Project: Adversarial Attacks

Spring 2020
(Due: April 24, 2020, 11:59am EST)

Please submit your project report to **blackboard** by 11:59am EST.

Instruction: The project report should be limited to 10 pages. We highly encourage you to type your solution in LaTeX. Template can be found on the course website. Consider using `overleaf.com`, which is an online LaTeX typing application. If you use Word, the margin must be at least 1 inch on each side, and the font should be 11 points or larger. The 10-page limit includes all text, tables, and figures. References are not included in the 10-page limit. If you want to write equations on paper, you need to scan it and put it in the report. We only accept the final 10-page PDF. **Violation of the page limit and format will not be graded**.

Objective

Here are the objectives for this project:

(a) Understanding the details of how some of the popular adversarial attacks operate on a Gaussian classifier.

(b) Implementing the Carlini-Wagner (CW) attack on a Gaussian classifier.

(c) Develop a defense system for the CW attack.

(d) Analyze the limitation of your defense system.

Exercise 1: Adversarial Attacks on Gaussian Classifier - Theory

To get started with this project, let us derive the adverarial attacks' solutions/iterates on 2-class Gaussian classifiers first, and observe some interesting theoretical properties of these attacks.

We recall the general form of the decision boundary of a 2-class Gaussian classifier:

$$\mathcal{H} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \;\middle|\; g(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T(\boldsymbol{W}_j - \boldsymbol{W}_t)\boldsymbol{x} + (\boldsymbol{w}_j - \boldsymbol{w}_t)^T\boldsymbol{x} + (w_{j,0} - w_{t,0}) = 0 \right\}, \quad j \neq t \tag{1}$$

where

$$\boldsymbol{W}_i = \boldsymbol{\Sigma}_i^{-1},\ \boldsymbol{w}_j = -\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i,\ w_{i,0} = \frac{1}{2}\boldsymbol{\mu}_i\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i + \frac{1}{2}\log(|\boldsymbol{\Sigma}_i|) - \log(\pi_i) \tag{2}$$

$\boldsymbol{\Sigma}_i$ is the covariance matrix of class i, $\boldsymbol{\mu}_i$ is the mean vector of class i, and π_i is the prior probability of class i. In the case of equal covariance matrix $\boldsymbol{\Sigma}$ among the two classes, (1) becomes linear

$$\mathcal{H} = \left\{ \boldsymbol{x} \in \mathbb{R}^d \;\middle|\; g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = 0 \right\} \tag{3}$$

where

$$\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_t),\ w_0 = -\frac{1}{2}(\boldsymbol{\mu}_j + \boldsymbol{\mu}_t)\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_t) + \log(\frac{\pi_j}{\pi_t}) \tag{4}$$

Tasks (15 points):

(a) We first study the min-distance attacks.

(i) Derive the minimum l_2 and l_∞ attacks in the linear case, based on (3).

(ii) Derive the iterates that solve the DeepFool attack, based on (1).

(iii) Find an example decision boundary where the DeepFool iterates **never** converge. A one-dimensional $g : \mathbb{R} \to \mathbb{R}$ is sufficient. Feel free to include visual illustrations.

(b) Now the max-loss attacks.

(i) Derive the max-loss l_∞ attack in the linear case, based on (3).

(ii) Derive the FGSM attack with $J(\boldsymbol{x}) = -g(\boldsymbol{x})$, based on (1).

(iii) Derive the iterates of the I-FGSM attack with $J(\boldsymbol{x}) = -g(\boldsymbol{x})$, based on (1).

(c) Finally, we examine the regularization based attacks.

(i) Derive the regularization-based attack in the linear case, based on (3).

(ii) Derive the iterates of the CW attack, based on (1).

Exercise 2: CW Attack on Gaussian Classifier - Non-Overlapping Patches

The goal of this exercise is to carry out the CW attack on the version of the cat-grass Gaussian classifier which operates on non-overlapping patches you trained and tested in homework 4. Before going straight to coding, let us examine the problem setup first.

First of all, let us define the notations. We denote an image as a vector $\boldsymbol{x} \in \mathbb{R}^d$. An image is of course an array. By denoting it as a vector we mean that we turn the array into a column vector. In Python, you can do reshape of the image into a vector. An image will contain d pixels. Let us now go to the i-th pixel. The i-th pixel is denoted as $x_i \in \mathbb{R}$. Okay, now let us define a neighborhood of x_i. For example, we can create an 8×8 neighborhood around x_i. Let us call this a **patch**, and denote it as $\boldsymbol{x}_i\mathbb{R}^{64}$. So now you have three variables:

- $\boldsymbol{x} \in \mathbb{R}^d$: The image, represented as a d-dimensional vector.
- $x_i \in \mathbb{R}$: The i-th pixel, which is a scalar.
- $\boldsymbol{x}_i \in \mathbb{R}^p$: A patch with center at the i-th pixel, represented as a p-dimensional vector.

Let us talk about how to attack. To attack the classifier, we just need to perturb each non-overlapping patch in the image independently using the CW attack method. More specifically, we are essentially solving for

$$\boldsymbol{x}_i^* = \underset{\boldsymbol{x}}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{x}_{i,0}\|_2^2 + \lambda \max(g_j(\boldsymbol{x}) - g_t(\boldsymbol{x}), 0) \tag{5}$$

where $\boldsymbol{x}_{i,0} \in \mathbb{R}^{64}$ is the vector representing the i^{th} patch in the original cat-grass image, and $\boldsymbol{x}_i^* \in \mathbb{R}^{64}$ is the optimally perturbed i^{th} patch vector.

Let us now implement the attack. Note that the algorithm can be implemented in a variety of ways, parts (a) and (b) below only outlines one way of implementation, and provide you with the necessary parameters and conventions we adopt; feel free to implement the algorithm in any way you like.

Tasks (15 points):

(a) The CW attack's optimization problem is solved by gradient descent; examine the expression (25) carefully. To get started, implement the calculation of the gradient of the objective function first, based on the results you obtained from exercise 1(c)(ii). Use regularization coefficient $\lambda = 5$ for now. You can fill in the blanks of the following code if you prefer.


```
def gradient(patch_vec, lambda, target_index, other necessary parameters):
    # Calculate g_j, g_t, determine if patch_vec is already in target class
    # If patch_vec is in target class, do not add any perturbation (return zero gradient!)
    # Else, calculate the gradient, using results from 1(c)(ii)
    return grad
```

(b) Write a function that implements the gradient descent algorithm on every non-overlapping patch in the image, using results from (a). Use $\alpha = 0.0001$. Terminate when the number of iterations reaches 300, or when the size of change $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\|_2$ is smaller than 0.001. You can fill in the blanks of the following code if you prefer.

```
def CW_attack(img_matrix, lambda, target_index, other necessary parameters)
    # Preprocess the data, and set up parameters
    # Start gradient descent
    while itr_num <= 300 and change >= 0.001:
        for i in range(M):
            for j in range(N):
                if i + 8 > M or j + 8 > N or i % 8 != 0 or j % 8 != 0:
                    continue
                # Perturb the patch with top left pixel (i,j) in img_matrix by
                # pert_patch = patch_vec + alpha*grad
        # Process the perturbed image matrix, display perturbed image, etc.
        change = np.linalg.norm(pert_img_curr - pert_img_prev)
        itr_num += 1
    # Return the attack image
```

Remark: he Python function `numpy.linalg.norm` can evaluate the norm of a vector.

Hint: The perturbed patch's entries might have values outside the bound $[0, 1]$. Use `numpy.clip` to avoid this problem, for example, write `np.clip(patch + alpha*grad, 0.0, 1.0)`. This is only one somewhat clumsy way to resolve the issue though, as there are many nicer functions to use, such as softmax, tanh, sigmoid, ReLU, ELU, but for the sake of simplicity, we will just use simple clipping.

(c) For $\lambda = 1, 5, 10$, obtain the following plots and data:

(i) The attack, i.e. the final perturbed image;

(ii) The perturbation added to the original image to obtain the attack;

(iii) Frobenius norm of the perturbation

(iv) The classifier's output on the attack, i.e., the number of patches classified as cat and as grass for the attack image;

(v) During gradient descent, plot the classifier's output on the perturbed image iterates per 50, 10, 5 iterations for $\lambda = 1, 5, 10$ respectively.

Comment on your plots and data. You can support your claims with more results than the above items.

(d) What do you observe when you increase the step size α? Explain your answer. You can include results from the program to support your claims.

Exercise 3: CW Attack on Gaussian Classifier - Overlapping Patches

In this exercise, we implement the CW attack on the overlapping-patches version of the cat-grass Gaussian classifier. This exercise is very similar to the previous one in terms of implementation, however, we should be careful with the problem formulation of the attack.

Since the classifier operates on every patch in the image, still using (5) as our attack's optimization problem here is not a wise decision. The reason is that, even though we still want to force the resulting image to be as "close" to the original image as possible, we also want to take into consideration the "interference" between the overlapping patches. For example, directly using (5) on each overlapping patch could cause the situation in Figure 1 to happen.

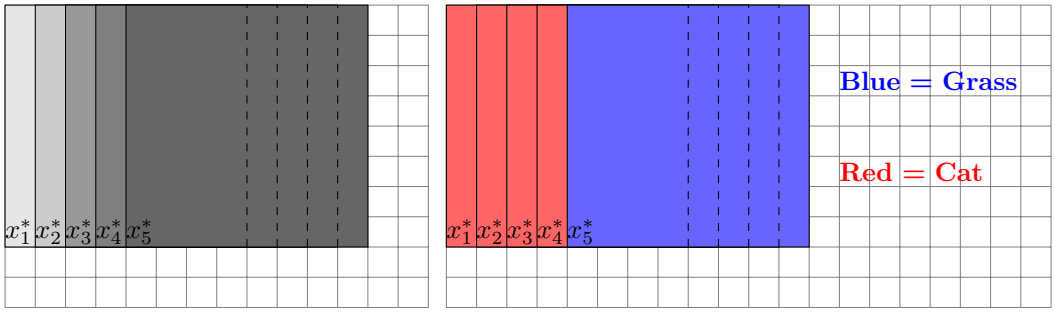

Figure 1: **Left**: five outputs from (5), all being perturbed from the **cat** class to the **grass** class by the algorithm; **Right**: classifier's decisions for the five patches, the first four are still classified as cat, contrary to our attack's intent, due to parts of them being overwritten by x_5^*

So here is what we are going to do in order to create an attack for **overlapping patches**. We are going to introduce an operator called the **patch extraction** operator $\boldsymbol{P}_i$. What this patch extraction operator does is to extract the i-th patch from the image. Formally, $\boldsymbol{P}_i$ is defined as an matrix such that

$$\boldsymbol{x}_i = \boldsymbol{P}_i \boldsymbol{x}. \tag{6}$$

In plain words, if you give me the image $\boldsymbol{x}$, I can extract the i-th patch by multiplying $\boldsymbol{x}$ with a matrix $\boldsymbol{P}_i$. This matrix $\boldsymbol{P}_i$ is a binary matrix with only one "1" on each row. The location of the "1" depends on which pixel you want to extract. For example, if $\boldsymbol{x}$ is a 3-element vector $\boldsymbol{x} = [x_1, x_2, x_3]^T$, and if $\boldsymbol{P}_i = [1, 0, 0]$, then $\boldsymbol{P}_i\boldsymbol{x}$ will extract the first pixel from $\boldsymbol{x}$. The dimension of $\boldsymbol{P}_i$ is p-by-d, where p is the size of the patch and d is the size of the input image.

With the patch extraction defined, let us consider the new optimization problem

$$\begin{aligned} \boldsymbol{x}^* &= \underset{\boldsymbol{x}}{\text{argmin}} \sum_{i=1}^{d} \left\{ \|\boldsymbol{P}_i(\boldsymbol{x} - \boldsymbol{x}_0)\|_2^2 + \lambda \max\left(g_j(\boldsymbol{P}_i\boldsymbol{x}) - g_t(\boldsymbol{P}_i\boldsymbol{x}), 0 \right) \right\} \\ &= \underset{\boldsymbol{x}}{\text{argmin}} \, \|\boldsymbol{x} - \boldsymbol{x}_0\|_2^2 + \lambda \sum_{i=1}^{L} \max(g_j(\boldsymbol{P}_i\boldsymbol{x}) - g_t(\boldsymbol{P}_i\boldsymbol{x}), 0) \end{aligned} \tag{7}$$

The difference between this equation and (5) is the presence of the matrices $\boldsymbol{P}_i$. The objective function says $\sum_{i=1}^{d} \|\boldsymbol{P}_i(\boldsymbol{x} - \boldsymbol{x}_0)\|_2^2$, plus the regularization $\sum_{i=1}^{d} \lambda \max\left(g_j(\boldsymbol{P}_i\boldsymbol{x}) - g_t(\boldsymbol{P}_i\boldsymbol{x}), 0 \right)$. This asks us to sum over **all** of the patches; we are now attacking the whole image at once, instead of attacking independent patches. More specifically, we simultaneously minimize the l_2 distance term for the whole image and the regularization term for every patch, thus we keep the perturbed image looking as "similar" to the original image as allowed, while making every patch in the perturbed image to be as "close" to the target class as possible. Note that this formulation resolves the issue raised in Figure 1, as we are taking into consideration the "interference" among the overlapping patches now.

Tasks (20 points):

(a) Calculate the gradient of the objective function in the second line of (7) with respect to $\boldsymbol{x}$.

(b) Implement the CW attack on the overlapping patches classifier. For stopping criteria, use maximum iteration 300 and change in iterates 0.01. If your implementation followed the outline in the previous part, much of your code should be reusable.

(c) Obtain the plots and data in the same way you did in exercise 2(c), except this time, plot the perturbed image iterates per 5, 5, 2 iterations for $\lambda = 0.5, 1, 5$ respectively. Again, comment on your plots and data. Moreover, how does the quality of the attack compare to that of the last exercise in general? Explain.

Exercise 4: Defending the Attack

Now that you have successfully attacked the classifier, we ask you to defend against the attack. Here are the tasks:

Tasks: (50 points)

- Develop a defense mechanism so that you can mitigate the attack described in Exercise 3.
- Analyze the limitation of your defense mechanism.
- Report your findings.

This is an open-ended question. Treat it like a research problem. We will grade it as if we were reviewers of ICML / NIPS. Here are the grading criteria.

- **Method Description**: what is your method? To answer this question you need to clearly describe the method and demonstrate your understanding of it. We are expecting to see equations and drawings to illustrate the idea. Do not just cite a paper and say that you are using that particular method. As reviewers of your method, we will challenge your choice of the method, fitness of the method to the problem, and the clarity of your description. More points will be given to *creative* solutions.
- **Comparisons**: comparison with other possible methods. We ask that you compare your proposed method with at least *two* baseline methods. You need to define the baselines and argue why they are the baselines. When reporting your results, we ask you to provide *quantitative arguments*. That is, whenever your wish to make a claim about your defense method, support it with tables and/or plots as thoroughly as you can. For instance, one should expect that as an attack's strength increases, the classifier's accuracy should decrease, but to support this claim, one should illustrate the relationship between classifier accuracy and attack strength with *tables* and *plots*. Finally, in your plots, when it makes sense, try to overlay your method with other methods for easier visual comparison. More points will be given to high quality experiments.
- **Ablation study**: if your method consists of multiple steps, or if you method relies on several ideas, we ask you to provide a comprehensive ablation study. That is, keep everything unchanged but substitute one component by something else. See what will happen to the final accuracy. Again, we want tables and curves, not hand-waving arguments.
- **Limitations**: limitations of your defense. We have discussed the concept of obfuscated gradient (or gradient masking) in the lecture. Is your method able to withstand this attack? If yes, why? If no, why? Provide concrete reasons, through experiments and/or proofs. More points will be given to a thorough analysis of obfuscated gradients.
- **Comments**: discuss your insights obtained from this project. Some example questions to reflect on: was the attack easy/difficult to defend against? What might be the major contributing causes of this easiness/difficulty? Besides the defenses you implemented, what are some defense ideas that you believe might work, but did not report on in the previous sections? Can some of the insights gained in this project be extended to adversarial defense for more complicated machine learning models, such as the nonlinear neural-network-based classifiers? What about those that probably cannot be? Why?
- **Writing**: clear writing means that your report is organized, and your ideas are *easy to follow*.

Summary of Adversarial Attacks

Table 1 below summarizes the adversarial attacks in the **2-class** case we studied in lecture.

Attack Name	**Optimization Problem**	**Solution/Iterate**
Min l_2 attack, linear	$\underset{\boldsymbol{x}}{\text{argmin}} \|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2$ subject to $\boldsymbol{w}^T\boldsymbol{x}+w_0=0$ (8)	$\boldsymbol{x}^* = \boldsymbol{x}_0 - \Big(\frac{\boldsymbol{w}^T\boldsymbol{x}_0+w_0}{\|\boldsymbol{w}\|_2^2}\Big)\boldsymbol{w}$ (9)
Min l_∞ attack, linear	$\underset{\boldsymbol{x}}{\text{argmin}} \|\boldsymbol{x}-\boldsymbol{x}_0\|_\infty$ subject to $\boldsymbol{w}^T\boldsymbol{x}+w_0=0$ (10)	$\boldsymbol{x}^* = \boldsymbol{x}_0 - \Big(\frac{\boldsymbol{w}^T\boldsymbol{x}_0+w_0}{\|\boldsymbol{w}\|_1}\Big)\text{sign}(\boldsymbol{w})$ (11)
DeepFool Attack, general	$\underset{\boldsymbol{x}}{\text{argmin}} \|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2$ subject to $g(\boldsymbol{x})=0$ (12)	$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \Big(\frac{g(\boldsymbol{x}^{(k)})}{\|\nabla_{\boldsymbol{x}} g(\boldsymbol{x}^{(k)})\|_2^2}\Big)\nabla_{\boldsymbol{x}} g(\boldsymbol{x}^{(k)})$ (13)
Max allowable l_∞ attack, linear	$\underset{\boldsymbol{x}}{\text{argmin}}\, \boldsymbol{w}^T\boldsymbol{x}+w_0$ subject to $\|\boldsymbol{x}-\boldsymbol{x}_0\|_\infty < \eta$ (14)	$\boldsymbol{x}^* = \boldsymbol{x}_0 - \eta\,\text{sign}(\boldsymbol{w})$ (15)
Fast Gradient Sign Method (FGSM), general, l_∞	$\max_{\boldsymbol{r}} \nabla_{\boldsymbol{x}} J(\boldsymbol{x}_0;\boldsymbol{w})^T\boldsymbol{r} + J(\boldsymbol{x}_0;\boldsymbol{w})$ subject to $\|\boldsymbol{r}\|_\infty < \eta$ (16)	$\boldsymbol{r} = -\eta\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_0;\boldsymbol{w}))$ $\boldsymbol{x}^* = \boldsymbol{x}_0 - \boldsymbol{r}$ (17)
Fast Gradient Sign Method (FGSM), general, l_2	$\max_{\boldsymbol{r}} \nabla_{\boldsymbol{x}} J(\boldsymbol{x}_0;\boldsymbol{w})^T\boldsymbol{r} + J(\boldsymbol{x}_0;\boldsymbol{w})$ subject to $\|\boldsymbol{r}\|_2 < \eta$ (18)	$\boldsymbol{r} = -\eta\,\frac{\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_0;\boldsymbol{w})}{\|\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_0;\boldsymbol{w})\|_2}$ $\boldsymbol{x}^* = \boldsymbol{x}_0 - \boldsymbol{r}$ (19)
Iterative Fast Gradient Sign Method (I-FGSM), general, l_∞	$\boldsymbol{x}^{(k+1)} = \underset{0\le\boldsymbol{x}\le 1}{\text{argmax}} \nabla_{\boldsymbol{x}} J(\boldsymbol{x}^{(k)};\boldsymbol{w})^T\boldsymbol{x}$ subject to $\|\boldsymbol{x}-\boldsymbol{x}_0\|_\infty < \eta$ (20)	$\boldsymbol{x}^{(k+1)} = \mathcal{P}_{[0,1]}\{\boldsymbol{x}_0 + \eta\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{x}^{(k)};\boldsymbol{w}))\}$ (21)
Regul'n-based Attack, linear	$\underset{\boldsymbol{x}}{\text{argmin}} \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2 + \lambda(\boldsymbol{w}^T\boldsymbol{x}+w_0)$ (22)	$\boldsymbol{x}^* = \boldsymbol{x}_0 - \lambda\boldsymbol{w}$ (23)
Carlini-Wagner (CW) Attack	$\underset{\boldsymbol{x}}{\text{argmin}}\, \varphi(\boldsymbol{x})$, where $\varphi(\boldsymbol{x}) = \|\boldsymbol{x}-\boldsymbol{x}_0\|_2^2 + \lambda\,\zeta(g_j(\boldsymbol{x}) - g_t(\boldsymbol{x}))$, $\zeta(y) = \max(y,0)$, and $j \neq t$ (24)	$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \alpha_k \nabla_{\boldsymbol{x}}\varphi(\boldsymbol{x}^{(k)})$ $\nabla_{\boldsymbol{x}}\varphi(\boldsymbol{x}) = 2(\boldsymbol{x}-\boldsymbol{x}_0) + \lambda\mathbb{I}\{g_j(\boldsymbol{x}) - g_t(\boldsymbol{x}) > 0\}\times (\nabla_{\boldsymbol{x}}(g_j(\boldsymbol{x}) - g_t(\boldsymbol{x})))$ (25)

Table 1: Adversarial Attacks on 2-Class Classifier