Convex Polytopes Are All You Need
To Defend Your Model

Alberto Hojel[*]
UC Berkeley
`ahojel@berkeley.edu`

Ryan Tabrizi[*]
UC Berkeley
`rtabrizi@berkeley.edu`

Heather Ding[*]
UC Berkeley
`rtabrizi@berkeley.edu`

Abstract

This technical report explores the challenges of adversarial attacks [2] on machine learning models and evaluates the provable defenses against these attacks. Considering the many consequences of model vulnerability in applications like healthcare and autonomous vehicles, it is essential to grasp the nature of these attacks and what solutions exist. This report focuses primarily on the attacks proposed in [3] and the provable defense in Wong and Kolter [4]. To better understand these attacks and defenses, we step through each proof and record justifications along the way. Altogether, we evaluate and explain adversarial objectives, defensive distillation [1], L-BFGS [3], and Fast Gradient Signed Method Approximation [2], outlining the processes of adversarial attacks, defenses, and their implications.

1 Introduction

Recent advancements in machine learning have achieved impressive feats, yet not much research has been done to ensure that these models work as expected. The consequences of malfunctioning models are particularly severe in the context of autonomous vehicles and healthcare, for instance. Moreover, such models can be infiltrated and exploited through what are called *adversarial attacks*. Specifically, adversarial examples are designed to invoke atypical behavior in models to undermine their safety and security. In the context of computer vision, adversarial examples introduced in [2] and [3] demonstrate how normal they can look to the human eye, yet cause incorrect classifications.

The relevance and importance of adversarial machine learning stem from the ongoing arms race between researchers who develop methods to strengthen classifiers against known attacks and those who invent new, more potent attacks capable of bypassing these defenses. This continuous cycle of attack and defense has driven the field forward but has also exposed the vulnerabilities of even the most sophisticated machine learning models.

To address these challenges, it is crucial to develop classifiers that are not only resilient to adversarial perturbations but also provably robust, ensuring that they can withstand attacks even when the adversary has complete knowledge of the classifier. This level of security is essential in maintaining the integrity and reliability of machine learning systems, particularly in safety-critical applications.

In this technical report, we build upon the pioneering work of Wong and Kolter (2018) [4], who proposed an approach for training provably robust deep ReLU classifiers. These classifiers are designed to be resistant to any norm-bounded adversarial perturbations within the training set, ensuring that they maintain their performance even in the presence of adversarial attacks. Furthermore, their method offers a provable technique for detecting any previously unseen adversarial examples with zero false negatives, although it may erroneously flag some non-adversarial examples.

The crux of Wong and Kolter's technique lies in constructing a convex outer bound on the "adversarial polytope" – the set of all final-layer activations achievable by applying a norm-bounded perturbation to the input. By ensuring that the class prediction of an example remains unchanged within this outer

[*]Denotes Equal Contribution.

bound, it is possible to prove that the example cannot be adversarial, as a small perturbation would not alter the class label.

In this project, we aim to delve deeper into Wong and Kolter's approach, providing a more accessible and descriptive explanation of their technique. By building upon their work, we hope to develop a nuanced overview of their method for training robust ReLU classifiers. Our ultimate goal is to contribute to the ongoing effort to protect machine learning systems against adversarial attacks, enhancing their security and reliability in real-world applications.

2 Towards Evaluating the Robustness of Neural Networks

In this work, Carlini and Wagner demonstrate that defensive distilled networks do not successfully protect against new adversarial attacks they propose.

2.1 Defensive Distillation

Previous work propose defensive distillation as a method to protect against adversarial attacks. A defensively distilled network is created by the following steps: take an existing neural network, use it to generate labels using a smoothed version of the softmax loss function, use the new labels to train on a new version of the same neural network.

2.2 L-BFGS

L-BFGS is a way to generate adversarial examples that successfully fool neural networks and have been proposed as an attack in previous papers. We formulate the optimization problem as follows:

$$\begin{aligned}&\text{minimize } \|x - x'\|_2^2\\&\text{subject to } C(x') = l,\\&x' \in [0,1]^n.\end{aligned}$$

In this formulation, the objective $\|x - x'\|_2$ minimizes the Euclidean distance between the original input and some perturbed input, such that our classifier classifies x' as some target class l. The box constraint ensures we still have valid pixel values. In practice, the following optimization problem is easier to solve for an optimal $c > 0$ found via line search:

$$\begin{aligned}&\text{minimize } c \cdot \|x - x'\|_2^2 + \text{loss}_{F,l}(x')\\&\text{subject to } x' \in [0,1]^n\end{aligned}$$

2.3 Newly Proposed L-2 Attack Algorithm

Carlini and Wagner propose a new L-2 attack method that successfully foils defensively distilled neural networks that remain visually indistinguishable from the original.

$$\min_w \|\frac{1}{2}(tanh(w) + 1) - x\|_2^2 + c \cdot f\left(\frac{1}{2}\left(\tanh(w) + 1\right)\right)$$

where f is defined as

$$f(x') = \max\left(\max_{i \neq t}\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa\right).$$

The term inside the L-2 norm minimizes the difference between the real input vector and the perturbed input vector, while the second term enforces our target class instead of the real class. κ determines how confident we want to be on our target class, the paper uses 0. Inside the function f, we are enforcing that after passing through all the layers($Z(x')$) will be forced into class t. We solve this optimization problem using gradient descent on multiple random starting points similar to the original vector to prevent getting stuck at a local minimum.

2.4 Provable Defenses

We now move on to more recent work by Wong and Kolter that propose a guaranteed defense against adversarial examples using the convex outer adversarial polytope.

The original optimization problem is nonconvex as the normball with which we define all possible adversarial attacks becomes nonconvex after undergoing the nonlinearities of the neural netweork. We see this in figure 1 where the outputed polytope is clearly not convex.

To create a relaxed polytope that is convex, we relax the ReLU activations whose convex hull, as shown in figure 2, is now convex. In doing so, we can arrive at a convex outer bound as seen in figure 1, which we can then use to prove robustness for different algorithms as we outline in the rest of the report.

Since the dual lower bounds the primal, we are able to assign scores to certain inputs that flag them as dangerous or not, as a negative solution to the optimization problem suggests that it has been classified as not the true class.

3 Deep Neural Network Classifiers

3.1 Generalized Description

A deep neural network classifier is a function that maps an input vector to an output vector corresponding to class probabilities. Mathematically, it is defined by a set of parameters Θ and a family of classifiers $\mathcal{F} := \{f_\theta : \theta \in \Theta\}$. 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. The goal is to find an f_θ that produces an output $\vec{y}_{\text{pred}} = f_\theta(\vec{x})$ that is close to the true label $\vec{y}_{\text{true}}$ according to a loss function.

For a given dataset $\mathcal{D}$, the optimal classifier minimizes the empirical risk function:

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

where $L : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a loss function that compares the model's prediction $f_\theta(\vec{x})$ to the true label $\vec{y}$.

3.2 Our Network

For this exploration, we will be considering a three-layer feedforward neural network with ReLU nonlinearity. That is, the network $f_\theta : \mathbb{R}^{n_1} \to \mathbb{R}^{n_3}$ consists of the layers

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

where $\vec{z}_1 = \vec{x}$ is the input for the network and $\overrightarrow{\hat{z}}_3$ 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

$$\begin{aligned}
\vec{z}_1 &\doteq \vec{x} \\
\overrightarrow{\hat{z}}_2 &\doteq W_1 \vec{z}_1 + \vec{b}_1 \\
\vec{z}_2 &\doteq \text{ReLU}\left(\overrightarrow{\hat{z}}_2\right) \\
\overrightarrow{\hat{z}}_3 &\doteq W_2 \vec{z}_2 + \vec{b}_2 \\
f_\theta(\vec{x}) &\doteq \overrightarrow{\hat{z}}_3
\end{aligned}$$

ReLU (short for Rectified Linear Unit) is defined as

$$\text{ReLU}(\vec{z}) \doteq \max\{\vec{z}, \vec{0}\}$$

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 define the parameter as

$$\theta \doteq \left(W_1, W_2, \vec{b}_1, \vec{b}_2\right)$$

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

4 Finding Adversarial Examples

The adversarial objective is a key concept in understanding the vulnerability of deep neural networks to adversarial attacks. The goal of an adversary is to find a perturbed input $\vec{x}'$ that is close to the original input $\vec{x}$, but results in an incorrect classification by the model. To achieve this, the adversary aims to maximize the loss function $L\left(f_\theta\left(\vec{x}'\right), \vec{y}_{\text{true}}\right)$, where $f\theta$ is the deep neural network classifier, and $\vec{y}_{\text{true}}$ is the true label for $\vec{x}'$.

The optimization problem that the adversary tries to solve can be formally expressed as:

$$\max_{\vec{x}'} \quad L\left(f_\theta\left(\vec{x}'\right), \vec{y}_{\text{true}}\right) \text{ s.t.} \qquad \|\vec{x} - \vec{x}'\|_\infty \leq \epsilon \tag{2}$$

The constraint $\|\vec{x} - \vec{x}'\|_\infty \leq \epsilon$ ensures that the adversarial input $\vec{x}'$ is close to the original input $\vec{x}$. Each element in the vector (corresponding to a pixel value in the case of some computer vision applications) should be no more than ϵ away from the original value (when using an infinity norm bound). This constraint maintains the visual similarity between $\vec{x}$ and $\vec{x}'$, making it difficult for humans to distinguish between the two inputs, while still causing the classifier to make incorrect predictions.

In summary, the adversarial objective encapsulates the goal of adversaries in crafting perturbed inputs that are visually indistinguishable from the original inputs but result in incorrect model predictions, thereby exposing the vulnerability of deep neural networks to adversarial attacks.

4.1 Fast Gradient Signed Method Approximation

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

$$\vec{x}_{\text{FGSM}} = \vec{x} + \epsilon \text{sgn}(\nabla \vec{x} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})).$$

This is similar to gradient ascent of the loss with respect to the input, except we only take a single step and use the sign of the gradient instead of the gradient itself.

The FGSM perturbation is the solution to a first-order approximation of the adversarial optimization problem, i.e.,

$$\begin{aligned} \vec{x}_{\text{FGSM}} = \arg\max_{\vec{x'}} &\Big[L(f_\theta(\vec{x}), \vec{y}_{\text{true}}) + \left(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top \vec{x'}\Big] \\ &\text{subject to} \quad \|\vec{x} - \vec{x'}\|_\infty \leq \epsilon. \end{aligned} \tag{3}$$

This can become evident through the following proof:

Let $\vec{x'} = \vec{x} + \epsilon\vec{v}$. The constraint in Equation 3 becomes:

$$\|\vec{x} - \vec{x} - \epsilon\vec{v}\|_\infty \leq \epsilon$$

Which can be simplified as follows:

$$\begin{aligned} \|\vec{x} - \vec{x} - \epsilon\vec{v}\|_\infty &\leq \epsilon, \\ \|-\epsilon\vec{v}\|_\infty &\leq \epsilon, \\ |-\epsilon|\|\vec{v}\|_\infty &\leq \epsilon, \\ \|\vec{v}\|_\infty &\leq 1. \end{aligned}$$

The objective function in Equation 3 becomes:

$$\begin{aligned} &\arg\max_{\vec{x}+\epsilon\vec{v}} \Big[L(f_\theta(\vec{x}), \vec{y}_{\text{true}}) + \left(\nabla \vec{x} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top (\vec{x'} + \epsilon\vec{v})\Big] \\ &\quad = \arg\max_{\vec{x}+\epsilon\vec{v}} \Big[L(f_\theta(\vec{x}), \vec{y}_{\text{true}}) + \left(\nabla \vec{x} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top \vec{x'} + \left(\nabla \vec{x} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top \epsilon\vec{v}\Big] \\ &\quad = \vec{x} + \epsilon \arg\max_{\vec{v}} \Big[\left(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top \vec{v}\Big] \end{aligned}$$

The optimization problem can be rewritten as:

$$\vec{x}_{\text{FGSM}} = \vec{x} + \epsilon \arg\max_{\vec{v}} \Big[\left(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top \vec{v} \Big], \\ \text{subject to} \quad \|\vec{v}\|_\infty \leq 1.$$

By dual norm properties, we have

$$\begin{matrix} \max_{\vec{v}} \vec{u}^\top \vec{v} \\ \text{s.t.} \quad \|\vec{v}\|_\infty \leq 1 \end{matrix} = \|\vec{u}\|_1,$$

where $\vec{u} = \nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})$. To achieve $\|\vec{u}\|_1$, we set

$$\vec{v} = \text{sgn}(\vec{u}) = \text{sgn}(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})).$$

We conclude that

$$\vec{x}_{\text{FGSM}} = \vec{x} + \epsilon \text{sgn}(\nabla \vec{x} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})).$$

By applying the FGSM, we have shown that it is a first-order approximation of the adversarial objective.

4.2 ℓ_2 norm constraints versus ℓ_∞

To approximate the ℓ_2 norm ball attack, we proceed similarly as the ℓ_∞ norm but with the following constraint:

$$\|\vec{x} - \vec{x'}\|_2 \leq \epsilon.$$

To simplify the constraint, we can use the same trick as before and let $\vec{x'} = \vec{x} + \epsilon\vec{v}$:

$$\begin{aligned} \|\vec{x} - \vec{x} - \epsilon\vec{v}\|_2 &\leq \epsilon, \\ \| - \epsilon\vec{v}\|_2 &\leq \epsilon, \\ | - \epsilon| \|\vec{v}\|_2 &\leq \epsilon, \\ \|\vec{v}\|_2 &\leq 1. \end{aligned}$$

To simplify the objective function, we again proceed as earlier, the objective function can be simplified to:

$$\vec{x}_{\text{FGSM}} = \vec{x} + \epsilon \arg\max_{\vec{v}} \Big[\left(\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\right)^\top \vec{v} \Big], \\ \text{subject to} \quad \|\vec{v}\|_2 \leq 1.$$

Since to maximize the dot product between two vectors we want them in the same direction, we obtain the following for $\vec{v}$:

$$\vec{v} = \frac{\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})}{\|\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\|_2}$$

It follows then that:

$$\vec{x}_{\text{FGSM}} = \vec{x} + \frac{\epsilon}{\|\nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}})\|_2} \nabla_{\vec{x}} L(f_\theta(\vec{x}), \vec{y}_{\text{true}}).$$

5 Rewriting the Adversary's Optimization Problem

We reformulate the adversarial problem as follows:

$$\begin{array}{ll} \min_{\vec{z}} & \vec{c}^{\top} \overrightarrow{\hat{z}}_3 \\ \text{s.t.} & \|\vec{z}_1 - \vec{x}\|_{\infty} \leq \epsilon \\ & \overrightarrow{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \text{ReLU}\left(\overrightarrow{\hat{z}}_2\right) \\ & \overrightarrow{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{array} \tag{4}$$

Here, we define the objective function as $\vec{c}^{\top} \overrightarrow{\hat{z}}_3$, where $\vec{c} = \vec{y}_{\text{true}} - \vec{y}_{\text{targ}}$. Both $\vec{y}_{\text{true}}$ and $\vec{y}_{\text{targ}}$ are one-hot vectors corresponding to the ground truth and adversarial label respectively. The objective function computes the difference between the classifier's scores assigned to the true class and the target class. If the adversary can make this objective negative, then the adversarial example's activation is higher than that of the ground truth, and the classifier will assign higher probability to the adversarial example. Furthermore, we only need to find one such adversarial example for a successful attack on the network.

It is important to note the slight abuse of notation in, where we use $\min_{\vec{z}}$ as shorthand for $\min_{\vec{z}_1, \overrightarrow{\hat{z}}_2, \vec{z}_2, \overrightarrow{\hat{z}}_3}$. This convention is maintained throughout the document to avoid clutter.

5.1 Primal Modification for Guaranteeing Target Classification

As stated earlier, a successful attack occurs when the objective value in (4) is negative. Perhaps the adversary wants to output a specific $\vec{y}_{targ}$ rather than simply preventing $\vec{y}_{true}$. In this case, we can solve the following optimization problem:

$$\begin{array}{ll} \min_{\vec{z}} & (\vec{y}_i - \vec{y}_{targ})^{\top} \overrightarrow{\hat{z}}_3 \ \forall i \neq \text{target} \\ \text{s.t.} & \|\vec{z}_1 - \vec{x}\|_{\infty} \leq \epsilon \\ & \overrightarrow{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \text{ReLU}\left(\overrightarrow{\hat{z}}_2\right) \\ & \overrightarrow{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \end{array} \tag{5}$$

Now, the activation corresponding to the target adversarial example is greater than that of all other classes, not only $\vec{y}_{\text{true}}$. In doing so, we guarantee $\vec{y}_{\text{targ}}$ as the predicted output. We can formulate this in standard form:

$$\begin{array}{ll} \min_{\vec{z}} & \vec{1}^{\top} \vec{s} \\ \text{s.t.} & \|\vec{z}_1 - \vec{x}\|_{\infty} \leq \epsilon \\ & \overrightarrow{\hat{z}}_2 = W_1 \vec{z}_1 + \vec{b}_1 \\ & \vec{z}_2 = \text{ReLU}\left(\overrightarrow{\hat{z}}_2\right) \\ & \overrightarrow{\hat{z}}_3 = W_2 \vec{z}_2 + \vec{b}_2 \\ & (\vec{y}_i - \vec{y}_{targ})^{\top} \overrightarrow{\hat{z}}_3 \leq \vec{s}_i \ \forall i \neq \text{targ} \end{array} \tag{6}$$

6 The Adversarial Polytope

We define the adversarial polytope $Z_{\epsilon}(x)$, which is the set of all final-layer activations attainable by perturbing input x with a change Δ of bounded ℓ_{∞} norm ϵ:

$$Z_{\epsilon}(x) = \{f_{\theta}(x + \Delta) : \|\Delta\|_{\infty} \leq \epsilon\}. \tag{7}$$

Optimizing over $Z_{\epsilon}(x)$ for multi-layer networks with non-linearities like ReLU is challenging because the set is non-convex as we see in figure 1. To address this issue, Wong and Kolter's work

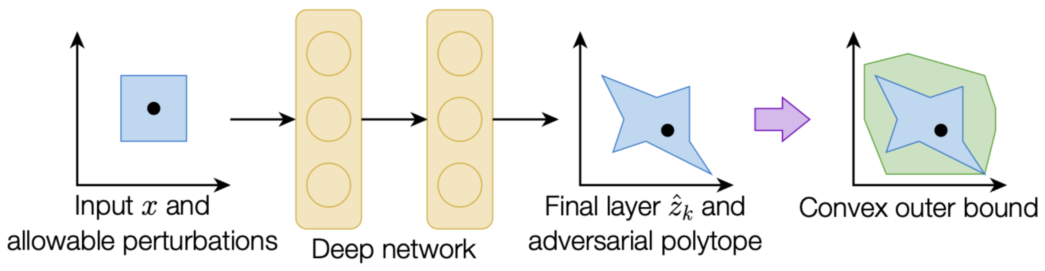
Input x and allowable perturbations
Deep network
Final layer $\hat{z}_k$ and adversarial polytope
Convex outer bound

Figure 1: The non-convex adversarial polytope and its corresponding convex outer bound as shown in [4].

[4] constructs a convex outer bound on the adversarial polytope. If one can prove that no point within this bound can change the model's prediction, then we guarantee that this example is not adversarial. In a sense, we consider all perturbations near the bounds of the non-convex polytope, as adversarial examples will often reside near these bounds. Ultimately, we will train a network to optimize the worst-case loss over this convex outer bound, thus enabling the application of robust optimization techniques despite the classifier's non-linearity.

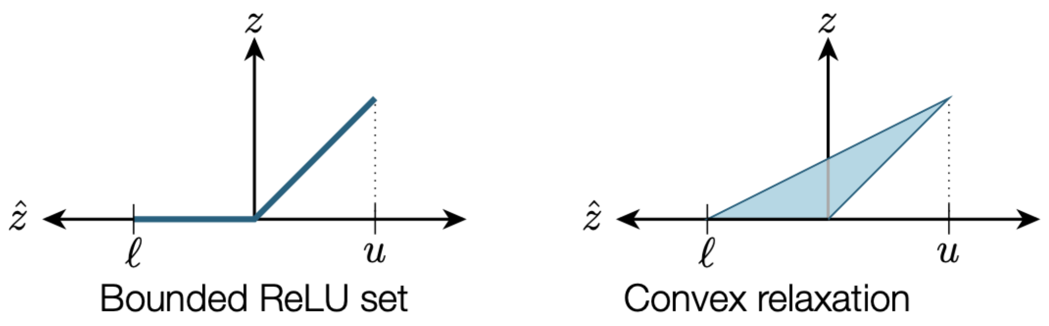

Figure 2: The convex relaxation of the ReLU non-linearity from [4].

To construct the convex outer bound, we start with a linear relaxation of the ReLU activations as proposed in [4]. Given known lower and upper bounds l and u for the pre-ReLU activations, we replace the ReLU equalities $z = \max\{0, \widehat{z}\}$ with their upper convex envelopes:

$$z \geq 0, \quad z \geq \widehat{z}, \quad -u\widehat{z} + (u - l)z \leq -ul.$$

We then analyze the relaxed constraint $\vec{z}_2 = \text{ReLU}(\overrightarrow{\widehat{z}}_2)$ on a case-by-case basis, utilizing the upper and lower bounds u_j and l_j for each $\widehat{z}_{2j}$. For each $j \in \{1, \ldots, n_2\}$, we introduce the convex hull $\mathcal{Z}_j$ of the original constraint:

$$\mathcal{Z}_j \doteq \text{conv}(\widehat{\mathcal{Z}}_j) = \text{conv}\left(\left\{ (z_{2j}, \widehat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} = \text{ReLU}\left(\widehat{z}_{2j}\right) \wedge l_j \leq \widehat{z}_{2j} \leq u_j \right\}\right) \tag{8}$$

We consider three cases:

1. If $l_j \leq u_j \leq 0$, the ReLU constraint is equivalent to fixing $z_{2j} = 0$. Thus, $\widehat{\mathcal{Z}}_j$ is already convex, and we can define:

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

2. If $0 \leq l_j \leq u_j$, we have $z_{2j} = \widehat{z}_{2j}$, so:

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

3. In the third case, $\widehat{\mathcal{Z}}_j$ is no longer convex. Its convex hull is a triangle, given by:

$$\mathcal{Z}_j = \left\{ (z_{2j}, \widehat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid z_{2j} \geq 0 \wedge z_{2j} \geq \widehat{z}_{2j} \right. \\ \left. \wedge -u_j \widehat{z}_{2j} + (u_j - l_j)\, z_{2j} \leq -u_j l_j \right\} \tag{9}$$

By exploiting the upper and lower bounds of the ReLU and the convex hull of the original constraint, we transform the non-convex problem into a convex one. This allows us to reason about the adversarial polytope bound and prove robustness through the problem's convexity.

6.1 Relaxation of the Adversary's Optimization Problem

Our relaxation of the problem (4) is thus

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

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. If we can prove robustness in the relaxed problem, then we have also proved robustness for the original problem.

6.2 Dualizing the Adversary's Optimization Problem

Although the relaxed adversarial optimization problem is convex and thus solvable, here we will show that the dual problem is much easier to solve and thus preferable. In this subpart we will give in-depth steps on finding the dual optimization problem.

6.2.1 Re-expressing the convex relaxation

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

Let the constraint: $\|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon$ become:

$$1_{B_\epsilon}(\vec{z_1}) = \begin{cases} 0 & \vec{z_1} \text{ st } \|\vec{z_1} - \vec{x}\|_\infty \leq \epsilon \\ \infty & \text{otherwise} \end{cases}$$

Let the constraint: $(z_{2j}, \widehat{z}_{2j}) \in \mathcal{Z}_j \quad \forall j \in \{1, \ldots, n_2\}$ become

$$1_{z_j}(z_{2j}, \widehat{z}_{2j}) = \begin{cases} 0 & (z_{2j}, \widehat{z}_{2j}) \in z_j \\ \infty & \text{otherwise} \end{cases}$$

Since the minimum objective will never choose ∞ for these two functions, the minimization problem becomes:

$$
\begin{aligned}
p^*(\vec{x},\vec{c}) = \min_{\vec{z}} \quad & c^\top \vec{\hat{z}}_3 + 1_{B_\epsilon}(\vec{z_1}) + 1_{z_j}(z_{2j}, \hat{z}_{2j}) \\
\text{s.t.} \quad & \vec{\hat{z}}_2 = W_1\vec{z}_1 + \vec{b}_1 \\
& \vec{\hat{z}}_3 = W_2\vec{z}_2 + \vec{b}_2
\end{aligned} \tag{10}
$$

6.2.2 Deriving the Lagrangian

Although the primal problem (10) can be computed with modern solvers, the input and hidden layers in classification problems can become very large and increase compute. To mitigate this, we can solve this problem through duality. We proceed with finding the Lagrangian:

$$
\begin{aligned}
\mathcal{L}(\vec{z},\vec{\nu}) =& \vec{c}^\top \vec{\hat{z}}_3 + \mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) + \sum_{j=1}^{n_2} \mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \hat{z}_{2j}) \\
&+ \vec{v_3}^\top(\hat{\vec{z_3}} - W_2\vec{z_2} - \vec{b_2}) + \vec{v_2}^\top(\hat{\vec{z_2}} - W_1\vec{z_1} - \vec{b_1}) \\
=& \vec{c}^\top \vec{\hat{z}}_3 + \mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) + \sum_{j=1}^{n_2} \mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \hat{z}_{2j}) \\
&+ \vec{v_3}^\top\hat{\vec{z_3}} - \vec{v_3}^\top W_2\vec{z_2} - \vec{v_3}^\top\vec{b_2} \\
&+ \vec{v_2}^\top\hat{\vec{z_2}} - \vec{v_2}^\top W_1\vec{z_1} - \vec{v_2}^\top\vec{b_1}
\end{aligned}
$$

Where $\vec{v_2}$ corresponds with the first constraint and $\vec{v_3}$ corresponds with the second.

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

6.2.3 Concluding with the Dual

Remember our "abuse of notation earlier". When we minimize over $\vec{z}$ we are in actuality minimizing over $\vec{z_1}, \vec{z_2}, \vec{z_3}, \hat{\vec{z_1}}, \hat{\vec{z_2}}, \hat{\vec{z_3}}$ By collecting like terms we can simplify as follows:

$$
\begin{aligned}
g(\vec{\nu}_2, \vec{\nu}_3) \doteq& \min_{\vec{z}} \mathcal{L}(\vec{z},\vec{\nu}) \\
=& \min_{\vec{z}}(\vec{c}^\top \vec{\hat{z}}_3 + \vec{\nu}_3^\top \vec{\hat{z}}_3 + \mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) - \vec{\nu}_2^\top W_1\vec{z}_1 \\
&+ \left(\sum_{j=1}^{n_2} \mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \hat{z}_{2j}) - \vec{\nu}_3^\top W_2\vec{z}_2 + \vec{\nu}_2^\top \vec{\hat{z}}_2\right) - \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i) \\
=& \min_{\vec{z_3}}\left((\vec{c} + \vec{\nu}_3)^\top \vec{\hat{z}}_3\right) + \min_{\vec{z_1}}\left(\mathbf{1}_{B_\epsilon(\vec{x})}(\vec{z}_1) - \vec{\nu}_2^\top W_1\vec{z}_1\right) \\
&+ \left(\sum_{j=1}^{n_2} \min_{z_{2j},\hat{z}_{2j}} \left(\mathbf{1}_{\mathcal{Z}_j}(z_{2j}, \hat{z}_{2j}) - \vec{\nu}_3^\top (W_2)_j z_{2j} + \nu_{2j}\hat{z}_{2j}\right)\right) \\
&- \sum_{i=1}^{2} \vec{\nu}_{i+1}^\top \vec{b}_i
\end{aligned} \tag{11}
$$

6.2.4 The Full Dual Problem

In this section, we will derive the Lagrangian dual in a simplified format, leveraging Fenchel conjugates. Looking at equation 11, we will simplify each minimization into a Fenchel conjugate or convert it into a constraint.

We have:

$$\begin{aligned}
&\min_{\vec{z}_1}\left(\mathbf{1}_{B_\varepsilon}(\vec{x})(\vec{z}_1)-\vec{\nu}_2^\top W_1\vec{z}_1\right)\\
&=-\sup_{\vec{z}_1}\left(\vec{\nu}_2^\top W_1\vec{z}_1-\mathbf{1}_{B_\varepsilon}(\vec{x})(\vec{z}_1)\right)\\
&=-\mathbf{1}^*_{B_\varepsilon}\left(W_1^\top\vec{\nu}_2\right)
\end{aligned}$$

And we have:

$$\begin{aligned}
&\sum_{j=1}^{n_2}\min_{z_{2j},\hat{z_{2j}}}\left(\mathbf{1}_{\mathcal{Z}_j}\left(z_{2j},\hat{z_{2j}}\right)-\vec{\nu}_3^\top\left(W_2\right)_j z_{2j}+\nu_{2j}\hat{z_{2j}}\right)\\
&=\sum_{j=1}^{n_2}-\sup_{z_{2j},\hat{z_{2j}}}\left(\vec{\nu}_3^\top\left(W_2\right)_j z_{2j}-\nu_{2j}\hat{z}_{2j}-\mathbf{1}_{\mathcal{Z}_j}\left(z_{2j},\hat{z_{2j}}\right)\right)\\
&=\sum_{j=1}^{n_2}-\mathbf{1}^*_{\mathcal{Z}_j}\left(\vec{\nu}_3^\top\left(W_2\right)_j,-\nu_{2j}\right)
\end{aligned}$$

Hence, our full Lagrangian dual can be written as:

$$\begin{aligned}
d^*(\vec{x},\vec{c})=\max_{\vec{\nu}}\min_{\vec{z}}\mathcal{L}(\vec{z},\vec{\nu})&=\max_{\vec{\nu}}\Bigg[\min_{\vec{z_3}}\left(\left(\vec{c}+\vec{\nu}_3\right)^\top\vec{z}_3\right)+\min_{\vec{z}_1}\left(\mathbf{1}_{B_\epsilon(\vec{x})}\left(\vec{z}_1\right)-\vec{\nu}_2^\top W_1\vec{z}_1\right)\\
&+\left(\sum_{j=1}^{n_2}\min_{z_{2j},\widehat{z}_{2j}}\left(\mathbf{1}_{\mathcal{Z}_j}\left(z_{2j},\widehat{z}_{2j}\right)-\vec{\nu}_3^\top\left(W_2\right)_j z_{2j}+\nu_{2j}\widehat{z}_{2j}\right)\right)-\sum_{i=1}^{2}\vec{\nu}_{i+1}^\top\vec{b}_i\Bigg]\\
&=\max_{\vec{\nu}}\left[-\mathbf{1}^*_{B_\varepsilon}(W_1^\top\vec{\nu}_2)+\sum_{j=1}^{n_2}-\mathbf{1}^*_{\mathcal{Z}_j}\left(\vec{\nu}_3^\top\left(W_2\right)_j,-\nu_{2j}\right)-\sum_{i=1}^{2}\vec{\nu}_{i+1}^\top\vec{b}_i\right]\\
&\text{s.t. }\vec{\nu}_3=-\vec{c}
\end{aligned}$$

7 Training a Robust Classifier

In this section, we describe the process of upper-bounding the loss function, the motivation behind this approach, and the methodology used to achieve it. This technique is crucial for enabling efficient robust optimization, particularly when training deep neural networks that are provably robust to adversarial examples.

The primary motivation behind upper-bounding the loss function is to facilitate the training of deep nonlinear classifiers in a robust optimization framework. In the context of adversarial attacks, we aim to minimize the worst-case loss due to some ϵ-perturbation of the original training input. By upper-bounding the hard loss function with a more tractable form, we can leverage standard gradient descent techniques to train a model that is significantly more robust to adversarial perturbations compared to those trained using the original loss function L.

7.1 Monotonic Loss Functions

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 $L\left(\vec{y},\vec{y}_{\text{true}}\right)\leq L\left(\vec{y}',\vec{y}_{\text{true}}\right)$

7.2 Translation-invariant Loss Functions

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

$$L\left(\vec{y},\vec{y}_{\text{true}}\right)=L\left(\vec{y}-a\mathbf{1},\vec{y}_{\text{true}}\right)$$

7.3 Upper Bounding

The upper bounding technique displayed in this section is generalized to a multi-layer deep neural classifier where $\hat{z}_k$ represents the output of the last layer.

We consider a monotonic, translation-invariant multi-class loss function $L : \mathbb{R}^{|y|} \times \mathbb{R}^{|y|} \to \mathbb{R}$. For any data point (x, y) and $\epsilon > 0$, we can upper-bound the worst-case adversarial loss as follows:

We start by expressing the loss of the worst-case adversarial attack using the adversarial polyptote:

$$\max_{\|\Delta\|_\infty \leq \epsilon} L\left(f_\theta(x+\Delta), y\right) = \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L\left(\hat{z}_k, y\right)$$

We now apply a mixture of the translation-invariance and monotonicity of the loss function. Since $L(x, y) \leq L(x - a1, y)$ for all a, we can re-write the worst-case adversarial loss as follows:

$$\begin{aligned}\max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L\left(\hat{z}_k, y\right) &\leq \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L\left(\hat{z}_k - \left(\hat{z}_k\right)_y 1, y\right) \\ &= \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L\left(\left(I - \mathbf{e}_y 1^T\right) \hat{z}_k, y\right) \\ &= \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L\left(C \hat{z}_k, y\right)\end{aligned}$$

where $C = \left(I - \mathbf{e}_y 1^T\right)$.

Furthermore, since L is a monotone loss function, we can upper bound the loss further by using the element-wise maximum over $\left[C\hat{z}_k\right]_i$ for $i \neq y$, and element-wise minimum for $i = y$. Specifically, we bound it as:

$$\max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} L\left(C \hat{z}_k, y\right) \leq L\left(h\left(\hat{z}_k\right)\right)$$

Where C_i is the i th row of C and $h\left(z_k\right)$ is defined element-wise as:

$$h\left(z_k\right)_i = \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} C_i \hat{z}_k$$

The above expression is equivalent to the adversarial problem in its maximization form. Recall that J is a lower bound (using $c = -C_i$):

$$J_\epsilon\left(x, g_\theta\left(-C_i\right)\right) \leq \min_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} -C_i^T \hat{z}_k$$

By multiplying both sides of the inequality by -1, we get the following upper bound:

$$-J_\epsilon\left(x, g_\theta\left(-C_i\right)\right) \geq \max_{\hat{z}_k \in \mathcal{Z}_\epsilon(x)} C_i^T \hat{z}_k$$

Applying this upper bound to $h\left(z_k\right)_i$, we conclude:

$$h\left(z_k\right)_i \leq -J_\epsilon\left(x, g_\theta\left(-C_i\right)\right)$$

By applying the upper bound to all elements of h, we obtain the final upper bound on the adversarial loss:

$$\max_{\|\Delta\|_\infty \leq \epsilon} L\left(f_\theta(x+\Delta), y\right) \leq L\left(-J_\epsilon\left(x, g_\theta\left(\mathbf{e}_y 1^T - I\right)\right), y\right)$$

Using the derived upper bound, we can formulate an efficient optimization approach for training provably robust deep networks. Given a dataset $(x_i, y_i)_{i=1,\ldots,N}$, we minimize the bound on the worst location (i.e., with the highest loss) in an ϵ-ball around each x_i. The resulting optimization problem can be solved more easily. Consequently, we obtain a network that is guaranteed to be robust to adversarial examples if we achieve low loss.

This methodology provides a foundation for developing provably robust deep networks, an essential step towards addressing the vulnerability of deep learning models to adversarial attacks.

8 Future Work

To provide a better intuition behind how the convex outer bound provably defends against adversarial attacks, we would like to have included a visualization as follows: the user could drag their cursor over the various norms within a defined norm ball that we've seen in the BFGS formulation, as well as the corresponding output in the convex outer bound. This will be worked on in the months to come.

References

[1] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[2] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations*, 2015.

[3] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2014.

[4] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292. PMLR, 2018.

9 Appendix

9.1 Defining Fenchel Conjugates

Throughout this paper, some Fenchel conjugates are leveraged to simplify notation. This section will include the derivations of those 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

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

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.

9.2 Fenchel conjugate of Absolute Value

To better understand the Fenchel conjugate, we consider a scalar example. Suppose $f : \mathbb{R} \to \mathbb{R}$, and $f(x) = |x|$. We can find $f^*(y)$ by casework.

First, we consider the case in which $y < -1$:

$$f^*(y) = \sup_{x} \{xy - |x| \mid x \in \mathbb{R}, y < -1\}$$

$$\implies f^*(y) = \sup_{x} \{xy - f(x) \mid x \in \mathbb{R}, y < -1\}$$

Then as $x \to -\infty$ for some $y < -1$, xy takes on a positive value greater than $|x|$ and $xy - |x|$ approaches ∞. Thus, $f^*(y) = \infty$ for $y < -1$

For $y = -1$, we observe that $xy - |x|$ is precisely 0 as $x \to -\infty$ since xy is effectively $|x|$ for $y = -1$ and a negative x. For $x \to \infty$, on the other hand, we get an increasingly negative value. Thus, $f^*(y) = 0$ for $y = -1$.

We see that for $-1 < y < 1$, xy will only be a fraction of $|x|$ and can never exceed 0 and is strictly equal to 0 when $x = 0$. Thus, $f^*(y)$ is also 0 in this case.

By symmetry, we can say that $f^*(y) = 0$ for $y = 1$ as $x \to \infty$, as well as that $f^*(y) = \infty$ for $y > 1$ as $x \to \infty$

9.3 Fenchel conjugate of L1 Norm

Now, suppose $g : \mathbb{R}^n \to \mathbb{R}$ and $g(\vec{x}) = \|\vec{x}\|_1$. Find $g^*(\vec{y})$.

$$g(x) = \|x\|_{\ell_1} = \sum_{i=1}^{n} |x_i|$$

$$\begin{aligned}
g^*(\vec{y}) &= \sup_{\vec{x}} \left\{ \vec{y}^\top \vec{x} - f(\vec{x}) \mid \vec{x} \in \mathbb{R}^n \right\} \\
&= \sup_{\vec{x}} \left\{ \sum_{i=1}^{n} y_i x_i - \sum_{i=1}^{n} |x_i| \mid \vec{x} \in \mathbb{R}^n \right\} \\
&= \sup_{\vec{x}} \left\{ \sum_{i=1}^{n} (y_i x_i - |x_i|) \mid \vec{x} \in \mathbb{R}^n \right\} \\
&= \sum_{i=1}^{n} \sup_{x_i} \left\{ (y_i x_i - |x_i|) \mid_i \in \mathbb{R} \right\} \\
&= \sum_{i=1}^{n} f^*(y_i)
\end{aligned}$$

Where we have already solved for $f^*(y_i)$ above in part B. If any of the y_i is greater than 1 or less than -1, our $g^*(\vec{y})$ is pushed to infinity, otherwise it is equal to 0.

9.4 Fenchel conjugate of Indicator Functions

Define the indicator function:

$$1_{B_\epsilon(\vec{x})}(\vec{v}) \begin{cases} 0 & \vec{z} \in B_\epsilon(\vec{x}) \\ +\infty & \text{otherwise} \end{cases}$$

Where $\vec{z} \in B_\epsilon(\vec{x})$ if $\vec{z} : \|\vec{z} - \vec{x}\|_\infty \le \epsilon$

Solving for the Fenchel congugate of that indicator function:

$$\begin{aligned}
1_{B_\epsilon(\vec{x})}(\vec{v}) &= \sup_{\vec{z}} \left\{ \vec{v}^\top \vec{z} - 1_{B_\epsilon(\vec{x})}(\vec{z}) \mid \vec{z} \in \mathbb{R}^n \right\} \\
&= \sup_{\vec{z} \in B_\epsilon(\vec{x})} \left\{ \vec{v}^\top \vec{z} \mid \vec{z} \in \mathbb{R}^n \right\} \\
&= \sup_{\vec{z} : \|\vec{z} - \vec{x}\|_\infty \le \epsilon} \left\{ \vec{v}^\top \vec{z} \mid \vec{z} \in \mathbb{R}^n \right\} \\
&= \sup_{\vec{z} : \|\vec{z} - \vec{x}\|_\infty \le \epsilon} \left\{ \vec{v}^\top \vec{z} + \vec{v}^\top \vec{x} - \vec{v}^\top \vec{x} \mid \vec{z} \in \mathbb{R}^n \right\} \\
&= \vec{v}^\top \vec{x} + \sup_{\vec{z} : \|\vec{z} - \vec{x}\|_\infty \le \epsilon} \left\{ \vec{v}^\top (\vec{z} - \vec{x}) \mid \vec{z} \in \mathbb{R}^n \right\} \\
&= \vec{v}^\top \vec{x} + \epsilon \|\vec{v}\|_1
\end{aligned}$$

The last step follows since the L1 and L∞ norms are duals:

9.5 Fenchel conjugate of 1_{z_j}

In this section we will derive the fenchel conjugate of the characteristic function of the set z_j. We approach through a case-by-case basis.

9.5.1 Case 1

We will show that when $l_j \le u_j \le 0$:

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

If $l_j \leq u_j \leq 0 \Rightarrow \widehat{z}_{2j} \leq 0$ and the ReLU constraint is equivalent to fixing $z_{2j} = 0$. Hence, $\widehat{\mathcal{Z}}_j$ is already convex:

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

Re-written as:

$$\mathcal{Z}_j = \{(\nu, \widehat{\nu}) \mid \nu = 0\}$$

With the definition of the characteristic function $\mathbf{1}_S$ for any set S as

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

We define:

$$\mathbf{1}_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) = \begin{cases} 0 & \text{if} \quad (\nu, \widehat{\nu}) \in \mathcal{Z}_j \\ +\infty & \text{otherwise} \end{cases} = \begin{cases} 0 & \text{if } \nu = 0 \\ +\infty & \text{otherwise} \end{cases}$$

We recall the definition of a Fenchel conjugate as:

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

Now, we compute the Fenchel conjugate of $\mathbf{1}_{\mathcal{Z}_j}$:

$$\begin{aligned} \mathbf{1}^*_{\mathcal{Z}_j}(\widehat{v}, -v) &= \sup_{x, \widehat{x}} \left\{ (\widehat{v}, -v)^\top (x, \widehat{x}) - \mathbf{1}_{\mathcal{Z}_j}(x, \widehat{x}) \right\} \\ &= \sup_{x, \widehat{x}} \left\{ \widehat{v}^\top \widehat{x} - v^\top x - \mathbf{1}_{\mathcal{Z}_j}(x, \widehat{x}) \right\} \end{aligned}$$

If $x \notin \mathcal{Z}_j$, then $\mathbf{1}_{\mathcal{Z}_j}(x, \widehat{x}) = \infty$, and the supremum is $-\infty$. Therefore, we can restrict the supremum to $(x, \widehat{x}) \in \mathcal{Z}_j$ with $x = 0$:

$$\begin{aligned} \mathbf{1}^*_{\mathcal{Z}_j}(\widehat{v}, -v) &= \sup_{\widehat{x}} \left\{ (\widehat{v}, -v)^\top (0, \widehat{x}) - \mathbf{1}_{\mathcal{Z}_j}(0, \widehat{x}) \right\} \\ &= \sup_{\widehat{x}} \left\{ (\widehat{v}, -v)^\top (0, \widehat{x}) \right\} \text{ because } \mathbf{1}_{\mathcal{Z}_j}(0, \widehat{x}) = 0 \\ &= \sup_{\widehat{x}} \left\{ \widehat{v}^\top \widehat{x} - v^\top 0 \right\} \\ &= \sup_{\widehat{x}} \left\{ \widehat{v}^\top \widehat{x} \right\} \end{aligned}$$

Now, we can analyze the supremum:

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

This result proves that $\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu)$ satisfies the given condition when $l_j \leq u_j \leq 0$.

9.5.2 Case 2

We now approach the next case, when $0 \leq l_j \leq u_j$:
If $0 \leq l_j \leq u_j$, then $\widehat{z}_j \geq 0$. Therefore, $z_j = \widehat{z}_j$ and $\mathcal{Z}_j$ is already convex:

$$\mathcal{Z}_j = \widehat{\mathcal{Z}}_j = \{(z_j, \widehat{z}_j) \in \mathbb{R} \times \mathbb{R} \mid z_j = \widehat{z}_j\}$$

Re-written as:

$$\mathcal{Z}_j = \{(\nu, \widehat{\nu}) \mid \nu = \widehat{\nu}\}$$

Therefore, we have:

$$\mathbf{1}_{\mathcal{Z}_j}(\nu, \widehat{\nu}) = \begin{cases} 0 & \text{if } (\nu, \widehat{\nu}) \in \mathcal{Z}_j \\ +\infty & \text{otherwise} \end{cases} = \begin{cases} 0 & \text{if } \nu = \widehat{\nu} \\ +\infty & \text{otherwise} \end{cases}$$

We will now analyze $\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu)$

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) = \sup_{x, \widehat{x}} \left\{ (\widehat{\nu}, -\nu)^\top (x, \widehat{x}) - \mathbf{1}_{\mathcal{Z}_j}(x, \widehat{x}) \right\}$$

If $x \neq \widehat{x}$, then $(x, \widehat{x}) \notin \mathcal{Z}_j$, and $\mathbf{1}_{\mathcal{Z}_j}(x, \widehat{x}) = +\infty$:

$$(\widehat{\nu}, -\nu)^\top (x, \widehat{x}) - \mathbf{1}_{\mathcal{Z}_j}(x, \widehat{x}) = -\infty$$

We can upper-bound by restricting $(x, \widehat{x}) \in \mathcal{Z}_j \Rightarrow x = \widehat{x}$:

$$\begin{aligned} \mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) &= \sup_{x} \left\{ (\widehat{\nu}, -\nu)^\top (x, x) - \mathbf{1}_{\mathcal{Z}_j}(x, x) \right\} \\ &= \sup_{x} \left\{ (\widehat{\nu}, -\nu)^\top (x, x) \right\} \text{ because } \mathbf{1}_{\mathcal{Z}_j}(x, x) = 0 \\ &= \sup_{x} \left\{ \widehat{\nu}^\top x - \nu^\top x \right\} \\ &= \sup \left\{ (\widehat{\nu} - \nu)^\top x \right\} \\ &= \begin{cases} 0 & \nu = \widehat{\nu} \\ +\infty & \text{otherwise} \end{cases} \end{aligned}$$

Hence, when $0 \leq l_j \leq u_j$:

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

9.5.3 Case 3

Finally, we approach the next case, when $l_j \leq 0 \leq u_j$:

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

Show that when $l_j \leq 0 \leq u_j$:

$$1^*_{\mathcal{Z}_j}(\hat{\nu}, -\hat{\nu}) \leq \begin{cases} \text{ReLu}\left(-l_j \nu\right) & \nu = \frac{\hat{\nu} u_j}{u_j - l_j} \\ +\infty & \text{otherwise} \end{cases}$$

If $l_j \leq 0 \leq u_j$: $\widehat{\mathcal{Z}}_j$ is no longer convex. Examining this set visually, it is clear that its convex hull is a triangle, given by

$$\begin{aligned} \mathcal{Z}_j = \{&(z_{2j}, \widehat{z}_{2j}) \in \mathbb{R} \times \mathbb{R} \mid \\ &z_{2j} \geq 0 \wedge z_{2j} \geq \widehat{z}_{2j} \wedge \\ &- u_j \widehat{z}_{2j} + (u_j - l_j) z_{2j} \leq -u_j l_j\} \end{aligned} \tag{12}$$

Note that the inequality

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

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

Similar to the previous parts, we can reason that the supremum will be achieved when the characteristic function outputs a value of 0 ($(z_j, \widehat{z_j}) \in \mathcal{Z}_j$):

$$\begin{aligned} 1^*_{\mathcal{Z}_j}(\hat{\nu}, -\nu) = &\sup_{x,\hat{x}} \left\{ (\hat{\nu}, -\nu)^\top (x, \hat{x}) - 1_{z_j}(x, \hat{x}) \right\} \\ &\text{if } x \notin Z_j \text{ then } 1_{z_j}(x, \hat{x}) = \infty \\ &(\hat{\nu}, -\nu)^\top (x, \hat{x}) - 1_{z_j}(x, \hat{x}) = -\infty \end{aligned}$$

Now, we want to find an upper bound for $\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu)$ given the following constraints:

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) = \sup_{(x,\widehat{x}) \in \mathcal{Z}_j} \left\{ (\widehat{\nu}, -\nu)^\top (x, \widehat{x}) \right\}$$

Since the optimum of a linear program can always be attained at one of the vertices of the feasible polytope, we only need to consider the vertices of the triangle in $\mathcal{Z}_j$. These vertices are $(l_j, 0)$, $(0, 0)$, and (u_j, u_j). Let's evaluate the inner product at each vertex:

At $(0, 0)$:

$$(\widehat{\nu}, -\nu)^\top (0, 0) = 0$$

Now, to analyze the $(l_j, 0)$ and (u_j, u_j) that are vertices of the feasible region we will look at the whole line between the points $(l_j, 0)$ and (u_j, u_j) which trivially include the points as well:

$$-u_j \widehat{x} + (u_j - l_j)\, x = -u_j l_j$$

Solving for x:

$$x = \frac{u_j \widehat{x} - u_j l_j}{u_j - l_j}$$

Now, let's plug the expression for x into the supremum:

$$(\widehat{\nu}, -\nu)^\top (x, \widehat{x}) = (\widehat{\nu}, -\nu)^\top \left(\frac{u_j \widehat{x} - u_j l_j}{u_j - l_j}, \widehat{x} \right)$$

Expanding the inner product, we get:

$$(\widehat{\nu}, -\nu)^\top \left(\frac{u_j \widehat{x} - u_j l_j}{u_j - l_j}, \widehat{x} \right) = \widehat{\nu} \frac{u_j \widehat{x} - u_j l_j}{u_j - l_j} - \nu \widehat{x}$$

Now, we want to find the value of $\widehat{x}$ that maximizes this expression. To do this, we can take the derivative with respect to $\widehat{x}$ and set it equal to zero:

$$\frac{d}{d\widehat{x}} \left(\widehat{\nu} \frac{u_j \widehat{x} - u_j l_j}{u_j - l_j} - \nu \widehat{x} \right) = 0$$

Calculating the derivative, we get:

$$\frac{d}{d\widehat{x}}\left(\widehat{\nu}\frac{u_j\widehat{x} - u_j l_j}{u_j - l_j} - \nu\widehat{x}\right) = \frac{\widehat{\nu}u_j}{u_j - l_j} - \nu$$

Setting the derivative equal to zero:

$$\frac{\widehat{\nu}u_j}{u_j - l_j} - \nu = 0$$

Solving for ν, we obtain:

$$\nu = \frac{\widehat{\nu}u_j}{u_j - l_j}$$

Now, we substitute this value of ν into the expression for the inner product to get the upper bound:

$$\begin{aligned}(\widehat{\nu}, -\nu)^\top \left(\frac{u_j\widehat{x} - u_j l_j}{u_j - l_j}, \widehat{x}\right) &= \widehat{\nu}\frac{u_j\widehat{x} - u_j l_j}{u_j - l_j} - \frac{\widehat{\nu}u_j}{u_j - l_j}\widehat{x} \\ &= \frac{\hat{\nu}u_j\hat{x}}{u_j - l_j} - \frac{\hat{\nu}u_j\hat{x}}{u_j - l_j} - \frac{\hat{\nu}u_j l_j}{u_j - l_j} \\ &= -\frac{\hat{\nu}u_j l_j}{u_j \cdot l_j} = -\nu l_j \end{aligned} \tag{13}$$

Hence, on the line that represents the upper bound of the triangle, and when the derivative of the Fenchel Conjugate is set to zero:

$$\nu = \frac{\widehat{\nu}u_j}{u_j - l_j}$$

and

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) = -\nu l_j$$

Given that $-l_j v \leq \text{ReLU}(-l_j v)$:

$$\mathbf{1}^*_{\mathcal{Z}_j}(\widehat{\nu}, -\nu) \leq \begin{cases}\text{ReLU}\left(-l_j\nu\right) & \nu = \frac{\widehat{\nu}u_j}{u_j - l_j} \\ +\infty & \text{otherwise}\end{cases}$$

Thus, we have shown that when $l_j \leq 0 \leq u_j$, the given inequality holds.

9.6 Finding ReLU bounds $\vec{u}$ and l_j

The dual problem from above assumed we have $\vec{u}$ and $\vec{l}$ in order to compute the relaxation on the ReLU non-linearity. To compute these bounds, we define the following notation for any matrix W with rows $\vec{w}_1^\top, \ldots, \vec{w}_k^\top$:

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

From our setup in section 6, we define the upperbound u_j of $\hat{z_{2j}}$ as

$$\begin{aligned} u_j = \max \quad & \hat{z}_{2j} \\ \text{s.t.} \quad & \hat{z}_{2j} = \left(W_1\vec{z}_1 + \vec{b}_1\right) j \\ & \|\vec{z}_1 - \vec{x}\|\infty \leq \epsilon \end{aligned} \tag{14}$$

$$\begin{aligned} \implies u_j = \max \quad & \left(W_1\vec{z}_1 + \vec{b}_1\right)j \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies u_j = \max \quad & \vec{w}_{1,j}^{\top}\vec{z}_1 + b_{1,j} \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies u_j = \vec{w}_{1,j}^{\top}\vec{x} + \max \quad & \vec{w}_{1,j}^{\top}\vec{z}_1 + b_{1,j} - \vec{w}_{1,j}^{\top}\vec{x} \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies u_j = \vec{w}_{1,j}^{\top}\vec{x} + b_{1,j} + \max \quad & \vec{w}_{1,j}^{\top}\left(\vec{z}_1 - \vec{x}\right) \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies u_j = \vec{w}_{1,j}^{\top}\vec{x} + b_{1,j} + \max \quad & \vec{w}_{1,j}^{\top}\left(\vec{z}_1 - \vec{x}\right) \\ \text{s.t.} \quad & \|\vec{z}_1 - \vec{x}\|_\infty \leq \epsilon \end{aligned}$$

$$\implies u_j = \vec{w}_{1,j}^{\top}\vec{x} + b_{1,j} + \epsilon\|\vec{w}_{1,j}\|_1$$

Where the second to last implication follows from l_∞ and l_1 being dual norms. We can generalize for all u_j in $\vec{u}$:

$$\vec{u} = W_1\vec{x} + \vec{b}_1 + \epsilon\|W_1\|_{:1} \tag{15}$$

as well as all l_j in $\vec{l}$

$$\vec{l} = W_1\vec{x} + \vec{b}_1 - \epsilon\|W_1\|_{:1} \tag{16}$$

where we use $-\epsilon$ since an arbitrary l_j will be the minimum of our original formulation shown in (14) instead of maximum.

More thoroughly,

$$\begin{aligned} l_j = \min \quad & \hat{z}2j \\ \text{s.t.} \quad & \hat{z}2j = \left(W_1\vec{z}_1 + \vec{b}_1\right)j \quad |\vec{z}_1 - \vec{x}|_\infty \leq \epsilon \end{aligned} \tag{17}$$

$$\begin{aligned} \implies l_j = \min \quad & \vec{w}_{1,j}^{\top}\vec{z}_1 + b_{1,j} \\ \text{s.t.} \quad & |\vec{z}_1 - \vec{x}|_\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies l_j = \min \quad & \vec{w}_{1,j}^{\top}\vec{x} + \vec{w}_{1,j}^{\top}\left(\vec{z}_1 - \vec{x}\right) + b_{1,j} - \vec{w}_{1,j}^{\top}\vec{x} \\ \text{s.t.} \quad & |\vec{z}_1 - \vec{x}|_\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies l_j = \vec{w}_{1,j}^{\top}\vec{x} + \min \quad & \vec{w}_{1,j}^{\top}\left(\vec{z}_1 - \vec{x}\right) + b_{1,j} \\ \text{s.t.} \quad & |\vec{z}_1 - \vec{x}|_\infty \leq \epsilon \end{aligned}$$

$$\begin{aligned} \implies l_j = \vec{w}_{1,j}^{\top}\vec{x} + \min \quad & \vec{w}_{1,j}^{\top}\left(\vec{z}_1 - \vec{x}\right) \\ \text{s.t.} \quad & |\vec{z}_1 - \vec{x}|\infty \leq \epsilon - b1, j/|\vec{w}_{1,j}|_1 \end{aligned}$$

$$\implies l_j = \vec{w}_{1,j}^{\top}\vec{x} - \epsilon|\vec{w}_{1,j}|1 + b1, j$$