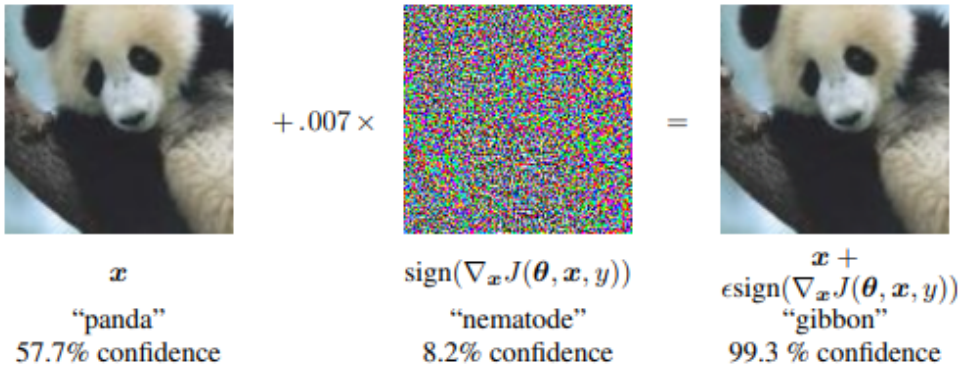
# CS510 Project 3: Adversarial Attacks and Detection

November 13, 2023

## Project Description

In this project, you will be performing adversarial attack and implementing an attack detection.

**Adversarial attacks** Adversarial attacks are inputs formed by applying intentionally crafted perturbations that result in the model outputting incorrect answers (i.e., *"panda"* $\rightarrow$ *"gibbon"*) [3].

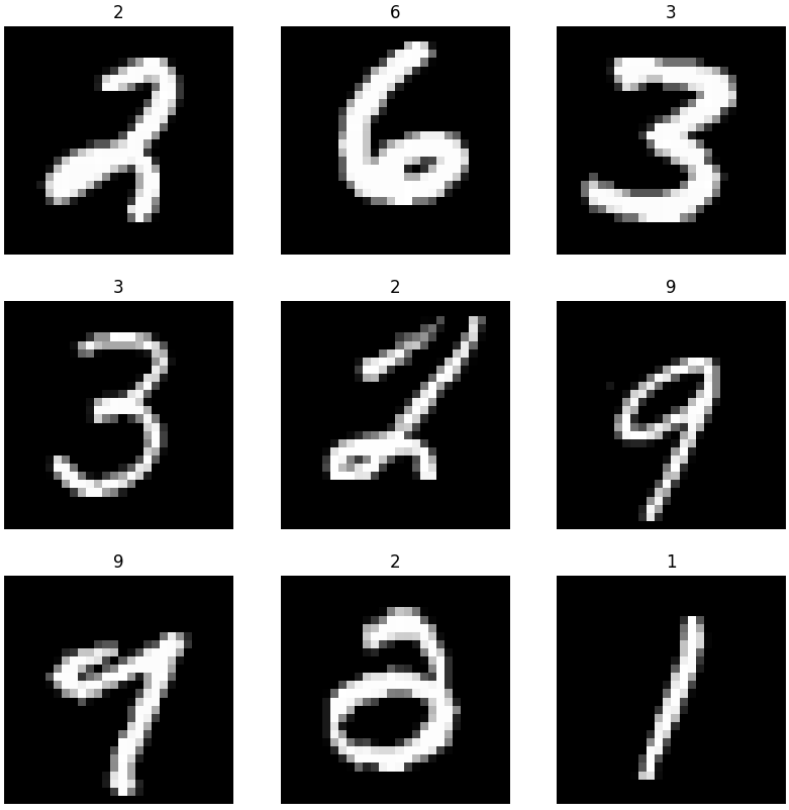



The above figure shows the attack process. The process is to create a filter which aims at maximizing model's outputting *"gibbon"* if added. The process is similar to the model training, i.e., use of gradient descending. However, we are not updating model's parameters (i.e., model tuning). The task is to modify the original images (as little as possible). Therefore, its gradient descent is conducted on images (not model weights) against the output cell of the target label (i.e., *"gibbon"*).

## Task 0: Dataset Preparation and Model Training

It is strongly recommended to use *Google Colab* environment. Download and open the skeleton file on *Google Colab*. Note that task 0 is already implemented. Hence, do not modify the code in this part.

We use *MNIST* dataset. It contains a myriad of black and white (i.e., greyscale) images of 0 to 10 hand-writings. Each image is consisted of 784 pixels (28 x 28) represented by a 0 to 255 integer. We normalize the images by rescaling into 0 to 1 float type variables. Therefore, each image holds a dimension of $28 \times 28 \times 1$. Below images show example images.



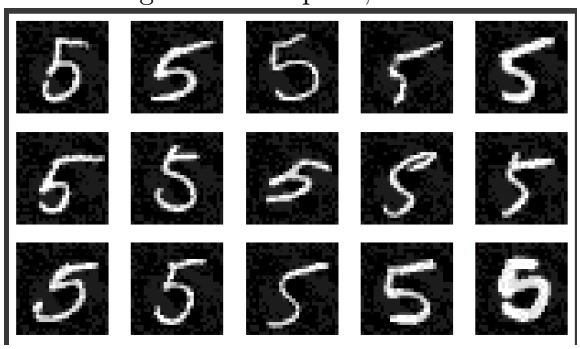

The model is composed of three dense layers (look at the *MyModel* function). In the first dense layer, a 784 size array (after flattened) is transformed into a size of 100 array. And then, it will be reduced into 20 and 10 (by second layer and final layer respectively). Note that it is a classification problem of 10 classes. The model is trained to output corresponding label in the final layer. The highest value among 10 outputs tells the model prediction (i.e., one-hot encoding).

## Task 1: Adversarial Attacks

The goal is to generate 200 adversarial samples. We select images labeled 5 and target 0. To fulfill this task, you need to implement an `make_noise_pattern()` function. This function is to generate perturbations. In that function, the gradient needs to follow;

$$\nabla_x J = \partial(m(x)[0])/\partial(x)$$

where $x$ is an original image and $m$ is the target model. After all, `make_noise_pattern()` returns $sign(\nabla_x J)$ Note that these adversarial images must look like 5 (albeit predicted 0). Below images are exemplars;



## Task 2: Attak Detection

Propose a detection method (please write your idea on a colab text cell). It does not have show state-of-the-art performance. We aim at 80% of f-1 score. In this project, make your own code to conduct an evaluation. Print the precision and recall [2]. For a preliminary data observation (or training another model if needed), all images must be separated from the performance test. Look at guidance in the skeleton file.

## Guidence

This section contains a few aspects with matrix operations. In my early experience, I felt most awkward with these. You can skip this part if you are already experienced.

**Batch images** In most cases, tensorflow operations are designed for batch operations. For example, the model takes batch images (rather than a single image).

Try below code (after finishing Part 0);

```
1 print (x_train[:20].shape)
2 y = model(x_train[:20])
3 print (y.shape)
```

At the first line, we select first 20 images from dataset that result in a $20 \times 28 \times 28 \times 1$ dimension. After line 2, the model will output 20 results, giving a $20 \times 10$ shape of output. To fetch the model's predictions, run `tf.math.argmax(y, axis = 1)`. It will give 20 number of predictions (note that if we set axis as 0, it will output 10 numbers, then what's wrong?). Here, the tip is when you work with matrix variables, keep monitor the shapes and compare with your expectations.

**Tensorflow Variables vs Numpy** For all tensorflow functions, you need to use *Tensorflow* variables. However, *numpy* library holds data. To convert you can use follwings;

```
x.numpy() // from tf variable to numpy
tf.Variable(x) // from numpy to tf variable
```

**Broadcasting Operations** Unlike *Python* array, *tensorflow* variables and *numpy* array support broadcasting. For example, if you want to multiply 3 to all values in the array, you can simply do `3 * x`. Using this aspect, you can conditionally select values from array. Assume you have `x = np.asarray([1, 4, 5])`. Then `x > 1` will result in an array of boolean varaibles, `[False, True, Turue]`. Finally `x[x>2]` gives us `[4, 5]`. You can also do the similar task with tensorflow variables. For detail, refer to [1].

## Grading

**Due Date** 12/8, 11:59PM EST

Submit your finished `cs510_project3.ipynb` (do not clear the outputs) on Brightspace by 12/8, 11:59PM.

## References


[1] Numpy broadcasting. `https://numpy.org/doc/stable/user/basics.broadcasting.html`.

[2] Precision and recll. `https://en.wikipedia.org/wiki/Precision_and_recall`.

[3] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.