A. Proofs of Lemmas and Theorems

Theorem 1. *For feedforward neural networks f, an input set $\mathcal{X}$ and sets $\mathcal{Z}_i, \hat{\mathcal{Z}}_i, \mathcal{J}_i, \mathcal{Y}_i, \hat{\mathcal{Y}}\rangle$ satisfying the containments in Equations 14-17, the set of vector-Jacobian products satisfies*

$$\{\nabla_x f(x)^T u \mid x \in \mathcal{X} \quad u \in B_{\beta^*}\} \subseteq \mathcal{Y}_0. \tag{18}$$

For such a $\mathcal{Y}_0$, the Lipschitz constant of f may be upper-bounded by maximizing the $\|\cdot\|_{\alpha^*}$ norm over the set $\mathcal{Y}_0$.

Proof. Suppose $\mathcal{Z}_i, \hat{\mathcal{Z}}_i, \mathcal{J}_i, \mathcal{Y}_i, \hat{\mathcal{Y}}_i$ satisfy the containments is equation 14-17. Now consider any $x \in \mathcal{X}$ and $u \in B_{\beta^*}$. The proof follows from repeated applications of the following statement: for any function g, if $\mathcal{A} \subseteq \mathcal{B}$, then $g(\mathcal{A}) \subseteq g(\mathcal{B})$. We iteratively apply this statement to the forward recursion to see that $\hat{Z}_i(x) \in \hat{\mathcal{Z}}_i$ for all i, and similarly for $Z_i(x) \in \mathcal{Z}_i$. From equation 17, $J_{i+1}^T(x) \in \mathcal{J}_{i+1}$ for all i. We may now perform the backward recursion to see that $Y_i(x, u) \in \mathcal{Y}_i$ and similarly for $\hat{Y}_i(x, u), \hat{\mathcal{Y}}_i$. Repeating this for all i yields the desired result. □

Lemma 1. *For any zonotope $Z \subset \mathbb{R}^d$ and any operator Φ operating over $\mathbb{R}^d$, if $\hat{Z}$ is a zonotope satisfying the containment*

$$\{\Phi(z) - \Lambda \odot z \mid z \in Z\} \subseteq \hat{Z} \tag{19}$$

then

$$\Phi(Z) \subseteq (\Lambda \odot Z) \oplus \hat{Z}.$$

Proof. By assumption, for every $z \in Z$, there exists a $\hat{z} \in \hat{Z}$ such that $\Phi(z) - \Lambda \odot z = \hat{z}$. This implies that, $\Phi(z) = (\Lambda \odot z) + \hat{z}$. By definition,

$$(\Lambda \odot Z) \oplus \hat{Z} := \{\Lambda \odot z + \hat{z} \mid z \in Z \quad \hat{z} \in \hat{Z}\},$$

so $\Phi(z) \in (\Lambda \odot Z) \oplus \hat{Z}$ for every $z \in Z$. □

Theorem 2. *When S is the set $\{(z_i, \phi(z_i)) \mid z_i \in [l_i, u_i]\}$, the solution to the vertical parallelogram fitting problem yields the optimal solution to Equation 20. Repeated calls to this subroutine yields the tightest hyperbox fitting the residuals as in equation 19.*

Proof. Suppose that $\{\Lambda_i, b_i, \mu_i\}_{i=1}^d$ is the set of solutions to the vertical parallelogram fitting problem for each set $S_i = \{(z_i, \phi(z_i) \mid z_i \in [l_i, u_i]\}$. Since the coordinate-wise bounds l_i, u_i are chosen such that $l \leq z \leq u$ for all $z \in Z$, the containment holds:

$$\{\Phi(z) - \Lambda \odot z - b \mid z \in \mathcal{Z}\} \subseteq \{\Phi(z) - \Lambda \odot z - b \mid l \leq z \leq u\}.$$

By definition of solutions to the vertical parallelogram fitting problem, the set of vectors $\{\Phi(z) - \Lambda \odot z - b \mid l \leq z \leq u\}$ is contained in the hyperbox $H(0, \mu)$. Adding b to each element of each set, we see that $\{\Phi(z) - \Lambda \odot z \mid l \leq z \leq u\}$ is contained in the hyperbox $H(b, \mu)$, thus satisfying the assumptions of Lemma 1. □

Lemma 2. *For any zonotope $Z \subseteq \mathbb{R}^d$ and hyperbox $H \subseteq \mathbb{R}^d$, if $\hat{Z}$ is a zonotope satisfying the containment*

$$\{x \odot z - \Lambda \odot z \mid x \in H \quad z \in Z\} \subseteq \hat{Z} \tag{21}$$

then

$$\{x \odot z \mid x \in H \quad z \in Z\} \subseteq (\Lambda \odot Z) \oplus \hat{Z}.$$

Proof. By assumption, for every $z \in Z$ and every $x \in H$, there exists a $\hat{z} \in \hat{Z}$ such that $x \odot z - \Lambda \odot z = \hat{z}$. This implies that, $x \odot z = (\Lambda \odot z) + \hat{z}$. And by definition

$$(\Lambda \odot Z) \oplus \hat{Z} := \{\Lambda \odot z + \hat{z} \mid z \in Z \quad \hat{z} \in \hat{Z}\},$$

so $x \odot z \in (\Lambda \odot Z) \oplus \hat{Z}$ for every $z \in Z$. □

Theorem 3. *When S is the set $\{(z, x \cdot z) \mid l^{(z)} \leq z \leq u^{(z)} \quad l^{(x)} \leq x \leq u^{(x)}\}$, the solution to the vertical-parallelogram fitting problem yields the optimal solution to Equation 22. Repeated calls to this subroutine yields the tightest hyperbox fitting the residuals as in Equation 21.*

Proof. Suppose that $\{(\Lambda_i, b_i, \mu_i)\}_{i=1}^d$ is the set of solutions to the vertical parallelogram fitting problem for each set $S_i = \{(z_i, x_i \odot z_i) \mid l_i^{(z)} \leq z_i \leq u_i^{(z)}, \quad l_i^{(x)} \leq x_i \leq u_i^{(x)}\}$. Let H be the hyperbox with lower and upper-bounds denoted by $l^{(x)}, u^{(x)}$. Since the coordinate-wise bounds l_i, u_i are chosen such that $l^{(z)} \leq z \leq u^{(z)}$ for all $z \in Z$, the containment holds:

$$\{x \odot z - \Lambda \odot z - b \mid z \in \mathcal{Z} \quad x \in H\} \subseteq \{\Phi(z) - \Lambda \odot z - b \mid l^{(z)} \leq z \leq u^{(z)} \quad x \in \mathcal{H}\}.$$

By definition of solutions to the vertical parallelogram fitting problem, the set of vectors $\{x \odot (z) - \Lambda \odot z - b \mid l^{(z)} \leq z \leq u^{(z)} \quad x \in H\}$ is contained in the hyperbox $H(0, \mu)$. Adding b to each element of each set, we see that $\{x \odot z - \Lambda \odot z \mid l \leq z \leq u \quad x \in H\}$ is contained in the hyperbox $H(b, \mu)$, thus satisfying the assumptions of Lemma 2. □

Theorem 4. *The problem of computing the maximal ℓ_1 norm of a zonotope is equivalent to the $\|\cdot\|_{\infty\to 1}$ matrix norm: both problems are NP-hard in general. Additionally, any approximation algorithm with approximation ratio α for the Grothendieck problem will yield an approximation algorithm with ratio α for the zonotope ℓ_1 maximization problem and vice versa.*

Proof. We prove this via a strict reduction in showing that any instance of one problem may be converted into an instance of the other and will keep the same optimal value. To do this, we first note that for any matrix M, with $(0||M)$ denoting the zero-column prepended to the columns of M, that $\|M\|_{\infty\to 1} = \|(0||M)\|_{\infty\to 1}$. This follows since

$$\|(0||M)\|_{\infty\to 1} = \max_{v \in B_\infty} \|(0||M)v\|_1 = max_{u \in B_\infty} \|Mu\| = \|M\|_{\infty\to 1}.$$

Next, it suffices to show that

$$\max_{z \in Z(c,E)} \|z\|_1 = \|(c||E)\|_{\infty\to 1}, \tag{23}$$

for if this were true, certainly any zonotope could be reduced to a matrix-norm maximization problem, and any matrix norm problem could first prepend the zero column to the matrix and be reduced to a zonotope norm-maximization problem. Any α-approximation algorithm for one problem could provide an α-approximation for any instance of the other via this reduction.

First we show that $\max_{z \in Z(c,E)} \|z\|_1 \leq \|(c||E)\|_{\infty\to 1}$. As the right-hand-side may be written

$$\|(c||E)\|_{\infty\to 1} = \max_{|v_0| \leq 1} \max_{v \in B_\infty} \|v_0 \cdot c + Ev\|_1$$

and whereas the left-hand side of Equation 23 is the same optimization with v_0 restricted to 1. Therefore the $(\leq)$ direction of Equation 23 holds. For the other direction, consider any integral solution to the RHS,

$$(v_0^*, v^*) \in \arg\max_{|v_0| \leq 1, v \in B_\infty} \|v_0 \cdot c + Ev\|_1.$$

Without loss of generality, v_0^* may be chosen to be 1, and the point $(c + Ev^*)$ is in $Z(c, E)$. Hence there's a point in $Z(c, E)$ with ℓ_1 norm at least that of $\|(c||E)\|_{\infty\to 1}$, thus proving the $(\geq)$ direction of equality 23. □

Theorem 5. *For a zonotope, $Z(c, E)$, the linear programming relaxation of $\max_{z \in Z(c,E)} \|z\|_1$ is computable in time $O(|E|)$ where $|E|$ denotes the number of elements in E.*

Proof. First we write down the Linear-programming relaxation of the zonotope-norm maximization problem and then relate this to the mapping of the zonotope through the absolute value operator, by our vertical-parallelogram fitting procedure. The final result follows from linear programs being efficiently solvable over zonotopes.

Consider some zonotope $Z(c, E) \subseteq \mathbb{R}^d$ which has coordinate-wise upper and lower bounds $[l_i, u_i]$ for every $i \in [d]$. We partition the coordinates into three sets of indices: S^-, S^+, S such that $S^- := \{i \mid u_i \leq 0\}$, $S^+ := \{i \mid l_i > 0\}$ and S is the set of indices not in either S^- or S^+. We may write down the familiar mixed-integer programming relaxation for the absolute value operator by introducing $|S|$ continuous variables, $\{t_i\}_{i\in S}$, and d integer variables $\{a_i\}_{i\in S}$, where $t_i \in \mathbb{R}$ and $a_i \in \{0, 1\}$:

$$\max \sum_{i\in S} t_i - \sum_{i\in S^-} z_i + \sum_{i\in S^+} z_i \tag{24}$$

$$t_i \geq z_i \tag{25}$$

$$t_i \geq -z_i \tag{26}$$

$$t_i \leq -z_i + 2 \cdot u_i \cdot a_i \tag{27}$$

$$t_i \leq z_i - 2 \cdot l_i \cdot (1 - a_i) \tag{28}$$

$$a_i \in \{0, 1\} \tag{29}$$

$$z \in Z(c, E) \tag{30}$$

Where the constraints enforce that $t_i = |z_i|$. The first two constraints require that $t_i \geq |z_i|$. To show $t_i \leq |z_i|$, we proceed by cases. When $z_i > 0$, then 27 implies that $a_i = 1$, for otherwise $t_i < 0$ contradicting the first constraint. This causes 28 to imply $t_i \leq z_i$. When $z_i < 0$, 28, $a_i = 0$, for otherwise 27 again implies that $t_i < 0$. This causes 27 to imply $t_i \leq -z_i$. When $z_i = 0$, either case can hold and $t_i = 0$. The linear programming relaxation lets a_i be in the range $[0, 1]$ instead of $\{0, 1\}$.

For any fixed z_i, we can compute the maximum value of t_i under this relaxation, which is a function of the now-continuous variable, a_i. By setting the upper bounds to equality, the optimal value of a_i is $a_i = \frac{z_i - l_i}{u_i - l_i}$ and t is then upper bounded by

$$t_i \leq \frac{-z_i + 2u_i \cdot (z_i - l_i)}{u_i - l_i}$$

. We observe that this is an equivalent relaxation to the upper-hull provided by the absolute value operator and our vertical-parallelogram fitting procedure (next section). This allows us to rewrite the optimization above as

$$\max_{z\in Z(c,E)} \sum_{i\in S} \frac{-z_i + 2u_i \cdot (z_i - l_i)}{u_i - l_i} - \sum_{i\in S^-} z_i + \sum_{i\in S^+} z_i$$

which we notice is a linear program over a zonotope. The objective vector may be developed in $O(d)$ time, and linear programs may be solvable over zonotopes in $O(|E|)$ time. □

B. Pseudocode

Algorithm 1 ZLip

Require: L-layer feedforward neural network f, input set $\mathcal{X}$, norm β
Returns: Zonotope $\mathcal{Y}_0 \supseteq \{\nabla_x f(x)^T v \mid x \in \mathcal{X}, \quad v \in B_{\beta^*}\}$

```
  function ZLip($f, \mathcal{X}, \beta$)
      $\mathcal{Z}_0 \leftarrow$ Zonotope($\mathcal{X}$)                          ▷ Cast input set to zonotope
      for $i \leftarrow 1$ to $L$ do                                     ▷ Forward pass (e.g., DeepZ)
          $\hat{\mathcal{Z}}_i \leftarrow$ map_affine($W_i, b_i, \mathcal{Z}i - 1$)
          $\mathcal{Z}_i \leftarrow$ map_nonlin($\sigma, \hat{\mathcal{Z}}_i$)
          $\mathcal{J}_i \leftarrow$ elementwise_jacobian($\sigma, \mathcal{Z}_i$)      ▷ Gradient range for $\nabla_z \sigma(\mathcal{Z}_i)$
      end for
      $\hat{\mathcal{Y}}_L \leftarrow$ Zonotope($\mathcal{B}_{\beta^*}$)                      ▷ Cast dual ball to zonotope
      for $i \leftarrow L$ to 1 do                                      ▷ Backward pass
          $\mathcal{Y}_{i-1} \leftarrow$ map_affine($W_i^T, 0, \hat{\mathcal{Y}}_i$)
          $\hat{\mathcal{Y}}_{i-1} \leftarrow$ elementwise_mul($\mathcal{J}_i, \mathcal{Y}_{i-1}$)
      end for
      $\hat{\mathcal{Y}}_0 \leftarrow$ map_affine($W_0^T, 0, \hat{\mathcal{Y}}_0$)
      return $\hat{\mathcal{Y}}_0$
  end function
```

Algorithm 2 Vertical Parallelogram Fitting

Require: Function $\sigma : \mathbb{R} \to \mathcal{P}(\mathbb{R})$, and interval $[c - |E|, c + |E|]$
Returns: Slope Λ^*, Altitude μ^*, center b^*

```
  function VP_Fit($\sigma, c, E$)
      $\mathcal{I} \leftarrow c \pm |E|$
      S $\leftarrow \{(x, \sigma(x)) \mid x \in \mathcal{I}\}$
      $h^-, h^+ \leftarrow$ conv_hull($S$)                          ▷ Possibly hard, depends on $\sigma$
      $x^* \leftarrow \text{argmax}_{x \in \mathcal{I}}\, h^+(x) - h^-(x)$
      $\mu^* \leftarrow h^+(x^*) - h^-(x^*)$                                        ▷ Altitude
      $\Lambda^* \leftarrow \delta(h^-(x^*)) \cap \delta(-h^+(x^*))$        ▷ Slope of parallogram's non-vert side
      $b^* \leftarrow \frac{1}{2} \cdot (h^+(x^*) + h^-(x^*))$                            ▷ Intercept
  return $\Lambda^*, \mu^*, b^*$
  end function
```

Algorithm 3 Auxiliary Functions

```
function MAP_NONLIN($\sigma, Z(c, E)$)                    ▷ $Z(c,E) \subseteq \mathbb{R}^d$, $\sigma$ is elementwise
    for $i \leftarrow 1$ to $d$ do
        $\Lambda_i, b_i^*, \mu_i^* \leftarrow$ VP_Fit($\sigma_i, c_i, E_i^T$)
    end for
    return $Z(b^*, \texttt{diag}(\mu^*)) \oplus (\Lambda \odot Z(c, E))$
end function

function ELEMENTWISE_MUL($H(l, u), Z(c, E)$)
    for $i \leftarrow 1$ to $d$ do
        $\Lambda_i, b_i^*, \mu_i^* \leftarrow$ VP_Fit($[l_i, u_i], c_i, E_i^T$)        ▷ Overloading VP_Fit signature
    end for
    return $Z(b^*, \texttt{diag}(\mu^*)) \oplus (\Lambda \odot Z(c, E))$
end function
```

C. Detailed Derivations for Vertical Parallelogram Fitting

We recall the algorithm from Section 6 for fitting a vertical parallelogram to a 2-dimensional set S. The first step was to compute the upper and lower convex hulls of S. For sets of the form $\{(x, f(x)) \mid x \in [l, u]\}$ for some differentiable function, this is equivalent to the biconjugate and the biconjugate of the negation of f. In this case, there exists a simple algorithm to yield the upper-convex hull. The lower-convex hull may be found the same way, but for the set $\{(x, -f(x))\}$. Observe that if f is convex over $[l, u]$, then the upper convex hull is the secant line between the endpoints $(l, f(l))$ and $(u, f(u))$ and the lower-convex hull is $f(x)$; vice versa for concave functions. For functions that are neither convex or concave over $[l, u]$, the upper convex hull may be piecewise continuous, alternating between secant-line segments and f. For function f, let the secant line of f between x_1 and x_2 be denoted as $Sec^f(x_1, x_2)$.

We take inspiration from the gift-wrapping procedure for finding convex hulls of finite 2-dimensional point sets. In gift-wrapping, the idea is to find the left-most point in the set and sweep a ray clockwise until an intersection with another point in the set is found. The sweep continues, with ray now starting at the newly intersected point until the left-most point is intersected again. Our procedure sweeps performs a sweep over rays of decreasing slope, noting that any intersecting point must lie on the set S and thus the ray is a secant line. Hence, the key subroutine to find the upper hull is to solve a maximization over slopes of secant lines. For a fixed x_0, the slope of the secant line $Sec^f(x_0, x)$ is $\frac{f(x)-f(x_0)}{x-x_0}$, and the maximization we seek to solve is a constrained variant of this,

$$\max_{x \in [x_0, u]} \frac{f(x) - f(x_0)}{x - x_0}. \tag{31}$$

When f is differentiable, then we can differentiate the above objective and set to zero and solve for x in the equality

$$f'(x) = \frac{f(x) - f(x_i)}{x - x_i} \tag{32}$$

This procedure may be repeated until the max is attained at an $x \geq u_i$, for which the final secant line spans between $(x_i, f(x_i))$ and $(u, f(u))$.

Once piecewise forms for the convex upper and lower hulls are formed, the maximal altitude can be computed by maximizing the piecewise function $h^+(x) - h^-(x)$. The proper slope for the tightest fitting vertical parallelogram is attained by considering an element in the intersection of subgradients of h^- and $-h^+$ at their maximal altitude.

C.1. Sigmoid/Tanh

We demonstrate the above procedure for finding convex upper and lower hulls of sets $\{(x, f(x) \mid x \in [l, u]\}$ where f is S-shaped like sigmoid and tanh. We say a function f is S-shaped if it is monotonically increasing and there exists an x' such that f is concave for all $x \geq x'$. We break this into cases, based on the values of $[l, u]$. If f is either convex or concave over the entire interval $[l, u]$, then the upper hull is either the secant line or f respectively, and vice versa for the lower hull. In this case, the maximum altitude is attained at the x where $f'(x)$ is equal to the slope of this secant line.

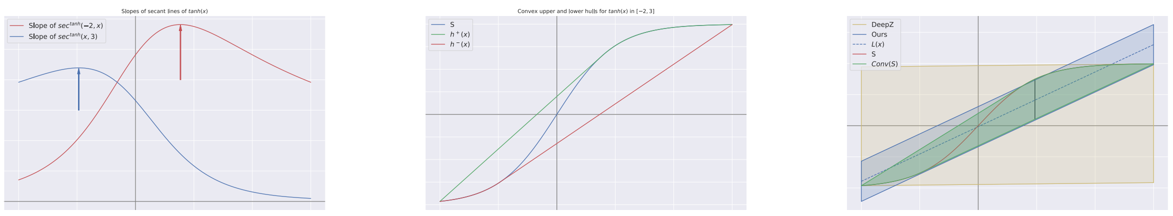

Figure 3. Procedure for computing the tightest fitting vertical-parallelogram to the set $S = \{(x, tanh(x)) \mid x \in [-2, 3]\}$. (Left) We plot the slope of the secant-line from -2 to x in red and the secant line from x to 3 in blue, where the maximum is marked with a vertical hash. (Middle) We plot the upper and lower convex hulls for S. (Right) We plot the convex hull of S and the parallelogram we produce versus that of prior work.

For cases where f is both convex and concave for portions of $[l, u]$, then we proceed with the iterative giftwrap procedure, starting at $x_0 = l$. For monotonically increasing tanh and sigmoid functions, this means that $l < 0 < u$ and we seek to find a point in the concave ($x \geq 0$) portion of f such that Equation 32 is satisfied. Since f is concave for $x \geq 0$, the function

$$f'(x) - \frac{f(x) - f(l)}{x - l}$$

only has one zero for $x \geq 0$, which may be found numerically[1]. Letting x^* be such an x, we have that the secant-line is given by $y = \frac{f(x^*)-f(l)}{x^*-l}(x - l) + f(l)$ and the upper convex hull for is

$$h^+(x) = \begin{cases} \frac{f(x^*)-f(l)}{x^*-l}(x - l) + f(l) & \text{if } x \in [l, x^*] \\ f(x) & \text{if } x \in [x^*, u] \end{cases}$$

A similar procedure may be considered for the lower convex hull, also yielding a convex hull like

$$h^-(x) = \begin{cases} f(x) & \text{if } x \in [l, x^\dagger] \\ \frac{f(x^\dagger)-f(l)}{x^\dagger-l}(x - l) + f(l) & \text{if } x \in [x^\dagger, u] \end{cases}$$

where $x^\dagger$ is the minimum amongst zeros for the function $f'(x) - \frac{f(x)-f(u)}{x-u}$. Then the altitude, $h^+(x) - h^-(x)$, is a concave piecewise function with three pieces, segmented into the intervals $[l, x^\dagger]$, $[x^\dagger, x^*]$, $[x^*, u]$. If the slope of the linear component of the upper hull is λ^+ and the slope of the linear component of the lower hull is λ^-, then the maximum is attained at one of four points: i) the point in $[l, x^\dagger]$ where $f'(x) = \lambda^-$; ii) the point in $[x^*, u]$ where $f'(x) = \lambda^+$; iii) x^*, or iv) $x^\dagger$. It is trivial to check all these points where f is the sigmoid or tanh function.

Regardless of which case we are in, once the upper and lower hulls have been computed and their difference has been maximized, the remaining steps are simple. It suffices to compute the subgradients of the lower hull and negative upper hull at that point, pick an element of their intersection and find the line that passes through the proper midpoint. This yields the right altitude and affine function for a vertical parallelogram.

C.2. Elementwise Multiplication

Now we consider the full set of cases for elementwise multiplication. Letting S be the 2-dimensional set $\{(x, y \cdot x) \mid x \in [l, u], \quad y \in [\alpha, \beta]\}$. When $\alpha = \beta$, then the set S is a line segment, and has vertical parallelogram with height 0 and slope α. When $\alpha < \beta$, and $l > 0$ the convex upper hull is $h^+(x) = \beta x$ and the convex lower hull is $h^-(x) = \alpha x$; vice versa for when $u < 0$. The only remaining case is when $\alpha < \beta$, and $l < 0 < u$, which is the case considered in the main paper, where $h^+(x)$ is the line passing through $(l, \alpha l)$ and $(u, \beta u)$; and the lower hull passes through $(l, \beta l)$ and $(u, \alpha u)$.

In all of the above cases, the upper and lower hulls are affine functions and their maximum over $[l, u]$ is attained at either l or u. The admissable slopes are exactly the range $[\alpha, \beta]$.

C.3. Absolute Value

[1]We keep track of the numerical error and ensure that the vertical height of the parallelogram accounts for this.

Now we consider the absolute value function, as used in the proof of Theorem 5. Consider the set $\{(x, |x|) \mid x \in [l, u]\}$. When $l \cdot u \geq 0$, then the set S lies on either the $y = x$ or $y = -x$ line, and the vertical parallelogram fit is equivalent to mapping through an affine function. On the other hand, when $l < 0 < u$, the set S has a lower convex hull of $h^-(x) = |x|$, because $|x|$ is a convex function. The upper hull is the secant line connecting $(l, |l|)$ and $(u, |u|)$, which may be written as $h^+(x) = \frac{u+l}{u-l}(x) - \frac{2ul}{u-l}$. The maximum of the altitude, $h^+(x) - h^-(x)$, is attained at $x = 0$, for which the maximum altitude is $-\frac{2ul}{u-l}$ and the height of the vertical parallelogram is $\mu = \frac{-ul}{u-l}$. The slope must be the slope of the (linear) upper hull, $\frac{u+l}{u-l}$ and the central line of the parallelogram must pass through the point $(0, \frac{-ul}{u-l})$.

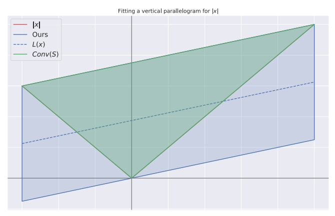

Figure 4. Convex hull and vertical parallelogram for the set $S = \{(x, |x|) \mid x \in [-3, 5]\}$.

D. Experiments

D.1. Model Architectures

Here we will describe the structure of each of the architectures considered. For networks with only fully connected and elementwise nonlinearities, we denote the architectures by $[n_{in}, n_1, \dots n_L \dots n_{out}]$, where n_i denotes the number of neurons in the i^{th} hidden layer, and ReLU nonlinearities are implied between each layer. We will use the notation "FC X" to denote fully connected layers with an output of X neurons, and $\text{Conv}_s(C \times W \times H)$ to denote convolutional layers with a stride of s and output dimension of C channels, and kernels of size $W \times H$. Transpose convolutional layers are denoted $\text{Conv}_s^T(C \times W \times H)$. For layers of the same size repeated k times, we'll denote this as $[layers]^{\times k}$.

Toy Networks: For the networks trained on the toy dataset, the input and output dimension are each 2, and the scalar-valued output is attained by taking the dot product with the vector $[+1, -1]$. These have varying depth, but all have architectures like

$$x \to \Big[FC100 \to ReLU\Big]^{\times k} \to FC2 \to z$$

where the number of hidden layers denotes the number of ReLU layers in the network.

Generators for MNIST and CIFAR: For MNIST and CIFAR, we trained 6 VAEs and 2 GANs. These each have the same architecture with the exception of varying input/output shapes. The VAEs each have a latent dimension of 20 and the GANs have an input dimension of 100. We use the notation D and C to denote the output dimnension and channel: $(784, 1)$ for MNIST and $(3072, 3)$ for CIFAR-10.

- **VAESmall:** $[D, 400, 200, 20, 200, 400, D]$. Where the decoder is just the $[20, 200, 400, D]$ subnetwork.
- **VAEMed:** $[D, 400, 200, 100, 50, 100, 200, 400, D]$ where the decoder is just the $[50, 100, 200, 400, D]$ subnetwork.
- **VAEBig:** $[D, 400, 200, 200, 200, 200, 100, 200, 200, 200, 200, 400, D]$, where the decoder is just the $[100, 200, 200, 200, 200, 400, D]$ subnetwork.
- **VAECNN:** $x \to \text{Conv}_2(16 \times 4 \times 4) \to ReLU \to \text{Conv}_2(32 \times 4 \times 4) \to ReLU \to FC50 \to FC800 \to ReLU \to \text{Conv}_2^T(16 \times 5 \times 5) \to ReLU \to \text{Conv}_2^T(C \times 4 \times 4) \to Sigmoid.$
- **VAETanh**: Same as VAEMed with $tanh$ nonlinearities in place of ReLU.
- **VC-Tanh**: Same as VAECNN with $tanh$ nonlinearities in place of ReLU.
- **FFGAN**: $[100, 256, 512, 1024, D]$.
- **DCGAN**: $x \to \text{Conv}_1^T(256 \times 4 \times 4) \to ReLU \to \text{Conv}_2^T(128 \times 4 \times 4) \to ReLU \to \text{Conv}_2^T(64 \times 4 \times 4) \to ReLU \to \text{Conv}_2^T(C \times 4 \times 4) \to tanh.$

Classifiers for MNIST and CIFAR: For MNIST and CIFAR, we trained three fully connected networks. We refer to these as $\{tiny, small, med, \} - *$. Each of these were trained with both the ReLU and tanh nonlinearities with both standard and PGD adversarial training. We will describe the training techniques in the next section.

- **Tiny***: $[D, 20, 20, 10]$.
- **Small***: $[D, 100, 100, 100, 10]$.
- **Med***: $[D] + [100] \times 6 + [10]$.

D.2. Datasets, Training Methods, and Computing Environment

Computing Environment: All networks were trained using Pytorch on a machine with $2\times$ GeForce RTX 2070 GPU's. All Lipschitz evaluations were performed using the CPU only, an Intel i7-9700K. Mixed integer programming evalutaions were performed using 4 cores and the Gurobi optimizer.

Datasets: The CIFAR-10 and MNIST datasets are standard. For the dataset taken from (Aziznejad et al., 2020), we generate 1000 points uniformly randomly in $[-1, 1]^2$ and attach the label 1 if the norm of the data point is $\leq \frac{2}{\pi}$.

Training Methods: We outline the methods used to train each set of networks:

- **Toy Dataset Nets:** We trained using the CrossEntropy loss and the Adam optimizer with a learning rate of 0.001, where we trained for 200 epochs.
- **VAEs:** For the VAE networks, we train with the standard VAE loss: a sum of binary cross-entropy between the reconstructed example and original input, and a KL-divergence term. We train for 50 epochs for all VAEs with the Adam optimizer and a learning rate of 0.001
- **GANs:** For the GANs, we train using the binary cross-entropy applied to the discriminator output. We train with Adam $(lr = 0.001)$ and 25 epochs.
- **Classifiers:** For the classifiers considered, we always train for 50 epochs using Adam $(lr = 0.001)$ and either the standard Cross-Entropy loss or the PGD loss for adversarial training. For MNIST networks, we allow an adversarial budget of ℓ_∞ norm of 0.1, and the adversary takes 10 steps with step-size 0.2. For CIFAR-10, the adversary has a budget of $\frac{8}{255}$ and takes 10 steps of stepsize $\frac{2}{255}$.

D.3. Varying Radius

We consider the effect of varying the radius of the region we evaluate Lipschitz constants over. We expect that as the radius increases, the bounds for the preactivations at each layer will become more loose. For ReLU networks, when zero is strictly contained in the preactivation bounds, a new degree of freedom needs to be introduced to the layerwise zonotope approximations. Hence, as more degrees of freedom are introduced, we expect the bound returned by ZLip to be looser and the runtime will be less efficient since the representation of each zonotope will be larger. However, other methods will likely also yield looser bounds as well. We also note that a more intelligent strategy that prunes the zonotope representation size to a more managable size may be employed, though we leave such performance improvements for future work.

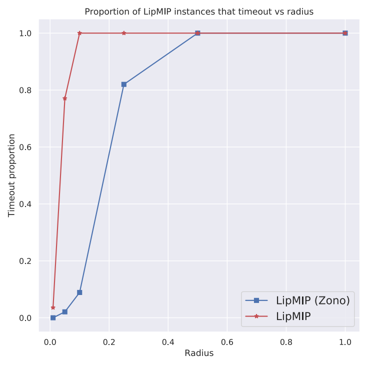

Figure 5. Proportion of timed out examples for LipMIP versus LipMIP using ZLip as a first step across varius radii.

Toy Dataset: First we examine the effect of changing the evaluation radius on the network trained on the circle dataset with 6 hidden layers of width 100 and the ReLU nonlinearity. We report the results in table 2, where we have evaluated over 64 elements from the test set and ℓ_∞ balls of each radius. We set a timeout of 120s for the mixed-integer programming approaches, at which point the tightest upper bound is returned, explaining the discrepancy between the results on the first two columns. The proportion of timed out examples is presented in Figure 5. We observe that for networks with small input dimension, CLEVER is fairly accurate and can be viewed as a surrogate for the true Lipschitz constant when the LipMIP results time out. In this case we see that for all but the global evaluation returned by SeqLip, all techniques provide looser bounds as the radius increases, however ZLip remains significantly tighter than Fast-Lip and even yields a tighter result than the tightest upper bound provided by LipMIP after this procedure times out.

Generative Models: Here we present results on the generative models when we vary the radius of the input region. We focus primarily on the VAEs described above for both the MNIST and CIFAR-10 datasets. These results for VAEMEd on MNIST are displayed in Figure 6. As expected, increasing radius size increases the estimated Lipschitz bound for both ZLip and Fast-Lip. The relative gap decreases as the radius increases, indicating that ZLip is comparatively tighter when few neurons are unstable, which we attribute to ZLip being able to perfectly map the affine layers. It is also expected that Fast-Lip does not get slower even as more neurons become unstable, as the representation size of a hyperbox in $\mathbb{R}^d$ will always be $O(d)$, and looser neuron bounds just increases the looseness of the approximation without increasing runtime.

Table 2. Evaluation of various Lipschitz computation techniques for the network with 6 hidden layers trained on the Circle dataset.

Lipschitz Values for 6 $\times$ 100 Circle Network						
Radius	LipMIP (Zono)	LipMIP	ZLip	Fast-Lip	SeqLip	CLEVER
0.01	2.62×10^2	2.97×10^2	2.98×10^2	2.82×10^3	2.72×10^3	2.95×10^2
0.05	3.69×10^2	2.79×10^3	4.22×10^2	9.94×10^3	2.72×10^3	3.61×10^2
0.1	4.55×10^2	6.09×10^3	6.17×10^2	1.29×10^4	2.72×10^3	4.39×10^2
0.25	8.91×10^2	1.67×10^4	1.55×10^3	2.00×10^4	2.72×10^3	4.65×10^2
0.5	2.51×10^3	2.20×10^4	3.70×10^3	2.46×10^4	2.72×10^3	6.72×10^2
1.0	6.49×10^3	2.82×10^4	5.81×10^3	3.10×10^4	2.72×10^3	9.04×10^2

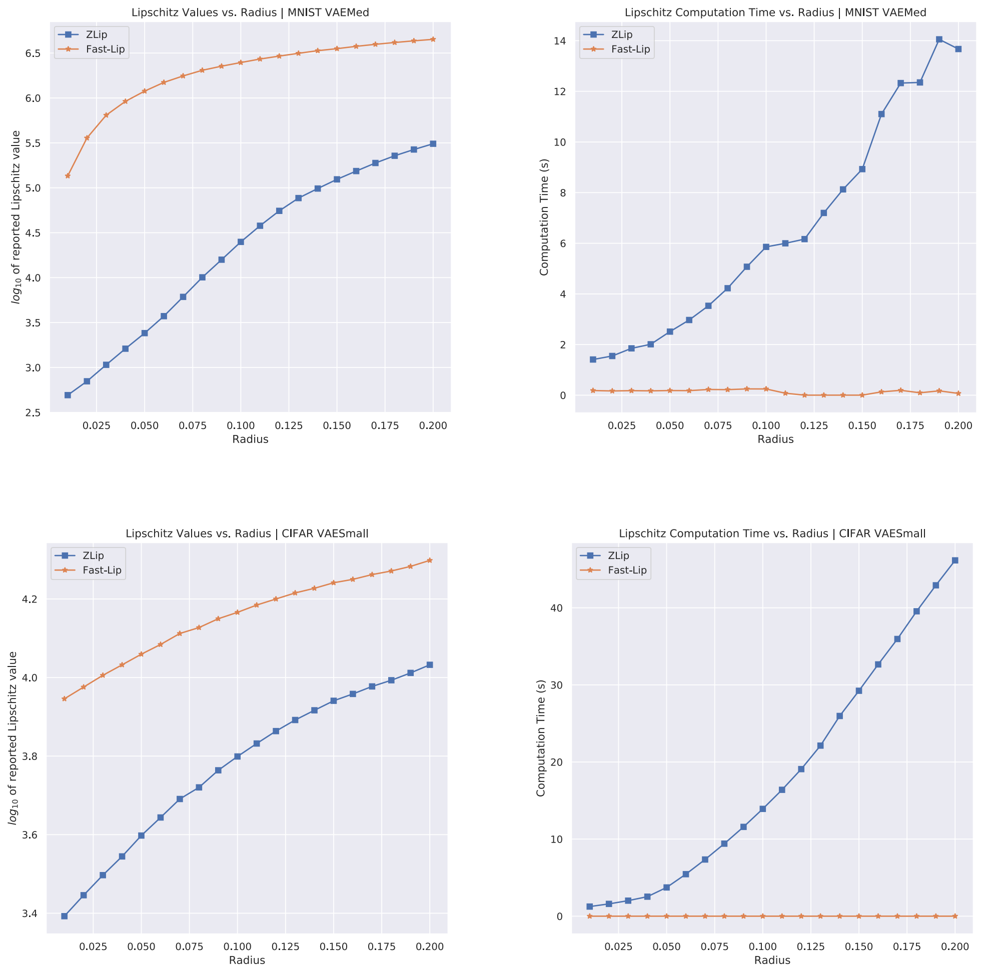

Figure 6. Reported Lipschitz constants and times for the MNIST MedVAE as we increase the radius of the region over which we evaluate Lipschitz constant. (Top Left) reports the values on a log-scale, noticing that both ZLip and Fast-Lip increase their estimate as the radius increases, the relative gap is largest for small radii.(Top Right) we display times in seconds versus the radius. The increase in running time of ZLip is due to the increase in number of unstable neurons, which increases the size of the representations of the zonotopes that must be passed through each layer. (Bottom) Reported Lipschitz constants and times for the CIFAR SmallVAE as we increase radius.

D.4. Ablating the choice of abstract domain:

ZLip operates by iteratively building *zonotopes* to satisfy the containments of equations 14-17, specifically we use zonotopes in both the forward pass (such as DeepZ, DiffAI), but also zonotopes in the backward pass. To examine the importance of zonotopes in both directions, we replace the zonotopes with hyperboxes in one or both of the directions. Note that using hyperboxes in both the forward and backward directions is FastLip. We denote the method that uses hyperboxes in the forward pass, but zonotopes in the backward pass as ‘Hyperbox -¿ Zono’, and vice versa.

Toy Dataset: In Figure 7 we compare the performance of the different techniques on networks trained on the toy dataset. The y-axis records the logarithm of the average ratio between each method’s Lipschitz estimate and ZLip’s Lipschitz estimate for each example. We evaluate on 100 random points for each x-value. On the left panel we vary the architecture size while evaluating on inputs that are hyperboxes with radius 0.1. For every network considered, on average, the abstract domain in the forward pass is more important, and this gap becomes more apparent as the size of the network increases. On the right panel, we fix a network and evaluate the performance as we vary the size of the certified region. For small radii, the choice of the forward domain is more important, however as the radius increases, the backward domain becomes more important. We conjecture this is because, for large radii, there is much uncertainty about which ReLU’s are fixed and the performance of zonotopes and hyperboxes in the forward pass becomes equivalent.

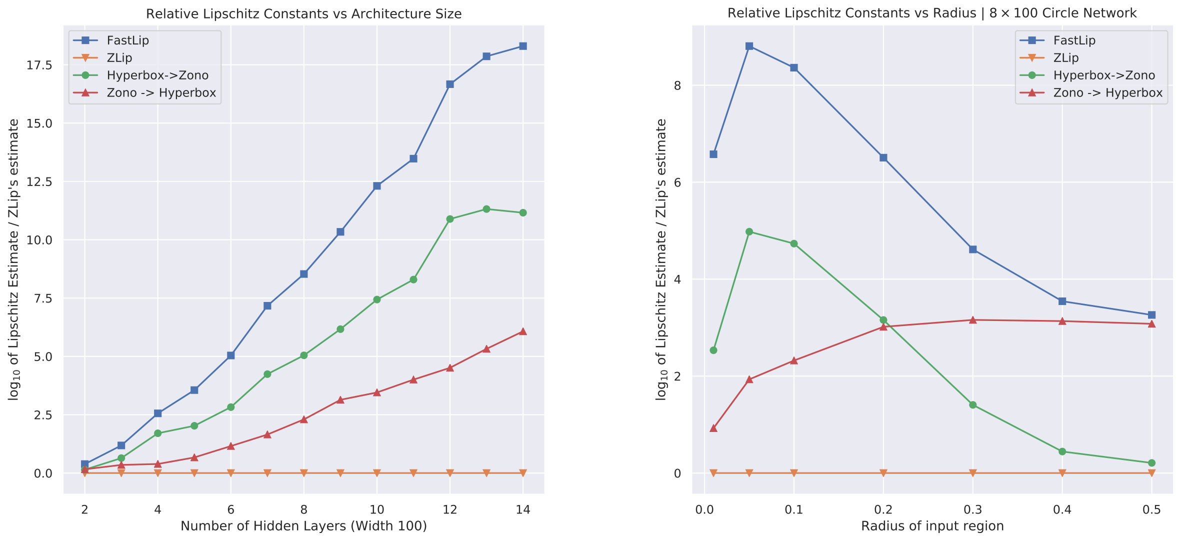

Figure 7. Log-mean-ratio of reported Lipschitz estimates (relative to ZLip) on varying networks trained on the Circle dataset. (Left) reports the values as we vary the network size, demonstrating that the forward domain is more important as the network size increases. (Right) reports the values for a fixed network as we vary the input radius size. Larger input radii have more uncertain ReLU neurons and the choice in forward domain becomes relatively less important.

MNIST Generative models: In Table 3 we evaluate the choice of abstract domains on MNIST generative models. We consider the decoders of the three different VAE’s trained on MNIST and evaluate over two different radii, considering hyperboxes of the denoted radius surrounding the encodings of random test MNIST examples. Here we again see that neither of the single-zonotope approaches is unilaterally better, however larger models tend to benefit more from using zonotopes in the forward pass.

Table 3. Mean ratio of Lipschitz estimates provided by other abstract domains relative to the estimate provided by ZLip, evaluated on MNIST VAE decoders. We see that as the model becomes larger, using zonotopes in the forward pass yields a tighter bound.

Radius	0.01			0.1		
Method	FastLip	$H \to Z$	$Z \to H$	FastLip	$H \to Z$	$Z \to H$
VAESmall	7.85	**1.33**	5.89	9.39	**3.08**	3.11
VAEMed	276.41	**12.71**	43.78	410.92	51.81	**12.25**
VAEBig	4238.83	123.21	**85.65**	26018.02	1357.93	**26.64**

D.5. Experimental Results on Classifiers

MNIST: For completeness and comparison against other networks on more realistic networks, we present results of our Lipschitz bounding technique versus several recent works for a variety of networks trained both with the standard classification loss as well as those trained adversarially. We evaluate the Lipschitz value returned by SeqLip, LipSDP, Fast-Lip, CLEVER, and ZLip for inputs of radius 0.1 centered at elements taken from the test set. If f is the trained classifier which has outputs in $\mathbb{R}^{10}$, we consider the Lipschitz constant of the network $f_i(\cdot) - f_{i+1}(\cdot)$ for each example where the true label is i. First we present the values for the MNIST networks trained with the standard CrossEntropy Loss in Table 4 and times are presented in Table 5. We remark that the bounds reported by CLEVER for networks with high dimension have been shown to be quite loose, so it is unclear what the correct Lipschitz value is for each of these networks. The most salient points here are that the values returned by ZLip are comparable to those returned by LipSDP at a significantly faster runtime. We also note that LipSDP errors when applied to networks as large as MedReLU or MedTanh.

For the PGD trained MNIST networks, we present the values and times in tables 6, 7. In direct comparison to the tables for the networks trained with CrossEntropy loss, we notice that all methods report lower values for the adversarially trained network. This tracks with prior work that adversarial regularization serves as a form of Lipschitz regularization.

CIFAR-10: The same experiments as above were performed on networks trained to classify the CIFAR-10 dataset. We present these results in Tables 8-11, but note that these results are qualitatively very similar to the results for the MNIST networks.

Table 4. Lipschitz values reported by various networks evaluated on the various Classifiers described above. All numbers report an average over regions of radius 0.1 centered at examples from the test set.

Lipschitz Estimates (MNIST) — CrossEntropy Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	3.51×10^3	2.94×10^3	7.36×10^3	1.15×10^1	5.59×10^3
SmallReLU	2.12×10^4	1.52×10^4	1.94×10^5	9.59×10^0	9.34×10^4
MedReLU	1.08×10^6	—	1.45×10^8	1.06×10^1	1.77×10^7
TinyTanh	1.33×10^4	1.14×10^4	2.56×10^4	2.97×10^1	1.98×10^4
SmallTanh	5.80×10^4	4.29×10^4	3.24×10^5	3.26×10^1	1.68×10^5
MedTanh	5.50×10^6	—	4.29×10^8	3.56×10^2	6.94×10^7

Table 5. Times for MNIST classifiers trained with the Cross-Entropy loss

Lipschitz Times (MNIST) — CrossEntropy Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	5.08×10^{-1}	1.78×10^{1}	1.75×10^{-3}	5.98×10^{1}	4.20×10^{-1}
SmallReLU	2.28×10^{0}	2.63×10^{2}	9.73×10^{-2}	1.01×10^{2}	9.36×10^{-1}
MedReLU	3.35×10^{0}	—	1.07×10^{-1}	1.75×10^{2}	2.13×10^{0}
TinyTanh	5.90×10^{1}	1.82×10^{1}	1.96×10^{-3}	5.98×10^{1}	4.22×10^{-1}
SmallTanh	4.56×10^{0}	2.28×10^{2}	1.05×10^{-1}	1.09×10^{2}	9.87×10^{-1}
MedTanh	9.91×10^{0}	—	1.08×10^{-1}	1.77×10^{2}	2.03×10^{0}

Table 6. Values for MNIST classifiers trained with the PGD loss

Lipschitz Estimates (MNIST) — PGD Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	3.97×10^{2}	2.66×10^{2}	3.48×10^{2}	1.07×10^{0}	1.42×10^{2}
SmallReLU	1.52×10^{3}	9.35×10^{2}	1.98×10^{4}	1.85×10^{0}	5.39×10^{3}
MedReLU	3.91×10^{4}	1.64×10^{4}	1.45×10^{7}	2.26×10^{0}	9.36×10^{5}
TinyTanh	1.38×10^{3}	8.61×10^{2}	6.86×10^{2}	3.14×10^{0}	5.17×10^{2}
SmallTanh	7.96×10^{3}	4.81×10^{3}	5.68×10^{4}	6.24×10^{0}	2.42×10^{4}
MedTanh	2.73×10^{5}	1.30×10^{5}	5.98×10^{7}	1.90×10^{1}	6.27×10^{6}

Table 7. Times for MNIST classifiers trained with the PGD loss

Lipschitz Times (MNIST) — PGD Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	1.18×10^{-2}	1.14×10^{1}	1.74×10^{-3}	2.71×10^{1}	1.10×10^{-1}
SmallReLU	2.71×10^{1}	1.77×10^{2}	1.13×10^{-1}	1.13×10^{2}	1.12×10^{0}
MedReLU	3.10×10^{0}	4.42×10^{2}	1.11×10^{-1}	1.76×10^{2}	2.17×10^{0}
TinyTanh	6.11×10^{-1}	1.62×10^{1}	1.84×10^{-3}	5.33×10^{1}	3.77×10^{-1}
SmallTanh	4.61×10^{0}	1.77×10^{2}	8.66×10^{-2}	7.72×10^{1}	7.31×10^{-1}
MedTanh	3.29×10^{0}	5.00×10^{2}	1.16×10^{-1}	1.86×10^{2}	2.15×10^{0}

Table 8. Values for CIFAR-10 classifiers trained with the Cross Entropy loss

Lipschitz Values (CIFAR-10) — CrossEntropy Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	1.11×10^{3}	8.51×10^{2}	1.13×10^{3}	6.05×10^{-1}	8.89×10^{2}
SmallReLU	1.27×10^{4}	7.23×10^{3}	1.55×10^{5}	1.43×10^{0}	6.32×10^{4}
MedReLU	4.30×10^{5}	1.78×10^{5}	2.42×10^{8}	2.22×10^{0}	2.13×10^{7}
TinyTanh	4.12×10^{3}	3.22×10^{3}	5.52×10^{3}	2.28×10^{0}	4.32×10^{3}
SmallTanh	2.61×10^{4}	1.77×10^{4}	1.72×10^{5}	5.73×10^{0}	7.82×10^{4}
MedTanh	1.28×10^{6}	—	1.86×10^{8}	1.26×10^{1}	2.22×10^{7}

Table 9. Values for CIFAR-10 classifiers trained with the PGD loss

Lipschitz Values (CIFAR-10) — PGD Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	3.01×10^{2}	1.92×10^{2}	4.63×10^{1}	1.17×10^{-1}	3.95×10^{1}
SmallReLU	2.03×10^{3}	1.22×10^{3}	2.15×10^{4}	3.62×10^{-1}	7.75×10^{3}
MedReLU	3.61×10^{4}	1.26×10^{4}	1.70×10^{7}	5.22×10^{-1}	1.26×10^{6}
TinyTanh	1.81×10^{3}	1.31×10^{3}	1.05×10^{3}	8.94×10^{-1}	8.91×10^{2}
SmallTanh	1.51×10^{4}	7.25×10^{3}	2.62×10^{4}	2.67×10^{0}	1.35×10^{4}
MedTanh	2.57×10^{5}	9.56×10^{4}	1.41×10^{7}	3.14×10^{0}	1.81×10^{6}

Table 10. Times for CIFAR-10 classifiers trained with the CrossEntropy loss

Lipschitz Times (CIFAR-10) — CrossEntropy Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	2.32×10^{-1}	5.02×10^{2}	9.59×10^{-2}	5.91×10^{1}	6.24×10^{-1}
SmallReLU	7.14×10^{0}	2.88×10^{3}	1.07×10^{-1}	1.06×10^{2}	1.16×10^{0}
MedReLU	7.45×10^{1}	4.81×10^{3}	6.38×10^{-2}	9.88×10^{1}	1.32×10^{0}
TinyTanh	6.45×10^{-1}	5.05×10^{2}	1.18×10^{-1}	7.28×10^{1}	8.66×10^{-1}
SmallTanh	3.11×10^{1}	3.25×10^{3}	1.17×10^{-1}	1.22×10^{2}	1.22×10^{0}
MedTanh	1.09×10^{0}	6.78×10^{3}	6.34×10^{-3}	3.98×10^{0}	1.89×10^{-1}

Table 11. Times for CIFAR-10 classifiers trained with the PGD loss

Lipschitz Times (CIFAR-10) — PGD Loss					
Network	SeqLip	SDP	Fast-Lip	CLEVER	ZLip
TinyReLU	1.47×10^{-1}	3.94×10^{2}	1.09×10^{-1}	6.74×10^{1}	7.93×10^{-1}
SmallReLU	4.11×10^{1}	2.52×10^{3}	9.89×10^{-2}	9.12×10^{1}	1.07×10^{0}
MedReLU	5.22×10^{1}	4.01×10^{3}	1.14×10^{-1}	1.89×10^{2}	2.39×10^{0}
TinyTanh	8.12×10^{-2}	4.60×10^{2}	1.13×10^{-1}	7.02×10^{1}	8.25×10^{-1}
SmallTanh	4.97×10^{0}	3.28×10^{3}	1.13×10^{-1}	1.22×10^{2}	1.23×10^{0}
MedTanh	1.13×10^{0}	3.90×10^{3}	5.77×10^{-3}	3.98×10^{0}	1.89×10^{-1}