EECS 127/227AT Optimization Models in Engineering UC Berkeley Spring 2024

Provably Robust Deep Classifiers

1 Introduction

In this project, we will be exploring how to make neural net classifiers robust by framing them as a non-convex optimization problem, then using the techniques we learned in this class to prove their robustness. In particular, we will follow a technique developed by Wong and Kolter [1].

2 Overview of Relevant Literature

In the first section of this project, you will read several research papers which are relevant to the topic of this project, and summarize and synthesize them into a related work section. The aim is to gain a better understanding of the topics discussed in this project and to get insights into the state-of-the-art development in our understanding of adversarial machine learning.

In the related works section, you will summarize the main results and findings for at least three papers. It is especially useful and interesting to write about any common threads you find across multiple papers. We will assign one below, along with some questions you may think about while reading. The remaining paper(s) you choose to read can be found via Google or other sources.[1]

The papers we assign are the following:

1. "Towards Evaluating the Robustness of Neural Networks" by Carlini and Wagner [2]. While writing your summary, please mention the following points.
   - Describe the technique of defensive distillation, focusing on the actual algorithm and the intuition for why it works.
   - Describe the L-BFGS algorithm and why it works.
   - Describe one of the attack algorithms proposed by Carlini and Wagner, for either ℓ^0, ℓ^2, or ℓ^∞ distances.
2. "Provable Defenses Against Adversarial Examples via the Convex Outer Adversarial Polytope" by Wong and Kolter [1]. While writing your summary, please mention the following points.
   (a) Describe the adversarial polytope. What is it, and how is it relaxed to make a convex adversarial polytope?
   (b) Summarize the primal optimization problem used to generate adversarial examples.
   (c) Describe the general approach which uses the dual of the above optimization problem to obtain a robust neural network.
   (d) What is the relationship between Wong and Kolter's paper [1] and Carlini and Wagner's original paper [2]?
3. The third (and beyond) papers you choose to read yourself must relate to the topic of adversarial machine learning, but the exact paper(s) are your choice. For any additional paper(s), please add a summary of the findings of the paper, and describe how the paper relates to [2] and [1]. What is the novel contribution of the new paper you have read? Why is it important/relevant to the field? What is an open question that remains after reading the paper?

[1] Given a research paper written recently, one way to see papers related to it is to go to Google Scholar and type in the paper title, then look at all papers which cite the paper you started with (via "Cited by `$NUM`"); these tend to continue the same research threads or demonstrate applications. Another way is to look up the papers cited by the papers you have already read, focusing on those which have relevant titles. Each paper collects a list of sources cited near the end of the paper.

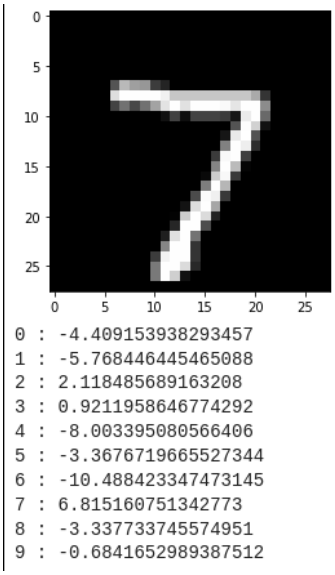

Figure 1: Example MNIST input and classifier output $\vec{y}_{\text{pred}}$. In this case, the classifier correctly predicts that the image is of the digit seven.

3 Problems

3.1 Deep Neural Network Classifiers

Formally, we define a set of parameters Θ, and a family of classifiers $\mathcal{F} := \{f_\theta : \theta \in \Theta\}$, such that each classifier $f_\theta \in \mathcal{F}$ is a function $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$, where n is the dimension of the input and m is the number of classes. We wish to find f_θ such that, given an input $\vec{x} \in \mathbb{R}^n$ and corresponding true label $\vec{y}_{\text{true}} \in \mathbb{R}^m$, the predicted label $\vec{y}_{\text{pred}} \doteq f_\theta(\vec{x}) \in \mathbb{R}^m$ is "close" to the true label $\vec{y}_{\text{true}}$ in some way which is characterized by a loss function, which we will shortly make precise. We say that f_θ is correct on $(\vec{x}, \vec{y}_{\text{true}})$ if and only if

$$\underset{i \in \{1,\ldots,n\}}{\operatorname{argmax}} (\vec{y}_{\text{pred}})_i = \underset{i \in \{1,\ldots,n\}}{\operatorname{argmax}} (\vec{y}_{\text{true}})_i \tag{1}$$

For concreteness, we consider the MNIST classification task. The inputs $\vec{x} \in \mathbb{R}^n$ are vectorized representation of the handwritten digit images (n is usually the number of pixels in the image); the labels $\vec{y}_{\text{pred}}, \vec{y}_{\text{true}} \in \mathbb{R}^{10}$ are "one-hot encodings" of the class. In particular, $\vec{y}_{\text{true}} = \vec{e}_{i_{\text{true}}}$ where i_{true} is the index corresponding to the correct digit (e.g. $i = k$ if $k - 1$ is the correct digit, because digits are 0–9 and indices are 1–10). See Figure 1. The optimal classifier minimizes the empirical risk function

$$\min_{\theta \in \Theta} \sum_{(\vec{x}, \vec{y}_{\text{true}}) \in \mathcal{D}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}}) \tag{2}$$

where $L : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a loss function that for each (data, label) tuple $(\vec{x}, \vec{y}_{\text{true}}) \in \mathcal{D}$ compares the model's prediction $f_\theta(\vec{x})$ to the true label $\vec{y}_{\text{true}}$. An example loss function L could be the zero-one loss

$$L(\vec{y}_{\text{pred}}, \vec{y}_{\text{true}}) \doteq \begin{cases} 1 & \operatorname{argmax}_{i \in \{1,\ldots,n\}} (\vec{y}_{\text{pred}})_i \neq \operatorname{argmax}_{i \in \{1,\ldots,n\}} (\vec{y}_{\text{true}})_i \\ 0 & \operatorname{argmax}_{i \in \{1,\ldots,n\}} (\vec{y}_{\text{pred}})_i = \operatorname{argmax}_{i \in \{1,\ldots,n\}} (\vec{y}_{\text{true}})_i \end{cases} \tag{3}$$

Then, the objective function in (2) simply counts the number of inputs $\vec{x} \in \mathcal{D}$ where the classifier's most confident class is not correct. Unfortunately, this function does not have useful gradients, so it is not used for training classifiers in practice. For this project, we will be considering a three-layer feedforward neural network with ReLU nonlinearity;

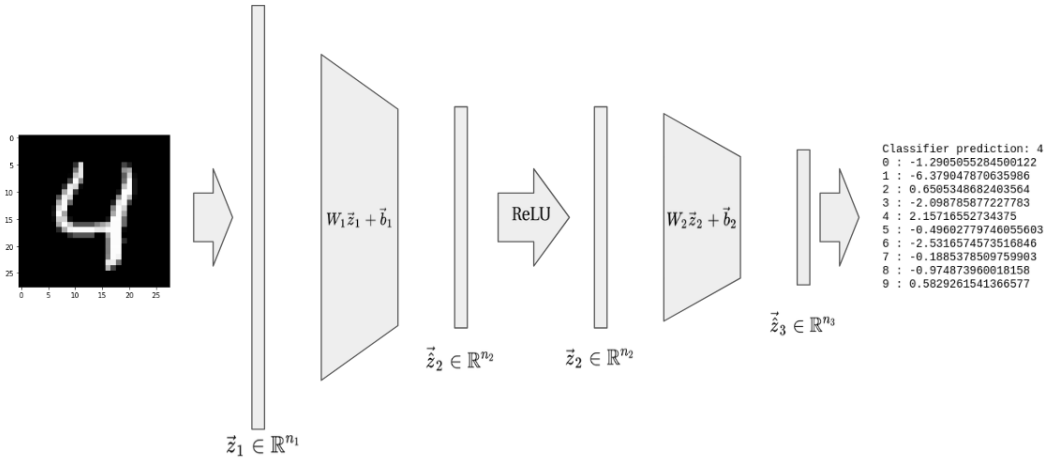

Figure 2: The deep neural network f_θ.

see Figure 2. That is, the network $f_\theta : \mathbb{R}^{n_1} \to \mathbb{R}^{n_3}$ consists of the layers

$$\vec{z}_1 \in \mathbb{R}^{n_1}, \vec{\tilde{z}}_2 \in \mathbb{R}^{n_2}, \vec{z}_2 \in \mathbb{R}^{n_2}, \vec{\tilde{z}}_3 \in \mathbb{R}^{n_3}, \tag{4}$$

where $\vec{z}_1 = \vec{x}$ is the input for the network and $\vec{z}_3$ is the output of f_θ, and the parameters

$$W_1 \in \mathbb{R}^{n_2 \times n_1}, W_2 \in \mathbb{R}^{n_3 \times n_2}, \vec{b}_1 \in \mathbb{R}^{n_2}, \vec{b}_2 \in \mathbb{R}^{n_3}, \tag{5}$$

which make up the affine transforms between layers. Explicitly, we define

$$\begin{aligned} \vec{z}_1 &\doteq \vec{x} \\ \vec{\tilde{z}}_2 &\doteq W_1\vec{z}_1 + \vec{b}_1 \\ \vec{z}_2 &\doteq \mathrm{ReLU}(\vec{\tilde{z}}_2) \\ \vec{\tilde{z}}_3 &\doteq W_2\vec{z}_2 + \vec{b}_2 \\ f_\theta(\vec{x}) &\doteq \vec{\tilde{z}}_3. \end{aligned} \tag{6}$$

ReLU (short for Rectified Linear Unit) is defined as

$$\mathrm{ReLU}(\vec{z}) \doteq \max\{\vec{z}, \vec{0}\} \tag{7}$$

where the maximum is taken elementwise.

Note that without the ReLU nonlinearity, the classifier f_θ would simply be a linear function of its input.

In the optimization problem (2), we define the parameter as

$$\theta \doteq (W_1, W_2, \vec{b}_1, \vec{b}_2) \tag{8}$$

Hence, $\Theta \doteq \mathbb{R}^{n_2 \times n_1} \times \mathbb{R}^{n_3 \times n_2} \times \mathbb{R}^{n_2} \times \mathbb{R}^{n_3}$.

3.1.1 Notation

Throughout this document we will use the following notation:

- For a vector $\vec{v}$, the scalar v_j is the jth element of $\vec{v}$.
- For a matrix A, the vector A_j is the jth column of A.

For example, $(W_2)_j$ is the jth column of the matrix W_2.

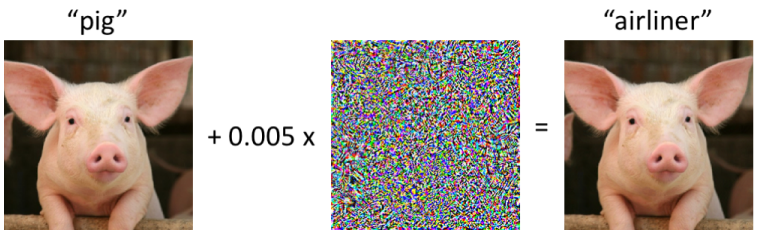

Figure 3: In this example, the original image is of a pig. However, an adversary is able to apply an imperceptible amount of noise and fool the classifier into believing the image is of an airliner. Image credit `gradientscience.com/intro_adversarial`.

3.2 Finding Adversarial Examples

In recent years, it has been found that classifier trained using standard methods are not very robust. That is, although a classifier may perform well when the inputs are sampled from real-world processes (e.g., images of real-world handwritten images), if they are artificially perturbed by even a small amount that is imperceptible to humans, they can easily be misled. For example, if we have trained a classifier for detecting handwritten digits, an adversary might be able to take an image of a four and slightly change some pixel values such that our classifier now thinks the image is of a two. See Figure 3 for another example of this.

More formally, the goal of an adversary is to find an example $\vec{x}'$ that is close to a real input $\vec{x}$, but is classified incorrectly. (The classifier is assumed to have correct output on $\vec{x}$.) That is, the adversary wishes to solve the following optimization problem:

$$\begin{aligned} \max_{\vec{x}'} \quad & L(f_\theta(\vec{x}'), \vec{y}_{\text{true}}) \\ \text{s.t.} \quad & \|\vec{x} - \vec{x}'\|_\infty \leq \epsilon \end{aligned} \tag{9}$$

where $\vec{y}_{\text{true}}$ is the vector that is one on the true label for $\vec{x}'$, and zero elsewhere. The constraint $\|\vec{x} - \vec{x}'\|_\infty \leq \epsilon$ says that the adversarial input $\vec{x}'$ must be close to $\vec{x}$; each vector element, which corresponds to a pixel value, must be no more than ϵ away from the original.

3.2.1 Fast Gradient Signed Method

One common way to approximate the solution to (9) is the Fast Gradient Signed Method (FGSM):

$$\vec{x}_{\text{FGSM}} \doteq \vec{x} + \epsilon \operatorname{sgn}(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})). \tag{10}$$

Note that this is very similar to gradient ascent of the loss w/r/t the input, except we only take a single step, and we use the sign of the gradient instead of the gradient itself.

1. Interpretation of FGSM

(a) Show that the FGSM perturbation (10) is the solution to a first-order approximation of the adversarial optimization problem, i.e.

$$\begin{aligned} \vec{x}_{\text{FGSM}} = \quad \operatorname*{argmax}_{\vec{x}'} \quad & L(f_\theta(\vec{x}), \vec{y}_{\text{true}}) + (\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}}))^\top \vec{x}' \\ \text{s.t.} \quad & \|\vec{x} - \vec{x}'\|_\infty \leq \epsilon. \end{aligned} \tag{11}$$

HINT: Consider setting $\vec{x}' = \vec{x} + \epsilon\vec{v}$ and expressing the optimization in terms of $\vec{v}$. Also, recall that the ℓ_1 and ℓ_∞ norms are dual.

(b) What would the solution to (11) be if we were optimizing over a ball instead of a box constraint (i.e. instead of $\|\vec{x} - \vec{x}'\|_\infty \leq \epsilon$, we had $\|\vec{x} - \vec{x}'\|_2 \leq \epsilon$)? Explain, in words, why problem (11) under the ℓ_2 and ℓ_∞ result in different solutions.

3.3 Convex relaxation of the adversarial optimization

Suppose now we are considering a specific adversary who wishes to fool the classifier into classifying a particular image $\vec{x}$ as an incorrect class i_{targ} instead of the true class i_{true} by perturbing x slightly. (In the MNIST example, maybe the adversary wants to fool the classifier into thinking an image of a four is really an image of a two.)

Using the same network f_θ as defined in (6), we rewrite the adversarial problem (9) as

$$\begin{aligned} \min_{\vec{z}} \quad & \vec{c}^\top \vec{\hat{z}}_3 \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \\ & \vec{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \text{ReLU}(\vec{\hat{z}}_2) \\ & \vec{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{aligned} \tag{12}$$

where for the objective we define

$$\vec{c} = \vec{y}_{\text{true}} - \vec{y}_{\text{targ}}. \tag{13}$$

Recall that $\vec{y}_{\text{true}}$ is defined as a vector that is one at the true class i_{true} and zero elsewhere. Similarly, $\vec{y}_{\text{targ}}$ is a zero-one vector that is one only at the adversary's target class i_{targ}. (This can be any incorrect class.) Thus, the objective

$$\vec{c}^\top \vec{\hat{z}}_3 = \vec{\hat{z}}_{3i_{\text{true}}} - \vec{\hat{z}}_{3i_{\text{targ}}} \tag{14}$$

is the difference between the classifier's score assigned on the true class and the target class—if the adversary can force this objective to be negative, then $\vec{\hat{z}}_{3i_{\text{targ}}} > \vec{\hat{z}}_{3i_{\text{true}}}$, so the classifier will no longer assign the input to the true class. In addition, if $\vec{\hat{z}}_{3i_{\text{targ}}} > \vec{\hat{z}}_{3i}$ for all other $i \neq i_{\text{targ}}$, the classifier is fooled into assigning the input to the target class.

Also, notice the slight abuse of notation in (12); when we say $\min_{\vec{z}}$ we really mean $\min_{\vec{z}_1, \hat{\vec{z}}_2, \vec{z}_2, \hat{\vec{z}}_3}$. We will keep this convention throughout this document in order to avoid clutter.

Note that because of the ReLU nonlinearity, the problem (12) is sometimes non-convex and thus difficult to find a solution to. However, if we knew upper and lower bounds u_j and l_j respectively for each $\hat{z}_{2j}$ (the jth element of $\vec{\hat{z}}_2$), we can instead relax the constraint $\vec{z}_2 = \text{ReLU}(\vec{\hat{z}}_2)$ to $\forall j \in \{1, \ldots, n_2\} : (z_{2j}, \hat{z}_{2j}) \in \mathcal{Z}_j$, where $\mathcal{Z}_j$ is the convex hull of the original constraint, i.e.

$$\mathcal{Z}_j \doteq \text{conv}(\widehat{\mathcal{Z}}_j) = \text{conv}(\{(z_{2j}, \hat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} = \text{ReLU}(\hat{z}_{2j}) \text{ and } l_j \leq \hat{z}_{2j} \leq u_j\}). \tag{15}$$

To make this explicit, we consider three cases. If $l_j \leq u_j \leq 0$, then we know $\hat{z}_{2j} \leq 0$, so the ReLU constraint is equivalent to fixing $z_{2j} = 0$. Thus, $\widehat{\mathcal{Z}}_j$ is already convex, and we can define

$$\mathcal{Z}_j = \widehat{\mathcal{Z}}_j = \{(z_{2j}, \hat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} = 0\}. \tag{16}$$

Notice we ignore the $l_j \leq \hat{z}_{2j} \leq u_j$ constraint in the definition of $\widehat{\mathcal{Z}}_j$ for simplicity. This does not affect the dual program, since we assume by definition of l_j and u_j that $l_j \leq \hat{z}_{2j} \leq u_j$ is already enforced by the other constraints. Similarly, if $0 \leq l_j \leq u_j$, we know $\hat{z}_{2j} \geq 0$ and thus $z_{2j} = \hat{z}_{2j}$, so

$$\mathcal{Z}_j = \widehat{\mathcal{Z}}_j = \{(z_{2j}, \hat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} = \hat{z}_{2j}\}. \tag{17}$$

In the third case, $\widehat{\mathcal{Z}}_j$ is no longer convex. Examining this set visually, it is clear that its convex hull is a triangle, given by

$$\mathcal{Z}_j = \{(z_{2j}, \hat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} \geq 0 \text{ and } z_{2j} \geq \hat{z}_{2j} \text{ and } -u_j \hat{z}_{2j} + (u_j - l_j) z_{2j} \leq -u_j l_j\}. \tag{18}$$

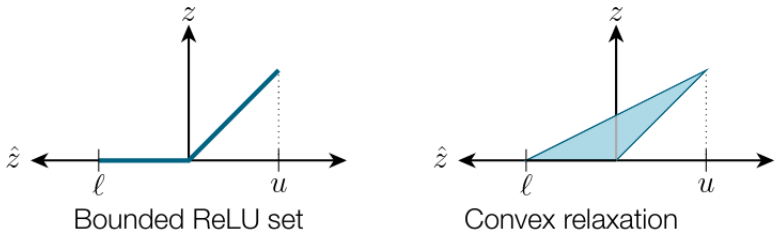

Figure 4: The convex relaxation of ReLU when $l_j \leq 0 \leq u_j$. Image credit [Wong & Kolter, 2018].

Note that the inequality

$$-u_j \widehat{z}_{2j} + (u_j - l_j) z_{2j} \leq -u_j l_j \tag{19}$$

defines the upper boundary of the triangle, i.e., the line going through $(l_j, 0)$ and (u_j, u_j). See Figure 4.

Our relaxation of the problem in (12) is thus

$$\begin{aligned} p^*(\vec{x}, \vec{c}) = \min_{\vec{z}} \quad & c^\top \vec{\widehat{z}}_3 \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \\ & \vec{\widehat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & (z_{2j}, \widehat{z}_{2j}) \in \mathcal{Z}_j \qquad \forall j \in \{1, \ldots, n_2\} \\ & \vec{\widehat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{aligned} \tag{20}$$

Note that since the feasible set of the relaxation is a superset of the original, the relaxed optimum is a lower bound for the original optimum.

2. Primal Modification for Guaranteeing Target Classification

How might we modify the problem in (12) if we wanted the objective being negative to mean that the classifier **must** output $\vec{y}_{\text{targ}}$? Note that this is different from fooling the classifier into **not** outputing $\vec{y}_{\text{true}}$.

3.4 Fenchel Conjugates

We take a slight detour to define what will be a useful notion to us: Fenchel conjugates. For any function $f : \mathbb{R}^n \to \mathbb{R}$, we define a Fenchel conjugate $f^* : \mathbb{R}^n \to \mathbb{R}$ by

$$f^*(\vec{y}) = \sup_{\vec{x}} \{\vec{y}^\top \vec{x} - f(\vec{x}) \mid \vec{x} \in \mathbb{R}^n\}. \tag{21}$$

This allows us to define f^* as a pointwise supremum of affine functions $\vec{y} \mapsto \vec{y}^\top \vec{x} - f(\vec{x})$, which ensures that $f^*(\vec{y})$ is convex in $\vec{y}$. In particular, the Fenchel conjugate is useful when formulating dual problems.

3. Practice with Fenchel Conjugates

Before proceeding to the next part, we will get some practice with taking Fenchel conjugates.

(a) Suppose $f : \mathbb{R} \to \mathbb{R}$, and $f(x) = |x|$. Find $f^*(y)$.
HINT: Consider case-work on the cases $\{y < -1, y = -1, -1 < y < 1, y = 1, y > 1\}$.

(b) Now, suppose $f : \mathbb{R}^n \to \mathbb{R}$ and $f(\vec{x}) = \|\vec{x}\|_1$. Find $f^*(\vec{y})$.
HINT: No need to reprove everything! Write the Fenchel conjugate of the ℓ_1 *norm in terms of the Fenchel conjugate of the absolute value function, and apply the result proved in the previous part.*

3.5 Using Lagrangian Duality

The relaxed optimization problem (20) is convex, so it can be solved efficiently with respect to the size of the layers $\vec{z}_i$ using standard tools. However, for many classification problems the input and intermediate layers can be large, so this may still be prohibitively slow. In this section, we will show that solving the dual problem instead is much easier.

For any set S, define the characteristic function $\mathbf{1}_S$ as

$$\mathbf{1}_S(x) \doteq \begin{cases} 0 & x \in S \\ +\infty & \text{otherwise.} \end{cases} \tag{22}$$

We also define the ℓ_∞ ball

$$B_\epsilon(\vec{x}) \doteq \{\vec{v} \in \mathbb{R}^n \mid \|\vec{v} - \vec{x}\|_\infty \leq \epsilon\}. \tag{23}$$

4. Dualizing the Classifier

(a) Show that we can re-express the convex relaxation (20) as

$$\begin{aligned} p^*(\vec{x}, \vec{c}) = \min_{\vec{z}} \quad & c^\top \vec{\widehat{z}}_3 + \mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) + \sum_{j=1}^{n_2} \mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \widehat{z}_{2j}) \\ \text{s.t.} \quad & \vec{\widehat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{\widehat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{aligned} \tag{24}$$

(b) Let $\vec{\nu}_3$ be the dual variable for the constraint $\vec{\widehat{z}}_3 = W_2\vec{z}_2 + \vec{b}_2$, and $\vec{\nu}_2$ be the dual variable for the constraint $\vec{\widehat{z}}_2 = W_1\vec{z}_1 + \vec{b}_1$. Show that the Lagrangian for the optimization problem (24) is

$$\begin{aligned} \mathcal{L}(\vec{z}, \vec{\nu}) = \vec{c}^\top \vec{\widehat{z}}_3 + \vec{\nu}_3^\top \vec{\widehat{z}}_3 + \mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) - \vec{\nu}_2^\top W_1 \vec{z}_1 \\ + \left(\sum_{j=1}^{n_2} \mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \widehat{z}_{2j}) - \vec{\nu}_3^\top W_2 \vec{z}_2 + \vec{\nu}_2^\top \vec{\widehat{z}}_2 \right) - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i. \end{aligned} \tag{25}$$

(c) Show that

$$\begin{aligned} g(\vec{\nu}_2, \vec{\nu}_3) &\doteq \min_{\vec{z}} \mathcal{L}(\vec{z}, \vec{\nu}) \\ &= \min_{\vec{\widehat{z}}_3} ((\vec{c} + \vec{\nu}_3)^\top \vec{\widehat{z}}_3) + \min_{\vec{z}_1} (\mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) - \vec{\nu}_2^\top W_1 \vec{z}_1) \\ &\quad + \left(\sum_{j=1}^{n_2} \min_{z_{2j}, \widehat{z}_{2j}} (\mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \widehat{z}_{2j}) - \vec{\nu}_3^\top (W_2)_j z_{2j} + \nu_{2j} \widehat{z}_{2j}) \right) - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i \end{aligned} \tag{26}$$

(d) Conclude that the Lagrangian dual to (24) is

$$\begin{aligned} d^*(\vec{x}, \vec{c}) = \max_{\vec{\nu}} \quad & -\mathbf{1}^*_{B_\epsilon(\vec{x})}(W_1^\top \vec{\nu}_2) + \sum_{j=1}^{n_2} -\mathbf{1}^*_{\mathcal{Z}_j}(\vec{\nu}_3^\top (W_2)_j, -\nu_{2j}) - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i \\ \text{s.t.} \quad & \vec{\nu}_3 = -\vec{c}. \end{aligned} \tag{27}$$

Defining $\vec{\widehat{\nu}}_i = W_i^\top \vec{\nu}_{i+1}$, this becomes

$$\begin{aligned} d^*(\vec{x}, \vec{c}) = \max_{\vec{\nu}} \quad & -\mathbf{1}^*_{B_\epsilon(\vec{x})}(\vec{\widehat{\nu}}_1) + \sum_{j=1}^{n_2} -\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}_{2j}, -\nu_{2j}) - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i \\ \text{s.t.} \quad & \vec{\nu}_3 = -\vec{c} \\ & \vec{\widehat{\nu}}_2 = W_2^\top \vec{\nu}_3 \\ & \vec{\widehat{\nu}}_1 = W_1^\top \vec{\nu}_2. \end{aligned} \tag{28}$$

HINT: Each of the minimizations in (26) can be written as either a Fenchel conjugate or converted to a constraint. Also, see Section 5.1.6 of Boyd and Vandenberghe's textbook for more on how the Fenchel conjugate relates to the Lagrangian dual.

3.6 Finding the Fenchel Conjugates

In the previous section, we derived the dual optimization problem in terms of the Fenchel conjugates of the characteristic functions $\mathbf{1}_{\mathcal{Z}_j}$ and $\mathbf{1}_{B_\epsilon(\vec{x})}$. Now, it only remains to find expressions for these conjugates.

5. Show that $\mathbf{1}^*_{B_\epsilon(\vec{x})}(\vec{\nu}) = \vec{\nu}^\top \vec{x} + \epsilon \left\|\vec{\nu}\right\|_1$.

The conjugate for $\mathbf{1}_{\mathcal{Z}_j}$ is a bit more complex, since it involves some casework and clever manipulations. Therefore we give the expressions here, but if interested see the last few exercises.

There are three cases to consider for $\mathbf{1}_{\mathcal{Z}_j}$:

- When $l_j \leq u_j \leq 0$,

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) \leq \begin{cases} 0 & \nu = 0 \\ +\infty & \text{otherwise.} \end{cases} \tag{29}$$

- When $0 \leq l_j \leq u_j$,

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) \leq \begin{cases} 0 & \nu = \widehat{\nu} \\ +\infty & \text{otherwise.} \end{cases} \tag{30}$$

- When $l_j \leq 0 \leq u_j$,

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) \leq \begin{cases} \mathrm{ReLU}(-l_j\nu) & \nu = \dfrac{\widehat{\nu} u_j}{u_j - l_j} \\ +\infty & \text{otherwise.} \end{cases} \tag{31}$$

Since we are only interested in finding a lower bound to the primal objective, these inequalities suffice.

3.7 The Dual Network

We are now ready to write the final expression for the dual problem (28). For notational convenience, let us define the following index sets:

$$\begin{aligned} S &\doteq \{j \in \{1,\dots,n_2\} \mid l_j \le 0 \le u_j\} \\ S^- &\doteq \{j \in \{1,\dots,n_2\} \mid l_j \le u_j \le 0\} \\ S^+ &\doteq \{j \in \{1,\dots,n_2\} \mid 0 \le l_j \le u_j\}. \end{aligned} \tag{32}$$

6. Expressing the Dual Network

(a) Deduce that the dual problem (28) can be expressed as

$$\begin{aligned} d^*(\vec{x},\vec{c}) = \max_{\vec{\nu}} \quad & -\vec{\widehat{\nu}}_1^\top \vec{x} - \epsilon\|\vec{\widehat{\nu}}_1\|_1 - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i + \sum_{j\in S} l_j \,\mathrm{ReLU}(\nu_{2j}) \\ \text{s.t.} \quad & \vec{\nu}_3 = -\vec{c} \\ & \vec{\widehat{\nu}}_2 = W_2^\top \vec{\nu}_3 \\ & \nu_{2j} = 0 && \forall j \in S^- \\ & \nu_{2j} = \widehat{\nu}_{2j} && \forall j \in S^+ \\ & \nu_{2j} = \frac{u_j}{u_j - l_j}\widehat{\nu}_{2j} && \forall j \in S \\ & \vec{\widehat{\nu}}_1 = W_1^\top \vec{\nu}_2 \end{aligned} \tag{33}$$

(b) Explain why the dual problem is convex and the unrelaxed primal problem (12) isn't, despite both having a ReLU non-linearity as part of the objective function and constraints respectively.

Observe now that $\vec{\nu}_3 = -\vec{c}$, and, given each $\vec{\nu}_{i+1}$, we can determine what values $\vec{\widehat{\nu}}_i$ and $\vec{\nu}_i$ are forced by the constraints to take on. That is, it is easy to find the optimal dual value $d^*(\vec{x},\vec{c})$, since there is only one feasible value for each of the dual variables $\vec{\nu}_3, \vec{\widehat{\nu}}_2, \vec{\nu}_2, \vec{\widehat{\nu}}_1$. We simply calculate what values each of these variables must take on, then compute the objective function. Once we have $d^*(\vec{x},\vec{c})$, we know by weak duality that this is a lower bound to $p^*(\vec{x},\vec{c})$, which is the relaxed primal problem (20) and again lower-bounds the primal adversarial problem (12) we are interested in. In fact, if we examine the constraints more carefully, we see that they exactly describe a backwards pass through the original network f_θ, albeit with a modified nonlinearity. In more detail, each constraint

$$\vec{\widehat{\nu}}_i = W_i^\top \vec{\nu}_{i+1} \tag{34}$$

can be thought of as the reverse of the affine layer

$$\vec{\widehat{z}}_{i+1} = W_i \vec{z}_i + \vec{b}_i \tag{35}$$

in the primal network, ignoring the bias term $\vec{b}_i$. Moreover, instead of the ReLU nonlinearity

$$\vec{z}_i = \mathrm{ReLU}(\vec{\widehat{z}}_i), \tag{36}$$

we have a different nonlinearity

$$\begin{aligned} \nu_{2j} &= 0 && \forall j \in S^- \\ \nu_{2j} &= \widehat{\nu}_{2j} && \forall j \in S^+ \\ \nu_{2j} &= \frac{u_j}{u_j - l_j}\widehat{\nu}_{2j} && \forall j \in S \end{aligned} \tag{37}$$

that depends on the bounds $\vec{l}$ and $\vec{u}$.

With this in mind, we define the backwards network $g_\theta(\vec{c})$ with layers $\vec{\nu}_3, \widehat{\vec{\nu}}_2, \vec{\nu}_2, \widehat{\vec{\nu}}_1$ as

$$\begin{aligned}
&\vec{\nu}_3 = -\vec{c} \\
&\widehat{\vec{\nu}}_2 = W_2^\top \vec{\nu}_3 \\
&\nu_{2j} = 0 && \forall j \in S^- \\
&\nu_{2j} = \widehat{\nu}_{2j} && \forall j \in S^+ \\
&\nu_{2j} = \frac{u_j}{u_j - l_j} \widehat{\nu}_{2j} && \forall j \in S \\
&\widehat{\vec{\nu}}_1 = W_1^\top \vec{\nu}_2
\end{aligned} \tag{38}$$

and define a loss

$$J_\epsilon(\vec{x}, g_\theta(\vec{c})) = -\widehat{\vec{\nu}}_1^\top \vec{x} - \epsilon \|\widehat{\vec{\nu}}_1\|_1 - \sum_{i=1}^{k-1} \vec{\nu}_{i+1}^\top \vec{b}_i + \sum_{j \in S} l_j \operatorname{ReLU}(\nu_{2j}) \tag{39}$$

where we think of $g_\theta(\vec{c})$ as returning all the layers $\widehat{\vec{\nu}}_i, \vec{\nu}_i$. Then, we can interpret the dual optimum $d^*(\vec{x}, \vec{c})$ as the loss incurred by the dual network g_θ on input $\vec{c}$, i.e.

$$d^*(\vec{x}, \vec{c}) = J_\epsilon(\vec{x}, g_\theta(\vec{c})). \tag{40}$$

3.8 Finding the bounds

Previously, we had written the dual assuming we knew the bounds l_j, u_j. Now it comes time to actually find these values.

7. Computing the ReLU Bounds

For any matrix W with rows $\vec{w}_1^\top, \ldots, \vec{w}_k^\top$, denote

$$\|W\|_{:1} \doteq [\|\vec{w}_1\|_1, \ldots, \|\vec{w}_k\|_1]^\top. \tag{41}$$

(Note that this is not a norm—the output is a vector, not a scalar.) Show that

$$\vec{u} = W_1\vec{x} + \vec{b}_1 + \epsilon \|W_1\|_{:1} \tag{42}$$

and

$$\vec{l} = W_1\vec{x} + \vec{b}_1 - \epsilon \|W_1\|_{:1} \tag{43}$$

are upper and lower bounds for $\vec{z}_2$, respectively.

Using this, whenever we need to compute $g_\theta(\vec{c})$, we can first obtain u and l using the formulas (42) and (43).

3.9 A certificate for robustness

Let us take a moment to recall what we have accomplished so far. We have found a way to compute $d^*(\vec{x}, \vec{c})$, which is the optimum for the dual program (28). By weak duality, this is a lower bound for $p^*(\vec{x}, \vec{c})$, which is the optimum for the relaxed primal problem (20). This in turn is a lower bound for

$$\begin{aligned} \min_{\vec{z}} \quad & \vec{c}^\top \vec{\hat{z}}_3 \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \\ & \vec{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \mathrm{ReLU}(\vec{\hat{z}}_2) \\ & \vec{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{aligned} \qquad \text{(12 again)}$$

which is the original adversarial problem. Recall that we choose

$$\vec{c} = \vec{y}_{\text{true}} - \vec{y}_{\text{targ}} \tag{44}$$

where $\vec{y}_{\text{true}}$ is one on the true class and zero elsewhere, and $\vec{y}_{\text{targ}}$ is one on the incorrect class i_{targ} and zero elsewhere. Then, the optimum for (12) is the worst-case difference between the classifier's confidence on the true class and class i_{targ}, assuming the input is fixed to an ϵ-ball around the original $\vec{x}$. Using the dual, we now have a guarantee on how badly the classifier will do in the worst case. As long as $d^*(\vec{x}, \vec{y}_{\text{true}} - \vec{y}_{\text{targ}})$ is positive, the classifier will have more confidence in i_{true} than in i_{targ}, for all ϵ-perturbations of the input $\vec{x}$.

More generally, we want to have a certificate that the classifier cannot be fooled into choosing any incorrect class $j \neq i_{\text{true}}$. This is simple to do using the tools we have developed; simply compute $d^*(\vec{x}, \vec{c}_j)$ for each $j \neq i_{\text{true}}$, where

$$\vec{c}_j = \vec{y}_{\text{true}} - \vec{e}_j \tag{45}$$

and $\vec{e}_j$ is one at index j and zero elsewhere.

By the same reasoning as above, if all of the $d^*(\vec{x}, \vec{c}_j)$ are positive, then the classifier is robust on input $\vec{x}$: there can be no perturbation $\vec{x}'$ with $\|\vec{x} - \vec{x}'\|_\infty < \epsilon$ such that the classifier is incorrect on $\vec{x}'$.

3.10 Training a robust classifier

Using the dual network (28), we are able to check the robustness of the primal network f_θ on any input x (see Exercise 8). In fact, we can do more than this—we can train a robust model by constructing a new loss in terms of J_ϵ and minimizing this w/r/t θ. We first assume some properties of the original loss L.

Definition 1

A multi-class loss function $L : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}$ is: *monotonic* if for all input $\vec{y}, \vec{y}'$ such that $y_i \leq y'_i$ for indices i corresponding to incorrect classes (i.e. $i \neq i_{\text{true}}$), and $y_{i_{\text{true}}} \geq y'_{i_{\text{true}}}$, we have

$$L(\vec{y}, \vec{y}_{\text{true}}) \leq L(\vec{y}', \vec{y}_{\text{true}}). \tag{46}$$

Definition 2

A multi-class loss function $L : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}$ is *translation-invariant* if for all $a \in \mathbb{R}$,

$$L(\vec{y}, \vec{y}_{\text{true}}) = L(\vec{y} - a\mathbf{1}, \vec{y}_{\text{true}}). \tag{47}$$

8. Show that cross-entropy loss, defined as

$$L(\vec{y}, \vec{y}_{\text{true}}) = -\sum_{i=1}^{m} (\vec{y}_{\text{true}})_i \log\left(\frac{e^{y_i}}{\sum_{j=1}^{m} e^{y_j}}\right) \tag{48}$$

is monotonic and translation-invariant. The cross-entropy loss is commonly used to train and/or measure the performance of classification models.

Now, we wish to train a robust classifier through the following optimization problem:

$$\min_\theta \sum_{x \in \mathcal{D}} \max_{\|x' - x\|_\infty \leq \epsilon} L(f_\theta(\vec{x}'), \vec{y}_{\text{true}}). \tag{49}$$

That is, we wish to find parameters that minimize the worst-case loss caused by an adversary limited to an ℓ_∞ ball around the original inputs $x \in \mathcal{D}$. Since this robust loss involves nested $\min$ and $\max$ terms, it is difficult to minimize it directly. Thus, we will instead minimize an upper bound.

Let us first extend our notation for the dual network g_θ (38) and the loss J_ϵ (39), such that the dual network g_θ takes matrices $C \in \mathbb{R}^{m \times d}$ as input. Then, if $\vec{c}_1, \ldots, \vec{c}_d$ are the columns of C, we define

$$\vec{J}_\epsilon(\vec{x}, g_\theta(C)) = [J_\epsilon(\vec{x}, g_\theta(\vec{c}_1)), \ldots, J_\epsilon(\vec{x}, g_\theta(\vec{c}_d))]^\top. \tag{50}$$

9. For monotonic and translation-invariant loss L, show that

$$\max_{\|x - x'\|_\infty \leq \epsilon} L(f_\theta(\vec{x}'), \vec{y}_{\text{true}}) \leq L(-\vec{J}_\epsilon(\vec{x}, g_\theta(\vec{y}_{\text{true}}\mathbf{1}^\top - I)), \vec{y}_{\text{true}}). \tag{51}$$

This shows that by solving the optimization problem

$$\min_\theta \sum_{x \in \mathcal{D}} L(-\vec{J}_\epsilon(\vec{x}, g_\theta(\vec{y}_{\text{true}}\mathbf{1}^\top - I)), \vec{y}_{\text{true}}), \tag{52}$$

which can be minimized using standard gradient descent, we are minimizing the worst-case loss due to some ϵ-perturbation of the original training input. Therefore, a model successfully trained using this loss would be much more robust to adversarial perturbations than a model trained using the original loss L.

3.11 The Fenchel Conjugate of $\mathbf{1}_{\mathcal{Z}_j}$

The next two exercises guide you through finding the Fenchel conjugate of $\mathbf{1}_{\mathcal{Z}_j}$, which was skipped in the previous section.

10. Show that when $l_j \leq u_j \leq 0$,

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) = \begin{cases} 0 & \nu = 0 \\ +\infty & \text{otherwise.} \end{cases} \tag{53}$$

On the other hand, show that when $0 \leq l_j \leq u_j$,

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) = \begin{cases} 0 & \nu = \widehat{\nu} \\ +\infty & \text{otherwise.} \end{cases} \tag{54}$$

In the remaining case, u_j and l_j straddle the origin, and $\mathcal{Z}_j$ is a non-degenerate triangle.

11. Show that when $l_j \leq 0 \leq u_j$,

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) \leq \begin{cases} \mathrm{ReLU}(-l_j\nu) & \nu = \dfrac{\widehat{\nu} u_j}{u_j - l_j} \\ +\infty & \text{otherwise.} \end{cases} \tag{55}$$

HINT: You may use the fact that the optimum of a linear program can always be attained at one of the vertices of the feasible polytope—in particular, the optimization problem for $\mathbf{1}^*_{\mathcal{Z}_j}$ attains the optimum at one of the three vertices of the triangle $\widehat{\mathcal{Z}}_j$. Deduce that the optimal $(y, \widehat{y})$ for the Fenchel conjugate maximization problem is either $(0, 0)$ or on the line

$$-u_j\widehat{y} + (u_j - l_j)y = -u_j l_j. \tag{56}$$

4 Extension

Now that you've seen some of the research landscape and worked through some of the resulting problems, the last part of the project is to complete an *extension* of the project material. An extension is a self-directed addition to the research elucidated in the related work and the problem set. You should include the extension material in your writeup after the problem set.

When selecting a topic for the extension, the following questions may be reasonable to ask:

- Was there a question that arose when reading related work or doing the project that seems like it might be interesting to investigate?
- Do the methods in the papers read for related work or the project itself extend to a particular interesting setting?
- Does a different method solve the same problems considered here? How can the methods be compared (on axes like efficiency, simplicity, etc)?

The extension would ideally answer at least one of these questions constructively, or some other question along the same lines. For example, in the first case, you would write down the question and try to answer it (via theory and/or experiments). In the second case, you would investigate the method in the paper applied to the setting of your interest. In the third case, you would apply the different method to the same paper, or compare it with some baseline methods.

Often, an extension idea will come from reading related work. Some broad ideas for possible extensions which heavily draw upon related work are:

- Try to replicate a piece of related work that's been done and that you have read about. Contrast this with the work done in the project.
- As we have done in this project, present in detail a simple (or complex if you want) case from a related paper that you read.

This is definitely not an extensive list. The overall guideline is that we must be able to understand what *you* have done from your work. It can help if you can crisply formulate a question that you are trying to answer. If you cannot, then it might not be a good extension.

Please note that *to successfully complete the extension, you do not need to have a breakthrough!* It is very difficult to come up with and answer a significant research question in a few weeks, so you should not feel discouraged if this is not possible for you. If you are ever feeling stuck, we encourage you to come to office hours to discuss your extension with us!

5 Deliverables

Your submission should contain:

1. A PDF of your final report. Your final report should be written in LaTeX. You are recommended to use the template that has been provided on the course website. You may choose not to typeset your equations, but in that case the document should include very high quality, cleanly handwritten scans of your work; we will not try to read illegible content. In particular, we prefer that you typeset your work, since it is an important skill to learn, and a nicely typeset project report (put on your website or CV) can be a strong showcase of the work you have done.

   Your report should include

   (a) An abstract, i.e., a paragraph length summary of what is in your document.

   (b) An introduction, which describes what the research problem studied in the project is and why it is important.

   (c) A literature review, whose guidelines were elaborated in Section 2.

   (d) Solutions to all guided portions of the project in Section 3, including relevant plots, figures, or code snippets that are needed to answer questions.

   (e) A detailed description of the work you did for your extension in Section 4.

   (f) A contributions section, i.e., a description of which members of your group did what work for the project.

2. Uploaded code, including Jupyter notebooks, for your work in the guided portion of the project (Section 3).

3. Uploaded code, including Jupyter notebooks, for your work in the project extension (Section 4).

All of these deliverables (project report PDF, code for the guided part of the project, and code for the extension) will have separate Gradescope assignments.

6 Rubric

To get any grade, you must submit a project report with:

- An abstract summarizing the report;
- An introduction section;
- A literature review section, which contains a literature review as detailed in Section 2;
- A results section, which contains the solutions for the guided portion of the project as detailed in Section 3;
- An extension section, which contains a summary of your extension as detailed in Section 4;
- A contribution section;

as well as upload your code (including Jupyter notebooks) for your work in the guided portion of the project as well as the extension (if applicable). Once these requirements are met, your grade is based on the quality of the report.

- To get a C, your project report must fulfill the following requirements:
  - Your introduction must describe the research problem clearly and in detail;
  - Your literature review must summarize both provided papers and at least one more, and contain mostly-correct answers to the provided questions in Section 2;
  - Your results for the guided portion of the project (Section 3) must contain:
    * Correct solutions for any seven problems from problem 1 — problem 11.
    * A mostly correct implementation for all parts of the Jupyter notebook, except possibly the section on robust training.
  - Your extension section may be blank (i.e., you do not need an extension to get a C).
- To get a B, your project report must fulfill the following requirements:
  - Your introduction must describe the research problem clearly and in detail;
  - Your literature review must summarize both provided papers and at least one more, and contain mostly-correct answers to the provided questions in Section 2;
  - Your results for the guided portion of the project (Section 3) must contain:
    * Correct solutions for any nine problems from problem 1 — problem 11.
    * A mostly correct implementation for all parts of the Jupyter notebook, except possibly the section on robust training.
  - Your extension section may be blank (i.e., you do not need an extension to get a B).
- To get a A, your project report must fulfill the following requirements:
  - Your introduction must describe the research problem clearly and in detail;
  - Your literature review must summarize both provided papers plus at least one more related work and contain completely correct answers to the provided questions in Section 2;
  - Your results for the guided portion of the project (Section 3) must contain completely correct mathematical solutions and/or code implementations for all parts of the problem set and Jupyter notebook;

- Your extension must contain significant, detailed, and organized work towards summarizing or synthesizing existing research, as per the guidelines in Section 4. In particular, it should start with a clear research question and document one or more attempts to answer this question. Note that the question does not need to be conclusively answered — this rubric item just asks for a thoughtful attempt to be made.

References

[1] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International conference on machine learning*, PMLR, 2018, pp. 5286–5295.

[2] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, Ieee, 2017, pp. 39–57.

