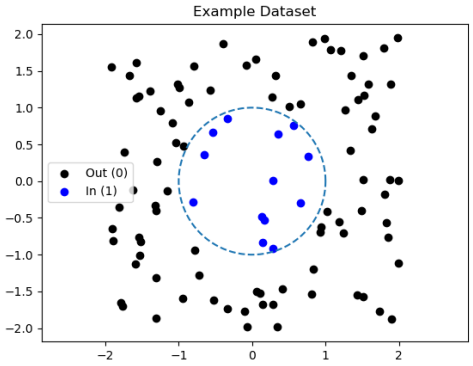
# Assignment 1: PGD

June 2024

## Part 1: Theoretical example

You are given a dataset consisting of points within a square centered at the origin with a side length of 2. Points inside a circle of radius 1 are labeled 1, and points outside this circle are labeled 0.




You want to train a model to learn to determine whether a given point lies within the circle boundary. You decide to train a fully connected neural network with the following structure: an input layer with 2 nodes (for the X and Y coordinate of a point), two hidden layers with 100 nodes each, and an output layer with 2 nodes.

Suppose an attacker wants to use a Projected Gradient Descent (PGD) attack to trick the model. For given points, the attacker generates adversarial points by perturbing their position with respect to the constraint parameter $\epsilon$, such that the model would classify these points to be the opposite side of the decision boundary, contrary to their true label. The $L_2$ norm is used for this exercise.

**Projected Gradient Descent Attack**

- Use **constrained optimization** to find adversarial examples
  - A: $\mathbb{R}^d \rightarrow \mathbb{R}^d$, st., $\mathbf{A}(\mathbf{x}) \rightarrow \mathbf{x}'$, $\mathbf{f}(\mathbf{x}) = t, \delta(\mathrm{x}', \mathrm{x}) \leq \epsilon$
- PGD
  - Initialize $\mathrm{x}^1 \leftarrow \mathrm{x}$
  - For $i = 1, \ldots, S$
    - Compute $\mathrm{x}^{i+1} \leftarrow \mathbf{\Pi}_{\mathcal{C}}(\mathrm{x}^i - \eta \cdot \mathrm{sign}(\nabla \mathbb{L}(\mathrm{f}, \mathrm{x}, \mathrm{c})))$, $\mathbf{\Pi}_{\mathcal{C}}(\cdot)$ is a projection operator
  - Return $\arg\max_i \mathbb{L}(\mathrm{x}_i)$

### Question 1: (1 pt)

For the point (0.2, 0.8), what is the minimum epsilon value required to cause a misclassification? What about the point (-0.6, 0.6)?

### Question 2: (1 pt)

What is the minimum epsilon value required for every point within the circle to be misclassified?

Using the minimum epsilon value computed just now on the point (0.2, 0.8) to generate an adversarial example, what is the maximum possible distance between the adversarial example and the circle boundary?

## Part 2: Coding assignment

In this assignment, we will launch the Projected Gradient Descent (PGD) attack on a realistic dataset.

### Introduction

By perturbing images, we can create adversarial images that will cause the image classification models to underperform. The perturbations would be negligible to human perception but enough to confuse a neural network. This phenomenon highlights one of the many limitations of machine-learning models in comparison to the human brain.

We will conduct a PGD attack on the Resnet-50 model [HZRS15] pre-trained on an image classification task on the ImageNet-1k dataset. This dataset spans 1000 object classes and the training set contains approximately 1000 images for each class. All input RGB images are resized to 232 x 232, and the center 224 x 224 is cropped out, and then normalized for the mean and standard deviation of the ImageNet dataset. Hence the Resnet model takes 3 x 224 x 224 tensors as input and outputs a 1000-dimension vector. After applying the softmax function on the output vector, we can interpret the resulting vector as a probability vector over the 1000 possible classes. One may take the arg max as the model's predicted class or look at the top k classes for a more lenient evaluation metric.

$$f\colon [0,1)^{3\times 224\times 224} \to \mathbb{R}^{1000}$$

In particular, we will be using $L_\infty$ norm for this exercise.

## Environment Setup

To set up your environment, first install the suitable version of Pytorch (click here for their main page)[1]. Then download the Python scripts. Your directory structure should look like this:

```
pgdattack
├─ resnet_attack_todo.py
└─ launch_resnet_attack.py
```

Next, run the following command to install the required dependencies.

```
pip install datasets
```

Then check the next section for the steps required for loading data.

Once your environment is set up, run the following command to verify that you have installed all required Python modules and files.

```
python launch_resnet_attack.py --test --batch_num 1 \
--batch_size 10 --results 'test_launch'
```

## Data Loading Instructions

The ImageNet dataset is available on Huggingface, but it is a gated dataset and you need a Huggingface account to access it (click here to create an account)[2]. Since it is an extremely large dataset, you may utilize streaming mode to avoid downloading large files to your computer, and the preprocessing procedure can be applied on the fly during the attack loop. This may lead to longer run times but significantly reduces the amount of storage required.

Once you have an account, head to the ILSVRC/imagenet-1k dataset ( click here[3] ) and select "accept conditions" to access its files and content. Then go to your account settings to create a user authentication token. Make sure to add ILSVRC/imagenet-1k to the list of readable datasets. In your command line interface, enter `huggingface-cli login` and paste your user token. Now you can freely access this dataset when executing the Python scripts from the command line.

[1] pytorch.org/get-started/locally/
[2] huggingface.co/join
[3] huggingface.co/datasets/ILSVRC/imagenet-1k

## Question 3: Generate Adversarial Images (10pts)

In this task, you are asked to complete the relevant parts of `resnet_attack_todo.py` makred with `TODO` and generate adversarial images.

You are given an entry script `launch_resnet_attack.py` that provides a pre-trained model and a data loader, where the input data is loaded and pre-processed. Run the script with the line argument `--results 'adv_images'` to specify the filename to which the resulting adversarial images are saved. You may also specify the batch size and the number of batches considering the amount of computer storage you have access to.

When executed, the script calls `ResnetPGDAttack.pgd_batch_attack()` specified in the file `resnet_attack_todo.py`, where the PGD attack is partially implemented. As is, the script will return the original images and save them under the `results` directory. Your task is to complete the implementation of `pgd_attack` to perform the gradient update step and the $L_\infty$ norm projection step in each iteration of the PGD attack, so that the script will return adversarial images as desired. The unimplemented code sections are marked by `TODO` comments.

After executing the entry script, your directory structure should look something like this:

```
pgdattack
├── resnet_attack_todo.py
├── launch_resnet_attack.py
└── results
    └── adv_images
```

### Evaluation

Make a plot of various epsilon values (at least three values between 0 and 1) vs accuracy on adversarial images. Clearly label the hyper-parameters used, such as the batch number, batch size, alpha value. See Figure 1 for an example. You are graded on the accuracy of both your plots and your code completion for the `resnet_attack_todo.py` file.

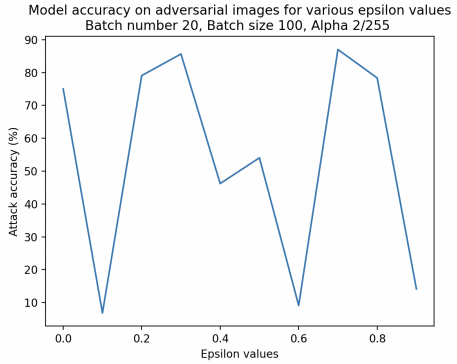



Figure 1: Example plot for Question 3

## Question 4: Adversarial Training (8pts)

For this task, you are asked to finetune the Resnet model by feeding the adversarial images you just created.

By feeding adversarial images as extra inputs, the model learns to resist the PGD attack and predict correctly even when perturbed images are presented as inputs. Therefore, adversarial images can be used to improve model robustness. Don't forget to clearly document your code!

### Evaluation

Your model will be assessed against 2000 hidden adversarial images (i.e. test cases). You pass the test if your model achieves an accuracy above 50% on the hidden adversarial images. You excel in the test if it achieves an accuracy above 80%. You are graded on both the correctness of your code and your model's test performance.

# Submission instructions

Upload a single PDF file containing the following content:

1. Answers to questions 1 and 2
2. Plots from questions 3
3. A link to the following files
   (a) The completed `resnet_attack_todo.py` file from question 3
   (b) All Python files you created for question 4
   (c) The fine-tuned model file from question 4

Please follow the instructions carefully to ensure the auto grader functions correctly.

# References

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.