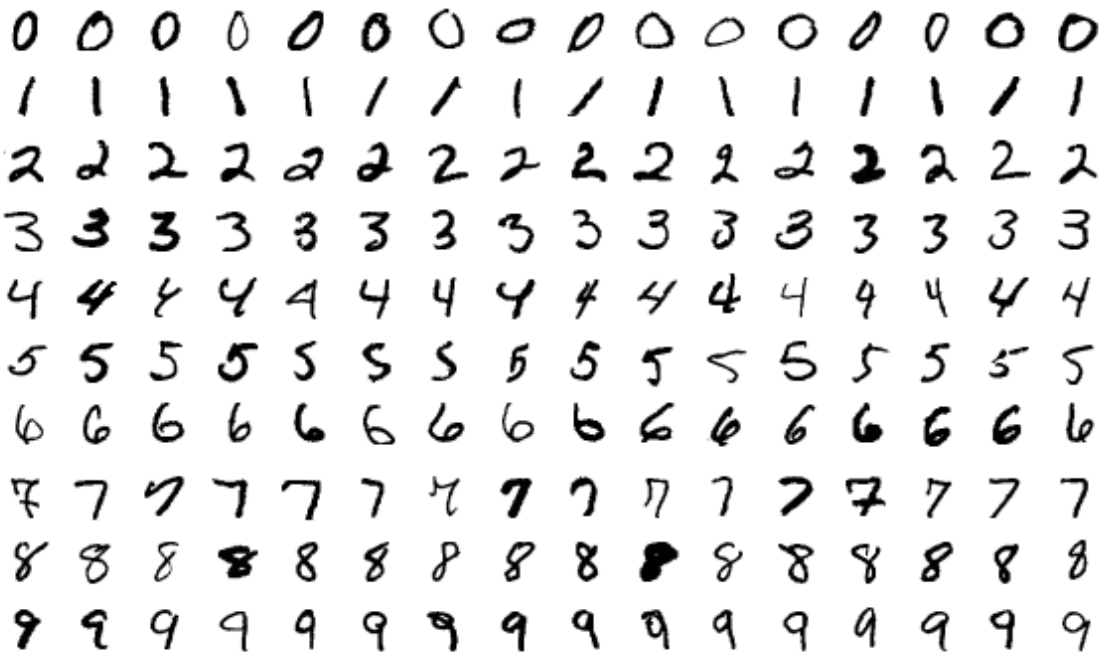
CS590 Project 4: Adversarial sample generation for AI models

Background

In this project, you need to break a digits classification neural network model through generating adversarial samples. This model takes a hand-written image from the data set MNIST (examples shown as follows) and outputs its corresponding digital label, from 0 to 9.



A neural network is a complex parameterized composite function. Learning the model is the same as finding good parameters. Usually good parameters are found by gradient descent of a particular loss.

In the attack scenario, you are given a model with learned parameters. The task is to find some malicious inputs against the neural network model. The definition of malicious input depends on specific field. In this example, it means an adversarial counterpart of a normal input, which looks similar to human but fools the model.

For example, given an image of digit "1", a good model recognizes it as number 1. We want to change the input image a little bit by adding a small perturbation on it. With carefully designed perturbation, the model will misclassify the perturbed input image "1" as number 7.

The quality of the perturbation is measured using some metric. Otherwise the attack doesn't make sense as long as a large enough perturbation can change the image into an arbitrary one. We follow the same criterion in this project as well.

Usually the generated adversarial sample is measured using l_∞ norm between the original image and the adversarial sample. In this project, we use the threshold $l_\infty <= 0.1$ as the criterion for a successful attack. For a grey-scale image, each pixel of the image has the value ranging from 0 to 255, which represents the brightness of pixels. The bound $l_\infty <= 0.1$ ($0.1 * 255 \approx 25$) means that for every pixel between your generated adversarial sample S_a and

the corresponding normal input S_n, the absolute difference should be no larger than 25.5, i.e., $(max|S_a - S_b|) < 25.5$.

Description

In this project, you are tasked to launch one black-box attack and two white-box attacks against a neural network model which recognizes hand-written numbers. For the white-box attack, you have the full knowledge of the model, i.e., weight parameters and architecture of the model. For black-box attacks, you know nothing about the model.

We will use python3 as the programing language, and tensorflow-1.4 as the neural network framework. Most part of the code has been provided, and you need to fill in the missing part. To complete this task, you are expected to know how neural network works, how to program in python and how to use tensorflow library.

For each task, a function is provided for mounting the attack. To complete each attack task, you need to fill in the corresponding code for generating adversarial samples. Your generated adversarial samples are considered successful on the following two conditions:

1. Adversarial samples lead the model to wrong labels (different from their true labels).
2. The perturbation added to the original image is within $l_\infty <= 0.1$ boundary.

Whether you will pass the task or not depends on your attack success rate.

You are only required to fill in the corresponding functions (task1(), task2() and task3()).

Task 1, Noise Attack (black-box attack)

You need to generate adversarial samples without knowing anything about the model. You can try different noises to add on the normal input. You are required to achieve 20% success rate.

Task 2, Gradient Based Attack (white-box attack)

In this task, you need to use the gradient as guidance to generate adversarial samples. You will need to design a loss function which can successfully generate adversarial samples, and use the loss function to calculate the gradient with respect to the input. The input image will be updated using the gradient.

Task 3, Transfer Attack (white-box attack)

In this task, a trick model is provided for mounting the attack. This model has a different softmax layer, whose gradients cannot be easily calculated as in Task 2. You need to use a substitute model to generate adversarial samples, which can be used to fool the trick model.

Todo

Fill the missing part of three task functions in main.py. Compare the results of these three methods in your report. Please submit your code along with the report in a compressed document.