Adversarial AI Projects for Undergraduates*

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

December 6, 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 30×30, 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 $\mathbf{x}$ with ground truth class labels , where each grey-level pixel $\mathbf{x}_i \in \{0, 1, ..., 255\}$ 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 $\mathbf{x}$ from MNIST from a random class t_0

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

(b) perform ZOO attack on $\mathbf{x}$ (Algo 1 of [5], a while loop)

(c) output $\mathbf{x}$ 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.

Notes regarding the ZOO attack [5]:

- See equations (5) (untargeted attack) and (6) and Algorithm 1.
- i, t in (5) are class indexes $\in \{0, 1, ..., 9\}$.
- i in (6) is a pixel index/coordinate of (input) image $\mathbf{x}$.
- Algorithm 1 will start with a clean, correctly classified MNIST image $\mathbf{x}$ from class $t_0 \in \{0, 1, ..., 9\}$ so that $f(\mathbf{x}) > 0$ initially.
- $\mathbf{e}_i$ is an image with every pixel zero except pixel i which equals 1.
- $\delta^* =$ arg $\max_\delta f(\mathbf{x} + \delta \mathbf{e}_i)$.
- Just select pixel index i at random and modify pixel grey-level $\mathbf{x}_i$ to minimize $f(\mathbf{x})$. You can search the entire range of values $\{0, 1, ..., 255\}$ or just in a small neighborhood above and below the current value $\mathbf{x}_i$.
- Stopping condition for Algo 1 is $f < 0$, i.e., the (untargeted aversarial) image is no longer classified to t_0.

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 [12]. Plot ROC of this defense and evaluate its AUC.

1.2 Python

- See our Python files available here
  `http://www.cse.psu.edu/~gik2/ONR-NROTC/python`
- Also see the notes re. the ZOO attack above.
- You can use another prebuilt model for MNIST, e.g. VGG-16:
  `https://pytorch.org/vision/stable/models.html`

2 MNIST, PyTorch, gradient based attacks

2.1 FGSM attack [7]

- Implement "low confidence" FGSM attack on MNIST[1]
- 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 θ
- 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)

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 8×8 blocks.
- Max-pool or average-pool each 8×8 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/`

[1] Tutorial on how to implement FGSM in PyTorch: `https://pytorch.org/tutorials/beginner/fgsm_tutorial.html`

- 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

[1] The CIFAR-10 dataset. https://www.cs.toronto.edu/ kriz/cifar.html, 2010.

[2] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proc. ACM CCS*, 2018.

[3] N. Carlini and D. Wagner. Adversarial examples are not easily detected: bypassing ten detection methods. In *Proc. AISec*, 2017.

[4] N. Carlini and D. Wagner. Towards Evaluating the Robustness of Neural Networks. In *Proc. IEEE Symposium on Security and Privacy*, 2017.

[5] 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.

[6] G. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. Goodfellow, and J. Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Proc. NeurIPS*, 2018.

[7] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *Proc. ICLR*, 2015.

[8] MW Graham and DJ Miller. Unsupervised learning of parsimonious mixtures on large spaces with integrated feature and component selection. *IEEE Transactions on Signal Processing*, pages 1289–1303, 2006.

[9] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proc. ICLR*, 2017.

[10] D. Hendrycks and K. Gimpel. Early methods for detecting adversarial images. In *Proc. ICLR - Workshop track*, 2017.

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

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

[13] 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.

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

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

[16] H. Wang, David J. Miller, and George Kesidis. Anomaly Detection of Test-Time Evasion Attacks using Class-conditional Generative Adversarial Networks. https://arxiv.org/abs/2105.10101.