosting”, so white-region counting defense will not work well as for ZOO attack of Project 1 (or as for JSMA attack [13] which also exhibits salt-and-pepper noise)

- First defense idea: for a grey-level hyperparameter $\theta \in (0, 255)$, count how many pixels are “whiter” than $\theta$

- Second defense idea: use an anomaly detector based only on softmax/output layer confidence as suggested in [10,9], see also [12]

2.2 CW attack [4]

- Implement CW attack and repeat project 2.1 for it

- Repeat for white-box CW attack [3,12]

- In addition to ROC AUC performance of the defense, compare work-factors of attack and defense (consider role of efficient optimization to compute CW attacks)

1 Tutorial on how to implement FGSM in PyTorch:
https://pytorch.org/tutorials/beginner/fgsm_tutorial.html
3 CIFAR-10 [1], PyTorch, null-model based defense

Now three color “channels” per pixel of CIFAR images (from 10 classes) [1]. Attacks are CW and FGSM, both low and high confidence [2, 6, 16]. Visualize the high-confidence attack images to check their success (i.e., attack images should still look like source samples on which they’re based)

Defense:

- Consider activations of “feature-extraction” layer, e.g., ResNet layer 3 (from the input) of 256 $8 \times 8$ blocks.
- Max-pool or average-pool each $8 \times 8$ block down to 2 values, i.e., down to $256 \times 2 = 512$ total activations.
- Build a class-conditional null using GMM modeling tool of https://scikit-learn.org/stable/
- For test samples, find probability under null based on decided-upon class and use this to detect TTEs
- Compare complexity and performance of these defense against the one based on softmax layer (Project 2.2) on FGSM and CW attacks
- Similar but more advanced defenses are discussed in [12, 16]
- Can use more advanced null models too [8]

References


