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 (Ailon 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 follow decision problem:*

Does there exist a set-cover of size t, i.e., $\exists C \subseteq [m]$, such that $\cup_{j\in C} S_j = U$ 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., $\exists C \subseteq [m]$, such that $\cup_{j\in C} S_j = U$ 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 α. 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 α times as large as t_0.

In our setting, we want to investigate the hardness of finding the lower bound with a guaranteed approximation ration, 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 α is as hard as finding an upper bound with an approximation ratio of α.

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 α is as hard as finding an upper bound t_U with an approximation ratio α.*

Proof. If we find a lower bound t_L with $\frac{t_0}{\alpha} \leq t_L \leq t_0$, by multiplying both sides by α, 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 α is at least as hard as finding an upper bound with an approximation ratio α. The converse is also true. □

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.

Theorem A.6 ((Raz & Safra, 1997; Alon et al., 2006; Dinur & Steurer, 2014)). *Unless* NP = P, *there is no polynomial time algorithm that gives a* $(1 - o(1)) \ln n$*-approximation to* SET-COVER *problem with universe size* n.

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:*

Does there exist a y *with* $\|x - y\|_1 \leq r$ *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:*

Does there exist a y *with* $\|x - y\|_1 \leq r$ *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 α. With a similar statement as in Proposition A.5, if we divide the answer r by α, 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.[2] The formal definition can be stated as follows,

Theorem A.11 ((Arora & Safra, 1998; Arora et al., 1998)). *Given a* SAT *formula* ϕ *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 acceptes or reject ϕ.
(II) : (Yes Case / Completeness) If ϕ is satisfiable, then there exists a proof so that all tests accept ϕ.
(III) : (No Case / Soundness) If ϕ is not satifiable, then no proof will cause more than $M/2$ tests to accept ϕ.

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 inapproximibility results, we refer the readers to (Ambühl et al., 2011).

[2] https://en.wikipedia.org/wiki/PCP_theorem

A.3. Warm-up

We state our hardness result for ROBUST-NET($\mathbb{B}$) (boolean inputs case) in this section. The reduction procedure for network with boolean 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 *hidden nodes.*

Proof. Consider a set-cover instance, let S denote a set of sets $\{S_1, S_2, \cdots, S_d\}$ 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 $\mathbf{1}_{i \in S_j}$ 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 ϕ_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 $u_j \in \{0, 1\}$, ϕ_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 $\sum_{j=1}^{d} \mathbf{1}_{i \in S_j} = \sum_{j=1}^{d} u_j$, because $u_j = 1$ indicates choosing set S_j and $u_j = 0$ otherwise.

For final output node w, we define an activation function ψ satisfies that

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

Since $v_i \in [n]$, ψ 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 to easy to see that $F(x) = 0$. Let $\alpha > 1$ denote a fixed parameter. Also, we have $F(y) > 0$ if and only if $C = \{j | y_j = 1\}$ 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. □

A.4. Main result

With the proof for ROBUST-NET($\mathbb{B}$) 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.*

Proof. Consider a set-cover instance, let S denote a set of sets $\{S_1, S_2, \cdots, S_d\}$ where $S_j \subseteq [n], \forall j \in [d]$. For each set S_j we create an input node u_j. For each $j \in [d]$, we create a hidden node t_j and connect u_j and t_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. Finally, we create an output node w and for each $i \in [n]$, we connect node v_i and node w.

Let $\delta = 1/d$. For each $j \in [n]$, we apply an activation function $\phi_{1,j}$ on t_j such that

$$\phi_{1,j} = -\max(0, \delta - u_j) + \max(0, u_j - 1 + \delta)$$

It is easy to see that

$$t_j = \phi_{1,j} = \begin{cases} u_j - \delta & \text{if } u_j \in [0, \delta] \\ u_j - (1 - \delta) & \text{if } u_j \in [1 - \delta, 1] \\ 0 & \text{otherwise .} \end{cases}$$

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 need an activation function $\phi_{2,i}$ on node v_i which satisfies that

$$\phi_{2,i} \in \begin{cases} [-\delta, 0], & \text{if } \forall j \in T_i, t_j \in [-\delta, 0], \\ [0, \delta], & \text{if } \exists j \in T_i, t_j \in [0, \delta]. \end{cases}$$

This can be implemented in the following way,

$$\phi_{2,i} = \max_{j \in T_i} t_j.$$

For the final output node w, we define it as

$$w = \min_{i \in [n]} v_i.$$

We use vector x to denote $\{0\}^d$ vector and it is to 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 A.16 (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 $F(y) > 0$.*

Proof. Without loss of generality, we let the set cover to be $\{S_1, S_2, ..., S_r\}$. Let $y_1 = y_2 = \cdots = y_r = 1$ and $y_{r+1} = y_{r+2} = ... = y_d = 0$. By the definition of t_j, we have $t_1 = t_2 = \cdots = t_r = \delta$. Since $\{S_1, S_2, \cdots, S_r\}$ 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 A.17 (Soundness)**.** *If for every $C \subseteq [d]$ with $\cup_{j \in C} S_j = U$ 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 $\boldsymbol{l}^{(k)}$ and $\boldsymbol{u}^{(k)}$, $\forall k \in [m-1]$ 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

$$f_j(\boldsymbol{x}) = \sum_{k=1}^{n_{m-1}} \mathbf{W}_{j,k}^{(m)}[\phi_{m-1}(\boldsymbol{x})]_k + \boldsymbol{b}_j^{(m)}, \tag{15}$$

$$= \sum_{k=1}^{n_{m-1}} \mathbf{W}_{j,k}^{(m)}\sigma(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(x) + \boldsymbol{b}_k^{(m-1)}) + \boldsymbol{b}_j^{(m)}, \tag{16}$$

$$= \sum_{k\in\mathcal{I}_{m-1}^+,\mathcal{I}_{m-1}^-,\mathcal{I}_{m-1}} \mathbf{W}_{j,k}^{(m)}\sigma(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) + \boldsymbol{b}_j^{(m)}. \tag{17}$$

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

$$\sigma(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(x) + \boldsymbol{b}_k^{(m-1)}) = \mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}.$$

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

$$\sigma(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) = 0.$$

Finally, for neurons belonging to Category (iii), i.e., $k \in \mathcal{I}_{m-1}$, we bound their outputs. If we adopt the linear upper and lower bounds in (1) and let $\boldsymbol{d}_k^{(m-1)} := \frac{\boldsymbol{u}_k^{(m-1)}}{\boldsymbol{u}_k^{(m-1)} - \boldsymbol{l}_k^{(m-1)}}$, we have

$$\boldsymbol{d}_k^{(m-1)}(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) \le \sigma(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) \le \boldsymbol{d}_k^{(m-1)}(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)} - \boldsymbol{l}_k^{(m-1)}). \tag{18}$$

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 $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$, parameters p, ϵ, there exists two explicit functions $f^L : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$ and $f^U : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$ (see Definition 3.4) such that $\forall j \in [n_m]$,*

$$f_j(\boldsymbol{x}) \le f_j^U(\boldsymbol{x}), \forall \boldsymbol{x} \in B_p(\boldsymbol{x}_0, \epsilon).$$

Notice that (18) can be used to construct an upper bound and lower bound of $f_j(\boldsymbol{x})$ by considering the signs of the weights $\mathbf{W}_{j,k}^{(m)}$. Let $f_j^{U,m-1}(\boldsymbol{x})$ be an upper bound of $f_j(\boldsymbol{x})$; $f_j^{U,m-1}(\boldsymbol{x})$ can be constructed by taking the right-hand-side (RHS) of

(18) if $\mathbf{W}_{j,k}^{(m)} > 0$ and taking the left-hand-side (LHS) of (18) if $\mathbf{W}_{j,k}^{(m)} < 0$:

$$
\begin{aligned}
& f_j^{U,m-1}(\boldsymbol{x}) \\
&= \sum_{k\in\mathcal{I}_{m-1}^+} \mathbf{W}_{j,k}^{(m)}(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) && (19)\\
&\quad + \sum_{k\in\mathcal{I}_{m-1}, \mathbf{W}_{j,k}^{(m)}>0} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)} - \boldsymbol{l}_k^{(m-1)}) \\
&\quad + \sum_{k\in\mathcal{I}_{m-1}, \mathbf{W}_{j,k}^{(m)}<0} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) + \boldsymbol{b}_j^{(m)} \\
&= \sum_{k=1}^{n_{m-1}} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}(\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \boldsymbol{b}_k^{(m-1)}) - \sum_{k\in\mathcal{I}_{m-1}, \mathbf{W}_{j,k}^{(m)}>0} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}\boldsymbol{l}_k^{(m-1)} + \boldsymbol{b}_j^{(m)}, && (20)\\
&= \sum_{k=1}^{n_{m-1}} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) && (21)\\
&\quad + \left(\sum_{k=1}^{n_{m-1}} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}\boldsymbol{b}_k^{(m-1)} - \sum_{k\in\mathcal{I}_{m-1}, \mathbf{W}_{j,k}^{(m)}>0} \mathbf{W}_{j,k}^{(m)}\boldsymbol{d}_k^{(m-1)}\boldsymbol{l}_k^{(m-1)} + \boldsymbol{b}_j^{(m)}\right),
\end{aligned}
$$

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

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

$$
\begin{aligned}
f_j^{U,m-1}(\boldsymbol{x}) &= \sum_{k=1}^{n_{m-1}} \mathbf{A}_{j,k}^{(m-1)}\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x}) + \left(\mathbf{A}_{j,:}^{(m-1)}\boldsymbol{b}^{(m-1)} - \mathbf{A}_{j,:}^{(m-1)}\mathbf{T}_{:,j}^{(m-1)} + \boldsymbol{b}_j^{(m)}\right) && (22)\\
&= \sum_{k=1}^{n_{m-1}} \mathbf{A}_{j,k}^{(m-1)}\Big(\sum_{r=1}^{n_{m-2}} \mathbf{W}_{k,r}^{(m-1)}[\phi_{m-2}(\boldsymbol{x})]_r\Big) + \left(\mathbf{A}_{j,:}^{(m-1)}\boldsymbol{b}^{(m-1)} - \mathbf{A}_{j,:}^{(m-1)}\mathbf{T}_{:,j}^{(m-1)} + \boldsymbol{b}_j^{(m)}\right) && (23)\\
&= \sum_{r=1}^{n_{m-2}}\sum_{k=1}^{n_{m-1}} \mathbf{A}_{j,k}^{(m-1)}\mathbf{W}_{k,r}^{(m-1)}[\phi_{m-2}(\boldsymbol{x})]_r + \left(\mathbf{A}_{j,:}^{(m-1)}\boldsymbol{b}^{(m-1)} - \mathbf{A}_{j,:}^{(m-1)}\mathbf{T}_{:,j}^{(m-1)} + \boldsymbol{b}_j^{(m)}\right) && (24)\\
&= \sum_{r=1}^{n_{m-2}} \widetilde{\boldsymbol{W}}_{j,r}^{(m-1)}[\phi_{m-2}(\boldsymbol{x})]_r + \widetilde{\boldsymbol{b}}_j^{(m-1)}. && (25)
\end{aligned}
$$

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

$$
\mathbf{T}_{k,j}^{(m-1)} = \begin{cases} \boldsymbol{l}_k^{(m-1)} & \text{if } k \in \mathcal{I}_{m-1},\ \mathbf{A}_{j,k}^{(m-1)} > 0 \\ 0 & \text{otherwise} \end{cases}
$$

since $0 \le \boldsymbol{d}_k^{(m-1)} \le 1$, $\mathbf{W}_{j,k}^{(m)} > 0$ is equivalent to $\mathbf{A}_{j,k}^{(m-1)} > 0$.

From (22) to (23), we simply write out the inner product $\mathbf{W}_{k,:}^{(m-1)}\phi_{m-2}(\boldsymbol{x})$ into a summation form, and from (23) to (24), we exchange the summation order of k and r. From (24) to (25), we let

$$
\begin{aligned}
\widetilde{\boldsymbol{W}}_{j,r}^{(m-1)} &= \sum_{k=1}^{n_{m-1}} \mathbf{A}_{j,k}^{(m-1)}\mathbf{W}_{k,r}^{(m-1)} && (26)\\
\widetilde{\boldsymbol{b}}_j^{(m-1)} &= \left(\mathbf{A}_{j,:}^{(m-1)}\boldsymbol{b}^{(m-1)} - \mathbf{A}_{j,:}^{(m-1)}\mathbf{T}_{:,j}^{(m-1)} + \boldsymbol{b}_j^{(m)}\right) && (27)
\end{aligned}
$$

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

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

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

$$f_j(\boldsymbol{x}) \leq f_j^U(\boldsymbol{x}),$$

where $f_j^U(\boldsymbol{x}) = [f^U(\boldsymbol{x})]_j$,

$$f_j^U(\boldsymbol{x}) = \mathbf{A}_{j,:}^{(0)}\boldsymbol{x} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{T}_{:,j}^{(k)})$$

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

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

$$\mathbf{D}^{(0)} = \boldsymbol{I}_{n_0}$$

$$\mathbf{D}_{r,r}^{(k)} = \begin{cases} \frac{\boldsymbol{u}_r^{(k)}}{\boldsymbol{u}_r^{(k)} - \boldsymbol{l}_r^{(k)}} & \text{if } r \in \mathcal{I}_k \\ 1 & \text{if } r \in \mathcal{I}_k^+ \\ 0 & \text{if } r \in \mathcal{I}_k^- \end{cases}$$

$$\mathbf{T}_{r,j}^{(k)} = \begin{cases} \boldsymbol{l}_r^{(k)} & \text{if } r \in \mathcal{I}_k,\ \mathbf{A}_{j,r}^{(k)} > 0 \\ 0 & \text{otherwise} \end{cases}$$

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 $f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$, parameters p, ϵ, there exists two explicit functions $f^L : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$ and $f^U : \mathbb{R}^{n_0} \to \mathbb{R}^{n_m}$ (see Definition 3.4) such that $\forall j \in [n_m]$,*

$$f_j^L(\boldsymbol{x}) \leq f_j(\boldsymbol{x}), \forall \boldsymbol{x} \in B_p(\boldsymbol{x}_0, \epsilon).$$

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

$$\mathbf{H}_{k,j}^{(m-1)} = \begin{cases} \boldsymbol{l}_k^{(m-1)} & \text{if } k \in \mathcal{I}_{m-1},\ \mathbf{A}_{j,k}^{(m-1)} < 0 \\ 0 & \text{otherwise.} \end{cases}$$

It is because the linear upper and lower bounds in (1) has the same slope $\frac{u}{u-l}$ on both sides (i.e. $\sigma(y)$ is bounded by two lines with the same slope but different intercept), which gives the same $\mathbf{A}$ matrix and $\mathbf{D}$ matrix in computing the upper

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

$$f_j^L(\boldsymbol{x}) \leq f_j(\boldsymbol{x}),$$

where $f_j^L(\boldsymbol{x}) = [f^L(\boldsymbol{x})]_j$,

$$f_j^L(\boldsymbol{x}) = \mathbf{A}_{j,:}^{(0)}\boldsymbol{x} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{H}_{:,j}^{(k)})$$

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

$$\mathbf{H}_{r,j}^{(k)} = \begin{cases} \boldsymbol{l}_r^{(k)} & \text{if } r \in \mathcal{I}_k,\ \mathbf{A}_{j,r}^{(k)} < 0 \\ 0 & \text{otherwise.} \end{cases}$$

C. Proof of Corollary 3.7

By Theorem 3.5, for $\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)$, we have $f_j^L(\boldsymbol{x}) \le f_j(\boldsymbol{x}) \le f_j^U(\boldsymbol{x})$. Thus,

$$f_j(\boldsymbol{x}) \le f_j^U(\boldsymbol{x}) \le \max_{\boldsymbol{x} \in B_p(\boldsymbol{x}, \epsilon)} f_j^U(\boldsymbol{x}), \tag{28}$$

$$f_j(\boldsymbol{x}) \ge f_j^L(\boldsymbol{x}) \ge \min_{\boldsymbol{x} \in B_p(\boldsymbol{x}, \epsilon)} f_j^L(\boldsymbol{x}). \tag{29}$$

Since $f_j^U(\boldsymbol{x}) = \mathbf{A}_{j,:}^{(0)}\boldsymbol{x} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{T}_{:,j}^{(k)})$,

$$\begin{aligned}
\gamma_j^U := \max_{\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)} f_j^U(\boldsymbol{x}) &= \max_{\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)} \left(\mathbf{A}_{j,:}^{(0)}\boldsymbol{x} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{T}_{:,j}^{(k)}) \right) \\
&= \left(\max_{\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)} \mathbf{A}_{j,:}^{(0)}\boldsymbol{x} \right) + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{T}_{:,j}^{(k)}) \qquad (30) \\
&= \epsilon \left(\max_{\boldsymbol{y} \in B_p(\mathbf{0}, 1)} \mathbf{A}_{j,:}^{(0)}\boldsymbol{y} \right) + \mathbf{A}_{j,:}^{(0)}\boldsymbol{x_0} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{T}_{:,j}^{(k)}) \qquad (31) \\
&= \epsilon \|\mathbf{A}_{j,:}^{(0)}\|_q + \mathbf{A}_{j,:}^{(0)}\boldsymbol{x_0} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{T}_{:,j}^{(k)}). \qquad (32)
\end{aligned}$$

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

$$\|\boldsymbol{z}\|_* = \{\sup_{\boldsymbol{y}} \boldsymbol{z}^\top \boldsymbol{y} \mid \|\boldsymbol{y}\| \le 1\},$$

and the fact that ℓ_q norm is dual of ℓ_p norm for $p, q \in [1, \infty]$, the term $\left(\max_{\boldsymbol{y} \in B_p(\mathbf{0},1)} \mathbf{A}_{j,:}^{(0)}\boldsymbol{y}\right)$ in (31) can be expressed as $\|\mathbf{A}_{j,:}^{(0)}\|_q$ in (32). Similarly,

$$\begin{aligned}
\gamma_j^L := \min_{\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)} f_j^L(\boldsymbol{x}) &= \min_{\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)} \left(\mathbf{A}_{j,:}^{(0)}\boldsymbol{x} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{H}_{:,j}^{(k)}) \right) \\
&= \left(\min_{\boldsymbol{x} \in B_p(\boldsymbol{x_0}, \epsilon)} \mathbf{A}_{j,:}^{(0)}\boldsymbol{x} \right) + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{H}_{:,j}^{(k)}) \\
&= \epsilon \left(\min_{\boldsymbol{y} \in B_p(\mathbf{0}, 1)} \mathbf{A}_{j,:}^{(0)}\boldsymbol{y} \right) + \mathbf{A}_{j,:}^{(0)}\boldsymbol{x_0} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{H}_{:,j}^{(k)}) \\
&= -\epsilon \left(\max_{\boldsymbol{y} \in B_p(\mathbf{0}, 1)} -\mathbf{A}_{j,:}^{(0)}\boldsymbol{y} \right) + \mathbf{A}_{j,:}^{(0)}\boldsymbol{x_0} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{H}_{:,j}^{(k)}) \qquad (33) \\
&= -\epsilon \|\mathbf{A}_{j,:}^{(0)}\|_q + \mathbf{A}_{j,:}^{(0)}\boldsymbol{x_0} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}(\boldsymbol{b}^{(k)} - \mathbf{H}_{:,j}^{(k)}). \qquad (34)
\end{aligned}$$

Again, from (33) to (34), we simply replace $\left(\max_{\boldsymbol{y} \in B_p(\mathbf{0},1)} -\mathbf{A}_{j,:}^{(0)}\boldsymbol{y}\right)$ by $\| - \mathbf{A}_{j,:}^{(0)}\|_q = \|\mathbf{A}_{j,:}^{(0)}\|_q$. Thus, if we use ν_j to denote the common term $\mathbf{A}_{j,:}^{(0)}\boldsymbol{x_0} + \boldsymbol{b}_j^{(m)} + \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}\boldsymbol{b}^{(k)}$, we have

$$\gamma_j^U = \epsilon \|\mathbf{A}_{j,:}^{(0)}\|_q - \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}\mathbf{T}_{:,j}^{(k)} + \nu_j, \qquad \text{(upper bound)}$$

$$\gamma_j^L = -\epsilon \|\mathbf{A}_{j,:}^{(0)}\|_q - \sum_{k=1}^{m-1} \mathbf{A}_{j,:}^{(k)}\mathbf{H}_{:,j}^{(k)} + \nu_j. \qquad \text{(lower bound)}$$

D. Algorithms

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

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

Require: weights and biases of m layers: $\mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(m)}$, $\boldsymbol{b}^{(1)}, \cdots, \boldsymbol{b}^{(m)}$, original class c, target class j

procedure FAST-LIN($\boldsymbol{x}_0, p, \epsilon_0$)
  Replace the last layer weights $\mathbf{W}^{(m)}$ with a row vector $\bar{\boldsymbol{w}} \leftarrow \mathbf{W}^{(m)}_{c,:} - \mathbf{W}^{(m)}_{j,:}$ (see Section 3.3.3)
  Initial $\epsilon \leftarrow \epsilon_0$
  while ϵ has not achieved a desired accuracy and iteration limit has not reached **do**
    $\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)} \leftarrow$ don't care
    for $k \leftarrow 1$ **to** m **do** ▷ Compute lower and upper bounds for ReLU unis for all m layers
      $\boldsymbol{l}^{(k)}, \boldsymbol{u}^{(k)} \leftarrow$COMPUTETWOSIDEBOUNDS($\boldsymbol{x}_0, \epsilon, p, \boldsymbol{l}^{(1:k-1)}, \boldsymbol{u}^{(1:k-1)}, k$)
    if $\boldsymbol{l}^{(m)} > 0$ **then** ▷ $\boldsymbol{l}^{(m)}$ is a scalar since the last layer weight is a row vector
      ϵ is a lower bound; increase ϵ using a binary search procedure
    else
      ϵ is not a lower bound; decrease ϵ using a binary search procedure
  $\widetilde{\epsilon}_j \leftarrow \epsilon$
  return $\widetilde{\epsilon}_j$ ▷ $\widetilde{\epsilon}_j$ is a certified lower bound β_L

procedure COMPUTETWOSIDEBOUNDS($\boldsymbol{x}_0, \epsilon, p, \boldsymbol{l}^{(1:m'-1)}, \boldsymbol{u}^{(1:m'-1)}, m'$)
  ▷ $\boldsymbol{x}_0 \in \mathbb{R}^{n_0}$: input data vector, p : ℓ_p norm, ϵ : maximum ℓ_p-norm perturbation
  ▷ $\boldsymbol{l}^{(k)}, \boldsymbol{u}^{(k)}$, $k \in [m']$: layer-wise bounds
  if $m' = 1$ **then** ▷ Step 1: Form $\mathbf{A}$ matrices
    $\mathbf{A}^{(0)} \leftarrow \mathbf{W}^{(1)}$ ▷ First layer bounds do not depend on $\boldsymbol{l}^{(0)}, \boldsymbol{u}^{(0)}$
  else
    for $k \leftarrow m' - 1$ **to** 1 **do**
      if $k = m' - 1$ **then** ▷ Construct $\mathbf{D}^{(m'-1)}, \mathbf{A}^{(m'-1)}, \mathbf{H}^{(m'-1)}, \mathbf{T}^{(m'-1)}$
        Construct diagonal matrix $\mathbf{D}^{(k)} \in \mathbb{R}^{n_k \times n_k}$ using $\boldsymbol{l}^{(k)}, \boldsymbol{u}^{(k)}$ according to Eq. (5).
        $\mathbf{A}^{(m'-1)} \leftarrow \mathbf{W}^{(m')}\mathbf{D}^{(m'-1)}$
      else ▷ Multiply all saved $\mathbf{A}^{(k)}$ by $\mathbf{A}^{(m'-1)}$
        $\mathbf{A}^{(k)} \leftarrow \mathbf{A}^{(m'-1)}\mathbf{A}^{(k)}$ ▷ We save $\mathbf{A}^{(k)}$ for next function call
      $\mathbf{T}^{(k)} \leftarrow \mathbf{0}$, $\mathbf{H}^{(k)} \leftarrow \mathbf{0}$ ▷ Initialize $\mathbf{T}^{(k)}$ and $\mathbf{H}^{(k)}$
      for all $r \in \mathcal{I}_k$ **do**
        for $j \leftarrow 1$ **to** n_k **do**
          if $\mathbf{A}^{(k)}_{j,r} > 0$ **then**
            $\mathbf{T}^{(k)}_{r,j} \leftarrow \boldsymbol{l}^{(k)}_r$
          else
            $\mathbf{H}^{(k)}_{r,j} \leftarrow \boldsymbol{l}^{(k)}_r$
  for $j = 1$ **to** $n_{m'}$ **do** ▷ Step 2: Compute γ^U and γ^L
    $\nu_j \leftarrow \mathbf{A}^{(0)}_{j,:}\boldsymbol{x_0} + \boldsymbol{b}^{(m')}_j$, $\mu^+_j \leftarrow 0$, $\mu^-_j \leftarrow 0$ ▷ Initialize ν_j, μ^+_j, μ^-_j
    for $k = 1$ **to** $m' - 1$ **do** ▷ This loop is skipped when $m' = 1$
      $\mu^+_j \leftarrow \mu^+_j - \mathbf{A}^{(k)}_{j,:}\mathbf{T}^{(k)}_{:,j}, \quad \mu^-_j \leftarrow \mu^-_j - \mathbf{A}^{(k)}_{j,:}\mathbf{H}^{(k)}_{:,j}$ ▷ According to Eq. (6)
      $\nu_j \leftarrow \nu_j + \mathbf{A}^{(k)}_{j,:}\boldsymbol{b}^{(k)}$ ▷ According to Eq. (7)
    ▷ ν_j, μ^+_j, μ^-_j satisfy Definition 3.6
    $\gamma^U_j \leftarrow \mu^+_j + \nu_j + \epsilon\|\mathbf{A}^{(0)}_{j,:}\|_q$
    $\gamma^L_j \leftarrow \mu^-_j + \nu_j - \epsilon\|\mathbf{A}^{(0)}_{j,:}\|_q$ ▷ Definition 3.6
  return γ^L, γ^U

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

Require: Weights of m layers: $\mathbf{W}^{(1)}, \cdots \mathbf{W}^{(m)}$, original class c, target class j

procedure FAST-LIP($\boldsymbol{x}_0, p, \epsilon$)
  Replace the last layer weights $\mathbf{W}^{(m)}$ with a row vector $\bar{\boldsymbol{w}} \leftarrow \mathbf{W}^{(m)}_{c,:} - \mathbf{W}^{(m)}_{j,:}$ (see Section 3.3.3)
  Run FAST-LIN to find layer-wise bounds $\boldsymbol{l}^{(i)}, \boldsymbol{u}^{(i)}$, and form $\mathcal{I}_i^+, \mathcal{I}_i^-, \mathcal{I}_i$ fo all $i \in [m]$
  $\mathbf{C}^{(0)} \leftarrow \mathbf{W}^{(1)}$, $\mathbf{L}^{(0)} \leftarrow \mathbf{0}$, $\mathbf{U}^{(0)} \leftarrow \mathbf{0}$
  for $l \leftarrow 1$ to $m-1$ **do**
    $\mathbf{C}^{(l)}, \mathbf{L}^{(l)}, \mathbf{U}^{(l)}$ = BOUNDLAYERGRAD($\mathbf{C}^{(l-1)}, \mathbf{L}^{(l-1)}, \mathbf{U}^{(l-1)}, \mathbf{W}^{(l+1)}, n_{l+1}, \mathcal{I}_l^+, \mathcal{I}_l^-, \mathcal{I}_l$)
  ▷ $\boldsymbol{v} \in \mathbb{R}^{n_0}$ because the last layer is replaced with a row vector $\bar{\boldsymbol{w}}$
  $\boldsymbol{v} \leftarrow \max(|\mathbf{C}^{(m-1)} + \mathbf{L}^{(m-1)}|, |\mathbf{C}^{(m-1)} + \mathbf{U}^{(m-1)}|)$ ▷ All operations are element-wise;
  $\widetilde{\epsilon}_j \leftarrow \min(\frac{g(\boldsymbol{x}_0)}{\|\boldsymbol{v}\|_q}, \epsilon)$ ▷ q is the dual norm of p, $\frac{1}{p} + \frac{1}{q} = 1$
  return $\widetilde{\epsilon}_j$ ▷ $\widetilde{\epsilon}_j$ is a certified lower bound β_L. We can also bisect $\widetilde{\epsilon}_j$ (omitted).
procedure BOUNDLAYERGRAD($\mathbf{C}, \mathbf{L}, \mathbf{U}, \mathbf{W}, n', \mathcal{I}^+, \mathcal{I}^-, \mathcal{I}$)
  for $k \in [n_0]$ **do** ▷ n_0 is the dimension of $\boldsymbol{x}_0$
    for $j \in [n']$ **do**

$$\mathbf{C}'_{j,k} \leftarrow \sum_{i \in \mathcal{I}^+} \mathbf{W}_{j,i}\mathbf{C}_{i,k}$$

$$\mathbf{U}'_{j,k} \leftarrow \sum_{i \in \mathcal{I}^+, \mathbf{W}_{j,i}>0} \mathbf{W}_{j,i}\mathbf{U}_{i,k} + \sum_{i \in \mathcal{I}^+, \mathbf{W}_{j,i}<0} \mathbf{W}_{j,i}\mathbf{L}_{i,k} + \sum_{i \in \mathcal{I}, \mathbf{W}_{j,i}(\mathbf{C}_{i,k}+\mathbf{U}_{i,k})>0} \mathbf{W}_{j,i}(\mathbf{C}_{i,k} + \mathbf{U}_{i,k})$$

$$\mathbf{L}'_{j,k} \leftarrow \sum_{i \in \mathcal{I}^+, \mathbf{W}_{j,i}>0} \mathbf{W}_{j,i}\mathbf{L}_{i,k} + \sum_{i \in \mathcal{I}^+, \mathbf{W}_{j,i}<0} \mathbf{W}_{j,i}\mathbf{U}_{i,k} + \sum_{i \in \mathcal{I}, \mathbf{W}_{j,i}(\mathbf{C}_{i,k}+\mathbf{L}_{i,k})<0} \mathbf{W}_{j,i}(\mathbf{C}_{i,k} + \mathbf{L}_{i,k})$$

  return $\mathbf{C}', \mathbf{L}', \mathbf{U}'$

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:

$$\begin{aligned}
&\|\nabla f_j(\boldsymbol{x})\|_q \\
&= \|\mathbf{W}^{(2)}_{j,:}\mathbf{\Lambda}^{(1)}\mathbf{W}^{(1)}\|_q \\
&= \|\mathbf{W}^{(2)}_{j,:}(\mathbf{\Lambda}^{(1)}_a + \mathbf{\Lambda}^{(1)}_u)\mathbf{W}^{(1)}\|_q && (35)\\
&\le \|\mathbf{W}^{(2)}_{j,:}\mathbf{\Lambda}^{(1)}_a\mathbf{W}^{(1)}\|_q + \|\mathbf{W}^{(2)}_{j,:}\mathbf{\Lambda}^{(1)}_u\mathbf{W}^{(1)}\|_q && (36)\\
&\le \|\mathbf{W}^{(2)}_{j,:}\mathbf{\Lambda}^{(1)}_a\mathbf{W}^{(1)}\|_q + \sum_{r \in \mathcal{I}_1} \|\mathbf{W}^{(2)}_{j,r}\mathbf{W}^{(1)}_{r,:}\|_q && (37)
\end{aligned}$$

where with a slight abuse of notation, we use $\mathbf{\Lambda}^{(1)}_a$ to denote the diagonal activation matrix for neurons who are always activated, i.e. its (r,r) entry $\mathbf{\Lambda}^{(1)}_{a(r,r)}$ is 1 if $r \in \mathcal{I}_1^+$ and 0 otherwise, and we use $\mathbf{\Lambda}^{(1)}_u$ 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 $\mathbf{\Lambda}^{(1)}_{u(r,r)}$ is 1 if $r \in \mathcal{I}_1$ and 0 otherwise. Therefore, we can write $\mathbf{\Lambda}^{(1)}$ as a sum of $\mathbf{\Lambda}^{(1)}_a$ and $\mathbf{\Lambda}^{(1)}_u$.

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

$$\|\mathbf{W}^{(2)}_{j,:}\mathbf{\Lambda}^{(1)}_u\mathbf{W}^{(1)}\| = \|\sum_{r \in \mathcal{I}_1} \mathbf{W}^{(2)}_{j,r}\mathbf{\Lambda}^{(1)}_{u(r,r)}\mathbf{W}^{(1)}_{r,:}\| \le \sum_{r \in \mathcal{I}_1} \|\mathbf{W}^{(2)}_{j,r}\mathbf{\Lambda}^{(1)}_{u(r,r)}\mathbf{W}^{(1)}_{r,:}\| \le \sum_{r \in \mathcal{I}_1} \|\mathbf{W}^{(2)}_{j,r}\mathbf{W}^{(1)}_{r,:}\|.$$

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 ℓ_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).

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 **lin**ear upper/lower bounds for ReLU, as discussed in Section 3.3 and Algorithm 1;
- **Fast-Lip**: Our proposed method based on bounding local **Lip**schitz 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 ℓ_2 and ℓ_∞ variants of Carlini and Wagner's attack (CW) (Carlini & Wagner, 2017c) to find the best ℓ_2 and ℓ_∞ distortions. We found that the CW ℓ_∞ attack usually finds adversarial examples with smaller ℓ_∞ 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 ℓ_1 distortions. We run CW ℓ_2 and ℓ_∞ 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 ℓ_p induced norm of weight matrices of each layer and use their product as the global Lipschitz constant L_q^j. A valid lower bound is given by $g(\boldsymbol{x_0})/L_q^j$ (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 ℓ_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 a 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 β_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 ℓ_∞ 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 F.1. Comparison of our proposed certified lower bounds **Fast-Lin** and **Fast-Lip**, **LP** and **LP-Full**, the estimated lower bounds by **CLEVER**, the exact minimum distortion by **Reluplex**, and the upper bounds by **Attack** algorithms (CW ℓ_∞ for $p = \infty$, CW ℓ_2 for $p = 2$, and EAD for $p = 1$) on 2, 3 layers toy MNIST networks with *only 20 neurons per layer*. Differences of lower bounds and speedup are measured on the two corresponding **bold** numbers in each row, representing the best answer from our proposed algorithms and LP based approaches. **Reluplex** is designed to verify ℓ_∞ robustness so we omit results for ℓ_2 and ℓ_1. Note that **LP-Full** and **Reluplex** *are very slow* and cannot scale to any practical networks, and the purpose of this table is to show how close our fast bounds are compared to the true minimum distortion provided by **Reluplex** and the bounds that are slightly tighter but very expensive (e.g. **LP-Full**).

Toy Networks			Average Magnitude of Distortions on 100 Images							
			Certified Bounds				difference	Exact	Uncertified	
Network	p	Target	Our bounds		Our baselines		ours vs.	Reluplex	CLEVER	Attacks
			Fast-Lin	Fast-Lip	LP	LP-Full	LP(-Full)	(Katz et al., 2017)	(Weng et al., 2018)	CW/EAD
MNIST $2 \times [20]$	∞	runner-up	**0.0191**	0.0167	**0.0197**	0.0197	-3.0%	0.04145	0.0235	0.04384
		rand	**0.0309**	0.0270	**0.0319**	0.0319	-3.2%	0.07765	0.0428	0.08060
		least	**0.0448**	0.0398	**0.0462**	0.0462	-3.1%	0.11711	0.0662	0.1224
	2	runner-up	**0.3879**	0.3677	0.4811	**0.5637**	-31.2%	-	0.4615	0.64669
		rand	**0.6278**	0.6057	0.7560	**0.9182**	-31.6%	-	0.8426	1.19630
		least	**0.9105**	0.8946	1.0997	**1.3421**	-32.2%	-	1.315	1.88830
	1	runner-up	2.3798	**2.8086**	2.5932	**2.8171**	-0.3%	-	3.168	5.38380
		rand	3.9297	**4.8561**	4.2681	**4.6822**	+3.7%	-	5.858	11.4760
		least	5.7298	**7.3879**	6.2062	**6.8358**	+8.1%	-	9.250	19.5960
MNIST $3 \times [20]$	∞	runner-up	**0.0158**	0.0094	0.0168	**0.0171**	-7.2%	0.04234	0.0223	0.04786
		rand	**0.0229**	0.0142	0.0241	**0.0246**	-6.9%	0.06824	0.0385	0.08114
		least	**0.0304**	0.0196	0.0319	**0.0326**	-6.9%	0.10449	0.0566	0.11213
	2	runner-up	**0.3228**	0.2142	0.3809	**0.4901**	-34.1%	-	0.4231	0.74117
		rand	**0.4652**	0.3273	0.5345	**0.7096**	-34.4%	-	0.7331	1.22570
		least	**0.6179**	0.4454	0.7083	**0.9424**	-34.4%	-	1.100	1.71090
	1	runner-up	**2.0189**	1.8819	2.2127	**2.5010**	-19.3%	-	2.950	6.13750
		rand	**2.8550**	2.8144	3.1000	**3.5740**	-20.1%	-	4.990	10.7220
		least	3.7504	**3.8043**	4.0434	**4.6967**	-19.0%	-	7.131	15.6850

(a) Comparison of bounds

Toy Networks			Average Running Time per Image					
			Certified Bounds				Exact	Speedup
Network	p	Target	Our bounds		Our baselines		Reluplex	ours vs.
			Fast-Lin	Fast-Lip	LP	LP-Full	(Katz et al., 2017)	LP-(full)
MNIST $2 \times [20]$	∞	runner-up	**3.09 ms**	3.49 ms	**217 ms**	1.74 s	134 s	70X
		rand	**3.25 ms**	5.53 ms	**234 ms**	1.93 s	38 s	72X
		least	**3.37 ms**	8.90 ms	**250 ms**	1.97 s	360 s	74X
	2	runner-up	**3.00 ms**	3.76 ms	1.10 s	**20.6 s**	-	6864X
		rand	**3.37 ms**	6.16 ms	1.20 s	**23.1 s**	-	6838X
		least	**3.29 ms**	9.89 ms	1.27 s	**26.4 s**	-	8021X
	1	runner-up	2.85 ms	**39.2 ms**	1.27 s	**16.1 s**	-	412X
		rand	3.32 ms	**54.8 ms**	1.59 s	**17.3 s**	-	316X
		least	3.46 ms	**68.1 ms**	1.74 s	**17.7 s**	-	260X
MNIST $3 \times [20]$	∞	runner-up	**5.58 ms**	3.64 ms	253 ms	**6.12 s**	4.7 hrs	1096X
		rand	**6.12 ms**	5.23 ms	291 ms	**7.16 s**	11.6 hrs	1171X
		least	**6.62 ms**	7.06 ms	307 ms	**7.30 s**	12.6 hrs	1102X
	2	runner-up	**5.35 ms**	3.95 ms	1.22 s	**57.5 s**	-	10742X
		rand	**5.86 ms**	5.81 ms	1.27 s	**66.3 s**	-	11325X
		least	**5.94 ms**	7.55 ms	1.34 s	**77.3 s**	-	13016X
	1	runner-up	**5.45 ms**	39.6 ms	1.27 s	**75.0 s**	-	13763X
		rand	**5.56 ms**	52.9 ms	1.47 s	**82.0 s**	-	14742X
		least	6.07 ms	**65.9 ms**	1.68 s	**85.9 s**	-	1304X

(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.

Large Networks			Average Magnitude of Distortion on 100 Images							Average Running Time per Image			
			Certified Bounds				diff	Uncertified		Certified Bounds			Speedup
Network	p	Target	Our bounds		LP	Op-norm	ours	CLEVER	Attacks	Our bounds		LP	ours
			Fast-Lin	Fast-Lip	(Baseline)	(Szegedy et al., 2013)	vs. LP	(Weng et al., 2018)	CW/EAD	Fast-Lin	Fast-Lip	(Baseline)	vs. LP
MNIST $2 \times [1024]$	∞	runner-up	**0.02256**	0.01802	**0.02493**	0.00159	-9.5%	0.0447	0.0856	**127 ms**	167 ms	**19.3 s**	151X
		rand	**0.03083**	0.02512	**0.03386**	0.00263	-8.9%	0.0708	0.1291	**156 ms**	219 ms	**20.8 s**	133X
		least	**0.03854**	0.03128	**0.04281**	0.00369	-10.0%	0.0925	0.1731	**129 ms**	377 ms	**22.2 s**	172X
	2	runner-up	**0.46034**	0.42027	**0.55591**	0.24327	-17.2%	0.8104	1.1874	**127 ms**	196 ms	**419 s**	3305X
		rand	**0.63299**	0.59033	**0.75164**	0.40201	-15.8%	1.2841	1.8779	**128 ms**	234 ms	**195 s**	1523X
		least	**0.79263**	0.73133	**0.94774**	0.56509	-16.4%	1.6716	2.4556	**163 ms**	305 ms	**156 s**	956X
	1	runner-up	2.78786	**3.46500**	**3.21866**	0.20601	+7.7%	4.5970	9.5295	117 ms	**1.17 s**	**38.9 s**	33X
		rand	3.88241	**5.10000**	**4.47158**	0.35957	+14.1%	7.4186	17.259	139 ms	**1.40 s**	**48.1 s**	34X
		least	4.90809	**6.36600**	**5.74140**	0.48774	+10.9%	9.9847	23.933	151 ms	**1.62 s**	**53.1 s**	33X
MNIST $3 \times [1024]$	∞	runner-up	**0.01830**	0.01021	**0.02013**	0.00004	-9.1%	0.0509	0.1037	**1.20 s**	1.81 s	**50.4 s**	42X
		rand	**0.02216**	0.01236	**0.02428**	0.00007	-8.7%	0.0717	0.1484	**1.12 s**	1.11 s	**52.7 s**	47X
		least	**0.02432**	0.01384	**0.02665**	0.00009	-8.7%	0.0825	0.1777	**1.02 s**	924 ms	**54.3 s**	53X
	2	runner-up	**0.35867**	0.22120	**0.41040**	0.06626	-12.6%	0.8402	1.3513	**898 ms**	1.59 s	**438 s**	487X
		rand	**0.43892**	0.26980	**0.49715**	0.10233	-11.7%	1.2441	2.0387	**906 ms**	914 ms	**714 s**	788X
		least	**0.48361**	0.30147	**0.54689**	0.13256	-11.6%	1.4401	2.4916	**925 ms**	1.01 s	**858 s**	928X
	1	runner-up	**2.08887**	1.80150	**2.36642**	0.00734	-11.7%	4.8370	10.159	**836 ms**	3.16 s	**91.1 s**	109X
		rand	**2.59898**	2.25950	**2.91766**	0.01133	-10.9%	7.2177	17.796	**863 ms**	3.84 s	**109 s**	126X
		least	**2.87560**	2.50000	**3.22548**	0.01499	-10.8%	8.3523	22.395	**900 ms**	4.20 s	**122 s**	136X
MNIST $4 \times [1024]$	∞	runner-up	**0.00715**	0.00219	-	0.00001	-	0.0485	0.08635	**1.90 s**	4.58 s	-	-
		rand	**0.00823**	0.00264	-	0.00001	-	0.0793	0.1303	**2.25 s**	3.08 s	-	-
		least	**0.00899**	0.00304	-	0.00001	-	0.1028	0.1680	**2.15 s**	3.02 s	-	-
	2	runner-up	**0.16338**	0.05244	-	0.11015	-	0.8689	1.2422	**2.23 s**	3.50 s	-	-
		rand	**0.18891**	0.06487	-	0.17734	-	1.4231	1.8921	**2.37 s**	2.72 s	-	-
		least	**0.20672**	0.07440	-	0.23710	-	1.8864	2.4451	**2.56 s**	2.77 s	-	-
	1	runner-up	**1.33794**	0.58480	-	0.00114	-	5.2685	10.079	**2.42 s**	2.71 s	-	-
		rand	**1.57649**	0.72800	-	0.00183	-	8.9764	17.200	**2.42 s**	2.91 s	-	-
		least	**1.73874**	0.82800	-	0.00244	-	11.867	23.910	**2.54 s**	3.54 s	-	-
CIFAR $5 \times [2048]$	∞	runner-up	**0.00137**	0.00020	-	0.00000	-	0.0062	0.00950	**24.2 s**	60.4 s	-	-
		rand	**0.00170**	0.00030	-	0.00000	-	0.0147	0.02351	**26.2 s**	78.1 s	-	-
		least	**0.00188**	0.00036	-	0.00000	-	0.0208	0.03416	**27.8 s**	79.0 s	-	-
	2	runner-up	**0.06122**	0.00951	-	0.00156	-	0.2712	0.3778	**34.0 s**	60.7 s	-	-
		rand	**0.07654**	0.01417	-	0.00333	-	0.6399	0.9497	**36.8 s**	49.4 s	-	-
		least	**0.08456**	0.01778	-	0.00489	-	0.9169	1.4379	**37.4 s**	49.8 s	-	-
	1	runner-up	**0.93835**	0.22632	-	0.00000	-	4.0755	7.6529	**36.5 s**	70.6 s	-	-
		rand	**1.18928**	0.31984	-	0.00000	-	9.7145	21.643	**37.5 s**	53.6 s	-	-
		least	**1.31904**	0.38887	-	0.00001	-	12.793	34.497	**38.3 s**	48.6 s	-	-
CIFAR $6 \times [2048]$	∞	runner-up	**0.00075**	0.00005	-	0.00000	-	0.0054	0.00770	**37.2 s**	106 s	-	-
		rand	**0.00090**	0.00007	-	0.00000	-	0.0131	0.01866	**37.0 s**	119 s	-	-
		least	**0.00095**	0.00008	-	0.00000	-	0.0199	0.02868	**37.2 s**	126 s	-	-
	2	runner-up	**0.03463**	0.00228	-	0.00476	-	0.2394	0.2979	**56.1 s**	99.5 s	-	-
		rand	**0.04129**	0.00331	-	0.01079	-	0.5860	0.7635	**60.2 s**	95.6 s	-	-
		least	**0.04387**	0.00385	-	0.01574	-	0.8756	1.2111	**61.8 s**	88.6 s	-	-
	1	runner-up	**0.59638**	0.05647	-	0.00000	-	3.3569	6.0112	**57.2 s**	108 s	-	-
		rand	**0.72178**	0.08212	-	0.00000	-	8.2507	17.160	**61.4 s**	88.2 s	-	-
		least	**0.77179**	0.09397	-	0.00000	-	12.603	28.958	**62.1 s**	65.1 s	-	-
CIFAR $7 \times [1024]$	∞	runner-up	**0.00119**	0.00006	-	0.00000	-	0.0062	0.0102	**10.5 s**	27.3 s	-	-
		rand	**0.00134**	0.00008	-	0.00000	-	0.0112	0.0218	**10.6 s**	29.2 s	-	-
		least	**0.00141**	0.00010	-	0.00000	-	0.0148	0.0333	**11.2 s**	30.9 s	-	-
	2	runner-up	**0.05279**	0.00308	-	0.00020	-	0.2661	0.3943	**16.3 s**	28.2 s	-	-
		rand	**0.05938**	0.00407	-	0.00029	-	0.5145	0.9730	**16.9 s**	27.3 s	-	-
		least	**0.06249**	0.00474	-	0.00038	-	0.6253	1.3709	**17.4 s**	27.6 s	-	-
	1	runner-up	**0.76647**	0.07028	-	0.00000	-	4.815	7.9987	**16.9 s**	27.8 s	-	-
		rand	**0.86467**	0.09239	-	0.00000	-	8.630	22.180	**17.6 s**	26.7 s	-	-
		least	**0.91127**	0.10639	-	0.00000	-	11.44	31.529	**17.5 s**	23.5 s	-	-
MNIST $3 \times [1024]$	∞	untargeted	**0.01808**	0.01016	**0.01985**	0.00004	-8.9%	0.0458	0.0993	**915 ms**	2.17 s	**227 s**	248X
	2		**0.35429**	0.21833	-	0.06541	-	0.7413	1.1118	**950 ms**	2.02 s	-	-
	1		**2.05645**	1.78300	**2.32921**	0.00679	-11.7%	3.9661	9.0044	**829 ms**	4.41 s	**537 s**	648X
CIFAR $5 \times [2048]$	∞	untargeted	**0.00136**	0.00020	-	0.00000	-	0.0056	0.00950	**24.1 s**	72.9 s	-	-
	2		**0.06097**	0.00932	-	0.00053	-	0.2426	0.3702	**34.2 s**	77.0 s	-	-
	1		**0.93429**	0.22535	-	0.00000	-	3.6704	7.3687	**35.6 s**	90.2 s	-	-