Project 1: Attack a self-driving car with an adversarial stop sign and noise

24-784 Special Topics: Trustworthy AI Autonomy

Prof. D. Zhao

Due: Friday Feb 10th, 23:59:59 EST

- Your online version and its timestamp will be used for assessment.
- We will use Gradescope to grade. The link is on the panel of CANVAS. This homework is to be completed **individually**. Please abide by the ACADEMIC INTEGRITY POLICY on the syllabus.
- Submit **your_code.py** or **your_code.ipynb** or **your_code.zip** to Gradescope under **Homework-1 Programming** and your solutions in **.pdf** format to **Homework-1**. Insert the generated images and plots in the **.pdf**. We will randomly test your code and manually check all answers.
- Please familiarize yourself with Python programming, if you have never used it previously. We only accept Python for the coding of all homeworks. To that end, attending Python recitation bootcamp at the beginning of this course is highly recommended.
- We advise you to start with the assignment early. All the submissions are to be done before the respective deadlines. For information about available credits for late days, refer to the syllabus in CANVAS.
- The accompanying Google Colab Notebook is available at: https://colab.research.google.com/drive/14uKttX2CpzmnCOveJgeUaaeIS_-8knRt?usp=sharing

1 Introduction

This homework presents you a hands-on opportunity to assess AI robustness by visualizing neural-network layers, generating adversarial examples and estimating the robustness of neural network when its inputs are perturbed by Gaussian noise. The rigorous formulations are discussed during lectures. Here, we use pre-built packages and demonstrate their use on pretrained models, such as AlexNet and ResNet-18.

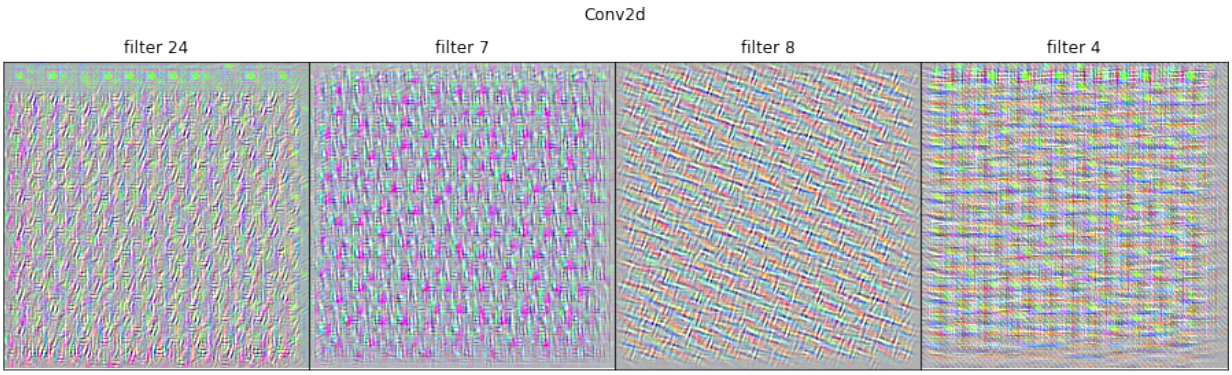


Figure 1: Visualization of CNN filters of AlexNet, Layer 3

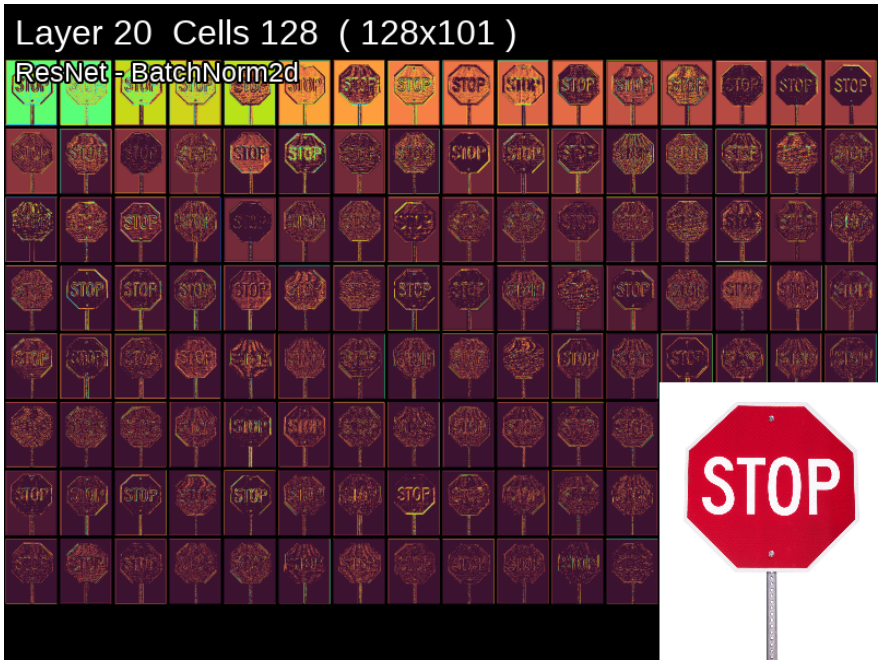


Figure 2: Visualization of the Layer 20 saliency map of ResNet18 with a stop sign image

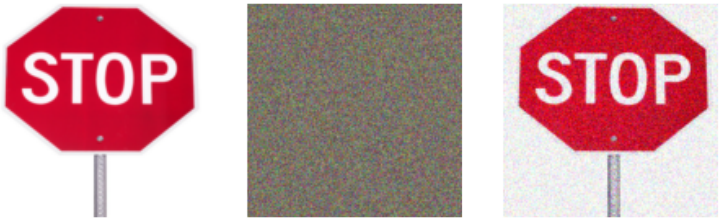


Figure 3: Left: original image (classified as 'street sign', class id 919). Middle: Adversarial perturbation. Right: Adversarial example (classified as 'street car', class id 829)

2 Model

2.1 Neural network visualization

For the task of visualization, we will focus on Convolution Neural Network (CNN) layers. CNN is a type of neural network that is often used to perform tasks such as image classification, object detection, instance segmentation, etc. The powerful CNN models are often really complex, hence termed as 'black box' models signifying the difficulties to understand how they are able to learn complex features or patterns present in the input. We will look at two ways of visualizing CNN models: activation maximization of the convolutional layers (Fig. 1) and saliency map given an input image (Fig. 2).

Maximal activation visualization generates an input image that maximizes the filter/kernel output activations. It allows us to visualize what sort of input patterns activate a particular filter. Saliency maps specifically plot the gradient of the predicted outcome from the model with respect to the input, or pixel values. By calculating the change in predicted class by applying small adjustments to pixel values across the image we can measure the relative importance of each pixel to the ultimate prediction by the model. We will use FlashTorch package for maximal activation and MapExtract for saliency map visualization.

2.2 Adversarial example

We will also implement simple adversarial attacks, both targeted and untargeted types. We will attack a pretrained ResNet-18 model, which we represent by $f(\cdot;\theta)$. For a given a pair of image and label (x, y), we want to design an adversary that can give adversarially perturbed image

$$\tilde{x} = x + \tilde{\delta},$$

where $\tilde{\delta}$ is the adversarial perturbation. The goal is either to make the perturbed image be wrongly predicted (i.e. $\tilde{x}$ is to be predicted as any class $y' \neq y$ for untargeted attack and as a specific target class y' for targeted attack). We will use ScratchAI package in the implementation.

2.3 Evaluation of the probabilistic robustness

Probabilistic robustness measures the how robust a trained network is in performing its task successfully when the input is subject to random perturbation. We will assess the

probabilistic robustness of ResNet-18 and seek to estimate its misclassification rate:

$$\mu = \mathbb{E}_{\delta \sim p}[1(f(x+\delta;\theta) \neq y)]$$

with high-degree of precision. We will compare Monte Carlo (MC), one of the most straightforward sampling methods, with Importance Sampling (IS). MC uses an estimator

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} 1(f(x+\delta_i;\theta) \neq y) \tag{1}$$

with n i.i.d. samples $\delta_i \sim p, i = 1, 2, \cdots, n$. On the other hand, IS uses samples $\delta_i \sim \tilde{p}, i = 1, 2, \cdots, n$ and computes

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} 1(f(x+\delta_i;\theta) \neq y) \cdot \frac{p(\delta_i)}{\tilde{p}(\delta_i)}. \tag{2}$$

The term $\frac{p(\delta_i)}{\tilde{p}(\delta_i)}$ is called the likelihood ratio, which is the ratio of the likelihood of sample δ_i under original distribution p to the proposal distribution $\tilde{p}$. For IS to work, the proposal distribution $p(\cdot)$ needs to be optimally chosen to attain estimation unbiasedness and reduced variance.

A good measure of evaluation efficiency is the relative error of the estimator $\hat{\mu}_n$ with n samples. It is computed as

$$RE(\hat{\mu}_n) = \frac{\sqrt{\mathrm{Var}(\hat{\mu}_n)}}{\mathbb{E}[\hat{\mu}_n]} = \frac{\mathrm{Std}(\hat{\mu}_n)}{\mu}, \tag{3}$$

which quantifies the the estimator variation as a percentage of a small but non-zero target value μ. An estimator $\hat{\mu}_n$ is good if its relative error quickly decays in n. We will compare the relative error of MC and IS in evaluating the probabilistic robustness of ResNet.

3 Resources

3.1 Dataset

We use a single stop sign image for this homework. In Colab you will download it here: https://static01.nyt.com/images/2011/12/11/magazine/11wmt1/mag-11WMT-t_CA0-jumbo.jpg.
In ImageNet, stop signs are considered 'street sign' class (class id 919).

3.2 Code

We provide a Google Colab notebook to help you set up the environment. The main packages to use are TorchVision, FlashTorch, MapExtrackt, and ScratchAI. These can be installed by running:

Figure 4: Stop sign image used for this homework

```
!pip install scratchai-nightly
!pip install torchvision
!pip install flashtorch
!pip install mapextrackt
```

Other standard packages like Numpy, Matplotlib, and PyTorch are also required for plotting, basic arithmetic, and running the models. Below are the main calls that you would need. More details are available in the documentation of these packages, listed in Section 5.

3.2.1 Loading pretrained models

We will use the pretrained AlexNet and ResNet-18 models. To load them, call:

```
from torchvision import models
resnet = models.resnet18(pretrained=True).eval()
alexnet = models.alexnet(pretrained=True).eval()
```

3.2.2 Maximal activation using FlashTorch

Maximal activation algorithm starts with random noise image and performs gradient ascent iterations to increase filter activations. This can be done with using FlashTorch package. Given a pretrained `model`, we visualize the maximal activation of specific `filters` at a `layer` by calling:

```
GradientAscent(model).visualize(layer, filters)
```

The `GradientAscent` is imported by calling:

```
from flashtorch.activmax import GradientAscent
```

3.2.3 Saliency Map using MapExtrackt

The following three lines of codes output the saliency map for certain `layer` of the predefined `model` when given an input image stored at `img_path`:

```
fe = FeatureExtractor(model)
fe.set_image(img_path)
fe.display_from_map(layer)
```

The `FeatureExtractor` can be imported by calling:

```
from MapExtrackt import FeatureExtractor
```

3.2.4 Uniform random perturbation using ScratchAI

To perturb a base image (`image`) with uniform perturbation of the interval -1 to 1, you can simply call:

```
perturbed_image = attacks.noise(image, eps=1)
```

We recommend importing all ScratchAI functions by calling

```
from scratchai import *
```

3.2.5 Adversarial attack using ScratchAI

To adversarially attack model `net_name` using `attack_method` for the image stored in `image_path`, call

```
one_call.attack(image_path, atk=attack_method, nstr=net_name, ret=True)
```

We will attack ResNet-18 in this homework, so we use `net_name = 'resnet18'`. We will implement two attack methods here: Fast Gradient Method (`attacks.FGM`) and Projected Gradient Descent ((`attacks.PGD`). Note that FGM method uses L_2 (instead of L_∞ for Fast Gradient Sign Method introduced in the lecture). The argument `ret` is there to indicate whether to return the resulting images, labels, and confidences of the attack or not.

4 Problem Set

In your experiments, set the random seed at 0 for reproducibility whenever you use functions like attacks.noise(), torch.randn(). You need to reset the random seeds to 0 each time you call those random functions. Examples of setting seed to 0 are shown below and also in Colab.

```
torch.manual_seed(0)
np.random.seed(0)
```

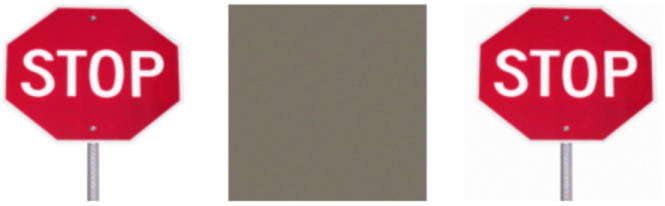
Exercise 1. Adversarial example generation for ResNet-18.

1. Load the public ResNet-18 model and use it to predict the label for the provided stop sign image. What is ResNet-18 prediction? What is its confidence?
2. Use the stop sign image as your base image x. Perform untargeted attack with uniformly random perturbation, i.e. $\tilde{x} = x + \delta$ where $\delta \sim \text{Uniform}(-\epsilon, \epsilon)$. Show one perturbed image for each $\epsilon \in \{0.1, 0.5, 1\}$. Does ResNet-18 misclassify any of these images?
3. Perform untargeted attack to ResNet-18 with FGM. Show the original image, the adversarial perturbation, and the perturbed image. What is ResNet-18 prediction on this adversarial example? What is its confidence?
4. Perform targeted attack to ResNet-18 with PGD, aiming for target label of 'street car' (class id 829). Show the original image, the adversarial perturbation, the perturbed image for each model. What is ResNet-18 confidence on this adversarial example?

Solution:

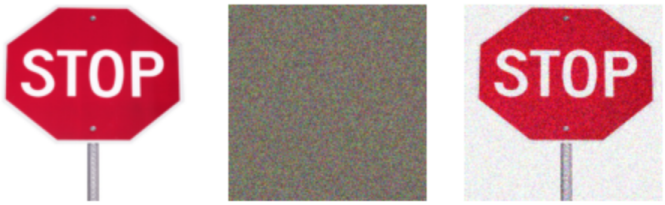
1. `>> ('street sign', 13.558080673217773)`
2. The output of the randomly perturbed attacks for $\epsilon = 0.1, 0.5, 1.0$ are the following:

```
('street sign', 14.593377113342285, 919)
```



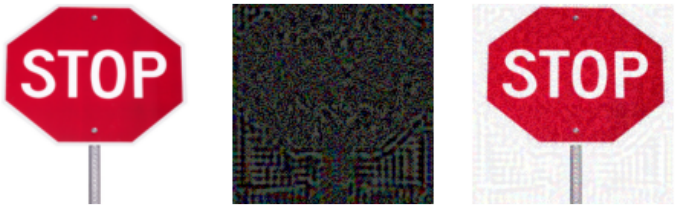
```
('doormat, welcome mat', 14.955455780029297, 539)
```

```
('doormat, welcome mat', 16.598552703857422, 539)
```



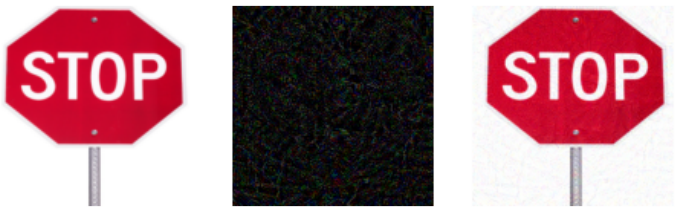
ResNet-18 misclassifies the perturbed image with $\epsilon = 0.5$ and $\epsilon = 1$.

3. The output of the FGM attack:



```
>> ('doormat, welcome mat', 14.294464111328125))
```

4. The output of the PGD attack:



```
>> ('streetcar, tram, tramcar, trolley, trolley car',
↪  30.324739456176758))
```

Exercise 2. Neural network visualization.

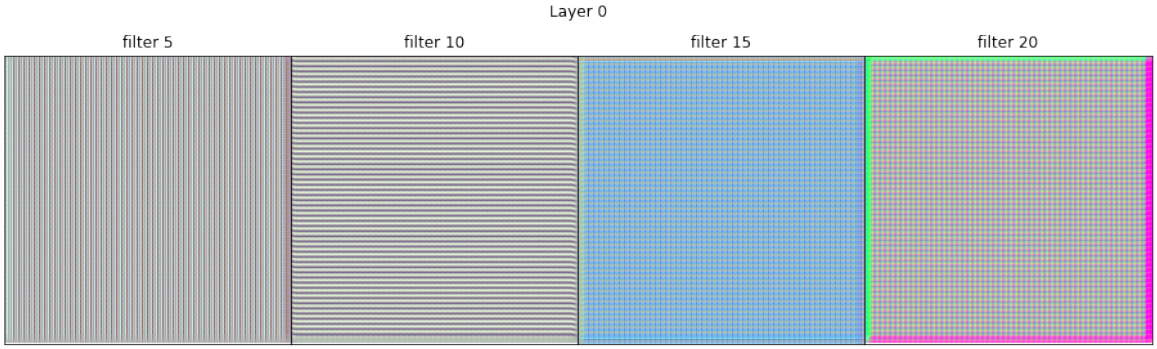
1. Use FlashTorch to visualize the maximal activation of filter 5, 10, 15, and 20 on the first convolutional layer (layer index 0) of AlexNet!

2. Still using FlashTorch, now visualize the maximal activation of filter 5, 10, 15, and 20 on the last convolutional layer (layer index 10) of AlexNet!

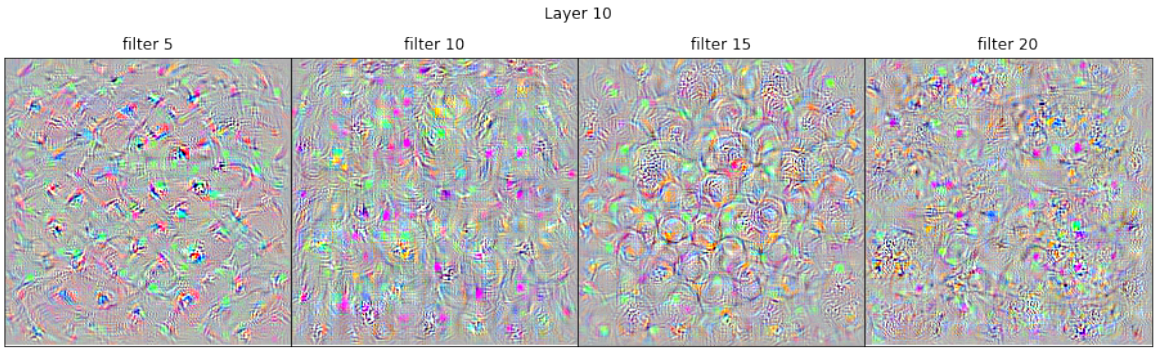
3. Now use MapExtrackt to visualize the salience map of the last convolutional layer (layer index 10) of AlexNet given the stop sign input!

Solution:

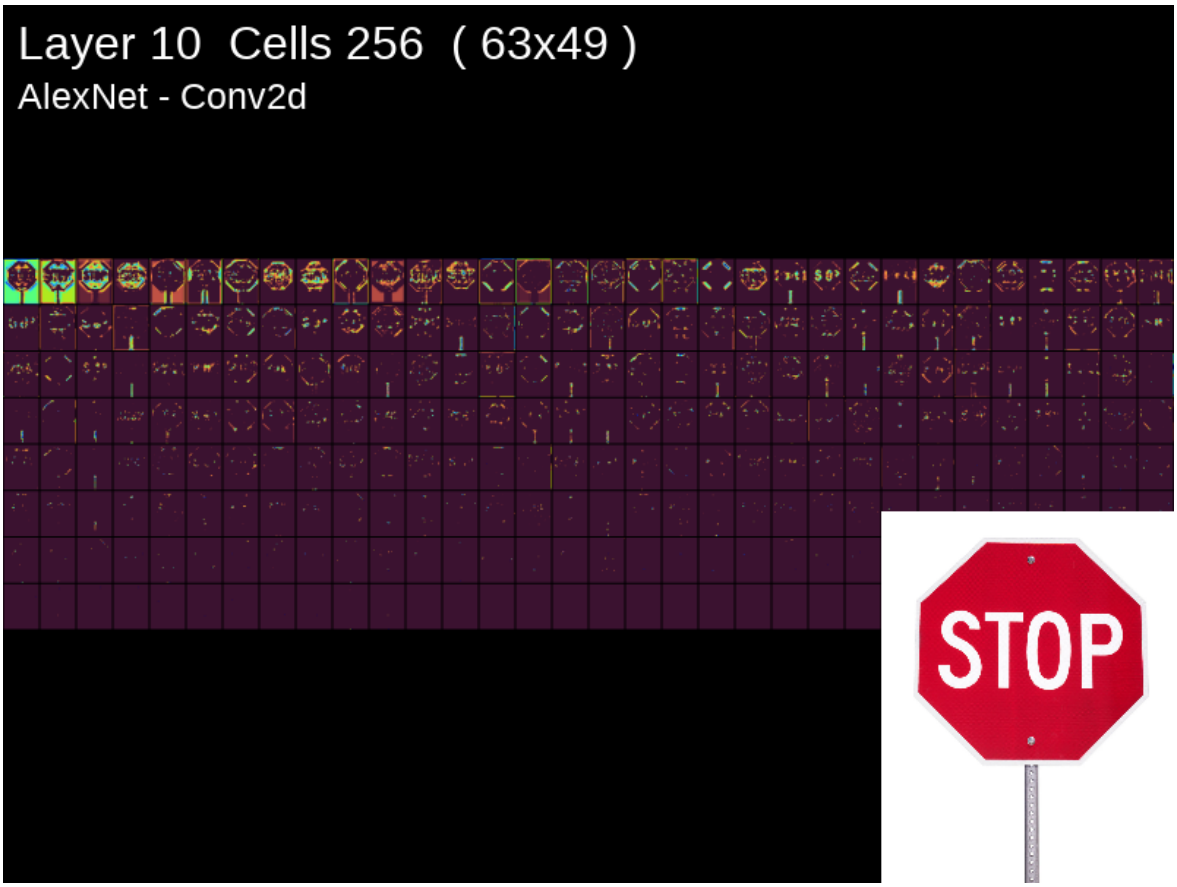
1. AlexNet layer 0, filters [5, 10, 20, 25]:



2. AlexNet layer 10, filters [5, 10, 20, 25]:



3. AlexNet saliency map of layer 10 with stop sign input:


Layer 10 Cells 256 (63x49)
AlexNet - Conv2d
STOP

Exercise 3. Evaluation of probabilistic robustness.

1. Estimate μ, the misclassification rate of ResNet-18 model for the stop sign image subject to random perturbation $\delta \sim p$. Use MC to estimate μ with $p = \mathcal{N}(0, \sigma^2 I)$ where $\sigma^2 = 0.2$ is the square of the standard deviation. Use $k = 10$ replications to compute the mean and standard deviation of your estimator. Plot the mean of your estimator $\hat{\mu}_n$ vs $n = [50, 100, 150, \cdots, 500]$. Plot also the confidence interval $[\hat{\mu}_n - \text{Std}(\hat{\mu}_n),\ \hat{\mu}_n + \text{Std}(\hat{\mu}_n)]$ on this plot. An example on how to plot confidence interval is already in Colab.

2. Suppose that $\mu = 0.03$. Compute and plot the relative error of $\hat{\mu}_n$ vs $n = [50, 100, 150, \cdots, 500]$.

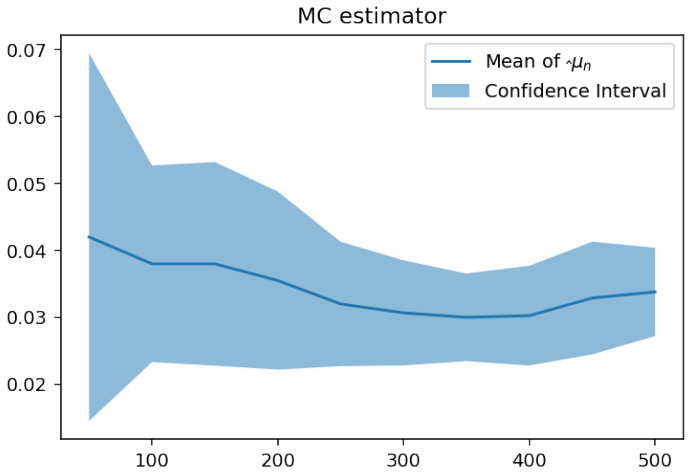
3. [Sample around adversarial example] Apply FGM attack again on ResNet-18 and denote by $\tilde{x}$ the adversarial example you obtain. Estimate μ with random perturbation $\delta \sim \tilde{p}$ where $\tilde{p} = \mathcal{N}(\frac{1}{3}\tilde{x}, \sigma^2 I)$. In other words, we construct an IS by sampling closed to the adversarial examples. Report the misclassification rate you observe with $n = [50, 100, 150, \cdots, 500]$.

4. Suppose now that we treat $\tilde{p}$ as IS distribution. Compute IS estimator $\hat{\mu}_n$ and plot $\hat{\mu}_n$ vs$n = [50, 100, 150, \cdots, 500]$. Show also the confidence interval on this plot.

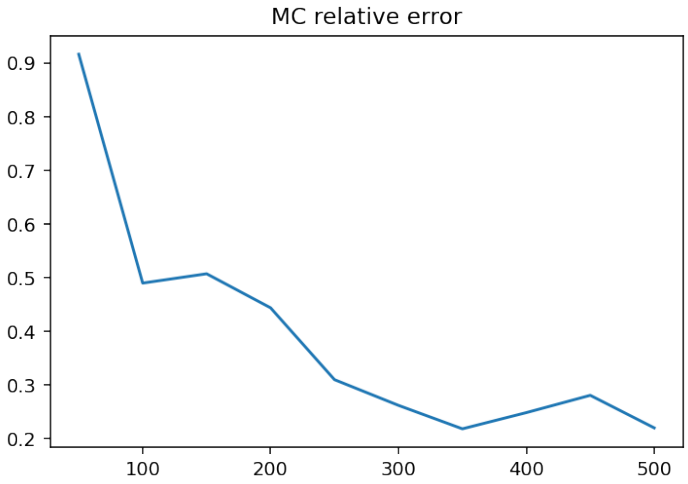
5. Compute and plot the relative error of the IS estimator $\hat{\mu}_n$ vs $n = [50, 100, 150, \cdots, 500]$. Is it better than the MC sampling? If not, then explain why.

Solution:

1. The mean and confidence interval of MC estimator:

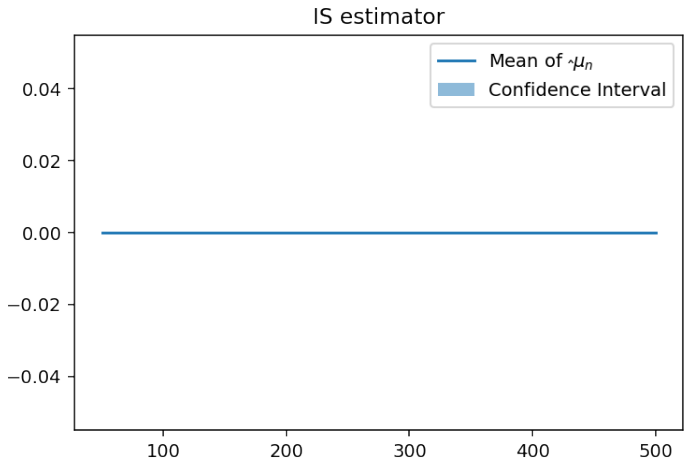


2. The relative error of MC estimator:

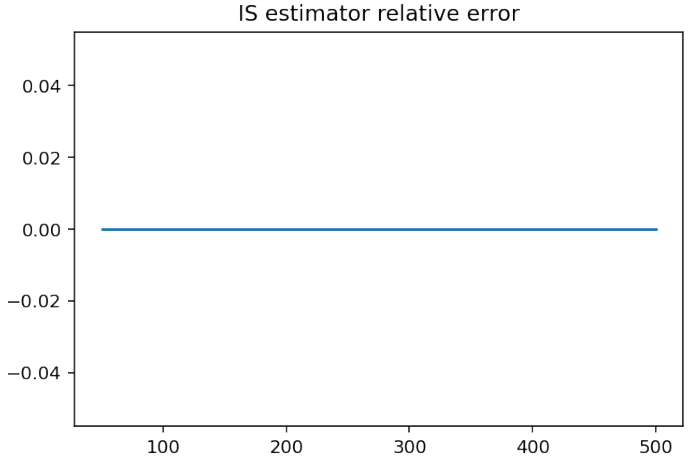


3. The misclassification rate of the close-to-adversarial examples
   means = [0.478 0.473 0.478 0.479 0.4804 0.478 0.484 0.4855 0.48733333 0.4904]

4. The mean and confidence interval of IS estimator:



5. The relative error of IS estimator:



In this case, IS estimator does a poor job in estimating μ, mainly due to: 1) the degeneracy of the likelihood ratio when computed for high-dimensional random variables and 2) overly simple mean-shift scheme, that does not give optimal IS result.

5 Reference

1. CMU 24784 Spring 2023 HW1 Google Colab Notebook, https://colab.research.google.com/drive/14uKttX2CpzmnCOveJgeUaaeIS_-8knRt?usp=sharing. Please copy it to your own account for editing.
2. TorchVision models, https://pytorch.org/vision/0.8/models.html
3. ScratchAI, https://github.com/iArunava/scratchai
4. FlashTorch, https://github.com/MisaOgura/flashtorch
5. MapExtrackt, https://github.com/lewis-morris/mapextrackt