Supplementary Material: Globally-Robust Neural Networks

Klas Leino [1] **Zifan Wang** [1] **Matt Fredrikson** [1]

A. Proofs

A.1. Proof of Theorem 1

Theorem 1. *If $\bar{F}^\epsilon(x) \neq \bot$, then $\bar{F}^\epsilon(x) = F(x)$ and F is ϵ-locally-robust at x.*

Proof. Let $j = F(x)$. Assume that $\bar{F}^\epsilon(x) \neq \bot$; this happens only if one of the outputs of f is greater than $\bar{f}_\bot^\epsilon(x)$ — from the definition of $f_\bot(x)$, it is clear that only $f_j(x)$ can be greater than $\bar{f}_\bot^\epsilon(x)$. Therefore $f_j(x) > \bar{f}_\bot^\epsilon(x)$, and so $\bar{F}^\epsilon(x) = j = F(x)$.

Now assume x' satisfies $||x - x'|| \leq \epsilon$. Let K_i be an upper bound on the Lipschitz constant of f_i. Then, $\forall i$

$$\frac{|f_i(x) - f_i(x')|}{\epsilon} \leq \frac{|f_i(x) - f_i(x')|}{||x - x'||} \leq K_i$$
$$\implies |f_i(x) - f_i(x')| \leq K_i \epsilon \tag{3}$$

We proceed to show that for any such x', $F(x')$ is also j. In other words, $\forall i \neq j$, $f_i(x') < f_j(x')$. By applying the definition of the Lipschitz constant as in (3), we obtain (4). Next, (5) follows from the fact that $\bar{f}_\bot^\epsilon(x) = \max_{i \neq j}\{y_i + (K_i + K_j)\epsilon\}$. We then obtain (6) from the fact that $f_j(x) > \bar{f}_\bot^\epsilon(x)$, as observed above. Finally, we again apply (3) to obtain (7).

$$f_i(x') \leq f_i(x) + |f_i(x) - f_i(x')| \leq f_i(x) + K_i\epsilon \tag{4}$$
$$\leq \bar{f}_\bot^\epsilon(x) - K_j\epsilon \tag{5}$$
$$< f_j(x) - K_j\epsilon \tag{6}$$
$$\leq f_j(x) - |f_j(x) - f_j(x')| \leq f_j(x') \tag{7}$$

Therefore, $f_i(x') < f_j(x')$, and so $F(x') = j$. This means that F is locally robust at x. □

A.2. Tighter Bounds for Theorem 1

Note that in the formulation of GloRo Nets given in Section 2.2, we assume that the predicted class, j, will decrease by the maximum amount within the ϵ-ball, while all other classes increase by their respective maximum amounts. This is a conservative assumption that guarantees local robustness; however, in practice, we can dispose of this assumption by instead calculating the Lipschitz constant of the margin by which the logit of the predicted class surpasses the other logits, $f_j - f_i$.

The *margin Lipschitz constant* of f, defined for a pair of classes, $i \neq j$, is given by Definition 4.

Definition 4. *Margin Lipschitz Constant For network, $f : \mathbb{R}^n \to \mathbb{R}^m$, and classes $i \neq j \in [m]$, K_{ij}^* is an upper bound on the margin Lipschitz constant of f if $\forall x_1, x_2$,*

$$\frac{|f_j(x_1) - f_i(x_1) - (f_j(x_2) - f_i(x_2))|}{||x_1 - x_2||} \leq K_{ij}^*$$

We now define a variant of GloRo Nets (Section 2.2) as follows: For input, x, let $j = F(x)$, i.e., j is the label assigned by the underlying model to be instrumented. Define $\bar{f}_i^\epsilon(x) ::= f_i(x)$, and $\bar{f}_\bot^\epsilon(x) ::= \max_{i \neq j}\{f_i(x) + \epsilon K_{ij}^*\}$.

Theorem 4. *Under this variant, if $\bar{F}^\epsilon(x) \neq \bot$, then $\bar{F}^\epsilon(x) = F(x)$ and F is ϵ-locally-robust at x.*

Proof. The proof is similar to the proof of Theorem 1 (Appendix A.1). Let $j = F(x)$. As before, when $\bar{F}^\epsilon(x) \neq \bot$, we see that $\bar{F}^\epsilon(x) = j = F(x)$.

Now assume x' satisfies $||x - x'|| \leq \epsilon$. Let K_{ij}^* be an upper bound on the margin Lipschitz constant. Then, $\forall i$

$$|f_j(x) - f_i(x) - (f_j(x') - f_i(x'))| \leq K_{ij}^*\epsilon \tag{8}$$

We proceed to show that for any such x', $F(x')$ is also j. In other words, $\forall i \neq j$, $f_i(x') < f_j(x')$. By applying (8), we obtain (9). Next, (10) follows from the fact that $\bar{f}_\bot^\epsilon(x) = \max_{i \neq j}\{f_i(x) + K_{ij}^*\epsilon\}$. We then obtain (11) from the fact that $f_j(x) > \bar{f}_\bot^\epsilon(x)$, as $\bar{F}^\epsilon(x) = j \neq \bot$.

$$\cancel{f_i(x)} + f_j(x) - \cancel{f_i(x)} - f_j(x') + f_i(x')$$
$$\leq f_i(x) + |f_j(x) - f_i(x) - (f_j(x') - f_i(x'))|$$
$$\leq f_i(x) + K_{ij}^*\epsilon \tag{9}$$
$$\leq \bar{f}_\bot^\epsilon(x) \tag{10}$$
$$< f_j(x) \tag{11}$$

Rearranging terms, we obtain that $f_i(x') < f_j(x')$. Thus, $F(x') = j$; this means that F is locally robust at x. □

[1]Carnegie Mellon University, Pittsburgh, Pennsylvania, USA. Correspondence to: Klas Leino <kleino@cs.cmu.edu>, Matt Fredrikson <mfredrik@cmu.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

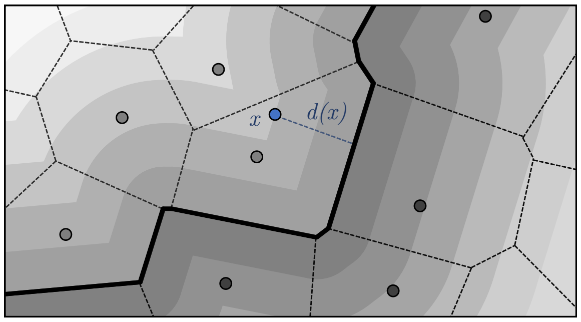

Figure A.1: Illustration of a function, g, constructed to satisfy Theorem 3. The points in S are shown in light and dark gray, with different shades indicating different labels. The Voronoi tessellation is outlined in black, and the faces belonging to the decision boundary are highlighted in bold. The level curves of g are shown in various shades of gray and correspond to points, x, at some fixed distance, $d(x)$, from the decision boundary.

A.3. Proof of Theorem 2

Theorem 2. *$\bar{F}^{\epsilon/2}$ is ϵ-globally-robust.*

Proof. Assume x_1 and x_2 satisfy $||x_1 - x_2|| \leq \epsilon$. Let $\bar{F}^{\epsilon/2}(x_1) = c_1$ and $\bar{F}^{\epsilon/2}(x_2) = c_2$.

If $c_1 = \bot$ or $c_2 = \bot$, global robustness is trivially satisfied.

Consider the case where $c_1 \neq \bot$, $c_2 \neq \bot$. Let x' be the midpoint between x_1 and x_2, i.e., $x' = (x_1 + x_2)/2$. Thus

$$||x_1 - x'|| = \left|\left|\frac{x_1 - x_2}{2}\right|\right| = \frac{||x_1 - x_2||}{2} \leq \frac{\epsilon}{2}.$$

By Theorem 1, this implies $F(x') = c_1$. By the same reasoning, $||x_2 - x'|| \leq \epsilon/2$, implying that $F(x') = c_2$. Thus, $c_1 = c_2$, so global robustness holds. □

A.4. Proof of Theorem 3

Theorem 3. *Let f be a binary classifier that predicts $1 \Longleftrightarrow f(x) > 0$. Let $K_L(x,\epsilon)$ be the local Lipschitz constant of f at point x with radius ϵ.*

Suppose that for some finite set of points, S, $\forall x \in S$, $|f(x)| > \epsilon K_L(x,\epsilon)$, i.e., all points in S can be verified via the local Lipschitz constant.

Then there exists a classifier, g, with global Lipschitz constant K_G, such that $\forall x \in S$, (1) g makes the same predictions as f on S, and (2) $|g(x)| > \epsilon K_G$, i.e., all points in S can be verified via the global Lipschitz constant.

Proof. Let T be the Voronoi tessellation generated by the points in S. Each Voronoi cell, $C_j \in T$, corresponds to the set of points that are closer to $p_j \in S$ than to any other point in S; and the face, $F_{ij} \in T$, which separates cells C_i and C_j, corresponds to the set of points that are equidistant from p_i and p_j.

Let $B = \{F_{ij} : \text{sign}(f(p_i)) \neq \text{sign}(f(p_j))\}$, i.e., the set of faces in the Voronoi tessellation that separate points that are classified differently by f (note that B corresponds to the boundary of the 1-nearest-neighbor classifier for the points in S).

Consider a point, x. Let $p_x \in S$ be the closest point in S to x, i.e., the point corresponding to the Voronoi cell containing x. Let $d(x) = ||\text{proj}(x \to B) - x||$; that is, $d(x)$ is the minimum distance from x to any point in any of the faces in B. Then define

$$g(x) = \text{sign}\big(f(p_x)\big)\frac{d(x)}{\epsilon}$$

First, observe that $g(x) > 0 \Longleftrightarrow f(x) > 0$ follows from the fact that $d(x)$ and ϵ are non-negative, thus the sign of $g(x)$ is derived from the sign of $f(x)$.

Next, we show that the global Lipschitz constant of g, K_G, is at most $1/\epsilon$, that is, $\forall x_1, x_2$,

$$\frac{|g(x_1) - g(x_2)|}{||x_1 - x_2||} \leq \frac{1}{\epsilon}$$

Consider two points, x_1 and x_2, and let p_1 and p_2 be the points in S corresponding to the respective Voronoi cells of x_1 and x_2.

First, consider the case where $\text{sign}(f(p_1)) \neq \text{sign}(f(p_2))$, i.e., x_1 and x_2 are on opposite sides of the boundary, B. In this case $|g(x_1) - g(x_2)| = {(d(x_1) + d(x_2))}/{\epsilon}$, and thus it suffices to show that $d(x_1) + d(x_2) \leq ||x_1 - x_2||$.

Assume for the sake of contradiction that $d(x_1) + d(x_2) > ||x_1 - x_2||$. Note that because x_1 and x_2 belong in Voronoi cells with different classifications from f, the line segment connecting x_1 and x_2 must cross the boundary, B, at some point c. Therefore, $||x_1 - c|| + ||x_2 - c|| = ||x_1 - x_2|| < d(x_1) + d(x_2)$; without loss of generality, this implies that $||x_1 - c|| < d(x_1)$. But since $c \in F \in B$, this contradicts that $d(x_1)$ is the minimum distance from x_1 to B.↯

Next, consider the case where $\text{sign}(f(p_1)) = \text{sign}(f(p_2))$. In this case $|g(x_1) - g(x_2)| = {|d(x_1) - d(x_2)|}/{\epsilon}$, and thus, without loss of generality, it suffices to show that $d(x_1) - d(x_2) \leq ||x_1 - x_2||$.

Assume for the sake of contradiction that $d(x_1) - d(x_2) > ||x_1 - x_2||$. Thus $d(x_1) > ||x_1 - x_2|| + d(x_2)$. However, this suggests that we can take the path from x_1 to x_2 to B with a smaller total distance than $d(x_1)$, contradicting that $d(x_1)$ is the minimum distance from x_1 to B.↯

We now show that $\forall p \in S$, $|g(p)| \geq 1$, i.e., $d(p) \geq \epsilon$. In other words, we must show that the distance of any point, $p \in S$ to the boundary, B, is at least ϵ. Consider a point, x, on some face, $F_{ij} \in B$. This point is equidistant from p_i

and $p_j \in S$, on which f makes different predictions; and every other point in S is at least as far from x as p_i and p_j. I.e., $||p_i - x|| = ||p_j - x|| \leq ||p - x||, \forall p \in S$. By the triangle inequality, $2||p_i - x|| \geq ||p_i - p_j||$, and by Lemma 1, $||p_i - p_j|| \geq 2\epsilon$. Thus $||p - x|| \geq \epsilon, \forall p \in S$; therefore every point on the boundary is at least ϵ from $p \in S$.

Putting everything together, we have that $\forall p \in S$, $|g(p)| \geq 1 \geq \epsilon K_G$. □

Note that while Theorem 3 is stated for binary classifiers, the result holds for categorical classifiers as well. We can modify the construction of g from the above proof in a straightforward way to accommodate categorical classifiers. In the case where there are m different classes, the output of g has m dimensions, each corresponding to a different class. Then, for x in a Voronoi cell corresponding to $p_x \in S$ with label, j, we define $g_j(x) ::= {}^{d(x)}/_{\epsilon}$ and $g_i(x) ::= 0 \; \forall i \neq j$. We can see that, for all pairs of classes, i and j, the Lipschitz constant of $g_i - g_j$ in this construction is the same as the Lipschitz constant of g in the above proof, since only one dimension of the output of g changes at once. Thus, we can use the global bound suggested in Appendix A.2 to certify the points in S.

A.5. Proof of Lemma 1

Lemma 1. *Suppose that for some classifier, F, and some set of points, S, $\forall x \in S$, F is ϵ-locally-robust at x. Then $\forall x_1, x_2 \in S$ such that $F(x_1) \neq F(x_2)$, $||x_1 - x_2|| > 2\epsilon$.*

Proof. Suppose that for some classifier, F, and some set of points, S, $\forall x \in S$, F is ϵ-locally-robust at x. Assume for the sake of contradiction that $\exists x_1, x_2 \in S$ such that $F(x_1) \neq F(x_2)$ but $||x_1 - x_2|| \leq 2\epsilon$. Consider the midpoint between x_1 and x_2, $x' = (x_1 + x_2)/2$. Note that

$$||x' - x_1|| = \frac{||x_1 - x_2||}{2} \leq \epsilon$$

Therefore, since F is ϵ-locally-robust at x_1, $F(x') = F(x_1)$. By the same argument, $F(x') = F(x_2)$. But this contradicts that $F(x_1) \neq F(x_2)$.↯ □

B. Hyperparameters

In this appendix, we describe hyperparameters used in the training of GloRo Nets to produce the results in Section 5. The full set of hyperparameters used for all experiments is shown in Tables B.1 and B.2. We explain each column as follows and discuss how a particular value is selected for each hyperparameter.

Architectures. To denote architectures, we use $c(C,K,S)$ to denote a convolutional layer with C output channels, a kernel of size $K \times K$, and strides of width S. We use `SAME` padding unless noted otherwise. We use $d(D)$ to denote a dense layer with D output dimensions. We use MinMax (Anil et al., 2019) or ReLU activations (see Appendix D for a comparison) after each layer except the top of the network, and do not include an explicit Softmax activation. Using this notation, the architectures referenced in Section 5 are as shown in the following list.

- 2C2F: c(16,4,2).c(32,4,2).d(100).d(10)
- 4C3F: c(32,3,1).c(32,4,2).c(64,3,1).c(64,4,2) .d(512).d(512).d(10)
- 6C2F: c(32,3,1).c(32,3,1).(32,4,2).(64,3,1) .c(64,3,1).c(64,4,2).d(512).d(10)
- 8C2F: c(64,3,1).c(64,3,1).c(64,4,2).c(128,3,1). c(128,4,2).c(256,3,1).(256,4,2).d(200)

We arrived at these architectures in the following way. 2C2F, 4C3F and 6C2F are used in the prior work (Lee et al., 2020; Croce et al., 2019; Wong & Kolter, 2018) to evaluate the verifiable robustness, and we used them to facilitate a direct comparison on training cost and verified accuracy. For Tiny-ImageNet, we additionally explored the architecture described in (Lee et al., 2020) for use with that dataset, but found that removing one dense and one convolutional layer (denoted by 8C2F in the list above) produced the same (or better) verified accuracy, but lowered the total training cost.

Data preprocessing. For all datasets, we scaled the features to the range [0,1]. On some datasets, we used the following data augmentation pipeline `ImageDataGenerator` from `tf.keras`, which is denoted by `default` in Table B.2 and B.1.

```
rotation_range=20
width_shift_range=0.2
height_shift_range=0.2
horizontal_flip=True
shear_range=0.1
zoom_range=0.1
```

When integrating our code with `tensorflow-dataset`, we use the following augmentaiton pipeline and denote it as `tfds` in Table B.2 and B.1.

```
horizontal_flip=True
zoom_range=0.25
random_brightness=0.2
```

Our use of augmentation follows the convention established in prior work (Wong et al., 2018; Lee et al., 2020): we only use it on CIFAR and tiny-imatenet, but not on MNIST.

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
2C2F GloRo	MNIST	`None`	0	512	500	0.3	0.3
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`orthogonal`	1e-3	`decay_to_1e-6`	`0.1,2.0,500`	`single`	5	
architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
4C3F GloRo	MNIST	`None`	0	512	200	1.74	1.58
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`orthogonal`	1e-3	`decay_to_1e-6`	`1.5`	`log`	5	
architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
6C2F GloRo	CIFAR-10	`tfds`	0	512	800	0.141	0.141
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`orthogonal`	1e-3	`decay_to_1e-6`	`1.2`	`log`	5	
architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
8C2F GloRo	Tiny-Imagenet	`default`	0	512	800	0.141	0.141
	optimizer	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	1e-4	`decay_to_5e-6`	`1.2,10,800`	`log`	5	

Table B.1: Hyperparameters used for training (MinMax) GloRo Nets.

ϵ scheduling. Prior work has also established a convention of gradually scaling ϵ up to a value that is potentially larger than the one used to measure verified accuracy leads to better results. We made use of the following schemes for accomplishing this.

- No scheduling: we use '`single`' to denote we ϵ_{train} for all epochs.
- Linear scheduling: we use a string '`x,y,e`' to denote the strategy that at training epoch t, we use $\epsilon_t = x + (y - x) * (t/e)$ if $t \leq e$. When $t > e$, we use the provided ϵ_{train} to keep training the model.
- Logarithmic scheduling: we use '`log`' to denote that we increase the epsilon with a logarithmic rate from 0 to ϵ_{train}.

We found that scheduling ϵ is often unnecessary when instead scheduling the TRADES parameter λ (discussed later in this section), which appears to be more effective for that loss. To select a scheme for scheduling ϵ, we compared the results of the three options listed above, and selected the one that achieved the highest verified accuracy. If there was no significant difference in this metric, then we instead selected the schedule with the least complexity, assuming the following order: `single`, (`x,y,e`), `log`. When applying (`x,y,e`) and `log`, we began the schedule on the first epoch, and ended it on $(\#\text{ epochs})/2$.

Initialization & optimization. In Table B.2, `default` refers to the Glorot uniform initialization, given by `tf.keras.initializers.GlorotUniform()`. The string '`ortho`' refers to an orthogonal initialization given by `tf.keras.initializers.Orthogonal()`. To select an initialization, we compared the verified accuracy achieved by either, and selected the one with the highest metric. In the case of a tie, we defaulted to the Glorot uniform initialization. We used the `adam` optimizer to perform gradient descent in all experiments, with the initial learning rate specified in Table B.2 and B.1, and default values for the other hyperparameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e{-}07$, `amsgrad` disabled).

Learning rate scheduling. We write '`decay_to_lb`' to denote a schedule that continuously decays the learning rate to `lb` at a negative-exponential rate, starting the decay at $(\#\text{epochs})/2$. To select `lb`, we searched over values $\texttt{lb} \in \{1\times10^{-7}, 5\times10^{-7}, 1\times10^{-6}, 5\times10^{-6}\}$, selecting the value that led to the best VRA. We note that for all datasets except Tiny-Imagenet, we used the default initial rate of 1×10^{-3}. On Tiny-Imagenet, we observed that after several

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
2C2F GloRo (CE)	MNIST	`None`	0	256	500	0.45	0.3
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	1e-3	`decay_to_1e-6`	`CE`	`single`	10	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
2C2F GloRo (T)	MNIST	`None`	0	256	500	0.45	0.3
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	1e-3	`decay_to_1e-6`	`0,2,500`	`single`	10	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
4C3F GloRo (CE)	MNIST	`None`	0	256	300	1.75	1.58
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	1e-3	`decay_to_5e-6`	`CE`	`single`	10	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
4C3F GloRo (T)	MNIST	`None`	0	256	300	1.75	1.58
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	1e-3	`decay_to_5e-6`	`0,3,300`	`single`	10	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
6C2F GloRo (CE)	CIFAR-10	`default`	20	256	800	0.1551	0.141
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`orthogonal`	1e-3	`decay_to_1e-6`	`CE`	`log`	5	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
6C2F GloRo (T)	CIFAR-10	`default`	20	256	800	0.1551	0.141
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	1e-3	`decay_to_1e-6`	`1.2`	`log`	5	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
8C2F GloRo (CE)	Tiny-Imagenet	`default`	0	256	250	0.16	0.141
	optimizer	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	2.5e-4	`decay_to_5e-7`	`CE`	`single`	5	

architecture	*dataset*	*data augmentation*	*warm-up*	*batch size*	*# epochs*	ϵ_{train}	ϵ_{test}
8C2F GloRo (T)	Tiny-Imagenet	`default`	0	256	500	0.16	0.141
	initialization	*init_lr*	*lr_decay*	*loss*	*ϵ schedule*	*power_iter*	
	`default`	2.5e-4	`decay_to_5e-7`	`1,10,500`	`single`	1	

Table B.2: Hyperparameters used for training (ReLU) GloRo Nets. We provide models trained with both TRADES (Definition 3) loss (denoted by "(T)") and with cross-entropy loss (denoted by "(CE)").

epochs at this rate, as well as at 5×10^{-4}, the loss failed to decrease, so again halved it to arrive at 2.5×10^{-4}.

Batch size & epochs. For all experiments, we used minibatches of 256 instances. Because our method does not impose a significant memory overhead, we found that this batch size made effective use of available hardware resources, increasing training time without impacting verified accuracy, when compared to minibatch sizes 128 and 512. Because the learning rate, ϵ, and λ schedules are all based on the total number of epochs, and can have a significant effect on the verified accuracy, we did not monitor convergence to determine when to stop training. Instead, we trained for epochs in the range $[100,1000]$ in increments of 100, and when verified accuracy failed to increase with more epochs, attempted training with fewer epochs (in increments of 50), stopping the search when the verified accuracy began to decrease again.

Warm-up. Lee et al. (2020) noted improved performance when models were pre-trained for a small number of epochs before optimizing the robust loss. We found that this helped in some cases with GloRo networks as well, in particular on CIFAR-10, where we used the same number of warm-up epochs as prior work.

λ scheduling. When using the TRADES loss described in Section 4, we found that scheduling λ often yielded superior results. We use '`x,y,e`' to denote that at epoch t, we set $\lambda_t=x+(y-x)*(t/e)$ if $t<e$ else $\lambda_t=y$. We write '`x`' to denote we use $\lambda=x$ all the time. To select the final λ, we trained on values in the range $[1,10]$ in increments of 1, and on finding the whole number that yielded the best result, attempted to further refine it by increasing in increments 0.1.

Power iteration. As discussed in Section 4, we use power iteration to compute the spectral norm for each layer to find the layer-wise Lipschitz constants. In Table B.2, *power_iter* denotes the number of iterations we run for each update during training. We tried values in the set $\{1,2,5,10\}$, breaking ties to favor fewer iterations for less training cost. After each epoch, we ran the power iteration until convergence (with tolerance 1×10^{-5}), and all of the verified accuracy results reported in Section 5 are calculated using a global bound based on running power iteration to convergence as well. Since the random variables used in the power iterations are initialized as `tf.Variables`, they are stored in `.h5` files together with the architecture and network parameters. Therefore, one can directly use the converged random variables from the training phase during the test phase.

Search strategy. Because of the number of hyperparameters involed in our evaluation, and limited hardware resources, we did not perform a global grid search over all combinations of hyperparameters discussed here. We plan to do so in future work, as it is possible that results could improve as we may have missed better settings than those explored to produce the numbers reported in our evaluation. Instead, we adopted a greedy strategy, prioritizing the choices that we believed would have the greatest impact on verified accuracy and training cost. In general, we explored parameter choices in the following order: ϵ schedule, λ schedule, # epochs, LR decay, warm-up, initialization, # power iterations, minibatch size.

Deterministic Guarantees

method	Clean (%)	PGD (%)	VRA(%)
MNIST ($\epsilon=0.3$)			
GloRo	99.0	97.8	95.7
BCP	93.4	89.5	84.7
KW	98.9	97.8	94.0
MMR	98.2	96.2	93.6
MNIST ($\epsilon=1.58$)			
GloRo	97.0	81.9	62.8
BCP	92.4	65.8	47.9
KW	88.1	67.9	44.5
BCOP	98.8	-	56.7
LMT	86.5	53.6	40.6
CIFAR ($\epsilon={}^{36}/_{255}$)			
GloRo	77.0	69.2	58.4
BCP	65.7	60.8	51.3
KW	60.1	56.2	50.9
Cayley	75.3	67.6	59.1
BCOP	72.4	64.4	58.7
LMT	63.1	58.3	38.1

Stochastic Guarantees

method	Clean (%)	PGD (%)	VRA(%)
CIFAR ($\epsilon=0.5$)			
RS	67.0	-	49.0
SmoothADV	-	-	63.0
MACER	81.0	-	59.0

Table C.1: Comprehensive VRA comparisons for deterministic guarantees and stochastic guarantees. Best results reported in the literature are included in the table.

C. Comprehensive VRA Comparisons

In Section 5 of the main paper, we compare, in depth, the performance of GloRo Nets to two approaches to deterministic certification that have been reported as achieving the state-of-the-art in recently published work on robustness certification (Lee et al., 2020).

For completeness, we present a brief overview of a wider range of approaches, providing the VRAs reported in the original respective papers. Table C.1 contains VRAs reported by several other approaches to deterministic certification, including the methods compared against in Section 5: KW (Wong

& Kolter, 2018) and BCP (Lee et al., 2020); work that was superseded by KW or BCP: MMR (Croce et al., 2019) and LMT (Tsuzuku et al., 2018); work that we became aware of after the completion of this work: BCOP (Li et al., 2019); and concurrent work: Cayley (Trockman & Kolter, 2021). In addition, we include work that provides a *stochastic* guarantee: Randomized Smoothing (RS) (Cohen et al., 2019a), SmoothADV (Salman et al., 2019), and MACER (Zhai et al., 2020). The results for stochastic certification typically use different radii, as reflected in Table C.1. We note that because these numbers are taken from the respective papers, the results in Table C.1 should be interpreted as ball-park figures, as they do not standardize the architecture, data scaling and augmentation, etc., and are thus not truly "apples-to-apples."

We find that GloRo Nets provide the highest VRA for both $\epsilon = 0.3$ and $\epsilon = 1.58$ on MNIST. GloRo Nets also match the result on CIFAR-10 from concurrent work, Cayley, coming within one percentage point of the VRA reported by Trockman & Kolter.

D. MinMax vs. ReLU GloRo Nets

Recently, Anil et al. (2019) proposed replacing ReLU activations with sorting activations to construct a class of *universal Lipschitz approximators*, that that can approximate any Lipschitz-bounded function over a given domain, and Cohen et al. (2019b) subsequently studied the application to robust training. We found that these advances in architecture complement our work, improving the VRA performance of GloRo Nets substantially compared to ReLU activations.

The results achieved by GloRo Nets in Figure 4a in Section 5 of the main paper are achieved using *MinMax* activations (Anil et al., 2019) rather than ReLU activations. Figure D.1a shows a comparison of the VRA that can be achieved by GloRo Nets using ReLU activations as opposed to MinMax activations. We see that in each case, the GloRo Nets using MinMax activations outperform those using ReLU activations by a several percentage points. Nonetheless, the ReLU-based GloRo Nets are still competitive with the VRA performance of KW and BCP.

We observed that MinMax activations result in a slight penalty to training and evaluation cost, as they are slightly slower to compute than ReLU activations. Figures D.1a and D.1b provide the cost in terms of time and memory incurred by GloRo Nets using each activation function. We see that the MinMax-based GloRo Nets are slightly slower and more memory-intensive; however, the difference is not particularly significant.

Finally, we compared the Lipschitz bounds obtained on MinMax and ReLU GloRo Nets, presented in Figure D.1c. We see that the Lipschitz bounds are fairly similar, in terms of both their magnitude a well as their tightness with respect to the empirical lower bounds.

E. Measuring Memory Usage

In our experiments, we used Tensorflow to train and evaluate standard and GloRo networks, and Pytorch to train and evaluate KW and BCP (since Wong & Kolter (2018) and Lee et al. (2020) implement their respective methods in Pytorch). To measure memory usage, we invoked `tf.contrib.memory_stats.MaxBytesInUse()` at the end of each epoch for standard and GloRo networks, and took the peak active use from `torch.cuda.memory_summary()` at the end of each epoch for KW and BCP.

We note that some differences may arise as a result of differences in memory efficiency between Tensorflow and Pytorch. In particular, Pytorch enables more control over memory management than does Tensorflow. In order to mitigate this difference as much as possible, we did not disable gradient tracking when evaluating certification times and memory usage in Pytorch. While gradient tracking is unnecessary for certification (it is only required for training), Tensorflow does not allow this optimization, so by forgoing it the performance results recorded in Figure 4b in Section 5 are more comparable across frameworks.

In Section 5.2, we note that Randomized Smoothing (RS) training times have been omitted. This is because RS essentially acts as a post-processing method on top of a pre-trained model. In practice the only difference between the training routine to produce a model for RS and standard training is the addition of Gaussian noise (mathcing the noise radius used for smoothing) to the data augmentation; we assume that this has a negligible impact on training cost.

F. Optimizing for Lipschitz Lower Bounds

Figure 4c in Section 5 gives empirical lower bounds on the global and average local Lipschitz constants on the models trained in our evaluation. We use optimization to obtain these lower bounds; further details are provided below.

Global Lower Bounds. We use the *margin Lipschitz constant*, K^*_{ij} (Definition 4 in Appendix A.2), which takes a different value for each pair of classes, i and j. To obtain the lower bound we optimize

$$\max_{x_1,x_2} \max_{i} \left\{ \frac{|f_{j_1}(x_1) - f_i(x_1) - (f_{j_1}(x_2) - f_i(x_2))|}{||x_1 - x_2||} \right\}$$

where $j_1 = F(x_1)$. Optimization is performed using Keras' default `adam` optimizer with 7,500 gradient steps. Both x_1 and x_2 are initialized to random points in the test set; we perform this optimization over 100 such initial pairs, and report the maximum value obtained over all initializations.

method	Model	Clean (%)	PGD (%)	VRA(%)	Sec./epoch	# Epochs	Mem. (MB)
MNIST ($\epsilon = 0.3$)							
ReLU GloRo (CE)	2C2F	98.4	96.9	94.6	0.7	500	0.7
ReLU GloRo (T)	2C2F	98.7	97.4	94.6	0.7	500	0.7
MinMax GloRo	2C2F	99.0	97.8	**95.7**	0.9	500	0.7
MNIST ($\epsilon = 1.58$)							
ReLU GloRo (CE)	4C3F	92.9	68.9	50.1	2.3	300	2.2
ReLU GloRo (T)	4C3F	92.8	67.0	51.9	2.0	300	2.2
MinMax GloRo	4C3F	97.0	81.9	**62.8**	3.7	300	2.7
CIFAR-10 ($\epsilon = 36/255$)							
ReLU GloRo (CE)	6C2F	70.7	63.8	49.3	3.2	800	2.6
ReLU GloRo (T)	6C2F	67.9	61.3	51.0	3.3	800	2.6
MinMax GloRo	6C2F	77.0	69.2	**58.4**	6.9	800	3.6
Tiny-Imagenet ($\epsilon = 36/255$)							
ReLU GloRo (CE)	8C2F	31.3	28.2	13.2	14.0	250	7.3
ReLU GloRo (T)	8C2F	27.4	25.6	15.6	13.7	500	7.3
MinMax GloRo	8C2F	35.5	32.3	**22.4**	40.3	800	10.4

(a)

method	Model	Time (sec.)	Mem. (MB)
ReLU GloRo	6C2F	0.2	2.5
MinMax GloRo	6C2F	0.4	1.8

(b)

method	global UB	global LB	local LB
MNIST ($\epsilon = 1.58$)			
Standard	$5.4 \cdot 10^4$	$1.4 \cdot 10^2$	17.1
ReLU GloRo	3.2	3.0	2.1
MinMax GloRo	2.3	1.9	0.8
CIFAR-10 ($\epsilon = 36/255$)			
Standard	$1.2 \cdot 10^7$	$1.1 \cdot 10^3$	96.2
ReLU GloRo	18.9	11.4	6.2
MinMax GloRo	15.8	11.0	3.7
Tiny-Imagenet ($\epsilon = 36/255$)			
Standard	$2.2 \cdot 10^7$	$3.6 \cdot 10^2$	40.7
ReLU GloRo	7.7	3.3	1.5
MinMax GloRo	12.5	5.9	0.8

(c)

Figure D.1: **(a)** Certifiable training evaluation results on benchmark datasets. Best results highlighted in bold. For ReLU GloRo Nets, we provide models trained with both TRADES (Definition 3) and with cross-entropy: "(T)" indicates that TRADES loss was used and "(CE)" indicates that cross-entropy was used. **(b)** Certification timing and memory usage results on CIFAR-10 ($\epsilon = 36/255$). **(c)** Upper and lower bounds on the global and average local Lipschitz constant. In (a) and (b), peak GPU Memory usage is calculated per-instance by dividing the total measurement by the training or certification batch size.

Local Lower Bounds. We use a variant of the *margin Lipschitz constant* (Definition 4 in Appendix A.2) analogous to the local Lipschitz constant at a point, x_0, with radius ϵ. To obtain this lower bound we optimize

$$\max_{x_1, x_2} \max_i \left\{ \frac{|f_j(x_1) - f_i(x_1) - (f_j(x_2) - f_i(x_2))|}{||x_1 - x_2||} \right\}$$

$$\text{subject to } ||x_1 - x_0|| \le \epsilon, ||x_2 - x_0|| \le \epsilon$$

where $j = F(x_0)$. Optimization is performed using Keras' default `adam` optimizer with 5,000 gradient steps. After each gradient step, x_1 and x_2 are projected onto the ϵ-ball centered at x_0. Both x_1 and x_2 are initialized to random points in the test set, and x_0 is a fixed random point in the test set. We perform this optimization over 100 random choices of x_0, and report the mean value.

Discussion. In Section 5.3, we observe that the global upper bound is fairly tight on the GloRo Net trained on MNIST, but decreasingly so on CIFAR-10 and Tiny-Imagenet. While this suggests that there is room for improvement in terms of the bounds obtained by GloRo Nets, we make note of two subtleties that may impact these findings.

First, the tightness decreases inversely with the dimensionality of the input. While it is reasonable to conclude that learning tight GloRo Nets in higher dimensions becomes increasingly difficult, it is worth noting that the optimization process described above also becomes more difficult in higher dimensions, meaning that some of the looseness may be attributable to looseness in the *lower* bound, rather than in the upper bound.

Second, the hyperparameters used may have an effect on the tightness of the Lipschitz bound. As seen in Appendix B, different hyperparameters were used on MNIST, CIFAR-10, and Tiny-Imagenet; some of these differences were selected based on impacting training time, which is of greater concern for larger datasets that naturally take longer to train. Specifically, we note that fewer power iterations were used for CIFAR-10, and even fewer for Tiny-Imagenet. While this is good for expediency, and still produces state-of-the-art VRA, we note that tighter bounds may be learned by putting more computation time into training, in the form of more power iterations (for example). More generally, this speculation suggests that slightly different training strategies, hyperparameters, etc., from the ones used in this work may be sufficient to improve the bounds and the VRA achieved by GloRo Nets. We conclude that future work should further explore this possibility.

References

Anil, C., Lucas, J., and Gross, R. Sorting out Lipschitz function approximation. In *ICML*, 2019.

Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International*

Conference on Machine Learning (ICML), 2019a.

Cohen, J. E., Huster, T., and Cohen, R. Universal lipschitz approximation in bounded depth neural networks. *arXiv preprint arXiv:1904.04861*, 2019b.

Croce, F., Andriushchenko, M., and Hein, M. Provable robustness of ReLU networks via maximization of linear regions. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

Lee, S., Lee, J., and Park, S. Lipschitz-certifiable training with a tight outer bound. In *NIPS*, 2020.

Li, Q., Haque, S., Anil, C., Lucas, J., Grosse, R. B., and Jacobsen, J.-H. Preventing gradient attenuation in lipschitz constrained convolutional networks. In *Advances in Neural Information Processing Systems*, 2019.

Salman, H., Li, J., Razenshteyn, I., Zhang, P., Zhang, H., Bubeck, S., and Yang, G. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, 2019.

Trockman, A. and Kolter, J. Z. Orthogonalizing convolutional layers with the cayley transform. In *International Conference on Learning Representations*, 2021.

Tsuzuku, Y., Sato, I., and Sugiyama, M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. In *NIPS*, 2018.

Wong, E. and Kolter, J. Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

Wong, E., Schmidt, F., Metzen, J. H., and Kolter, J. Z. Scaling provable adversarial defenses. In *NIPS*, 2018.

Zhai, R., Dan, C., He, D., Zhang, H., Gong, B., Ravikumar, P., Hsieh, C.-J., and Wang, L. Macer: Attack-free and scalable robust training via maximizing certified radius. In *International Conference on Learning Representations*, 2020.