A. Hardness

In this section we show that finding the minimum adversarial distortion with a certified approximation ratio is hard. We first introduce some basic definitions and theorems in Section A.1. We provide some backgrounds about in-approximability reduction in Section A.2. Section A.3 gives a warmup proof for boolean case and then Section A.4 provides the proof of our main hardness result (for network with real inputs).

A.1. Definitions

We provide some basic definitions and theorems in this section. First, we define the classic 3SAT problem.

**Definition A.1 (3SAT problem).** Given \(n\) variables and \(m\) clauses in a conjunctive normal form CNF formula with the size of each clause at most 3, the goal is to decide whether there exists an assignment to the \(n\) Boolean variables to make the CNF formula to be satisfied.

For the 3SAT problem in Definition A.1, we introduce the Exponential Time Hypothesis (ETH), which is a common concept in complexity field.

**Hypothesis A.2 (Exponential Time Hypothesis (ETH) (Impagliazzo et al., 1998)).** There is a \(\delta > 0\) such that the 3SAT problem defined in Definition A.1 cannot be solved in \(O(2^{\delta n})\) time.

ETH had been used in many different problems, e.g. clustering (Alon et al., 2018; Cohen-Addad et al., 2018), low-rank approximation (Razenshteyn et al., 2016; Song et al., 2017a;b; 2018). For more details, we refer the readers to a survey (Lokshtanov et al., 2013).

Then we define another classical question in complexity theory, the SET-COVER problem, which we will use in our proof. The exact SET-COVER problem is one of Karp’s 21 NP-complete problems known to be NP-complete in 1972:

**Definition A.3 (SET-COVER).** The inputs are \(U, S; U = \{1, 2, \cdots, n\}\) is a universe, \(P(U)\) is the power set of \(U\), and \(S = \{S_1, \cdots, S_m\} \subseteq P(U)\) is a family of subsets, \(\cup_{j \in [m]} S_j = U\). The goal is to give a YES/NO answer to the following decision problem:

**Does there exist a set-cover of size \(t\), i.e., \(\exists C \subseteq [m], \text{such that } \cup_{j \in C} S_j = U \text{ with } |C| = t?**

Alternatively, we can also state the problem as finding the minimum set cover size \(t_0\), via a binary search on \(t\) using the answers of the decision problem in A.3. The Approximate SET-COVER problem is defined as follows.

**Definition A.4 (Approximate SET-COVER).** The inputs are \(U, S; U = \{1, 2, \cdots, n\}\) is a universe, \(P(U)\) is the power set of \(U\), and \(S = \{S_1, \cdots, S_m\} \subseteq P(U)\) is a family of subsets, \(\cup_{j \in [m]} S_j = U\). The goal is to distinguish between the following two cases:

(I): There exists a small set-cover, i.e., \(\exists C \subseteq [m], \text{such that } \cup_{j \in C} S_j = U \text{ with } |C| \leq t\).

(II): Every set-cover is large, i.e., every \(C \subseteq [m]\) with \(\cup_{j \in C} S_j = U\) satisfies that \(|C| > \alpha t\), where \(\alpha > 1\).

An oracle that solves the Approximate SET-COVER problem outputs an answer \(t_U \geq t_0\) but \(t_U \leq \alpha t_0\) using a binary search, where \(t_U\) is an upper bound of \(t_0\) with a guaranteed approximation ratio \(\alpha\). For example, we can use a greedy (rather than exact) algorithm to solve the SET-COVER problem, which cannot always find the smallest size of set cover \(t_0\), but the size \(t_U\) given by the greedy algorithm is at most \(\alpha\) times as large as \(t_0\).

In our setting, we want to investigate the hardness of finding the lower bound with a guaranteed approximation ratio, but an approximate algorithm for SET-COVER gives us an upper bound of \(t_0\) instead of an lower bound of \(t_0\). However, in the following proposition, we show that finding an lower bound with an approximation ratio of \(\alpha\) is as hard as finding an upper bound with an approximation ratio of \(\alpha\).

**Proposition A.5.** Finding a lower bound \(t_L\) for the size of the minimal set-cover (that has size \(t_0\)) with an approximation ratio \(\alpha\) is as hard as finding an upper bound \(t_U\) with an approximation ratio \(\alpha\).

**Proof.** If we find a lower bound \(t_L\) with \(t_L \leq t_U \leq t_0\), by multiplying both sides by \(\alpha\), we also find an upper bound \(t_U = \alpha t_L\) which satisfies that \(t_0 \leq t_U \leq \alpha t_0\). So finding an lower bound with an approximation ratio \(\alpha\) is at least as hard as finding an upper bound with an approximation ratio \(\alpha\). The converse is also true. \(\square\)

SET-COVER is a well-studied problem in the literature. Here we introduce a theorem from (Raz & Safra, 1997; Alon et al., 2006; Dinur & Steurer, 2014) which implies the hardness of approximating SET-COVER.
We now formally define our neural network robustness verification problems.

**Definition A.7 (ROBUST-NET(\(\mathbb{R}\))).** Given an \(n\) hidden nodes ReLU neural network \(F(x) : \mathbb{R}^d \to \mathbb{R}\) where all weights are fixed, for a query input vector \(x \in \mathbb{R}^d\) with \(F(x) \leq 0\). The goal is to give a YES/NO answer to the following decision problem:

\[
\text{Does there exist } a \ y \text{ with } ||x - y||_1 \leq r \text{ such that } F(y) > 0?\]

With an oracle of the decision problem available, we can figure out the smallest \(r\) (defined as \(r_0\)) such that there exists a vector \(y\) with \(||x - y||_1 \leq r\) and \(F(y) > 0\) via a binary search.

We also define a binary variant of the ROBUST-NET problem, denoted as ROBUST-NET(\(\mathbb{B}\)). The proof for this variant is more straightforward than the real case, and will help the reader understand the proof for the real case.

**Definition A.8 (ROBUST-NET(\(\mathbb{B}\))).** Given an \(n\) hidden nodes ReLU neural network \(F(x) : \{0,1\}^d \to \{0,1\}\) where weights are all fixed, for a query input vector \(x \in \{0,1\}^d\) with \(F(x) = 0\). The goal is to give a YES/NO answer to the following decision problem:

\[
\text{Does there exist } a \ y \text{ with } ||x - y||_1 \leq r \text{ such that } F(y) = 1?\]

Then, we define the approximate version of our neural network robustness verification problems:

**Definition A.9 (Approximate ROBUST-NET(\(\mathbb{B}\))).** Given an \(n\) hidden nodes ReLU neural network \(F(x) : \{0,1\}^d \to \{0,1\}\) where weights are all fixed, for a query input vector \(x \in \{0,1\}^d\) with \(F(x) = 0\). The goal is to distinguish the following two cases:

(I): There exists a point \(y\) such that \(||x - y||_1 \leq r\) and \(F(y) = 1\).

(II): For all \(y\) satisfies \(||x - y||_1 \leq \alpha r\), the \(F(y) = 0\), where \(\alpha > 1\).

**Definition A.10 (Approximate ROBUST-NET(\(\mathbb{R}\))).** Given an \(n\) hidden nodes ReLU neural network \(F(x) : \mathbb{R}^d \to \mathbb{R}\) where weights are all fixed, for a query input vector \(x \in \mathbb{R}^d\) with \(F(x) \leq 0\). The goal is to distinguish the following two cases:

(I): There exists a point \(y\) such that \(||x - y||_1 \leq r\) and \(F(y) > 0\).

(II): For all \(y\) satisfies \(||x - y||_1 \leq \alpha r\), the \(F(y) \leq 0\), where \(\alpha > 1\).

As an analogy to SET-COVER, an oracle that solves the Approximate ROBUST-NET(\(\mathbb{R}\)) problem can output an answer \(r \geq r_0\) but \(r \leq \alpha r_0\), which is an upper bound of \(r_0\) with a guaranteed approximation ratio \(\alpha\). With a similar statement as in Proposition A.5, if we divide the answer \(r\) by \(\alpha\), then we get a lower bound \(r' = \frac{r}{\alpha}\) where \(r' \geq \frac{r_0}{\alpha}\), which is a lower bound with a guaranteed approximation ratio. If we can solve Approximate ROBUST-NET(\(\mathbb{R}\)), we can get a lower bound with a guaranteed approximation ratio, which is the desired goal of our paper.

### A.2. Background of the PCP theorem

The famous Probabilistically Checkable Proofs (PCP) theorem is the cornerstone of the theory of computational hardness of approximation, which investigates the inherent difficulty in designing efficient approximation algorithms for various optimization problems. The formal definition can be stated as follows,

**Theorem A.11 ((Arora & Safra, 1998; Arora et al., 1998)).** Given a SAT formula \(\phi\) of size \(n\) we can in time polynomial in \(n\) construct a set of \(M\) tests satisfying the following:

(I) : Each test queries a constant number \(d\) of bits from a proof, and based on the outcome of the queries it either accepts or reject \(\phi\).

(II) : (Yes Case / Completeness) If \(\phi\) is satisfiable, then there exists a proof so that all tests accept \(\phi\).

(III) : (No Case / Soundness) If \(\phi\) is not satisfiable, then no proof will cause more than \(M/2\) tests to accept \(\phi\).

Note that PCP kind of reduction is slightly different from NP reduction, for more examples (e.g. maximum edge biclique, sparsest cut) about how to use PCP theorem to prove inapproximability results, we refer the readers to (Ambühl et al., 2011).

---

3https://en.wikipedia.org/wiki/PCP_theorem
A.3. Warm-up

We state our hardness result for ROBUST-NET(\(\mathbb{R}\)) (boolean inputs case) in this section. The reduction procedure for network with real inputs is more straightforward and easier to understand than the real inputs case.

**Theorem A.12.** Unless NP = P, there is no polynomial time algorithm to give a \((1 - o(1)) \ln n\)-approximation to ROBUST-NET(\(\mathbb{B}\)) problem (Definition A.9) with \(n\) nodes.

**Proof.** Consider a set-cover instance, let \(S\) denote a set of sets \(\{S_1, S_2, \ldots, S_d\}\) where \(S_j \subseteq [n], \forall j \in [d]\).

For each set \(S_j\) we create an input node \(u_j\). For each element \(i \in [n]\), we create a hidden node \(v_i\). For each \(i \in [n]\) and \(j \in [d]\), if \(i \in S_j\), then we connect \(u_j\) and \(v_i\). We also create an output node \(w\), for each \(i \in [n]\), we connect node \(v_i\) and node \(w\).

Let \(1_{i \in S_j}\) denote the indicator function that it is 1 if \(i \in S_j\) and 0 otherwise. Let \(T_i\) denote the set that \(T_i = \{j \mid i \in S_j, \forall j \in [d]\}\). For each \(i \in [n]\), we define an activation function \(\phi_i\) satisfies that

\[
\phi_i = \begin{cases} 
1, & \text{if } \sum_{j \in T_i} u_j \geq 1, \\
0, & \text{otherwise}.
\end{cases}
\]

Since \(u_j \in \{0, 1\}\), \(\phi_i\) can be implemented in this way using ReLU activations:

\[
\phi_i = 1 - \max \left(0, 1 - \sum_{j \in T_i} u_j\right).
\]

Note that \(\sum_{j=1}^d 1_{i \in S_j} = \sum_{j=1}^d u_j\), because \(u_j = 1\) indicates choosing set \(S_j\) and \(u_j = 0\) otherwise.

For final output node \(w\), we define an activation function \(\psi\) satisfies that

\[
\psi = \begin{cases} 
1, & \text{if } \sum_{i=1}^n v_i \geq n, \\
0, & \text{otherwise}.
\end{cases}
\]

Since \(v_i \in [n]\), \(\psi\) can be implemented as

\[
\psi = \max \left(0, \sum_{i=1}^n v_i - n + 1\right).
\]

We use vector \(x\) to denote \(\{0\}^d\) vector and it is easy to see that \(F(x) = 0\). Let \(\alpha > 1\) denote a fixed parameter. Also, we have \(F(y) > 0\) if and only if \(C = \{j \mid y_j = 1\}\) is a set-cover. According to our construction, we can have the following two claims,

**Claim A.13 (Completeness).** If there exists a set-cover \(C \subseteq [d]\) with \(\cup_{j \in C} S_j = [n]\) and \(|C| \leq r\), then there exists a point \(y \in \{0, 1\}^d\) such that \(\|x - y\|_1 \leq r\) and \(F(y) > 0\).

**Claim A.14 (Soundness).** If for every \(C \subseteq [d]\) with \(\cup_{j \in C} S_j = U\) satisfies that \(|C| > \alpha \cdot t\), then for all \(y \in \{0, 1\}^d\) satisfies that \(\|x - y\|_1 \leq \alpha r\), \(F(y) \leq 0\) holds.

Therefore, using Theorem A.11, Theorem A.6, Claim A.13 and Claim A.14 completes the proof. \(\square\)

A.4. Main result

With the proof for ROBUST-NET(\(\mathbb{R}\)) as a warm-up, we now prove our main hardness result for ROBUST-NET(\(\mathbb{R}\)) in this section.

**Theorem A.15.** Unless NP = P, there is no polynomial time algorithm to give an \((1 - o(1)) \ln n\)-approximation to ROBUST-NET(\(\mathbb{R}\)) problem (Definition A.10) with \(n\) hidden nodes.
We use vector 
\[ x \in \mathbb{R}^n \]
and node \( u \) as follows.

According to our construction, we can have the following two claims.

1. **Claim**: The set-cover, we know that
   \[ \mathcal{C} \subseteq \mathbb{R}^n \]
   can be implemented in the following way.

   For the neuron it as \( e_n \), we define the adversarial point is found.

   \[ y = \alpha(1 - \delta) \]

   Then \( F(y) = w = \min_{i \in [n]} v_i \]

   We use vector \( x \) to denote \( \{0\}^d \) vector and it is easy to see that \( F(x) = -\delta < 0 \). Let \( \alpha > 1 \) denote a fixed parameter.

   According to our construction, we can have the following two claims.

   **Claim A16** (Completeness). If there exists a set-cover \( C \subseteq [d] \) with \( \bigcup_{x \in C} S_x = [n] \) and \( |C| \leq r \), then there exists a point \( y \in [0,1]^d \) such that \( \|x - y\|_1 \leq r \) and \( F(y) > 0 \).

   **Proof.** Without loss of generality, we let the set cover to be \( \{S_1, S_2, ..., S_r\} \). Let \( y_1 = y_2 = \cdots = y_r = 1 \)

   \[ y_{r+1} = y_{r+2} = \cdots = y_d = 0 \]

   By the definition of \( t_j \), we have \( t_1 = t_2 = \cdots = t_r = \delta \). Since \( \{S_1, S_2, ..., S_r\} \) is a set-cover, we know that \( v_i = \delta \) for all \( i \in [n] \). Then \( F(y) = w = \min_{i \in [n]} v_i = \delta > 0 \). Since we also have \( \|y\|_1 = r \), the adversarial point is found.

   **Claim A17** (Soundness). If for every \( C \subseteq [d] \) with \( \bigcup_{x \in C} S_x = U \) satisfies that \( |C| > \alpha \cdot r \), then for all \( y \in [0,1]^d \) satisfies that \( \|x - y\|_1 \leq \alpha r(1 - 1/d) \), \( F(y) \leq 0 \) holds.

   **Proof.** Proof by contradiction. We assume that there exists \( y \) such that \( F(y) > 0 \) and \( \|y\|_1 \leq \alpha r(1 - 1/d) \). Since \( F(y) > 0 \), we have for all \( i, v_i > 0 \). Thus there exists \( j \in T_i \) such that \( t_j > 0 \). Let \( \pi : [n] \to Q \) denote a mapping \( (Q \subseteq [d] \) will be decided later). This means that for each \( i \in [n] \), there exists \( j \in T_i \), such that \( 1 - \delta < y_j \leq 1 \), and we let \( \pi(i) \) denote that \( j \).

   We define set \( Q \subseteq [d] \) as follows

   \[ Q = \{j \mid \exists i \in [n], \text{ s.t. } \pi(i) = j \in T_i \text{ and } t_j > 0\}. \]

   Since \( \sum_{j \in [d]} |y_j| = \|y\|_1 \leq \alpha r(1 - 1/d) \), we have

   \[ \sum_{j \in Q} |y_j| \leq \sum_{j \in [d]} |y_j| \leq \alpha r(1 - 1/d) \]
where the first step follows by $|Q| \leq d$.

Because for all $j \in Q$, $|y_j| > 1 - \delta = 1 - 1/d$, we have

$$|Q| \leq \frac{\alpha r (1 - 1/d)}{(1 - 1/d)} = \alpha \cdot r.$$ 

So \( \{S_j\}_{j \in Q} \) is a set-cover with size less than or equal to \( \alpha \cdot r \), which is a contradiction.

Therefore, using Theorem A.11, Theorem A.6, Claim A.16 and Claim A.17 completes the proof.

By making a stronger assumption of ETH, we can have the following stronger result which excludes all \( 2^{o(n^c)} \) time algorithms, where \( c > 0 \) is some fixed constant:

**Corollary A.18.** Assuming Exponential Time Hypothesis (ETH, see Hypothesis A.2), there is no \( 2^{o(n^c)} \) time algorithm that gives a \( (1 - o(1)) \ln n \)-approximation to ROBUST-NET problem with \( n \) hidden nodes, where \( c > 0 \) is some fixed constant.

**Proof.** It follows by the construction in Theorem A.15 and (Moshkovitz, 2012a;b).

Note that in (Moshkovitz, 2012a), an additional conjecture, Projection Games Conjecture (PGC) is required for the proof, but the result was improved in (Moshkovitz, 2012b) and PGC is not a requirement any more.
B. Proof of Theorem 3.5

For a $m$-layer ReLU network, assume we know all the pre-ReLU activation bounds $l^{(k)}$ and $u^{(k)}$, $\forall k \in [m-1]$ for a $m$-layer ReLU network and we want to compute the bounds of the the $j$ th output at $m$ th layer.

The $j$ th output can be written as

$$f_j(x) = \sum_{k=1}^{m-1} W_{j,k}^{(m)}[\phi_{m-1}(x)]_k + b_j^{(m)}, \quad (15)$$

$$= \sum_{k=1}^{m-1} W_{j,k}^{(m)} \sigma(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}) + b_j^{(m)}, \quad (16)$$

$$= \sum_{k \in I_{k,m-1}^+} W_{j,k}^{(m)} \sigma(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}) + b_j^{(m)}.$$  \( \tag{17} \)

For neurons belonging to category (i), i.e., $k \in I_{m-1}^+$,

$$\sigma(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}) = W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}.$$  \( \tag{18} \)

For neurons belonging to category (ii), i.e., $k \in I_{m-1}^-$,

$$\sigma(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}) = 0.$$

Finally, for neurons belonging to Category (iii), i.e., $k \in I_{m-1}$, we bound their outputs. If we adopt the linear upper and lower bounds in (1) and let $d_k^{(m-1)} := \frac{x_k^{(m-1)} - l_k^{(m-1)}}{u_k^{(m-1)} - l_k^{(m-1)}}$, we have

$$d_k^{(m-1)}(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}) \leq \sigma(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}) \leq d_k^{(m-1)}(W_{k,:}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)} - l_k^{(m-1)}). \quad (18)$$

B.1. Upper bound

The goal of this section is to prove Lemma B.1.

**Lemma B.1** (Upper bound with explicit function). Given an $m$-layer ReLU neural network function $f : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_m}$, parameters $p, \epsilon$, there exists two explicit functions $f^U : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_m}$ and $f^L : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_m}$ (see Definition 3.4) such that $\forall j \in [n_m]$,

$$f_j(x) \leq f_j^U(x), \forall x \in B_p(x_0, \epsilon).$$

Notice that (18) can be used to construct an upper bound and lower bound of $f_j(x)$ by considering the signs of the weights $W_{j,k}^{(m)}$. Let $f_j^{U,m-1}(x)$ be an upper bound of $f_j(x)$; $f_j^{L,m-1}(x)$ can be constructed by taking the right-hand-side (RHS) of
(18) if $W_{j,k}^{(m)} > 0$ and taking the left-hand-side (LHS) of (18) if $W_{j,k}^{(m)} < 0$:

$$f_j^{U,m-1}(x) = \sum_{k \in I_{m-1}^+} W_{j,k}^{(m)} \phi_{m-2}(x) + b_k^{(m-1)}$$

$$= \sum_{k \in I_{m-1}^+} W_{j,k}^{(m)} \phi_{m-2}(x) + b_k^{(m-1)}$$

$$+ \sum_{k \in I_{m-1}^+} W_{j,k}^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)} - l_k^{(m-1)}$$

$$+ \sum_{k \in I_{m-1}^+} W_{j,k}^{(m)} d_k^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)} + b_j^{(m)}$$

$$= \sum_{k \in I_{m-1}^-} W_{j,k}^{(m)} d_k^{(m-1)} \phi_{m-2}(x) + b_k^{(m-1)}$$

$$+ \sum_{k \in I_{m-1}^-} W_{j,k}^{(m)} d_k^{(m)} \phi_{m-2}(x) + b_k^{(m)}$$

where we set $d_k^{(m-1)} = 1$ for $k \in I_{m-1}^+$ and set $d_k^{(m-1)} = 0$ for $k \in I_{m-1}^-$ from (19) to (20) and collect the constant terms (independent of $x$) in the parenthesis from (20) to (21).

If we let $A^{(m-1)} = W^{(m)} D^{(m-1)}$, where $D^{(m-1)}$ is a diagonal matrix with diagonals being $d_k^{(m-1)}$, then we can rewrite $f_j^{U,m-1}(x)$ into the following:

$$f_j^{U,m-1}(x) = \sum_{k=1}^{n_{m-1}} A_{j,k}^{(m-1)} W_{k,:}^{(m)} \phi_{m-2}(x) + \left( A_{j,:}^{(m-1)} b^{(m-1)} - A_{j,:}^{(m-1)} T_{:,j}^{(m-1)} + b_j^{(m)} \right)$$

$$= \sum_{k=1}^{n_{m-1}} A_{j,k}^{(m-1)} W_{k,:}^{(m)} \phi_{m-2}(x) + \left( A_{j,:}^{(m-1)} b^{(m-1)} - A_{j,:}^{(m-1)} T_{:,j}^{(m-1)} + b_j^{(m)} \right)$$

$$= \sum_{k=1}^{n_{m-1}} A_{j,k}^{(m-1)} W_{k,:}^{(m)} \phi_{m-2}(x) + \left( A_{j,:}^{(m-1)} b^{(m-1)} - A_{j,:}^{(m-1)} T_{:,j}^{(m-1)} + b_j^{(m)} \right)$$

$$= \sum_{k=1}^{n_{m-1}} A_{j,k}^{(m-1)} W_{k,:}^{(m)} \phi_{m-2}(x) + \left( A_{j,:}^{(m-1)} b^{(m-1)} - A_{j,:}^{(m-1)} T_{:,j}^{(m-1)} + b_j^{(m)} \right)$$

$$= \sum_{k=1}^{n_{m-1}} A_{j,k}^{(m-1)} W_{k,:}^{(m)} \phi_{m-2}(x) + \left( A_{j,:}^{(m-1)} b^{(m-1)} - A_{j,:}^{(m-1)} T_{:,j}^{(m-1)} + b_j^{(m)} \right)$$

From (21) to (22), we rewrite the summation terms in the parenthesis into matrix-vector multiplications and for each $j \in [n_m]$ let

$$T_{k,j}^{(m-1)} = \begin{cases} t_k^{(m-1)} & \text{if } k \in I_{m-1}^+, A_{j,k}^{(m-1)} > 0 \\ 0 & \text{otherwise} \end{cases}$$

since $0 \leq d_k^{(m-1)} \leq 1$, $W_{j,k}^{(m)} > 0$ is equivalent to $A_{j,k}^{(m-1)} > 0$.

From (22) to (23), we simply write out the inner product $W_{k,:}^{(m)} \phi_{m-2}(x)$ into a summation form, and from (23) to (24), we exchange the summation order of $k$ and $r$. From (24) to (25), we let

$$W_{j,r}^{(m-1)} = \sum_{k=1}^{n_{m-2}} A_{j,k}^{(m-1)} W_{k,r}^{(m-1)}$$

$$b_j^{(m-1)} = \left( A_{j,:}^{(m-1)} b^{(m-1)} - A_{j,:}^{(m-1)} T_{:,j}^{(m-1)} + b_j^{(m)} \right)$$

Towards Fast Computation of Certified Robustness for ReLU Networks
and now we have (25) in the same form as (15).

Indeed, in (15), the running index is $k$ and we are looking at the $m$ th layer, with weights $W_{j,k}^{(m)}$, activation functions $\phi_{m-1}(x)$ and bias term $b_{j}^{(m)}$; in (25), the running index is $r$ and we are looking at the $m-1$ th layer with equivalent weights $\tilde{W}_{j,r}^{(m-1)}$, activation functions $\phi_{m-2}(x)$ and equivalent bias $\tilde{b}_{j}^{(m-1)}$. Thus, we can use the same technique from (15) to (25) and obtain an upper bound on the $f_{j}^{U,-1}(x)$ and repeat this procedure until obtaining $f_{j}^{U,1}(x)$, where

$$f_{j}(x) \leq f_{j}^{U,-m-1}(x) \leq f_{j}^{U,-m-2}(x) \leq \ldots \leq f_{j}^{U,1}(x).$$

Let the final upper bound $f_{j}^{U}(x) = f_{j}^{U,1}(x)$, and now we have

$$f_{j}(x) \leq f_{j}^{U}(x),$$

where $f_{j}^{U}(x) = [f^{U}(x)]_{j}$,

$$f_{j}^{U}(x) = A_{j}^{(0)} x + b_{j}^{(m)} + \sum_{k=1}^{m-1} A_{j}^{(k)} (b_{k} - T_{r,j}^{(k)})$$

and for $k = 1, \ldots, m-1$,

$$A^{(m-1)} = W^{(m)} D^{(m-1)}, A^{(k-1)} = A^{(k)} W^{(k)} D^{(k-1)},$$

$$D^{(0)} = I_{n_{0}}$$

$$D_{r,r}^{(k)} = \begin{cases} 
\frac{u_{(r)}^{(k)}}{u_{r}^{(k)} - t_{r}^{(k)}} & \text{if } r \in I_{k} \\
1 & \text{if } r \in I_{k}^{+} \\
0 & \text{if } r \in I_{k}^{-}
\end{cases}$$

$$T_{r,j}^{(k)} = \begin{cases} 
1 & \text{if } r \in I_{k}, A_{j,r}^{(k)} > 0 \\
0 & \text{otherwise}
\end{cases}$$

B.2. Lower bound

The goal of this section is to prove Lemma B.2.

**Lemma B.2 (Lower bound with explicit function).** Given an $m$-layer ReLU neural network function $f : \mathbb{R}^{n_{0}} \rightarrow \mathbb{R}^{n_{m}}$, parameters $p, \epsilon$, there exists two explicit functions $f^{L} : \mathbb{R}^{n_{0}} \rightarrow \mathbb{R}^{n_{m}}$ and $f^{U} : \mathbb{R}^{n_{0}} \rightarrow \mathbb{R}^{n_{m}}$ (see Definition 3.4) such that for all $j \in [n_{m}]$,

$$f_{j}^{L}(x) \leq f_{j}(x), \forall x \in B_{p}(x_{0}, \epsilon).$$

Similar to deriving the upper bound of $f_{j}(x)$, we consider the signs of the weights $W_{j,k}^{(m)}$ to derive the lower bound. Let $f_{j}^{L,-m-1}(x)$ be a lower bound of $f_{j}(x)$: $f_{j}^{L,-m-1}(x)$ can be constructed by taking the right-hand-side (RHS) of (18) if $W_{j,k}^{(m)} < 0$ and taking the left-hand-side (LHS) of (18) if $W_{j,k}^{(m)} > 0$. Following the procedure in (19) to (25) (except that now the additional bias term is from the set $k \in I_{m-1}, W_{j,k}^{(m)} < 0$), the lower bound is similar to the upper bound we have derived but but replace $T^{(m-1)}$ by $H^{(m-1)}$, where for each $j \in [n_{m}]$,

$$H_{k,j}^{(m-1)} = \begin{cases} 
t_{k}^{(m-1)} & \text{if } k \in I_{m-1}, A_{j,k}^{(m-1)} < 0 \\
0 & \text{otherwise}
\end{cases}$$

It is because the linear upper and lower bounds in (1) has the same slope $\frac{m}{n_{m}}$ on both sides (i.e. $\sigma(y)$ is bounded by two lines with the same slope but different intercept), which gives the same $A$ matrix and $D$ matrix in computing the upper
Towards Fast Computation of Certified Robustness for ReLU Networks

bound and lower bound of $f_j(x)$. This is the key to facilitate a faster computation under this linear approximation (1). Thus, the lower bound for $f_j(x)$ is:

$$f_j^L(x) \leq f_j(x),$$

where $f_j^L(x) = [f^L(x)]_j$,

$$f_j^L(x) = A_j^{(0)} x + b_j^{(m)} + \sum_{k=1}^{m-1} A_j^{(k)} (b^{(k)} - H_j^{(k)})$$

and for $k = 1, \ldots, m-1$,

$$H_j^{(k)} = \begin{cases} I_j^{(k)} & \text{if } r \in \mathcal{I}_k, A_j^{(k)} < 0 \\ 0 & \text{otherwise.} \end{cases}$$
C. Proof of Corollary 3.7

By Theorem 3.5, for \( x \in B_p(x_0, \epsilon) \), we have \( f_j^U(x) \leq f_j(x) \leq f_j^L(x) \). Thus,

\[
\begin{align*}
\gamma_j^U := \max_{x \in B_p(x_0, \epsilon)} f_j^U(x) &= \max_{x \in B_p(x_0, \epsilon)} \left( \max_{x \in B_p(x_0, \epsilon)} A_{j_i}^{(0)} x + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} - T_{j_i}^{(k)} \right) \\
&= \epsilon \left( \max_{y \in B_p(0,1)} A_{j_i}^{(0)} y + A_{j_i}^{(0)} x_0 + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} - T_{j_i}^{(k)} \right) \\
&= \epsilon \|A_{j_i}^{(0)}\|_q + A_{j_i}^{(0)} x_0 + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} - T_{j_i}^{(k)}. 
\end{align*}
\]

(30)

(31)

From (30) to (31), we do a transformation of variable \( y := \frac{x-x_0}{\epsilon} \) and therefore \( y \in B_p(0,1) \). By the definition of dual norm \( \|\cdot\|_* \):

\[
\|z\|_* = \{\sup_{y} z^T y ~|~ ||y|| \leq 1\},
\]

and the fact that \( \ell_q \) norm is dual of \( \ell_p \) norm for \( p, q \in [1, \infty) \), the term \( \left( \max_{y \in B_p(0,1)} A_{j_i}^{(0)} y \right) \) in (31) can be expressed as \( \|A_{j_i}^{(0)}\|_q \) in (32). Similarly,

\[
\begin{align*}
\gamma_j^L := \min_{x \in B_p(x_0, \epsilon)} f_j^L(x) &= \min_{x \in B_p(x_0, \epsilon)} \left( \min_{x \in B_p(x_0, \epsilon)} A_{j_i}^{(0)} x + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} - H_{j_i}^{(k)} \right) \\
&= \epsilon \left( \min_{y \in B_p(0,1)} -A_{j_i}^{(0)} y + A_{j_i}^{(0)} x_0 + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} - H_{j_i}^{(k)} \right) \\
&= -\epsilon \|A_{j_i}^{(0)}\|_q + A_{j_i}^{(0)} x_0 + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} - H_{j_i}^{(k)}. 
\end{align*}
\]

(33)

(34)

Again, from (33) to (34), we simply replace \( \left( \max_{y \in B_p(0,1)} -A_{j_i}^{(0)} y \right) \) by \( -A_{j_i}^{(0)}\|_q = \|A_{j_i}^{(0)}\|_q \). Thus, if we use \( \nu_j \) to denote the common term \( A_{j_i}^{(0)} x_0 + b_j^{(m)} + \sum_{k=1}^m A_{j_i}^{(k)} b^{(k)} \), we have

\[
\begin{align*}
\gamma_j^U &= \epsilon \|A_{j_i}^{(0)}\|_q - \sum_{k=1}^m A_{j_i}^{(k)} T_{j_i}^{(k)} + \nu_j, \quad \text{(upper bound)} \\
\gamma_j^L &= -\epsilon \|A_{j_i}^{(0)}\|_q - \sum_{k=1}^m A_{j_i}^{(k)} H_{j_i}^{(k)} + \nu_j. \quad \text{(lower bound)}
\end{align*}
\]
D. Algorithms

We present our full algorithms, **Fast-Lin** in Algorithm 1 and **Fast-Lip** in Algorithm 2.

---

**Algorithm 1 Fast Bounding via Linear Upper/Lower Bounds for ReLU (Fast-Lin)**

**Require:** weights and biases of \( m \) layers: \( W^{(1)}, \ldots, W^{(m)}, b^{(1)}, \ldots, b^{(m)} \), original class \( c \), target class \( j \)

```
1: procedure FAST-LIN(x₀, p, c₀)
2:   Replace the last layer weights \( W^{(m)} \) with a row vector \( \bar{w} \leftarrow W^{(m)}_i - w^{(m)}_j \) (see Section 3.3.3)
3:   Initial \( \epsilon \leftarrow \epsilon_0 \)
4:   while \( \epsilon \) has not achieved a desired accuracy and iteration limit has not reached do
5:     \( I^{(0)}, u^{(0)} \leftarrow \emptyset \)
6:     for \( k \leftarrow 1 \) to \( m \) do ▶ Compute lower and upper bounds for ReLU unis for all \( m \) layers
7:       \( l^{(k)}, u^{(k)} \leftarrow \text{COMPUTE TWO-SIDE BOUNDS}(x₀, \epsilon, p, l^{(k-1)}, u^{(k-1)}, k) \)
8:     if \( I^{(m)} > 0 \) then ▶ \( I^{(m)} \) is a scalar since the last layer weight is a row vector
9:       if \( \epsilon \) is a lower bound; increase \( \epsilon \) using a binary search procedure
10: else
11:     if \( \epsilon \) is not a lower bound; decrease \( \epsilon \) using a binary search procedure
12: end
13: end
14: return \( \bar{e}_j \) ▶ \( \bar{e}_j \) is a certified lower bound \( \beta_L \)
```

**Algorithm 2 Fast Bounding via Lipschitz Upper/Lower Bounds for ReLU (Fast-Lip)**

```
1: procedure COMPUTE TWO-SIDE BOUNDS(x₀, \( \epsilon, p, l^{(1)}, u^{(1)}, m' \))
2:   \( x₀ \in \mathbb{R}^n_0 \) : input data vector, \( p \) : \( \ell_p \)-norm, \( \epsilon \) : maximum \( \ell_p \)-norm perturbation
3:   \( l^{(k)}, u^{(k)}, k \in \mathbb{R}^{m'} \) : layer-wise bounds
4:   \( \text{Step 1: Form A matrices} \)
5:   if \( m' = 1 \) then
6:     \( A^{(0)} \leftarrow W^{(1)} \)
7:     First layer bounds do not depend on \( I^{(0)}, u^{(0)} \)
8:   else
9:   for \( k \leftarrow 1 \) to \( m' - 1 \) do:
10: if \( k = m' - 1 \) then
11:   Construct diagonal matrix \( D^{(k)} \in \mathbb{R}^{n_k \times n_k} \) using \( I^{(k)}, u^{(k)} \) according to Eq. (5).
12:   \( A^{(m' - 1)} \leftarrow W^{(m')} D^{(m' - 1)} \)
13: else
14:   \( A^{(k)} \leftarrow A^{(m' - 1)} A^{(k)} \)
15:   \( T^{(k)} \leftarrow 0, H^{(k)} \leftarrow 0 \)
16: for all \( r \in \mathbb{Z}_k \) do
17:   for \( j \leftarrow 1 \) to \( n_k \) do
18:     if \( A^{(k)}_{j,r} > 0 \) then
19:       \( T^{(k)}_{j,r} \leftarrow I^{(k)}_{j,r} \)
20:     else
21:       \( H^{(k)}_{j,r} \leftarrow I^{(k)}_{j,r} \)
22:   end
23: end
24: for \( j \leftarrow 1 \) to \( n_m \) do
25:   \( \nu_j \leftarrow A^{(0)}_{j,0} + b^{(m')} , \mu_j^+ \leftarrow 0, \mu_j^- \leftarrow 0 \)
26: for \( k \leftarrow 1 \) to \( m' - 1 \) do
27:   \( \mu_j^+ \leftarrow \mu_j^+ - A^{(k)}_{j,:} T^{(k)}_{:,j} , \mu_j^- \leftarrow \mu_j^- - A^{(k)}_{j,:} H^{(k)}_{:,j} \)
28:   \( \nu_j \leftarrow \nu_j + A^{(k)}_{j,:} b^{(k)} \)
29: end
30: \( \gamma^{U}_j \leftarrow \mu_j^+ + \nu_j + \epsilon \| A^{(0)}_{j,:} \|_q \)
31: \( \gamma^{L}_j \leftarrow \mu_j^- + \nu_j + \epsilon \| A^{(0)}_{j,:} \|_q \)
32: end
33: return \( \gamma^L, \gamma^U \)
```

---

Towards Fast Computation of Certified Robustness for ReLU Networks
Towards Fast Computation of Certified Robustness for ReLU Networks

Algorithm 2 Fast Bounding via Upper Bounding Local Lipschitz Constant (Fast-Lip)

Require: Weights of \( m \) layers: \( W^{(1)}, \ldots, W^{(m)} \), original class \( c \), target class \( j \)
1: procedure Fast-Lip(\( x_0, p, c \))
2: Replace the last layer weights \( W^{(m)} \) with a row vector \( \bar{w} \leftarrow W^{(m)}_{c, :} - W^{(m)}_{j, :} \) (see Section 3.3.3)
3: Run Fast-LIN to find layer-wise bounds \( I^{(i)}, u^{(i)} \), and form \( \mathcal{I}_+^{(l)}, \mathcal{I}_-^{(l)}, \mathcal{I}_l \) for all \( i \in [m] \)
4: \( C^{(0)} \leftarrow W^{(1)}, L^{(0)} \leftarrow 0, U^{(0)} \leftarrow 0 \)
5: for \( l \leftarrow 1 \) to \( m - 1 \) do
6: \( (C^{(l)}, L^{(l)}, U^{(l)}) = \text{BOUNDAYERGRAD}(C^{(l-1)}, L^{(l-1)}, U^{(l-1)}, W^{(l-1)}, W^{(l+1)}, n_{l+1}, \mathcal{I}_+^{(l)}, \mathcal{I}_-^{(l)}, \mathcal{I}_l) \)
7: \( v \leftarrow \max(\|C^{(m-1)} + L^{(m-1)}\|, \|C^{(m-1)} + U^{(m-1)}\|) \) \( \triangleright \) All operations are element-wise;
8: \( \bar{e}_j \leftarrow \min\left(\frac{q(\sigma_{p q})}{\|w\|_{q}}, \epsilon\right) \) \( \triangleright q \) is the dual norm of \( p, \frac{1}{p} + \frac{1}{q} = 1 \)
9: return \( \bar{e}_j \)
10: end

procedure BOUNDAYERGRAD(C, L, U, W, n', \( \mathcal{I}_+^{(3)}, \mathcal{I}_-^{(3)} \))
12: for \( k \in [n_0] \) do
13: for \( j \in [n'] \) do
14: \( C_{j, k} \leftarrow \sum_{i \in \mathcal{I}^+} W_{j, i} C_{i, k} \)
15: \( U_{j, k} \leftarrow \sum_{i \in \mathcal{I}^+} W_{j, i} U_{i, k} + \sum_{i \in \mathcal{I}^+} W_{j, i} L_{i, k} + \sum_{i \in \mathcal{I}^-} W_{j, i} (C_{i, k} + L_{i, k}) \)
16: \( L_{j, k} \leftarrow \sum_{i \in \mathcal{I}^+} W_{j, i} L_{i, k} + \sum_{i \in \mathcal{I}^+} W_{j, i} U_{i, k} + \sum_{i \in \mathcal{I}^-} W_{j, i} (C_{i, k} + L_{i, k}) \)
17: return \( C', L', U' \)

E. An alternative bound on the Lipschitz constant

Using the property of norm, we can derive an upper bound of the gradient norm of a 2-layer ReLU network in the following:

\[
\|\nabla f_j(x)\|_q = \|W^{(2)}_{j, :} A^{(1)}_{a} W^{(1)}\|_q
\]

\[
= \|W^{(2)}_{j, :} (A^{(1)}_{a} + A^{(1)}_{u}) W^{(1)}\|_q
\]

\[
\leq \|W^{(2)}_{j, :} A^{(1)}_{a} W^{(1)}\|_q + \|W^{(2)}_{j, :} A^{(1)}_{u} W^{(1)}\|_q
\]

\[
\leq \|W^{(2)}_{j, :} A^{(1)}_{u} W^{(1)}\|_q + \sum_{r \in \mathcal{X}_1} \|W^{(2)}_{j, r} W^{(1)}_{r, :}\|_q
\]

where with a slight abuse of notation, we use \( A^{(1)}_{a} \) to denote the diagonal activation matrix for neurons who are always activated, i.e. its \((r, r)\) entry \( A^{(1)}_{a(r, r)} \) is 1 if \( r \in \mathcal{I}_1^+ \) and 0 otherwise, and we use \( A^{(1)}_{u} \) to denote the diagonal activation matrix for neurons whose status are uncertain because they could possibly be active or inactive, i.e. its \((r, r)\) entry \( A^{(1)}_{u(r, r)} \) is 1 if \( r \in \mathcal{I}_1 \) and 0 otherwise. Therefore, we can write \( A^{(1)} \) as a sum of \( A^{(1)}_a \) and \( A^{(1)}_u \).

Note that (35) to (36) is from the sub-additive property of a norm, and (36) to (37) uses the sub-additive property of a norm again and set the uncertain neurons encoding all to 1 because

\[
\|W^{(2)}_{j, :} A^{(1)}_{u} W^{(1)}\|_q = \|W^{(2)}_{j, :} A^{(1)}_{u(r, r)} W^{(1)}_{r, :}\|_q \leq \sum_{r \in \mathcal{X}_1} \|W^{(2)}_{j, r} A^{(1)}_{u(r, r)} W^{(1)}_{r, :}\|_q \leq \sum_{r \in \mathcal{X}_1} \|W^{(2)}_{j, r} W^{(1)}_{r, :}\|_q
\]

Notice that (37) can be used as an upper bound of Lipschitz constant and is applicable to compute a certified lower bound for minimum adversarial distortion of a general \( \ell_p \) norm attack. However, this bound is expected to be less tight because we simply include all the uncertain neurons to get an upper bound on the norm in (37).
Towards Fast Computation of Certified Robustness for ReLU Networks

F. Details of Experiments in Section 4

F.1. Methods

Below, we give detailed descriptions on the methods that we compare in Table 1, Table F.1 and Table F.2:

- **Fast-Lin**: Our proposed method of directly bounding network output via linear upper/lower bounds for ReLU, as discussed in Section 3.3 and Algorithm 1;
- **Fast-Lip**: Our proposed method based on bounding local Lipschitz constant, in Section 3.4 and Algorithm 2;
- **Reluplex**: Reluplex (Katz et al., 2017) is a satisfiability modulo theory (SMT) based solver which delivers a true minimum distortion, but is very computationally expensive;
- **LP-Full**: A linear programming baseline method with formulation borrowed from (Wong & Kolter, 2018). Note that we solve the primal LP formulation exactly to get a best possible bound. This variant solves full relaxed LP problems at every layer to give a final “adversarial polytope”. Similar to our proposed methods, it only gives a lower bound. We extend this formulation to \( p = 2 \) case, where the input constraint becomes quadratic and requires a quadratic constrained programming (QCP) solver, which is usually slower than LP solvers.
- **LP**: Similar to LP-Full, but this variant solves only one LP problem for the full network at the output neurons and the layer-wise bounds for the neurons in hidden layers are solved by Fast-Lin. We also extend it to \( p = 2 \) case with QCP constraints on the inputs. LP and LP-Full are served as our baselines to compare with Fast-Lin and Fast-Lip;
- **Attacks**: Any successful adversarial example gives a valid upper bound for the minimum adversarial distortion. For larger networks where Reluplex is not feasible, we run adversarial attacks and obtain an upper bound of minimal adversarial distortions to compare with. We apply the \( \ell_2 \) and \( \ell_\infty \) variants of Carlini and Wagner’s attack (CW) (Carlini & Wagner, 2017c) to find the best \( \ell_2 \) and \( \ell_\infty \) distortions. We found that the CW \( \ell_\infty \) attack usually finds adversarial examples with smaller \( \ell_\infty \) distortions than using PGD (projected gradient descent). We use EAD (Chen et al., 2018b), a Elastic-Net regularized attack, to find adversarial examples with small \( \ell_1 \) distortions. We run CW \( \ell_2 \) and \( \ell_\infty \) attacks for 3,000 iterations and EAD attacks for 2,000 iterations;
- **CLEVER**: CLEVER (Weng et al., 2018) is an attack-agnostic robustness score based on local Lipschitz constant estimation and provides an estimated lower-bound. It is capable of performing robustness evaluation for large-scale networks but is not a certified lower bound;
- **Op-norm**: Operator norms of weight matrices were first used in (Szegedy et al., 2013) to give a robustness lower bound. We compute the \( \ell_p \) induced norm of weight matrices of each layer and use their product as the global Lipschitz constant \( L_p \). A valid lower bound is given by \( g(x_0)/L_p \) (see Section 3.4). We only need to pre-compute the operator norms once for all the examples.

F.2. Setup

We use MNIST and CIFAR datasets and evaluate the performance of each method in MLP networks with up to 7 layers or over 10,000 neurons, which is the largest network size for non-trivial and guaranteed robustness verification to date. We use the same number of hidden neurons for each layer and denote a \( m \)-layer network with \( n \) hidden neurons in each layer as \( m \times [n] \). Each network is trained with a grid search of learning rates from \( \{0.1, 0.05, 0.02, 0.01, 0.005\} \) and weight decays from \( \{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\} \) and we select the network with the best validation accuracy. We consider both targeted and untargeted robustness under \( \ell_p \) distortions \( (p = 1, 2, \infty) \); for targeted robustness, we consider three target classes: a random class, a least likely class and a runner-up class (the class with second largest probability). The reported average scores are an average of 100 images from the test set, with images classified wrongly skipped. Reported time is per image. We use binary search to find the certified lower bounds in Fast-Lin, Fast-Lip, LP and LP-Full, and the maximum number of search iterations is set to 15.

We implement our algorithm using Python (with Numpy and Numba)\(^3\), while for the LP based method we use the highly efficient Gurobi commercial LP solver with Python Interface. All experiments are conducted in single thread mode (we

\(^3\)https://github.com/huanzhang12/CertiFiedReLURobustness
disable the concurrent solver in Gurobi) on an Intel Xeon E5-2683v3 (2.0 GHz) CPU. Despite the inefficiency of Python, we still achieve two orders of magnitudes speedup compared with LP, while achieving a very similar lower bound. Our methods are automatically parallelized by Numba and can gain further speedups on a multi-core CPU, but we disabled this parallelization for a fair comparison to other methods.

F.3. Discussions

In Table 1a (full Table: Table F.1), we compare the lower bound \( \beta_L \) computed by each algorithm to the true minimum distortion \( r_0 \) found by Reluplex. We are only able to verify 2 and 3 layer MNIST with 20 neurons per hidden layer within reasonable time using Reluplex. It is worth noting that the input dimension (784) is very large compared to the network evaluated in (Katz et al., 2017) with only 5 inputs. Lower bounds found by Fast-Lin is very close to LP, and the gaps are within 2-3X from the true minimum distortion \( r_0 \) found by Reluplex. The upper bound given by CW \( \ell_\infty \) are also very close to \( r_0 \).

In Table 1b (full Table: Table F.2), we compare Fast-Lin, Fast-Lip with LP and Op-norm on larger networks with up to over ten thousands hidden neurons. Fast-Lin and Fast-Lip are significantly faster than LP and are able to verify much larger networks (LP becomes very slow to solve exactly on 4-layer MNIST with 4096 hidden neurons, and is infeasible for even larger CIFAR models). Fast-Lin achieves a very similar bound comparing with results of LP over all smaller models, but being over two orders of magnitude faster. We found that Fast-Lip can achieve better bounds when \( p = 1 \) in two-layers networks, and is comparable to Fast-Lin in shallow networks. Meanwhile, we also found that Fast-Lin scales better than Fast-Lip for deeper networks, where Fast-Lin usually provides a good bound even when the number of layers is large. For deeper networks, neurons in the last few layers are likely to have uncertain activations, making Fast-Lip being too pessimistic. However, Fast-Lip outperforms the global Lipschitz constant based bound (Op-norm) which quickly goes down to 0 when the network goes deeper, as Fast-Lip is bounding the local Lipschitz constant to compute robustness lower bound. In Table F.2, we also apply our method to MNIST and CIFAR models to compare the minimum distortion for untargeted attacks. The computational benefit of Fast-Lin and Fast-Lip is more significant than LP because LP needs to solve \( n_m \) objectives (where \( n_m \) is the total number of classes), whereas the cost of our methods stay mostly unchanged as we get the bounds for all network outputs simultaneously.

In Table 2, we compute our two proposed lower bounds on neural networks with defending techniques to evaluate the effects of defending techniques (e.g. how much robustness is increased). We train the network with two defending methods, defensive distillation (DD) (Papernot et al., 2016) and adversarial training (Madry et al., 2018) based on robust optimization. For DD we use a temperature of 100, and for adversarial training, we train the network for 100 epochs with adversarial examples crafted by 10 iterations of PGD with \( \epsilon = 0.3 \). The test accuracy for the adversarially trained models dropped from 98.5% to 97.3%, and from 98.6% to 98.1%, for 3 and 4 layer MLP models, respectively. We observe that both defending techniques can increase the computed robustness lower bounds, however adversarial training is significantly more effective than defensive distillation. The lower bounds computed by Fast-Lin are close to the desired robustness guarantee \( \epsilon = 0.3 \).
<table>
<thead>
<tr>
<th>Toy Networks</th>
<th>Average Magnitude of Distortions on 100 Images</th>
<th>Average Running Time per Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
<td>Certified Bounds</td>
<td>Uncertified Attacks</td>
</tr>
<tr>
<td>MNIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 x 20 [20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Runner-Up</td>
<td>Rand</td>
</tr>
<tr>
<td></td>
<td>0.0151</td>
<td>0.0179</td>
</tr>
<tr>
<td></td>
<td>0.0309</td>
<td>0.0319</td>
</tr>
<tr>
<td></td>
<td>0.0448</td>
<td>0.0462</td>
</tr>
<tr>
<td></td>
<td>0.3879</td>
<td>0.4811</td>
</tr>
<tr>
<td></td>
<td>0.6278</td>
<td>0.7560</td>
</tr>
<tr>
<td></td>
<td>0.9105</td>
<td>1.0997</td>
</tr>
<tr>
<td></td>
<td>2.3798</td>
<td>2.5932</td>
</tr>
<tr>
<td></td>
<td>3.9297</td>
<td>4.2681</td>
</tr>
<tr>
<td></td>
<td>7.5298</td>
<td>6.2062</td>
</tr>
<tr>
<td></td>
<td>0.0158</td>
<td>0.0168</td>
</tr>
<tr>
<td></td>
<td>0.0229</td>
<td>0.0241</td>
</tr>
<tr>
<td></td>
<td>0.0304</td>
<td>0.0319</td>
</tr>
<tr>
<td></td>
<td>0.3228</td>
<td>0.3809</td>
</tr>
<tr>
<td></td>
<td>0.4652</td>
<td>0.5345</td>
</tr>
<tr>
<td></td>
<td>0.6179</td>
<td>0.7083</td>
</tr>
<tr>
<td></td>
<td>2.0189</td>
<td>2.1272</td>
</tr>
<tr>
<td></td>
<td>2.8550</td>
<td>3.1000</td>
</tr>
<tr>
<td></td>
<td>3.7504</td>
<td>4.0434</td>
</tr>
<tr>
<td>MNIST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 x 20 [20]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Runner-Up</td>
<td>Rand</td>
</tr>
<tr>
<td></td>
<td>0.0158</td>
<td>0.0168</td>
</tr>
<tr>
<td></td>
<td>0.0229</td>
<td>0.0241</td>
</tr>
<tr>
<td></td>
<td>0.0304</td>
<td>0.0319</td>
</tr>
<tr>
<td></td>
<td>0.3228</td>
<td>0.3809</td>
</tr>
<tr>
<td></td>
<td>0.4652</td>
<td>0.5345</td>
</tr>
<tr>
<td></td>
<td>0.6179</td>
<td>0.7083</td>
</tr>
<tr>
<td></td>
<td>2.0189</td>
<td>2.1272</td>
</tr>
<tr>
<td></td>
<td>2.8550</td>
<td>3.1000</td>
</tr>
<tr>
<td></td>
<td>3.7504</td>
<td>4.0434</td>
</tr>
</tbody>
</table>

(a) Comparison of bounds

(b) Comparison of time
Table F.2. Comparison of our proposed certified lower bounds Fast-Lin and Fast-Lip with other lower bounds (LP, Op-norm, CLEVER) and upper bounds (Attack) algorithms: CW for $p = 2, \infty$, EAD for $p = 1$) on networks with 2-7 layers, where each layer has 1024 or 2048 nodes. Differences of lower bounds and speedup are measured on the two corresponding bold numbers in each row. Note that LP-Full and Reluplex are computationally infeasible for all the networks reported here, and “-” indicates the method is computationally infeasible for that network. For Op-norm, computation time for each image is negligible as the operator norms can be pre-computed.

<table>
<thead>
<tr>
<th>Network</th>
<th>$p$</th>
<th>Target</th>
<th>Certified Bounds</th>
<th>Uncertified Bounds</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>2</td>
<td>runner-up</td>
<td>0.02526</td>
<td>0.80529</td>
<td>127 ms 167 ms 19.5 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>least</td>
<td>0.00683</td>
<td>0.00859</td>
<td>156 ms 219 ms 20.8 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>runner-up</td>
<td>0.40034</td>
<td>0.48267</td>
<td>129 ms 377 ms 22.2 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>least</td>
<td>0.63299</td>
<td>0.50933</td>
<td>127 ms 190 ms 419 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>runner-up</td>
<td>0.79263</td>
<td>0.71313</td>
<td>128 ms 234 ms 195 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>least</td>
<td>0.39670</td>
<td>0.26501</td>
<td>163 ms 305 ms 156 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>runner-up</td>
<td>3.88241</td>
<td>5.10000</td>
<td>177 ms 1.137 s 38.9 s</td>
</tr>
<tr>
<td>MNIST</td>
<td>2</td>
<td>least</td>
<td>4.90809</td>
<td>6.36600</td>
<td>139 ms 1.40 s 48.1 s</td>
</tr>
</tbody>
</table>

| CIFAR | 5 | runner-up | 0.01630 | 0.02013 | 1.20 s 1.81 s 50.4 s | 42X |
| CIFAR | 5 | least    | 0.02216 | 0.02428 | 1.12 s 1.11 s 52.7 s | 47X |
| CIFAR | 5 | runner-up | 0.24232 | 0.02669 | 1.02 s 924 ms 54.3 s | 53X |
| CIFAR | 5 | least    | 0.35807 | 0.41940 | 898 ms 1.59 s 438 s | 497X |
| CIFAR | 4 | runner-up | 0.25989 | 0.29590 | 863 ms 3.16 s 91.1 s | 109X |
| CIFAR | 4 | least    | 0.43502 | 0.49715 | 906 ms 9.41 s 714 s | 788X |
| CIFAR | 4 | runner-up | 1.05833 | 0.11175 | 925 ms 1.01 s 858 s | 928X |
| CIFAR | 4 | least    | 1.37974 | 0.72800 | 1.30 s 1.22 s 122 s | 130X |
| CIFAR | 5 | runner-up | 0.00673 | 0.00219 | 2.19 s 2.55 s 11.0 s | - |
| CIFAR | 5 | least    | 0.00682 | 0.00264 | 2.25 s 3.08 s 12.2 s | - |
| CIFAR | 5 | runner-up | 0.00099 | 0.00304 | 2.15 s 3.02 s 11.0 s | - |
| CIFAR | 5 | least    | 0.00244 | 0.00000 | 2.54 s 3.54 s 14.5 s | - |
| CIFAR | 5 | runner-up | 0.00170 | 0.00030 | 26.2 s 78.1 s 2.9 s | - |
| CIFAR | 5 | least    | 0.00188 | 0.00036 | 27.8 s 79.0 s 3.0 s | - |
| CIFAR | 6 | runner-up | 0.00612 | 0.00051 | 34.0 s 60.7 s 2.8 s | - |
| CIFAR | 6 | least    | 0.00764 | 0.00147 | 36.8 s 49.4 s 3.9 s | - |
| CIFAR | 6 | runner-up | 0.00456 | 0.00177 | 37.4 s 49.8 s 3.1 s | - |
| CIFAR | 6 | least    | 0.00635 | 0.00262 | 37.5 s 51.6 s 3.0 s | - |
| CIFAR | 7 | runner-up | 0.00075 | 0.00005 | 37.2 s 106 s 3.3 s | - |
| CIFAR | 7 | least    | 0.00099 | 0.00007 | 37.0 s 119 s 3.3 s | - |
| CIFAR | 6 | runner-up | 0.00346 | 0.00028 | 56.1 s 99.5 s 3.5 s | - |
| CIFAR | 6 | least    | 0.00412 | 0.00031 | 60.2 s 95.6 s 3.5 s | - |
| CIFAR | 7 | runner-up | 0.00563 | 0.00057 | 61.9 s 88.6 s 3.5 s | - |
| CIFAR | 7 | least    | 0.00718 | 0.00062 | 62.1 s 65.1 s 3.2 s | - |

Towards Fast Computation of Certified Robustness for ReLU Networks