EECS 127/227AT Optimization Models in Engineering
Spring 2020 Project

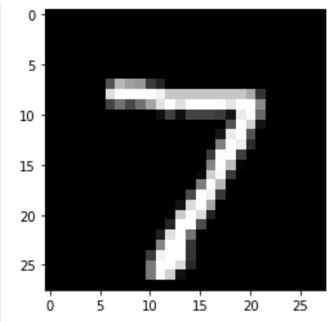
Provably Robust Deep Classifiers

1 Deep Network Classifiers

In this project, we will be exploring the robustness of deep neural classifiers. In particular, we will see a technique developed by Wong and Kolter[1] that can prove a given classifier is not easily fooled by small perturbations of its inputs.

Formally, a classifier is a function $f_\theta : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, where θ are the parameters for the classifier, n is the dimension of the input, and m is the number of classes. Given an input $\vec{x} \in \mathbb{R}^n$ and using parameter values θ, the classifier outputs a confidence vector $\vec{y} \in \mathbb{R}^n$, where the ith element represents the classifier's confidence that $\vec{x}$ belongs to class i.

For concreteness, we consider the MNIST classification task. The inputs $\vec{x} \in \mathbb{R}^n$ are images of handwritten digits (which have been flattened into vectors); the output is a vector in $\mathbb{R}^{10}$ where the first element is the classifier's confidence that $\vec{x}$ is an image of the digit zero, the second is the confidence that $\vec{x}$ is an image of the digit one, and so on. The classifier's final prediction is the class it is most confident on. See Figure 1.



```
0 : -4.409153938293457
1 : -5.768446445465088
2 : 2.118485689163208
3 : 0.9211958646774292
4 : -8.003395080566406
5 : -3.3676719665527344
6 : -10.488423347473145
7 : 6.815160751342773
8 : -3.337733745574951
9 : -0.6841652989387512
```

Figure 1: Example MNIST input and classifier output. In this case the classifier is correct, since the image is of the digit seven, and this is the class that the classifier is most confident in.

[1] J. Z. Kolter and E. Wong. 2018. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*. 5283–5292.

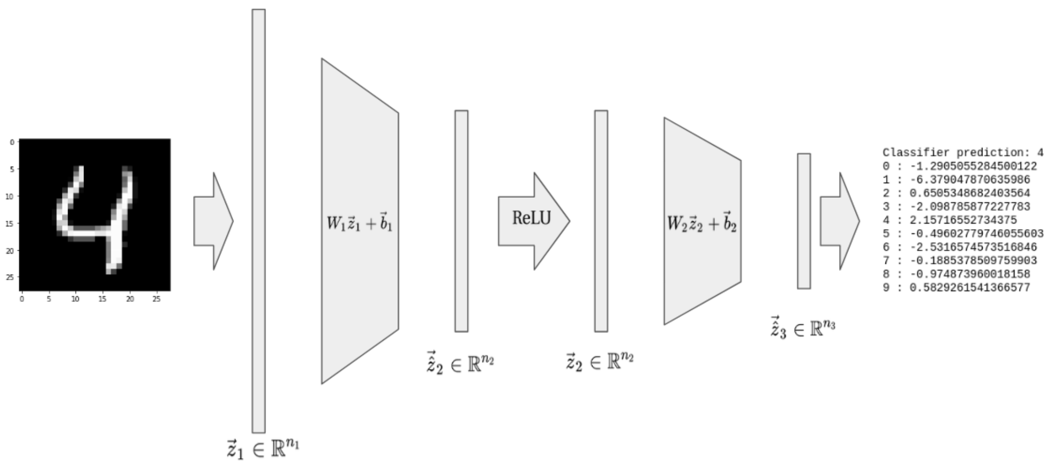
We can think of classifiers as solutions to the optimization problem

$$\min_{\theta} \sum_{\vec{x} \in \mathcal{D}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}}) \tag{1}$$

where L is some loss function, the set $\mathcal{D}$ is the dataset of input images, and $\vec{y}_{\text{true}}$ is the vector that is one on the ground truth class and zero elsewhere. An example loss function L could be the zero-one loss

$$L(\vec{x}) := \begin{cases} 1 & \operatorname{argmax}_{i\in[n]} f_\theta(\vec{x})_i \neq \operatorname{argmax}_{i\in[n]} (y_{\text{true}})_i \\ 0 & \text{otherwise.} \end{cases}$$

Then, the objective function in (1) 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.


$W_1\vec{z}_1 + \vec{b}_1$
ReLU
$W_2\vec{z}_2 + \vec{b}_2$
$\vec{z}_1 \in \mathbb{R}^{n_1}$
$\vec{\hat{z}}_2 \in \mathbb{R}^{n_2}$
$\vec{z}_2 \in \mathbb{R}^{n_2}$
$\vec{\hat{z}}_3 \in \mathbb{R}^{n_3}$

```
Classifier prediction: 4
0 : -1.2905055284500122
1 : -6.379047870635986
2 : 0.6505348682403564
3 : -2.098785877227783
4 : 2.15716552734375
5 : -0.49602779746055603
6 : -2.5316574573516846
7 : -0.1885378509759903
8 : -0.974873960018158
9 : 0.5829261541366577
```

Figure 2: The deep neural network f_θ.

For this project, we will be considering a three-layer feedforward neural network with ReLU non-linearity; see Figure 2. That is, the network $f_\theta : \mathbb{R}^{n_1} \longrightarrow \mathbb{R}^{n_3}$ consists of the layers

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

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

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

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

$$\begin{aligned} \vec{z}_1 &:= \vec{x} \\ \vec{\hat{z}}_2 &:= W_1\vec{z}_1 + \vec{b}_1 \\ \vec{z}_2 &:= \operatorname{ReLU}(\vec{\hat{z}}_2) \\ \vec{\hat{z}}_3 &:= W_2\vec{z}_2 + \vec{b}_2 \\ f_\theta(\vec{x}) &:= \vec{\hat{z}}_3. \end{aligned} \tag{2}$$

ReLU (short for Rectified Linear Unit) is defined as

$$\mathrm{ReLU}(\vec{z}) := \max\{\vec{z}, \vec{0}\}$$

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 (1), we search over all possibilities for

$$\theta := (W_1, W_2, \vec{b}_1, \vec{b}_2) \in \mathbb{R}^{n_2 \times n_1} \times \mathbb{R}^{n_3 \times n_2} \times \mathbb{R}^{n_2} \times \mathbb{R}^{n_3}$$

in order to find the parameters that best minimize the total loss.

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. The scalar A_{jk} is the element of A at the jth column and kth row.

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

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 mislead. 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.

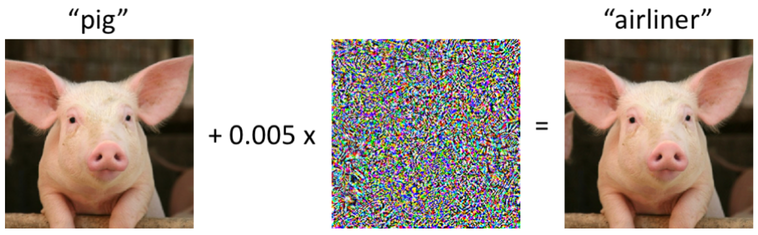


Figure 3: In this example, the original image is of a pig. However, an adversary is able to apply an imperceptible amount of noise and fool the classifier into believing the image is of an airliner. Image credit `gradientscience.com/intro_adversarial`.

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:

$$\begin{aligned} \max_{\vec{x}'} \quad & L(f_\theta(\vec{x}'), \vec{y}_{\text{true}}) \\ \text{s.t.} \quad & \|\vec{x} - \vec{x}'\|_\infty \leq \varepsilon \end{aligned} \tag{3}$$

where $\vec{y}_{\text{true}}$ is the vector that is one on the true label for $\vec{x}'$, and zero elsewhere. The constraint $\|\vec{x} - \vec{x}'\|_\infty \leq \varepsilon$ 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.

2.1 Fast Gradient Signed Method

One common way to approximate the solution to (3) is the Fast Gradient Signed Method (FGSM):

$$\vec{x}_{\text{FGSM}} := \vec{x} + \varepsilon \operatorname{sgn}(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})). \tag{4}$$

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.

Exercise 1. *Show that the FGSM perturbation* (4) *is the solution to a first-order approximation of the adversarial optimization problem, i.e.*

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

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

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 (2), we rewrite the adversarial problem (3) as

$$\begin{aligned} \min_{\vec{z}} \quad & \vec{c}^\top \vec{\hat{z}}_3 \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \varepsilon \\ & \vec{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \text{ReLU}(\vec{\hat{z}}_2) \\ & \vec{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{aligned} \tag{5}$$

where for the objective we define

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

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

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

is the difference between the classifier's confidences on the true class and the target class—if the adversary can force this objective to be negative, then the classifier is fooled into assigning the input to the target class instead of the true class.

Also, notice the slight abuse of notation in (5); 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 (5) is 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 $\hat{\vec{z}}_2$), we can instead relax the constraint $\vec{z}_2 = \mathrm{ReLU}(\hat{\vec{z}}_2)$ to $\forall j \in \{1, \dots, n_2\}$: $(z_{2j}, \hat{z}_{2j}) \in \mathcal{Z}_j$, where $\mathcal{Z}_j$ is the convex hull of the original constraint, i.e.

$$\mathcal{Z}_j := \mathrm{conv}(\hat{\mathcal{Z}}_j) = \mathrm{conv}\{(z_{2j}, \hat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} = \mathrm{ReLU}(\hat{z}_{2j})\}.$$

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

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

Similarly, if $0 \le l_j \le u_j$, we know $\hat{z}_{2j} \ge 0$ and thus $z_{2j} = \hat{z}_{2j}$, so

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

In the third case, $\hat{\mathcal{Z}}_j$ is no longer convex. Examining this set visually, it is clear that its convex

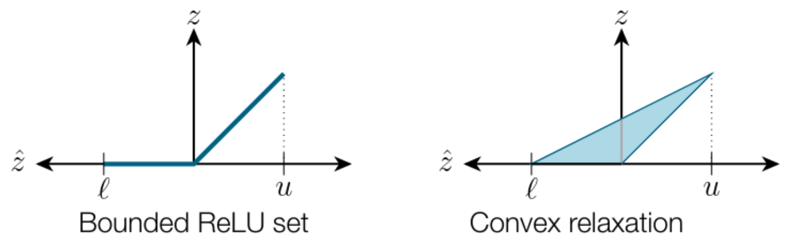


Figure 4: The convex relaxation of ReLU when $l_j \le 0 \le u_j$. Image credit [Wong & Kolter, 2018].

hull is a triangle, given by

$$\mathcal{Z}_j = \{(z_{2j}, \hat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} \ge 0 \wedge z_{2j} \ge \hat{z}_{2j} \wedge -u_j\hat{z}_{2j} + (u_j - l_j)z_{2j} \le -u_j l_j\}.$$

Note that the inequality

$$-u_j\hat{z}_{2j} + (u_j - l_j)z_{2j} \le -u_j l_j$$

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 (5) is thus

$$\begin{aligned} p^*(\vec{x}, \vec{c}) = \min_{\vec{z}} \quad & c^\top \hat{\vec{z}}_3 \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \le \varepsilon \\ & \hat{\vec{z}}_2 = W_1\vec{z}_1 + \vec{b}_1 \\ & (z_{2j}, \hat{z}_{2j}) \in \mathcal{Z}_j \qquad \forall j \in \{1, \dots, n_2\} \\ & \hat{\vec{z}}_3 = W_2\vec{z}_2 + \vec{b}_2 \end{aligned} \tag{6}$$

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.

4 Using Lagrangian Duality

The relaxed optimization problem (6) is convex, so it can be solved efficiently w/r/t 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

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

we also define the ℓ_∞ ball

$$B_\varepsilon(\vec{x}) := \{\vec{v} \in \mathbb{R}^n \mid \|\vec{v} - \vec{x}\|_\infty \leq \varepsilon\}.$$

Exercise 2. *Show that we can reëxpress the convex relaxation as*

$$\begin{aligned} p^*(\vec{x}, \vec{c}) = \min_{\vec{z}} \quad & c^\top \vec{\hat{z}}_3 + \mathbf{1}_{B_\varepsilon(\vec{x})}(\vec{z}_1) + \sum_{j=1}^{n_2} \mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \hat{z}_{2j}) \\ s.t. \quad & \vec{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{aligned} \tag{7}$$

Recall that for a function f, the Fenchel conjugate f^* is defined as

$$f^*(\vec{y}) = \max_{\vec{x}} \vec{y}^\top \vec{x} - f(\vec{x}).$$

Exercise 3. *(a) Let $\vec{\nu}_3$ be the dual variable for the constraint $\vec{\hat{z}}_3 = W_2\vec{z}_2 + \vec{b}_2$, and $\vec{\nu}_2$ be the dual variable for the constraint $\vec{\hat{z}}_2 = W_1\vec{z}_1 + \vec{b}_1$. Show that the Lagrangian for the optimization problem (7) is*

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

(b) Show that

$$\begin{aligned} g(\vec{\nu}_2, \vec{\nu}_3) & := \min_{\vec{z}} \mathcal{L}(\vec{z}, \vec{\nu}) \\ & = \min_{\vec{\hat{z}}_3}((\vec{c} + \vec{\nu}_3)^\top \vec{\hat{z}}_3) + \min_{\vec{z}_1}(\mathbf{1}_{B_\varepsilon(\vec{x})}(\vec{z}_1) - \vec{\nu}_2^\top W_1 \vec{z}_1) \\ & \quad + \left(\sum_{j=1}^{n_2} \min_{z_{2j}, \hat{z}_{2j}} (\mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \hat{z}_{2j}) - \vec{\nu}_3^\top (W_2)_j z_{2j} + \nu_{2j} \hat{z}_{2j}) \right) - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i \end{aligned} \tag{8}$$

(c) Conclude that the Lagrangian dual to (7) is

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

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

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

Hint: *Each of the minimizations in* (8) *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.

5 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 $\mathbf{1}_{B_\varepsilon(\vec{x})}$. Now, it only remains to find expressions for these conjugates.

Exercise 4. *Show that* $\mathbf{1}^*_{B_\varepsilon(\vec{x})}(\vec{\nu}) = \vec{\nu}^\top \vec{x} + \varepsilon \|\vec{y}\|_1$.

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

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

- *When* $l_j \le u_j \le 0$,

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

- *When* $0 \le l_j \le u_j$,

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

- *When* $l_j \le 0 \le u_j$,

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

6 The Dual Network

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

$$\begin{aligned} S &:= \{j \in [n_2] \mid l_j \le 0 \le u_j\} \\ S^- &:= \{j \in [n_2] \mid l_j \le u_j \le 0\} \\ S^+ &:= \{j \in [n_2] \mid 0 \le l_j \le u_j\}. \end{aligned}$$

Exercise 5. *Deduce that the dual problem can be expressed as*

$$\begin{aligned} d^*(\vec{x},\vec{c}) = \max_{\vec{\nu}} \quad & -\hat{\vec{\nu}}_1^\top \vec{x} - \varepsilon\|\hat{\vec{\nu}}_1\|_1 - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i + \sum_{j\in S} l_j \operatorname{ReLU}(\nu_{2j}) \\ s.t. \quad & \vec{\nu}_3 = -\vec{c} \\ & \hat{\vec{\nu}}_2 = W_2^\top \vec{\nu}_3 \\ & \nu_{2j} = 0 && \forall j \in S^- \\ & \nu_{2j} = \hat{\nu}_j && \forall j \in S^+ \\ & \nu_{2j} = \frac{u_j}{u_j - l_j}\hat{\nu}_j && \forall j \in S \\ & \hat{\vec{\nu}}_1 = W_1^\top \vec{\nu}_2 \end{aligned} \tag{9}$$

Observe now that $\vec{\nu}_3 = -\vec{c}$, and, given each $\vec{\nu}_{i+1}$, we can determine what values $\hat{\vec{\nu}}_i$ 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 $\vec{\nu}_3, \hat{\vec{\nu}}_2, \vec{\nu}_2, \hat{\vec{\nu}}_1$. 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 (6) and again lower-bounds the primal adversarial problem (5) 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

$$\hat{\vec{\nu}}_i = W_i^\top \vec{\nu}_{i+1}$$

can be thought of as the reverse of the affine layer

$$\hat{\vec{z}}_{i+1} = W_i \vec{z}_i + \vec{b}_i$$

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

$$\vec{z}_i = \operatorname{ReLU}(\hat{\vec{z}}_i),$$

we have a different nonlinearity

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

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 $\vec{\nu}_3, \hat{\vec{\nu}}_2, \vec{\nu}_2, \hat{\vec{\nu}}_1$ as

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

and define a loss

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

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

7 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.

Exercise 6. *For any matrix W with rows $\vec{w}_1^\top, \ldots, \vec{w}_k^\top$, denote*

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

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

Show that

$$\vec{u} = W_1\vec{x} + \vec{b}_1 + \varepsilon\|W_1\|_{:1} \tag{10}$$

and

$$\vec{l} = W_1\vec{x} + \vec{b}_1 - \varepsilon\|W_1\|_{:1} \tag{11}$$

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

Using this, whenever we need to compute $g_\theta(\vec{c})$, we can first obtain u and l using the formulas (10) and (11).

8 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 (9). By weak duality, this is a lower bound for $p^*(\vec{x}, \vec{c})$, which is the optimum for the relaxed primal problem (6). This in turn is a lower bound for

$$\begin{aligned} \min_{\vec{z}} \quad & \vec{c}^\top \vec{\hat{z}}_3 \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \le \varepsilon \\ & \vec{\hat{z}}_2 = W_1\vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \operatorname{ReLU}(\vec{\hat{z}}_2) \\ & \vec{\hat{z}}_3 = W_2\vec{z}_2 + \vec{b}_2 \end{aligned} \tag{5 again}$$

which is the original adversarial problem. Recall that we choose

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

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 (5) 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

$$c_j = \vec{y}_{\text{true}} - \vec{e}_j$$

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 $\|\vec{x} - \vec{x}'\|_\infty < \varepsilon$ such that the classifier is incorrect on $\vec{x}'$.

A Optional: Training a robust classifier

Using the dual network (9), 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 y, 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

$$L(y, y_{\text{true}}) \leq L(y', y_{\text{true}}).$$

Definition 2. A multi-class loss function $L : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}$ is *translation-invariant* if for all $a \in \mathbb{R}$,

$$L(y, y_{\text{true}}) = L(y - a\mathbf{1}, y_{\text{true}}).$$

Exercise 7. *[**Optional**] Show that cross-entropy loss, defined as*

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

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:

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

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.

Exercise 8. *[**Optional**] For monotonic and translation-invariant loss L, show that*

$$\max_{\|x-x'\|_\infty \leq \varepsilon} L(f_\theta(\vec{x}'), y_{\text{true}}) \leq L(-J_\varepsilon(\vec{x}, g_\theta(e_y \mathbf{1}^\top - I)), y_{\text{true}}).$$

This shows that by solving the optimization problem

$$\min_\theta \sum_{x \in \mathcal{D}} L(-J_\varepsilon(\vec{x}, g_\theta(e_y \mathbf{1}^\top - I)), y_{\text{true}}),$$

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.

B Optional: 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 main section.

Exercise 9. ***[Optional]*** *Show that when $l_j \leq u_j \leq 0$,*

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

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

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

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

Exercise 10. ***[Optional]*** *Show that when $l_j \leq 0 \leq u_j$,*

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

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 $\hat{\mathcal{Z}}_j$. Deduce that the optimal $(y, \hat{y})$ for the Fenchel conjugate maximization problem is either $(0, 0)$ or on the line*

$$-u_j \hat{y} + (u_j - l_j) y = -u_j l_j.$$