Privacy & Machine Learning — Adversarial Machine Learning

Name: Ziyu Shu

December 17, 2019

Files

The assignment archive contains the following Python source files:

- `main.py`. This file is the main assignment source file.
- `nets.py`. This file defines the neural network architectures and some useful related functions.
- `attacks.py`. This file contains attack code used in the assignment.

Step 0: Training a Neural Net for MNIST Classification

In this problem, you will train a neural network to do MNIST classification. The code for this problem uses the following command format.

```
python3 main.py Step0 <nn_desc> <target_train_size> <num_epoch>
```

Here `<nn_desc>` is a neural network description string (no whitespaces). It can take two forms: `simple,<num_hidden>,<l2_reg_const>` or `deep`. The latter specifies the deep neural network architecture (see `get_deeper_classifier()` in `nets.py` for details), whereas the former specifies a simple neural network architecture (see `get_simple_classifier()` in `nets.py` for details) with one hidden layer with `<num_hidden>` neurons and an L_2 regularization constant of `<l2_reg_const>`. Also, `<target_train_size>` is the number of records in the target model's training dataset and `<num_epoch>` is the number of epoch to train for.

For example, suppose you run the following command.

```
python3 main.py Step0 simple,64,0.001 50000 100
```

This will train the target model on 50000 MNIST images for 100 epochs. The target model architecture is a neural network with a single hidden layer of 64 neurons which uses L_2 regularization with a constant of 0.001.[1] (The loss function is the categorical cross-entropy.)

1. Run the following command:

   ```
   python3 main.py Step0 simple,512,0.001 50000 100
   ```

   This will train the model and save it on the filesystem. Note that 'Step0' is used to denote training. The command line for subsequent steps (`Step1`, `Step2`, etc.) will load the model trained for this problem!

[1]By default, the code will provide detailed output about the training process and the accuracy of the target model.

Step 1: Fun with Iterative Pixel Perturbation Attacks

In this step, you will use an attack that produces adversarial examples by iteratively modifying pixels. (The code is in `turn_off_pixels_iterative_attack()` which is located in `attacks.py`.)

At each step, the "turn off" pixel attack computes the gradient of the loss for a given target label (with respect to the input), and uses this gradient to determine which pixel (that is currently "on") to turn "off". Each MNIST image is represented as an array of pixels, each taking a value in $\{0, 1, \ldots, 255\}$. We define as "on" a pixel with value larger than some threshold t (by default $t = 10$). Turning off a pixel is defined as setting its value to 0. The attack ends after `max_iter` iterations or as soon as the terminate condition is reached (e.g., if the model's prediction on the perturbed image matches the target label).

To run the code , use the following command.

```
python3 main.py Step1 <nn_desc> <train_sz> <num_epoch> <input_idx> <target_label>
```

Here `<input_idx>` is the input (benign) image that the attack will create an adversarial example from and `<target_label>` is the target label. The code will automatically load the model from file, so you need to have completed step 0 first!

1. Now, let's run the attack using the following command with various input images and target labels.

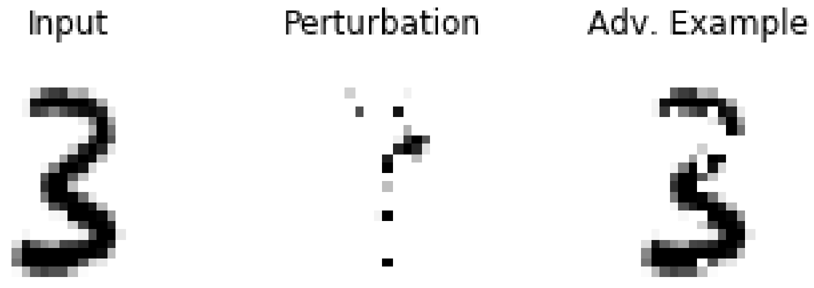
   ```
   python3 hw3.py problem1 simple,512,0.001 50000 100 <input_idx> <target_label>
   ```

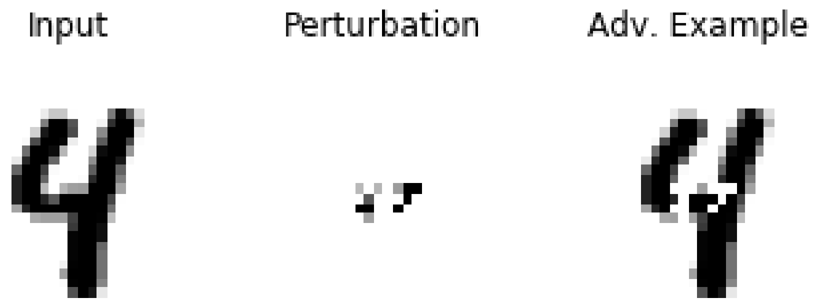
   Note: it is important that the architecture, size of training data, and number of epochs match what you ran for step 0. (The code uses these arguments to locate the model to load.)

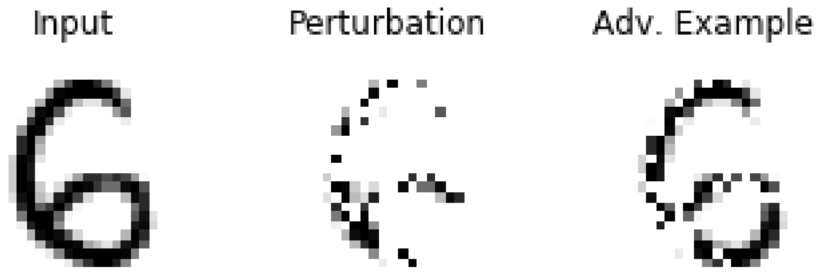
   For example, try:

   ```
   python3 main.py Step1 simple,512,0.001 50000 100 0 8
   python3 main.py Step1 simple,512,0.001 50000 100 0 5
   python3 main.py Step1 simple,512,0.001 50000 100 4 1
   python3 main.py Step1 simple,512,0.001 50000 100 9 9
   python3 main.py Step1 simple,512,0.001 50000 100 14 7
   ```

   The code will plot the adversarial examples (see `plots/`) and print the distortion according to your proposed metric.

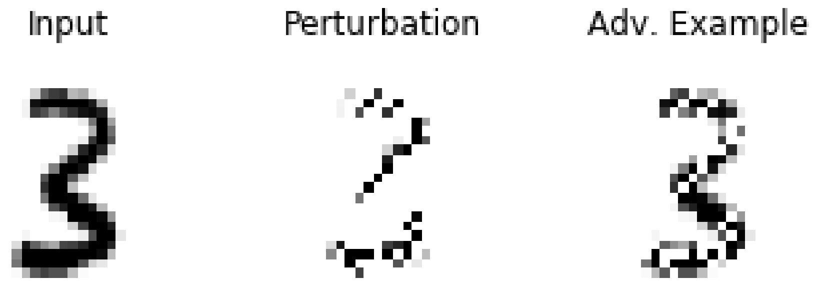

Input
Perturbation
Adv. Example

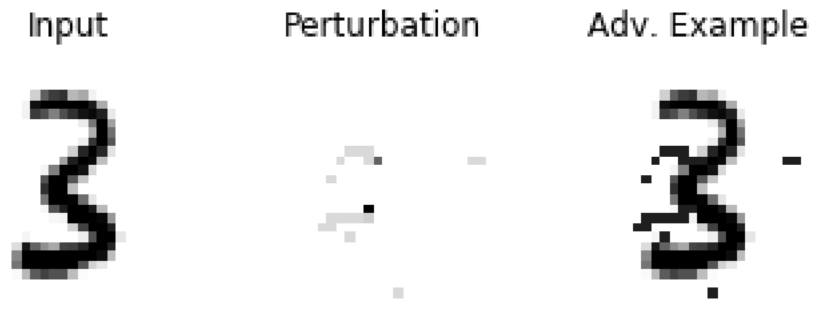

Input
Perturbation
Adv. Example

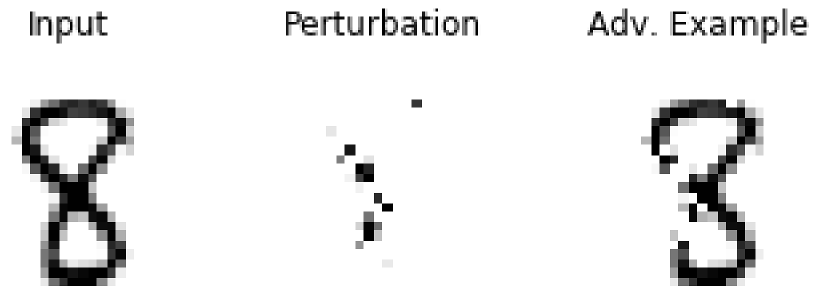


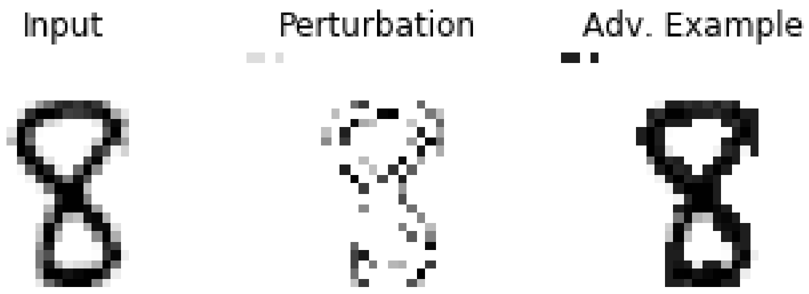
2. The "turn on" pixel attack is implemented in `turn_on_pixels_iterative_attack()`.

   Now run the code again and paste both adversarial examples you obtained (the one from the "turn-off" attack and the one from the "turn-on" attack).




Input
Perturbation
Adv. Example


Input
Perturbation
Adv. Example


Input
Perturbation
Adv. Example

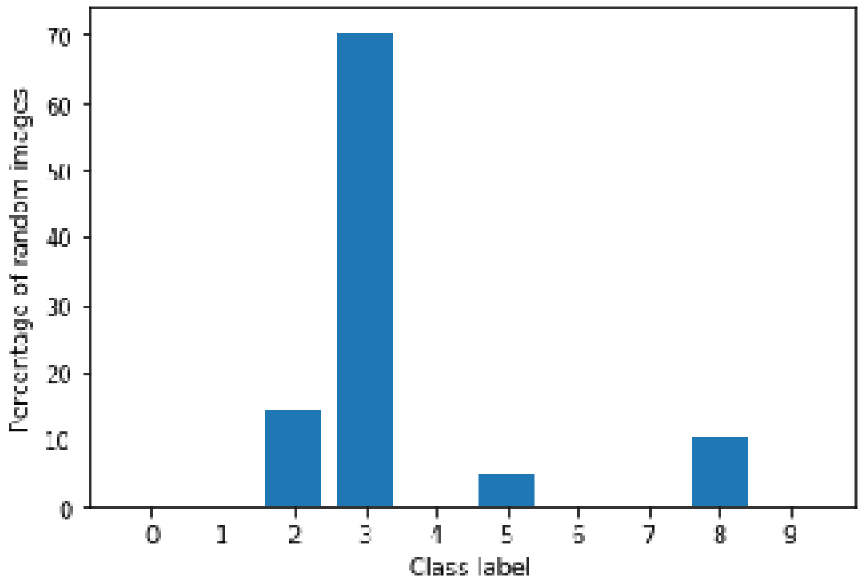
Step 2: Strange Predictions & Adversarial Examples

In this step, we will look at strange behavior of neural nets using our MNIST classification model. Specifically, we will study the behavior of the model when given random images as input.

1. Locate the `random_image()` function in main of `main.py`. The purpose of this function is to generate a random image in the input domain of MNIST. Each image is represented as a 1×784 array of pixels (integers) with each pixel taking a value in $\{0, 1, \ldots, 255\}$. Run the following command for some target label.

   ```
   python3 main.py Step2 simple,512,0.001 50000 100 <target_label>
   ```

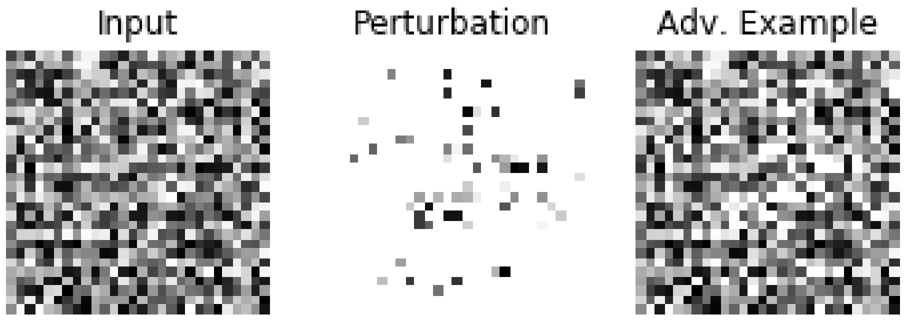
   The code will plot the distribution of predictions for random images (estimated over a large number of samples).

   After this step, turn off the `show_distribution` flag.

2. Run the previous command again. The code will generate a random image and use the "turn off" pixel attack to produce an adversarial example for it (given the target label chosen).

   Paste the plot here. What do notice? Do adversarial examples produced this way exhibit features that you expect given the target label?



target is 5.

Is it easier to generate perturbations for digits that are well-represented in distribution you obtained in step 2.1?

References

[1] Carlini, N., and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 39–57.

[2] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[3] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (2016), IEEE, pp. 372–387.