Rethinking Lipschitz Neural Networks and Certified Robustness: A Boolean Function Perspective

Bohang Zhang[1,3,4] **Du Jiang**[1] **Di He**[1,†] **Liwei Wang**[1,2,†]
[1]Key Laboratory of Machine Perception, MOE, School of Intelligence Science and Technology, Peking University [2]Center for Data Science, Peking University
[3] Peng Cheng Laboratory [4] Pazhou Laboratory (Huangpu) [†]Corresponding author
{zhangbohang,dujiang,dihe}@pku.edu.cn, wanglw@cis.pku.edu.cn

Abstract

Designing neural networks with bounded Lipschitz constant is a promising way to obtain certifiably robust classifiers against adversarial examples. However, the relevant progress for the important ℓ_∞ perturbation setting is rather limited, and a principled understanding of how to design expressive ℓ_∞ Lipschitz networks is still lacking. In this paper, we bridge the gap by studying certified ℓ_∞ robustness from a novel perspective of representing Boolean functions. We derive two fundamental impossibility results that hold for any standard Lipschitz network: one for robust classification on finite datasets, and the other for Lipschitz function approximation. These results identify that networks built upon norm-bounded affine layers and Lipschitz activations intrinsically lose expressive power even in the two-dimensional case, and shed light on how recently proposed Lipschitz networks (e.g., GroupSort and ℓ_∞-distance nets) bypass these impossibilities by leveraging *order statistic functions*. Finally, based on these insights, we develop a unified Lipschitz network that generalizes prior works, and design a practical version that can be efficiently trained (making certified robust training free). Extensive experiments show that our approach is scalable, efficient, and consistently yields better certified robustness across multiple datasets and perturbation radii than prior Lipschitz networks.

1 Introduction

Modern neural networks, despite their great success in various applications [22, 13], typically suffer from a severe drawback of lacking adversarial robustness. In classification tasks, given an image $\boldsymbol{x}$ correctly classified by a neural network, there often exists a small adversarial perturbation $\boldsymbol{\delta}$, making the perturbed image $\boldsymbol{x} + \boldsymbol{\delta}$ look indistinguishable to humans but fool the network to predict a wrong class with high confidence [57, 4].

It is well-known that the adversarial robustness of a neural network is closely related to its Lipschitz continuity [8, 61] (see Section 3.1). Accordingly, training neural networks with bounded Lipschitz constant has been considered a promising way to address the problem. A variety of works studied Lipschitz architectures for the ordinary Euclidean norm [61, 60, 36, 54], and recent works even established state-of-the-art (deterministic) certified ℓ_2 robustness [55, 40]. However, when it comes to the more critical (realistic) ℓ_∞ perturbation setting, the progress seems to be rather limited. In fact, standard Lipschitz ReLU networks have been shown to perform poorly in terms of ℓ_∞ robustness [61, 24, 2]. While other more advanced Lipschitz networks have been proposed [2, 10], the achieved results are still not satisfactory even on simple datasets like MNIST. Until recently, Zhang et al. [73, 74] designed a quite *unusual* Lipschitz network based on a heuristic choice of the ℓ_∞*-distance function*, which surprisingly established state-of-the-art certified ℓ_∞ robustness on multiple datasets over prior works. Yet, it remains unclear why previous Lipschitz networks typically failed, and what is the essential reason behind the success of this particular ℓ_∞-distance network structure.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Theoretical contributions. In this work, we systematically investigate how to design expressive Lipschitz neural networks (w.r.t. ℓ_∞-norm) through the novel lens of representing discrete Boolean functions, which provides a deep understanding on the aforementioned problems. Specifically, we first figure out a fundamental limitation of standard Lipschitz networks in representing a class of logical operations called *symmetric Boolean functions* (SBF), which comprises the basic logical AND/OR as special cases. We prove that for any non-constant SBF of d variables, there exists a finite dataset of size $\mathcal{O}(d)$ such that the certified ℓ_∞ robust radius must vanish as $\mathcal{O}(1/d)$ for any classifier induced by a standard Lipschitz network. Remarkably, since logical AND/OR operations correspond to perhaps the most basic classifiers, our result indicates an intrinsic difficulty of such networks in fitting high-dimensional real-world datasets with guaranteed certified ℓ_∞ robustness.

Our analysis can be readily extended into the Lipschitz function approximation setting. We point out the relationship between *monotonic* SBF and the *order statistics* (which are 1-Lipschitz functions), and then prove that any d-dimensional order statistic (including the max/min function) on a compact domain cannot be approximated by standard Lipschitz networks with error $\mathcal{O}(1 - 1/d)$, regardless of the network size. This impossibility result is significant in that: (i) it applies to all Lipschitz activations (thus extending prior works [2, 24]), (ii) it resolves an open problem raised recently in [43], and (iii) a *quantitative* lower bound of approximation error is established.

Equipped by the above impossibility results, we proceed to examine two advanced Lipschitz architectures: the GroupSort network [2] and the recently proposed ℓ_∞-distance net [73, 74]. We find that besides the linear operation, both networks incorporate other Lipschitz aggregation operations into the neuron design, especially the order statistic functions, thus shedding light on how they work. However, for the MaxMin network [2] — a computationally efficient version of the GroupSort network implemented in practice, representing Boolean functions and order statistics is possible only when the network is very *deep*. In particular, we prove that representing certain d-dimensional Boolean functions requires a depth of $\Omega(d)$, implying that shallow MaxMin networks are not Lipschitz-universal function approximators. In contrast, we show a *two-layer* ℓ_∞-distance net suffices to represent any order statistic function on a compact domain or even all Boolean functions. This strongly justifies the empirical success of ℓ_∞-distance net over GroupSort (MaxMin) networks.

Practical contributions. Our theoretical insights can also guide in designing better Lipschitz network architectures. Inspired by the importance of order statistics, we propose a general form of Lipschitz network, called SortNet, that extends both GroupSort and ℓ_∞ distance networks and incorporates them into a unified framework. Yet, the full-sort operation is computationally expensive and leads to optimization difficulties (as with the GroupSort network). We further propose a specialized SortNet that can be efficiently trained, by assigning each weight vector $\boldsymbol{w}$ using *geometric series*, i.e. w_i proportional to ρ^i for some $0 \leq \rho < 1$. This leads to a restricted version of SortNet but still covers ℓ_∞-distance net as a special case. For this particular SortNet, we skillfully derive a stochastic estimation that gives an unbiased approximation of the neuron output without performing sorting operations explicitly. This eventually yields an efficient training strategy with similar cost as training standard networks, thus making certified robust training free. Extensive experiments demonstrate that the proposed SortNet is scalable, efficient, and consistently achieves better certified robustness than prior Lipschitz networks across multiple datasets and perturbation radii. In particular, our approach even scales on a variant of ImageNet, and surpasses the best-known result [69] with a 22-fold decrease in training time thanks to our "free" certified training approach.

The contribution and organization of this paper can be summarized as follows:

- We develop a systematic study for the expressive power of Lipschitz neural networks using the tools of Boolean function theory. We prove the impossibility results of standard Lipschitz networks in two settings: a) certified ℓ_∞ robustness on discrete datasets (Section 3.2); b) Lipschitz function approximation (Section 3.3).
- We provide insights into how recently proposed networks can bypass the impossibility results. In particular, we show that a *two-layer* ℓ_∞-distance net can precisely represent any Boolean functions, while *shallow* GroupSort networks cannot (Section 3.4).
- We propose SortNet, a Lipschitz network that generalizes GroupSort and ℓ_∞-distance net. For a special type of SortNet, we derive a stochastic training approach that bypasses the difficulties in calculating sorting operations explicitly and makes certified training free (Section 4).
- Extensive experiments demonstrate that SortNet exhibits better certified robustness on several benchmark datasets over baseline methods with high training efficiency (Section 5).

2 Related Work

Extensive studies have been devoted to developing neural networks with certified robustness guarantees. Existing approaches can be mainly divided into the following three categories.

Certified defenses for standard networks. A variety of works focus on establishing certified robustness for standard neural networks. However, exactly calculating the certified radius of a standard ReLU network is known to be NP-hard [28]. Researchers thus developed a class of relaxation-based approaches that provide a tight lower bound estimate of the certified robustness efficiently. These approaches typically use convex relaxation to calculate a bound of the neuron outputs under input perturbations layer by layer [65, 66, 64, 53, 42, 19, 62, 76]. See also [3, 15, 16, 47, 48, 68, 11, 35, 63, 23, 51] for more advanced approaches. However, most of these works suffer from high computational costs and are hard to scale up to large datasets. Currently, the only scalable convex relaxation approach is based on *interval bound propagation* (IBP) [42, 21, 75, 69, 52], but the produced bound is known to be loose [50], and a recent study showed that IBP cannot achieve enough certified robustness on simple datasets for any standard ReLU network [41].

Certified defenses using Lipschitz networks. On the other hand, Lipschitz networks inherently imply certified robustness, resulting in a *much simpler* certification process based on the output margin (see Proposition 3.1). Yet, most prior works can only handle the ℓ_2-norm Lipschitz situation by leveraging specific mathematical properties such as the spectral norm [8, 71, 20, 61, 17, 46, 2, 36, 40] or orthogonality of weight matrices [39, 60, 54, 55]. For the ℓ_∞-norm, standard Lipschitz networks were shown to give only a vanishingly small certified radius [61]. Huster et al. [24] found that standard Lipschitz ReLU networks cannot represent certain simple functions such as the absolute value, which inspired the first expressive Lipschitz architecture called the GroupSort network [2]. Since then, GroupSort has been extensively investigated [10, 58], but its performance is still much worse than the above relaxation-based approaches even on MNIST. Recently, Zhang et al. [73, 74] first proposed a practical 1-Lipschitz architecture w.r.t. ℓ_∞-norm based on a special neuron called the ℓ_∞-distance neuron, which can scale to TinyImageNet with state-of-the-art certified robustness over relaxation-based approaches. However, despite its practical success, it is rather puzzling how such a simple architecture can work while prior approaches all failed. Answering this question may require an in-depth re-examination of Lipschitz networks (w.r.t. ℓ_∞-norm), which is the focus of this paper.

Certified defenses via randomized smoothing. As a rather different and parallel research line, randomized smoothing typically provides *probabilistic* certified ℓ_2 robustness guarantees. Due to the wide applicability, randomized smoothing has been scaled up to ImageNet and achieves state-of-the-art certified accuracy for ℓ_2 perturbations [34, 38, 9, 49, 72, 27]. However, certifying robustness with high probability requires sampling a large number of noisy inputs (e.g., 10^5) for a single image, leading to a high computational cost at inference. Moreover, theoretical results pointed out that it cannot achieve non-trivial certified ℓ_∞ robustness if the perturbation radius is larger than $\Omega(d^{-1/2})$ where d is the input dimension [70, 5, 30, 67].

3 The Expressive Power of Lipschitz Neural Networks

3.1 Preliminaries

Notations. We use boldface letters to denote vectors (e.g., $\boldsymbol{x}$) or vector functions (e.g., $\boldsymbol{f}$), and use x_i (or f_i) to denote its i-th element. For a unary function σ, $\sigma(\boldsymbol{x})$ applies $\sigma(\cdot)$ element-wise on vector $\boldsymbol{x}$. The ℓ_p-norm ($p \geq 1$) and ℓ_∞-norm of a vector $\boldsymbol{x}$ are defined as $\|\boldsymbol{x}\|_p = (\sum_i |x_i|^p)^{1/p}$ and $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$, respectively. The matrix ∞-norm is defined as $\|\mathbf{W}\|_\infty = \max_i \|\mathbf{W}_{i,:}\|_1$ where $\mathbf{W}_{i,:}$ is the i-th row of the matrix $\mathbf{W}$. The k-th largest element of a vector $\boldsymbol{x}$ is denoted as $x_{(k)}$. We use $[n]$ to denote the set $\{1, \cdots, n\}$, and use $\boldsymbol{e}_i$ to denote the unit vector with the i-th element being one. We adopt the big O notations by using $\mathcal{O}(\cdot)$, $\Omega(\cdot)$, and $\Theta(\cdot)$ to hide universal constants.

Lipschitzness. A mapping $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is said to be L-Lipschitz continuous w.r.t. norm $\|\cdot\|$ if for any pair of inputs $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$,

$$\|\boldsymbol{f}(\boldsymbol{x}_1) - \boldsymbol{f}(\boldsymbol{x}_2)\| \leq L\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|. \tag{1}$$

If the mapping $\boldsymbol{f}$ represented by a neural network has a small Lipschitz constant L, then (1) implies that the change of network output can be strictly controlled under input perturbations, resulting in *certified* robustness guarantees as shown in the following proposition.

Proposition 3.1. (Certified robustness of Lipschitz networks) *For a neural network $\boldsymbol{f}$ with Lipschitz constant L under ℓ_p-norm $\|\cdot\|_p$, define the resulting classifier g as $g(\boldsymbol{x}) := \arg\max_k f_k(\boldsymbol{x})$ for an input $\boldsymbol{x}$. Then g is provably robust under perturbations $\|\boldsymbol{\delta}\|_p < \frac{c}{L}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$, i.e.*

$$g(\boldsymbol{x}+\boldsymbol{\delta}) = g(\boldsymbol{x}) \quad \text{for all } \boldsymbol{\delta} \text{ with } \|\boldsymbol{\delta}\|_p < c/L \cdot \operatorname{margin}(\boldsymbol{f}(\boldsymbol{x})). \tag{2}$$

Here $c = \sqrt[p]{2}/2$ is a constant depending only on the norm $\|\cdot\|_p$, which is $1/2$ for the ℓ_∞-norm, and $\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$ is the margin between the largest and second largest output logits.

The proof of Proposition 3.1 is simple and can be found in Appendix B.1 or [39, Appendix P]. It can be seen that the robust radius is inversely proportional to the Lipschitz constant L.

Standard Lipschitz networks. Throughout this paper, we refer to standard neural networks as neural networks formed by affine layers (e.g., fully-connected or convolutional layers) and element-wise activation functions. Based on the Lipschitz property of composite functions, most prior works enforce the 1-Lipschitzness of a multi-layer neural network by constraining each layer to be a 1-Lipschitz mapping. For the ℓ_∞-norm, it is further equivalent to constraining the weight matrices to have bounded ∞-norm, plus using Lipschitz activation functions [2], which can be formalized as

$$\boldsymbol{x}^{(l)} = \sigma^{(l)}(\mathbf{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}) \quad \text{s.t. } \|\mathbf{W}^{(l)}\|_\infty \le 1 \text{ and } \sigma^{(l)} \text{ being 1-Lipschitz, } l \in [M]. \tag{3}$$

Here M is the number of layers and usually $\sigma^{(M)}(x) = x$ is the identity function. The network takes $\boldsymbol{x}^{(0)} := \boldsymbol{x}$ as the input and outputs $\boldsymbol{x}^{(M)}$. For K-class classification problems, $\boldsymbol{x}^{(M)} \in \mathbb{R}^K$ and the network predicts the class $g(\boldsymbol{x}) := \arg\max_{k\in[K]} x_k^{(M)}$. We refer to the resulting network as a standard Lipschitz network.

3.2 Certified robustness on discrete Boolean datasets

In this section, we will construct a class of counterexamples for which certified ℓ_∞ robustness can be arbitrarily poor using standard Lipschitz networks. We focus on the *Boolean dataset*, a discrete dataset where both inputs and labels are Boolean-valued and the relationship between inputs and their labels $(\boldsymbol{x}^{(i)}, y^{(i)}) \in \{0,1\}^d \times \{0,1\}$ can be described using a Boolean function $g^{\mathrm{B}} : \{0,1\}^d \to \{0,1\}$. The key motivation lies in the finding that Boolean vectors correspond to the vertices of a d-dimensional hypercube, and thus are geometrically related to the ℓ_∞-distance metric. In particular, the ℓ_∞-distance between any two different data points in a Boolean dataset is always 1, which means that the dataset is *well-separated*. This yields the following proposition, stating that it is always possible to achieve *optimal* certified ℓ_∞ robustness on Boolean datasets by using Lipschitz classifiers.

Proposition 3.2. *For any Boolean dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^n$, there exists a classifier $\hat{g} : \mathbb{R}^d \to \{0,1\}$ induced by a 1-Lipschitz mapping $\hat{\boldsymbol{f}} : \mathbb{R}^d \to \mathbb{R}^2$, such that $\hat{g}$ can fit the whole dataset with $\operatorname{margin}(\hat{\boldsymbol{f}}(\boldsymbol{x}^{(i)})) = 1\ \forall i \in [n]$, thus achieving a certified ℓ_∞ radius of $1/2$ by Proposition 3.1.*

The key observation in the proof (Appendix B.2) is that one can construct a so-called *nearest neighbor classifier* that achieves a large margin on the whole dataset and is *1-Lipschitz*. Based on Proposition 3.2, it is natural to ask whether standard Lipschitz networks of the form (3) can perform well on Boolean datasets. Unfortunately, we show it is not the case, even on a class of simple datasets constructed using *symmetric* Boolean functions.

Definition 3.3. A Boolean function $g^{\mathrm{B}} : \{0,1\}^d \to \{0,1\}$ is symmetric if it is invariant under input permutation, i.e. $g^{\mathrm{B}}(x_1, \cdots, x_d) = g^{\mathrm{B}}(x_{\pi(1)}, \cdots, x_{\pi(d)})$ for any $\boldsymbol{x} \in \{0,1\}^d$ and $\pi \in S_d$.

Example 3.4. Two of the most basic operations in Boolean algebra are the logical AND/OR, both of which belong to the class of symmetric Boolean functions. Other important examples include the exclusive-or (XOR, also called the parity function), NAND, NOR, and the majority function (or generally, the threshold functions). See Appendix A.2 for a detailed description of these examples.

Theorem 3.5. *For any non-constant symmetric Boolean function $g^B : \{0,1\}^d \to \{0,1\}$, there exists a Boolean dataset with labels $y^{(i)} = g^B(\boldsymbol{x}^{(i)})$, such that no standard Lipschitz network can achieve a certified ℓ_∞ robust radius larger than $1/2d$ on the dataset.*

Implications. Theorem 3.5 shows that the certified radius of standard Lipschitz networks must vanish as dimension d grows, which is in stark contrast to the constant radius given by Proposition 3.2. Also, note that the logical AND/OR functions are perhaps the most basic classifiers (which simply make predictions based on the existence of binary input features). It is thus not surprising to see that standard Lipschitz networks perform poorly on real-world datasets (e.g., even the simple MNIST dataset where input pixels are almost Boolean-valued (black/white) [61]).

We give an elegant proof of Theorem 3.5 in Appendix B.3, where we also prove that the bound of $1/2d$ is *tight* in Proposition B.8. Moreover, we discuss the sample complexity by proving that a dataset of size $\mathcal{O}(d)$ already suffices to give $\mathcal{O}(1/d)$ certified radius (Corollary B.10). To the best of our knowledge, Theorem 3.5 is the first impossibility result that targets the certified ℓ_∞ robustness of standard Lipschitz networks with an *quantitative* upper bound on the certified radius. In the next section, we will extend our analysis to the function approximation setting and make discussions with literature results [24, 2].

3.3 Lipschitz function approximation

Classic approximation theory has shown that standard neural networks are universal function approximators [12, 37], in that they can approximate any continuous function on a compact domain arbitrarily well. For 1-Lipschitz neural networks, an analogous question is whether they can approximate all 1-Lipschitz functions accordingly. Unfortunately, the result in Section 3.2 already implies a negative answer. Indeed, by combining Proposition 3.2 and Theorem 3.5, $\hat{\boldsymbol{f}}$ is clearly a 1-Lipschitz function that cannot be approximated by any standard Lipschitz network.

To gain further insights into the structure of unrepresentable 1-Lipschitz functions, let us consider the *continuousization* of discrete Boolean functions. For the symmetric case, one needs to find a class of 1-Lipschitz continuous functions that are also invariant under permutations. It can be found that a simple class of candidates is the *order statistics*, i.e. the k-th largest element of a vector. One can check that the k-th order statistic $x_{(k)}$ is indeed 1-Lipschitz and is precisely the continuousization of the k-threshold Boolean function defined as $g^{\mathrm{B},k}(\boldsymbol{x}) := \mathbb{I}(\sum_i x_i \geq k)$. In particular, $x_{(1)} = \max_i x_i$ and $x_{(d)} = \min_i x_i$ corresponds to the logical OR/AND functions, respectively. Importantly, note that any Boolean function that is both symmetric and *monotonic* is a k-threshold function, and vice versa (Appendix A.2). Therefore, k-threshold functions can be regarded as the most elementary Boolean functions, suggesting that the ability to express order statistics is necessary for neural networks.

Unfortunately, using a similar analysis as the previous section, we have the following theorem:

Theorem 3.6. *Any standard Lipschitz network $f : \mathbb{R}^d \to \mathbb{R}$ cannot approximate the simple 1-Lipschitz function $\boldsymbol{x} \to x_{(k)}$ for arbitrary $k \in [d]$ on a bounded domain $\mathcal{K} = [0,1]^d$ if $d \geq 2$. Moreover, there exists a point $\widehat{\boldsymbol{x}} \in \mathcal{K}$, such that*

$$|f(\widehat{\boldsymbol{x}}) - \widehat{x}_{(k)}| \geq \frac{1}{2} - \frac{1}{2d}. \tag{4}$$

We give a proof in Appendix B.4. The above theorem indicates that order statistics cannot be *uniformly* approximated using any standard Lipschitz network regardless of the network size. Moreover, note that the trivial constant function $\tilde{f}(\boldsymbol{x}) = 1/2$ already achieves an approximation error of $1/2$ uniformly on $\mathcal{K}$, implying that standard Lipschitz networks can hardly improve upon trivial solutions.

Remark 3.7. Theorem 3.6 can be easily generalized to weaker forms of non-uniform approximation, e.g., using the ℓ_p-norm as distance metrics [45], by proving that there exists a hypercube $\mathcal{B}_\infty(\widehat{\boldsymbol{x}})$ centered at $\widehat{\boldsymbol{x}}$ with length $\Theta(1)$, such that $|f(\tilde{\boldsymbol{x}}) - \tilde{x}_{(k)}| \geq \Theta(1)$ holds for all $\tilde{x} \in \mathcal{B}_\infty(\widehat{\boldsymbol{x}})$ when $d \geq 2$. See Corollary B.13 for details.

Discussion with prior works [2, 24, 43]. The work of Anil et al. also gave negative results on the expressive power of standard Lipschitz networks[1] [2, Theorem 1]. They proved a different (weaker) version of Theorem 3.6, showing that if the activation function σ is *monotonic*, the network cannot *precisely* represent non-linear 1-Lipschitz functions whose gradient norm is 1 almost everywhere (e.g., the absolute value function proved before by [24]). They did not give a quantitative approximation error. The intuition is that any monotonic non-linear 1-Lipschitz activation (e.g., ReLU) must have regions with slopes less than 1, leading to *gradient attenuation* during backpropagation. The authors thus attributed the reason to the activation function, which is not *gradient-norm preserving* (GNP). However, such an explanation is still not fully satisfactory, as GNP can be simply achieved using a non-monotonic activation (e.g., $\sigma(x) = |x|$). Consequently, one may expect that a standard Lipschitz network built on a suitable (non-monotonic) activation function can have sufficient expressive power. Such an idea is recently explored in [43], where the authors proved that using a general 1-Lipschitz piecewise linear activation with 3 linear regions, the corresponding network achieves the maximum expressive power compared with other Lipschitz activations and can approximate any *one-dimensional* 1-Lipschitz function. They pose the high dimension setting as an open problem.

[1] The result is described w.r.t. ℓ_2-norm, but with some effort, it can be extended to the ℓ_∞-norm case.

Unfortunately, Theorem 3.6 addressed the open problem with negative answer, stating that such networks are not expressive even for the two-dimensional setting. It also implies that GNP is not *sufficient* to explain the failure of standard Lipschitz networks. Instead, we draw a rather different conclusion, arguing that the lack of expressiveness is due to the inability of *weight-constrained* affine transformations to perform basic Boolean operations (even in two dimensions). A further justification is given in Section 3.4. Finally, compared with Anil et al. [2], the form of Theorem 3.6 is more fundamental, in the sense that it does not make assumptions on the activation function, and it gives a quantitative error bound on the approximation that is arbitrarily close to the plain fit $f(\boldsymbol{x}) = 1/2$.

3.4 Investigating more advanced Lipschitz networks

Seeing the above impossibility results, we then examine two representative works of (non-standard) Lipschitz networks in literature: the GroupSort network [2] and the recently proposed ℓ_∞-distance net [73, 74]. Notably, both networks are Lipschitz-universal function approximators and thus fully expressive. The GroupSort network makes minimum changes to standard Lipschitz networks (3), by replacing element-wise activation σ with GroupSort layers. GroupSort partitions the input vector into groups, sorts the sub-vector of each group in descending order, and finally concatenates the resulting sub-vectors. Since sorting is computationally expensive, the authors considered a practical version of GroupSort with a group size of 2, called MaxMin, which simply calculates the maximum and minimum pair by pair [6]. ℓ_∞-distance net, on the other hand, is fundamentally different from standard Lipschitz networks. Each neuron in an ℓ_∞-distance net is designed based on the ℓ_∞-distance function $y = \|\boldsymbol{x} - \boldsymbol{w}\|_\infty + b$ (with parameters $\{\boldsymbol{w}, b\}$). Despite the somewhat unusual structure, ℓ_∞-distance net has been shown to substantially outperform GroupSort (MaxMin) in terms of certified ℓ_∞ robustness according to [73], a puzzling thing to be understood.

We provide a possible explanation for this. We find that both networks incorporate order statistics into the neuron design, either explicitly (GroupSort) or implicitly (ℓ_∞-distance net), thus bypassing the impossibility result in Theorem 3.6. Indeed, the sorting operations in GroupSort explicitly calculate order statistics. As for ℓ_∞-distance net, we show its basic neuron can implicitly represent the max function on a bounded domain, by assigning the weight $\boldsymbol{w} = -c\mathbf{1}$ and the bias $b = -c$ with a sufficiently large constant c:

$$y = \|\boldsymbol{x} - \boldsymbol{w}\|_\infty + b = \max_i |x_i - (-c)| - c = \max_i x_i \quad \text{if } c \geq \max_i -x_i, \tag{5}$$

and thus can represent the logical OR operation. In general, we have the following theorem:

Theorem 3.8. *A two-layer ℓ_∞-distance net can exactly represent the following functions:* (i) *any discrete Boolean function;* (ii) *any continuous order-statistic function on a compact domain.*

We give a proof in Appendix B.5. Our proof uses the fundamental result in Boolean algebra that any Boolean function can be written in its disjunctive normal form (DNF, see Appendix A.1), which can be further reduced to using only the composition of logical OR operations of *literals* and thus be realized by a two-layer ℓ_∞-distance net. To represent order statistics, we formulate them as nested max-min functions, which can also be realized by a two-layer ℓ_∞-distance net. Therefore, the construction in our proof provides a novel understanding of the mechanism behind the success of ℓ_∞-distance nets, since *each ℓ_∞-distance neuron can be regarded as a basic "logical gate" and the whole network can simulate any Boolean circuit*.

For GroupSort networks with a group size $G \geq d$, a similar result holds. However, it is not the case for practically used MaxMin networks ($G = 2$), where we have the following impossibility results:

Theorem 3.9. *An M-layer MaxMin network $f : \mathbb{R}^d \to \mathbb{R}$ cannot approximate any k-th order statistic function on a bounded domain $\mathcal{K} = [0, 1]^d$ if $M \leq \lceil \log_2 d \rceil$ (no matter how wide the network is). Moreover, there exists a point $\widehat{\boldsymbol{x}} \in \mathcal{K}$, such that*

$$|f(\widehat{\boldsymbol{x}}) - \widehat{x}_{(k)}| \geq \frac{1}{2} - \frac{2^{M-2}}{d} \geq \frac{1}{4} \quad \textit{if } M \leq \lfloor \log_2 d \rfloor. \tag{6}$$

Theorem 3.10. *Let M_d be the minimum depth such that an M_d-layer MaxMin network can represent any (discrete) d-dimensional Boolean function. Then $M_d = \Omega(d)$.*

The above theorems show that MaxMin networks must be *very deep* in order to represent Boolean functions and order statistics, which is in stark contrast to Theorem 3.8, as a constant depth is sufficient for ℓ_∞-distance nets. Based on Theorem 3.10, we directly have the following corollary:

Corollary 3.11. *The function class induced by M_d-layer MaxMin networks is not a universal approximator to the d-dimensional 1-Lipschitz functions if $M_d = o(d)$.*

The proofs of Theorems 3.9 and 3.10 are deferred to Appendix B.6 and B.7, respectively. In particular, the proof of Theorem 3.10 is non-trivial and makes elegant use of Boolean circuit theory, so we present a proof sketch here. The key insight is that for any Boolean function, if it can be expressed by some MaxMin network f, then it can be expressed by a special MaxMin network with the same topology as f such that all the weight vectors $\boldsymbol{w}$ are sparse with at most one non-zero element, either 1 or -1 (Corollary B.23). This implies that *weight vectors have no use* in representing Boolean functions and thus MaxMin networks reduce to *2-ary Boolean circuits*, i.e. directed acyclic graphs whose internal nodes are logical gates including NOT and the 2-ary AND/OR. Note that for a 2-ary Boolean circuit that has M layers and outputs a scalar, the number of nodes will not exceed $2^{M+1}-1$ (achieved by a complete binary tree). However, the classic result in Boolean circuit theory (Shannon 1942) showed that for most Boolean functions of d variables, a lower bound on the minimum size of 2-ary Boolean circuits is $\Omega(2^d/d)$, which thus yields $M = \Omega(d)$ and concludes the proof.

In Appendix, we also discuss the tightness of the above theorems. We prove that a depth of $\mathcal{O}(\log_2 d)$ is sufficient to represent any order statistic function using Boolean circuit theory (Theorem B.19), and a straightforward construction using DNF shows that a depth of $\mathcal{O}(d)$ is sufficient to represent any Boolean functions (Proposition B.20). Thus both theorems are tight.

Unfortunately, training deep MaxMin networks is known to be challenging due to optimization difficulties [10]. Consequently, prior works only use a shallow MaxMin network with no more than 4 layers [2, 10], which severely lacks expressive power. One possible solution is to increase the group size, and several works explored this aspect using toy examples and observed significant benefits empirically [10, 58]. However, a large group size involves computationally expensive sorting operations and makes the network hard to train [2], limiting its value in practice.

4 A Unified Framework of Lipschitz Neural Networks

The above theoretical results have justified order statistics as a crucial component in representing a class of Boolean functions, shedding light on how GroupSort and ℓ_∞-distance net work. Based on these insights, in this section, we will propose a unified framework of Lipschitz networks that take the respective advantage of prior Lipschitz architectures, and then give a practical (specialized) version that enables efficient training.

Consider a general Lipschitz network constructed using the following three basic types of 1-Lipschitz operations: (i) norm-bounded affine transformations, e.g. $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ ($\|\boldsymbol{w}\|_1 \leq 1$) and $y = x + b$; (ii) 1-Lipschitz unary activation functions σ; (iii) order statistics. The first two types are extensively used in standard Lipschitz networks, while the last type is motivated by Section 3 and is of crucial importance. We propose the following network which naturally combines the above components:

Definition 4.1. (SortNet) Define an M-layer fully-connected SortNet $\boldsymbol{f}$ as follows. The network takes $\boldsymbol{x} = \boldsymbol{x}^{(0)}$ as input, and the k-th unit in the l-th hidden layer $x_k^{(l)}$ is computed by

$$x_k^{(l)} = (\boldsymbol{w}^{(l,k)})^{\mathrm{T}} \operatorname{sort}(\sigma(\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l,k)})), \quad \text{s.t. } \|\boldsymbol{w}^{(l,k)}\|_1 \leq 1, \quad l \in [M], k \in [d_l] \tag{7}$$

where d_l is the size of the l-th layer, and $\operatorname{sort}(\boldsymbol{x}) := (x_{(1)}, \cdots, x_{(d)})^{\mathrm{T}}$ calculates all order statistics of $\boldsymbol{x} \in \mathbb{R}^d$. The network outputs $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^{(M)} + \boldsymbol{b}^{\text{out}}$. Here $\{\boldsymbol{w}^{(l,k)}\}$, $\{\boldsymbol{b}^{(l,k)}\}$ and $\boldsymbol{b}^{\text{out}}$ are parameters.

It is easy to see that SortNet is 1-Lipschitz w.r.t. ℓ_∞-norm. We now show that SortNet is a general architecture that extends both GroupSort and ℓ_∞-distance networks into a unified framework.

Proposition 4.2. *Any GroupSort network with an arbitrary group size on a compact input domain can be represented by a SortNet with the same topological structure using activation $\sigma(x) = x$.*

Proposition 4.3. *Any ℓ_∞-distance net can be represented by a SortNet with the same topological structure by fixing the weights $\boldsymbol{w}^{(l,k)} = \boldsymbol{e}_1$ and using the absolute-value activation $\sigma(x) = |x|$.*

See Appendix C for a proof. As a result, SortNet can exploit the respective advantage of these Lipschitz networks. Compared with GroupSort, SortNet can freely use activation functions such as the absolute value, thus easily addressing the problem claimed in [24, 2]. Moreover, unlike GroupSort, the bias vector $\boldsymbol{b}^{(l,k)}$ in SortNet (7) can be assigned diversely for different neurons in the same layer. In this way, one can control the sorting behavior of each neuron individually by varying the bias value without disturbing the output of other neurons, which is very beneficial (see Appendix C.1 for details). Compared with ℓ_∞-distance net, SortNet adds linear transformation and incorporates all order statistics (rather than only the maximum), thus can represent certain functions more effectively.

A practical version of SortNet. As with GroupSort networks, we also design a practical (specialized) version of SortNet which enjoys efficient training. But different from the MaxMin network that reduces the group size, we keep the full-dimensional order statistics as they are crucial for the expressive power (Section 3.4). The key observation is that in (7), the only required computation is the linear combination of order statistics (i.e. $\boldsymbol{w}^{\mathrm{T}} \operatorname{sort}(\cdot)$), rather than the entire sorting results (i.e. $\operatorname{sort}(\cdot)$). We find that for certain carefully designed choices of the weight vector $\boldsymbol{w}$, there exist efficient approximation algorithms that can give a good estimation of $\boldsymbol{w}^{\mathrm{T}} \operatorname{sort}(\cdot)$. In particular, we propose an assignment of the weight vector that follows geometric series, i.e. w_i proportional to ρ^i, in which case we have the following result:

Proposition 4.4. *Let $\boldsymbol{w} \in \mathbb{R}^d$ be a vector satisfying $w_i = (1-\rho)\rho^{i-1}, i \in [d]$ for some $0 \leq \rho < 1$. Then for any vector $\boldsymbol{x} \in \mathbb{R}_+^d$ with non-negative elements,*

$$\boldsymbol{w}^{\mathrm{T}} \operatorname{sort}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{s} \sim \mathrm{Ber}(1\text{-}\rho)}[\max_i s_i x_i]. \tag{8}$$

Here $\boldsymbol{s}$ is a random vector following independent Bernoulli distribution with probability $1-\rho$ being 1 and ρ being 0.

Proof. Without loss of generality, assume $x_1, \cdots, x_d$ are different from each other. Denote $j_1, \cdots, j_d$ as the sorting indices such that $\operatorname{sort}(\boldsymbol{x}) = (x_{j_1}, \cdots, x_{j_d})$. Then

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{s} \sim \mathrm{Ber}(\rho)}[\max_i s_i x_i] &= \textstyle\sum_{k \in [d]} \Pr_{\boldsymbol{s} \sim \mathrm{Ber}(\rho)} [\max_i s_i x_i = x_{j_k}]\, x_{j_k} \\
&= \textstyle\sum_{k \in [d]} \Pr_{\boldsymbol{s} \sim \mathrm{Ber}(\rho)} [s_{j_k} = 1 \text{ and } s_{j_i} = 0\ \forall 1 \leq i < k]\, x_{j_k} \\
&= \textstyle\sum_{k \in [d]} (1-\rho)\rho^{k-1} \cdot x_{(k)} = \boldsymbol{w}^{\mathrm{T}} \operatorname{sort}(\boldsymbol{x}).
\end{aligned}$$

□

It is easy to check that the weight $\boldsymbol{w}$ in the above proposition satisfies $\|\boldsymbol{w}\|_1 \leq 1$, which guarantees the Lipschitzness. The non-negative condition on $\boldsymbol{x}$ in Proposition 4.4 holds when using a suitable activation function in neuron (7), such as the absolute value function.

Proposition 4.4 suggests that one can use $\max_i s_i x_i$ to give an *unbiased* estimation of $\boldsymbol{w}^{\mathrm{T}} \operatorname{sort}(\boldsymbol{x})$. In this way, the expensive sorting operation is avoided and replaced by a max operation, thus significantly reducing the computational cost in training. We give an efficiently GPU implementation for training SortNet in Appendix D. Note that $\boldsymbol{s}$ is a random Bernoulli vector, so the above calculation is similar to applying a mask on the input of each neuron, like dropout [56]. It means that the introduced stochasticity may further prevent overfitting and benefit generalization performance.

Regarding the value of ρ. When $\rho = 0$, only the maximum value is taken into the computation and the resulting network can recover the ℓ_∞-distance net by choosing the activation function $\sigma(x) = |x|$. This means the specialized SortNet still extends ℓ_∞-distance net and thus has sufficient expressive power. When $\rho > 0$, all order statistics become utilized. A simple way of selecting ρ is to regard it as a hyper-parameter and set its value by cross-validation, which is adopted in our experiments. One can also consider treating ρ as learnable parameters for each neuron that participate in the optimization process, but this involves calculating the gradient of ρ which may be complicated due to the stochastic sampling procedure (8). We will leave the study as future work.

5 Experiments

In this section, we perform extensive empirical evaluations of the proposed SortNet architecture as well as various prior works in the certified ℓ_∞ robustness area. To show the scalability of different approaches, we consider a variety of benchmark datasets, including MNIST [33], CIFAR-10 [29], TinyImageNet [32], and ImageNet (64×64) [7]. Due to space limitations, a complete training recipe is given in Appendix E. Our code and trained models are released at `https://github.com/zbh2047/SortNet`.

5.1 Experimental setting

SortNet model configuration. Since SortNet generalizes the ℓ_∞-distance net, we simply follow the same model configurations as [73] and consider two types of models. The first one is a simple SortNet consisting of M fully-connected layers with a hidden size of 5120, which is used in MNIST and CIFAR-10. Like [73], we choose $M = 5$ for MNIST and $M = 6$ for CIFAR-10. Since SortNet is Lipschitz, we directly apply the margin-based certification method to calculate the certified accuracy

(Proposition 3.1). To achieve the best results on ImageNet-like datasets, in our second type of model we consider using a composite architecture consisting of a base SortNet backbone and a prediction head (denoted as SortNet+MLP). Following [73], the SortNet backbone has 5 layers with a width of 5120 neurons, which serves as a robust feature extractor. The top prediction head is a lightweight 2-layer perceptron with 512 hidden neurons (or 2048 for ImageNet), which takes the robust features as input to give classification results. We also try a larger SortNet backbone, denoted as SortNet+MLP (2x), that has roughly four times the training cost (see Appendix E.2 for architectural details). We use the same approach as [73] to train and certify these models, i.e. by combining margin-based certification for the SortNet backbone and interval bound propagation for the top MLP [21].

Baseline methods and metrics. We compare SortNet with representative literature approaches including relaxation-based certification (for standard networks), margin-based certification (using Lipschitz networks), and mixed-integer linear programming (MILP) [59]. In Appendix G, we also discuss randomized smoothing approaches [9, 49], which provide probabilistic guarantees rather than deterministic ones. For each method in these tables, we report five metrics: (i) training efficiency, measured by the wall-clock time per training epoch; (ii) certification efficiency, measured by the time needed to calculate the certified accuracy on the test dataset; (iii) the clean test accuracy without perturbation (denoted as Clean); (iv) the robust test accuracy under 100-step PGD attack (denoted as PGD); (v) the certified robust test accuracy (denoted as Certified). For a fair comparison, we reproduce most of baseline methods using the official codes and report the wall-clock time under the same NVIDIA-RTX 3090 GPU. These results are presented in Tables 1, 2 and 3. In Appendix H, we also show the training variance of each setting by running 8 sets of experiments independently, and full results (including the median performance) are reported in Table 9 and 10.

5.2 Experimental results

Performance on MNIST. The results are presented in Table 1. Following the common practice, we consider both a small perturbation radius $\epsilon = 0.1$ and a larger one $\epsilon = 0.3$. It can be seen that the SortNet models can achieve **98.14%** ($\epsilon = 0.1$) and **93.40%** ($\epsilon = 0.3$) certified accuracy, respectively, both of which outperform all previous baseline methods. In contrast, the GroupSort network can only achieve a trivial certified accuracy for $\epsilon = 0.3$. This matches our theory in Section 3.4, indicating that the expressive power of shallow MaxMin networks is insufficient in real-world applications.

Performance on CIFAR-10. The results are presented in Table 2. Following the common practice, we consider two perturbation radii: $\epsilon = 2/255$ and $\epsilon = 8/255$. Our models can achieve **56.94%** ($\epsilon = 2/255$) and **40.39%** ($\epsilon = 8/255$) certified accuracy, respectively. Moreover, the training approach proposed in Section 4 is very efficient, e.g., with a training time of **13**$\sim$**14** seconds per epoch. For both radii, our models perform the best among all existing approaches that can be certified in a reasonable time. Compared with relaxation-based methods, the certified accuracy of SortNet models is much higher (typically $+3 \sim +6$ point for both radii), despite our training speed being several times faster. Such results may indicate that certified ℓ_∞ robustness can be better achieved by designing suitable Lipschitz models than by devising relaxation procedures for non-Lipschitz models.

Performance on TinyImageNet and ImageNet. To demonstrate the scalability of SortNet models, we finally run experiments on two large-scale datasets: Tiny-ImageNet and ImageNet (64×64). Notably, the ImageNet dataset has 1000 classes and contains 1.28 million images for training and 50,000 images for testing. Due to both the large size and the huge number of classes, achieving certified ℓ_∞ robustness on the ImageNet level has long been a challenging task.

Table 3 presents our results along with existing baselines. Among them, we achieve **18.18%** certified accuracy on TinyImageNet and achieve **9.54%** certified accuracy on ImageNet, both of which establish state-of-the-art results. The gap is most prominent on ImageNet, where our small SortNet+MLP model already outperforms the largest model of [69] while being **22** times faster to train. Even for the largest model (SortNet+MLP 2x), the training is still 7 times faster, resulting in a training overhead of 4 days using two GPUs. We suspect that continuing to increase the model size will yield better results, given the noticeable improvement of the larger model over the smaller one.

Comparing with ℓ_∞-distance net. As can be seen, SortNet models consistently achieve better certified accuracy than ℓ_∞-distance nets for all different datasets and perturbation levels, and the performance gap is quite prominent compared with the original work [73]. Very recently, a follow-up paper [74] significantly improved the performance of ℓ_∞-distance net using a carefully designed

Table 1: Comparison of our results with existing methods on MNIST dataset.

Method		Train Time (s)	MNIST ($\epsilon = 0.1$)			MNIST ($\epsilon = 0.3$)		
			Clean	PGD	Certified	Clean	PGD	Certified
Relaxation	IBP [21]	17.5	98.92	97.98	97.25	97.88	93.22	91.79
	IBP [52]	34.7	98.84	–	97.95	97.67	–	93.10
	CROWN-IBP [75]	60.3	98.83	98.19	97.76	98.18	93.95	92.98
Lipschitz	GroupSort (MaxMin) [2]	–	97.0	84.0	79.0	97.0	34.0	2.0
	ℓ_∞-dist Net [73]	17.2	98.66	97.85	97.73	98.54	94.62	92.64
	ℓ_∞-dist Net+MLP [73]	17.2	98.86	97.77	97.60	98.56	95.05	93.09
	ℓ_∞-dist Net [74]	17.0	98.93	98.03	97.95	98.56	94.73	93.20
	SortNet	**10.6**	99.01	98.21	**98.14**	98.46	94.64	**93.40**
MILP	COLT [3]	–	99.2	–	97.1	97.3	–	85.7

Table 2: Comparison of our results with existing methods on CIFAR-10 dataset.

Method		Time (s)		$\epsilon = 2/255$			$\epsilon = 8/255$		
		Train	Certify	Clean	PGD	Certified	Clean	PGD	Certified
Relaxation	CAP [66]	659.0	7,570	68.28	–	53.89	28.67	–	21.78
	IBP [21]	19.0	2.74	61.46	50.28	44.79	50.99	31.27	29.19
	IBP [52]	70.4	4.02	66.84	–	52.85	48.94	–	34.97
	CROWN-IBP [75]	87.2	7.01	71.52	59.72	53.97	45.98	34.58	33.06
	CROWN-IBP [69]	45.0	4.02	–	–	–	46.29	35.69	33.38
Lipschitz	ℓ_∞-dist Net [73]	19.7	1.73	60.33	51.55	50.94	56.80	36.19	33.30
	ℓ_∞-dist Net+MLP [73]	19.7	1.74	65.62	51.47	51.05	50.80	36.51	35.42
	ℓ_∞-dist Net [74]	18.9	1.73	60.61	54.28	54.12	54.30	41.84	40.06
	SortNet	14.0	8.00	65.96	57.03	56.67	54.84	41.50	**40.39**
	SortNet+MLP	**13.4**	8.01	67.72	57.83	**56.94**	54.13	41.58	39.99
MILP	COLT [3]	252.0	$\sim 10^5$	78.4	–	60.5	51.7	–	27.5

Table 3: Comparison of our results with existing methods on TinyImageNet and ImageNet datasets.

Method	TinyImageNet ($\epsilon = 1/255$)				ImageNet 64×64 ($\epsilon = 1/255$)			
	Time (s)	Clean	PGD	Certified	Time (s)	Clean	PGD	Certified
IBP [21]	735	26.46	20.60	14.85	11,026	15.96	9.12	6.13
IBP [52]	284	25.71	–	17.64	–	–	–	–
CROWN-IBP [69]	1,256	27.82	20.52	15.86	16,269	16.23	10.26	8.73
ℓ_∞-dist Net+MLP [73]	55	21.82	–	16.31	–	–	–	–
ℓ_∞-dist Net [74]	55	12.57	11.09	11.04	–	–	–	–
SortNet+MLP	**39**	24.17	20.57	17.92	**715**	13.48	10.93	9.02
SortNet+MLP (2x larger)	156	25.69	21.57	**18.18**	2,192	14.79	11.93	**9.54**

training strategy, creating a strong baseline on CIFAR-10. However, we find their approach does not suit the ImageNet-like datasets when the number of classes is large (see Appendix E.6). In contrast, SortNet models enjoy great scalability ranging from MNIST to ImageNet and consistently outperform [74]. The improvement is also remarkable for $\epsilon = 2/255$ on CIFAR-10 ($+7.11\%$ and $+2.82\%$ in clean / certified accuracy).

In Appendix F, we conduct ablation studies on CIFAR-10 by varying the value of ρ and comparing SortNet models ($\rho > 0$) with ℓ_∞-distance net ($\rho = 0$), under *the same training strategy* in this paper without additional tricks. We observe a large gain in certified accuracy when switching from ℓ_∞-distance net to general SortNet. This empirically indicates that incorporating other order statistics has extra benefits in certified ℓ_∞ robustness than using only the maximum (the first order statistic).

6 Conclusion

In this paper, we study certified ℓ_∞ robustness from the novel perspective of representing Boolean functions. Our analysis points out an inherent problem in the expressive power of standard Lipschitz networks, and provides novel insights on how recently proposed Lipschitz networks resolve the problem. We also answer several previous open problems, such as (i) the expressive power of standard Lipschitz networks with general activations [43] and (ii) the lower bound on the depth of MaxMin networks to become universal approximators [58, 43]. Finally, guided by the theoretical results, we design a new Lipschitz network with better empirical performance than prior works.

Limitations, Open Problems, and Broader Impact

Regarding the ℓ_p-norm. One major limitation of this work is that we only focus on ℓ_∞ robustness. While such results may shed light on general ℓ_p-norm settings when p is large, it does not apply to the standard ℓ_2-norm. In particular, in this case MaxMin is *equivalent* to the absolute value activation function in terms of expressive power [2], which contrasts to the ℓ_∞-norm case for which MaxMin networks are strcitly more expressive. Moreover, empirical results suggest that these ℓ_2 Lipschitz networks may have sufficient expressive power [55] (although it remains a fantastic open problem to prove that MaxMin networks with bounded matrix 2-norm are universal approximators).

Based on the above finding, this work reflects an interesting "phase transition" in the expressive power of standard Lipschitz networks when p is switched from 2 to a large number. Coincidentally, a similar limitation is also proved when using randomized smoothing, which suffers from the curse of dimensionality when $p > 2$ [70]. This raises an interesting question of why the effect of p is very similar for both methods and how things change as p increases.

Beyond standard Lipschitz networks. Another limitation is that our results apply only for standard Lipschitz networks. When the Lipschitz constant is constrained using carefully designed bounding methods [47, 18, 31, 51] (rather than a simple stacking of 1-Lipschitz layers), the robustness certification will be less efficient, but the resulting networks are likely to bypass the impossibility results in this paper. It is an interesting direction to study whether we can just use *standard networks* with a carefully-designed Lipschitz bounding method that can achieve good certified robustness while still keeping adequate efficiency.

Other promising directions. On the application side, it is interesting to study how to design efficient training approaches for the general SortNet models with learnable weights or learnable ρ. Another meaningful question is how to encode inductive biases into these Lipschitz networks (e.g., designing convolutional architectures) to better suit image tasks.

Broader impact. Interestingly, our theoretical results point out a surprising connection between MaxMin/ℓ_∞-distance networks and Boolean circuits. We believe the value of this paper may go beyond the certified robustness community and link to the field of theoretical computer science.

Acknowledgement

This work is supported by National Science Foundation of China (NSFC62276005), The Major Key Project of PCL (PCL2021A12), Exploratory Research Project of Zhejiang Lab (No. 2022RC0AN02), and Project 2020BD006 supported by PKUBaidu Fund. Bohang Zhang would like to thank Ruichen Li and Yuxin Dong for helpful discussions. We also thank all the anonymous reviewers for the very careful and detailed reviews as well as the valuable suggestions. Their help has further enhanced our work.

References

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. An 0 (n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 1–9, 1983.

[2] Cem Anil, James Lucas, and Roger Grosse. Sorting out Lipschitz function approximation. In *International Conference on Machine Learning*, pages 291–301, 2019.

[3] Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations*, 2020.

[4] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[5] Avrim Blum, Travis Dick, Naren Manoj, and Hongyang Zhang. Random smoothing might be unable to certify ℓ_∞ robustness for high-dimensional images. *arXiv preprint arXiv:2002.03517*, 2020.

[6] Artem Chernodub and Dimitri Nowicki. Norm-preserving orthogonal permutation linear unit activation functions (oplu). *arXiv preprint arXiv:1604.02313*, 2016.

[7] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[8] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863, 2017.

[9] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pages 1310–1320. PMLR, 2019.

[10] Jeremy EJ Cohen, Todd Huster, and Ra Cohen. Universal Lipschitz approximation in bounded depth neural networks. *arXiv preprint arXiv:1904.04861*, 2019.

[11] Francesco Croce, Maksym Andriushchenko, and Matthias Hein. Provable robustness of ReLU networks via maximization of linear regions. In *the 22nd International Conference on Artificial Intelligence and Statistics*, pages 2057–2066. PMLR, 2019.

[12] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, 2019.

[14] Gavin Weiguang Ding, Yash Sharma, Kry Yik Chau Lui, and Ruitong Huang. MMA training: Direct input space margin maximization through adversarial training. In *International Conference on Learning Representations*, 2020.

[15] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *arXiv preprint arXiv:1803.06567*, 2018.

[16] Krishnamurthy Dj Dvijotham, Robert Stanforth, Sven Gowal, Chongli Qin, Soham De, and Pushmeet Kohli. Efficient neural network verification with exactness characterization. In *Uncertainty in Artificial Intelligence*, pages 497–507. PMLR, 2020.

[17] Farzan Farnia, Jesse Zhang, and David Tse. Generalizable adversarial training via spectral normalization. In *International Conference on Learning Representations*, 2019.

[18] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of Lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[19] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[20] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing Lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

[21] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Yujia Huang, Huan Zhang, Yuanyuan Shi, J Zico Kolter, and Anima Anandkumar. Training certifiably robust neural networks with efficient local Lipschitz bounds. *Advances in Neural Information Processing Systems*, 34, 2021.

[24] Todd Huster, Cho-Yu Jason Chiang, and Ritu Chadha. Limitations of the Lipschitz constant as a defense against adversarial examples. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 16–29. Springer, 2018.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[26] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[27] Jongheon Jeong and Jinwoo Shin. Consistency regularization for certified robustness of smoothed classifiers. *Advances in Neural Information Processing Systems*, 33, 2020.

[28] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[29] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. *Technical Report TR-2009*, 2009.

[30] Aounon Kumar, Alexander Levine, Tom Goldstein, and Soheil Feizi. Curse of dimensionality on randomized smoothing for certifiable robustness. In *International Conference on Machine Learning*, pages 5458–5467. PMLR, 2020.

[31] Fabian Latorre, Paul Rolland, and Volkan Cevher. Lipschitz constant estimation of neural networks via sparse polynomial optimization. In *International Conference on Learning Representations*, 2020.

[32] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[33] Yann LeCun, Corinna Cortes, and Chris Burges. MNIST handwritten digit database, 1998.

[34] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[35] Sungyoon Lee, Jaewook Lee, and Saerom Park. Lipschitz-certifiable training with a tight outer bound. *Advances in Neural Information Processing Systems*, 33, 2020.

[36] Klas Leino, Zifan Wang, and Matt Fredrikson. Globally-robust neural networks. In *International Conference on Machine Learning (ICML)*, 2021.

[37] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[38] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive noise. *Advances in Neural Information Processing Systems*, 32:9464–9474, 2019.

[39] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Joern-Henrik Jacobsen. Preventing gradient attenuation in Lipschitz constrained convolutional networks. *Advances in Neural Information Processing Systems*, 32:15390–15402, 2019.

[40] Laurent Meunier, Blaise Delattre, Alexandre Araujo, and Alexandre Allauzen. A dynamical system perspective for Lipschitz neural networks. *arxiv:2110.12690*, 2022.

[41] Matthew Mirman, Maximilian Baader, and Martin Vechev. The fundamental limits of interval arithmetic for neural networks. *arXiv preprint arXiv:2112.05235*, 2021.

[42] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3578–3586. PMLR, 2018.

[43] Sebastian Neumayer, Alexis Goujon, Pakshal Bohra, and Michael Unser. Approximation of Lipschitz functions using deep spline neural networks. *arXiv preprint arXiv:2204.06233*, 2022.

[44] Ryan O'Donnell and Karl Wimmer. Approximation by DNF: examples and counterexamples. In *International Colloquium on Automata, Languages, and Programming*, pages 195–206. Springer, 2007.

[45] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica*, 8:143–195, 1999.

[46] Haifeng Qian and Mark N. Wegman. L2-nonexpansive neural networks. In *International Conference on Learning Representations*, 2019.

[47] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[48] Aditi Raghunathan, Jacob Steinhardt, and Percy S. Liang. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems*, pages 10900–10910, 2018.

[49] Hadi Salman, Greg Yang, Jerry Li, Pengchuan Zhang, Huan Zhang, Ilya Razenshteyn, and Sébastien Bubeck. Provably robust deep learning via adversarially trained smoothed classifiers. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 11292–11303, 2019.

[50] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. *Advances in Neural Information Processing Systems*, 32:9835–9846, 2019.

[51] Zhouxing Shi, Yihan Wang, Huan Zhang, Zico Kolter, and Cho-Jui Hsieh. Efficiently computing local Lipschitz constants of neural networks via bound propagation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[52] Zhouxing Shi, Yihan Wang, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Fast certified robust training with short warmup. *Advances in Neural Information Processing Systems*, 34, 2021.

[53] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10802–10813, 2018.

[54] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on Machine Learning*, volume 139, pages 9756–9766. PMLR, 2021.

[55] Sahil Singla, Surbhi Singla, and Soheil Feizi. Improved deterministic l2 robustness on CIFAR-10 and CIFAR-100. In *International Conference on Learning Representations*, 2022.

[56] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[57] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[58] Ugo Tanielian and Gerard Biau. Approximating Lipschitz continuous functions with groupsort neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 442–450. PMLR, 2021.

[59] Vincent Tjeng, Kai Y Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*, 2019.

[60] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley transform. In *International Conference on Learning Representations*, 2021.

[61] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Advances in Neural Information Processing Systems*, 31:6541–6550, 2018.

[62] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[63] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in Neural Information Processing Systems*, 34, 2021.

[64] Lily Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning*, pages 5276–5285. PMLR, 2018.

[65] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[66] Eric Wong, Frank R Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8410–8419, 2018.

[67] Yihan Wu, Aleksandar Bojchevski, Aleksei Kuvshinov, and Stephan Günnemann. Completing the picture: Randomized smoothing suffers from the curse of dimensionality for a large family of distributions. In *International Conference on Artificial Intelligence and Statistics*, pages 3763–3771. PMLR, 2021.

[68] Kai Y Xiao, Vincent Tjeng, Nur Muhammad Mahi Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing ReLU stability. In *International Conference on Learning Representations*, 2019.

[69] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33, 2020.

[70] Greg Yang, Tony Duan, J Edward Hu, Hadi Salman, Ilya Razenshteyn, and Jerry Li. Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning*, pages 10693–10705. PMLR, 2020.

[71] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[72] Runtian Zhai, Chen Dan, Di He, Huan Zhang, Boqing Gong, Pradeep Ravikumar, Cho-Jui Hsieh, and Liwei Wang. MACER: Attack-free and scalable robust training via maximizing certified radius. In *International Conference on Learning Representations*, 2020.

[73] Bohang Zhang, Tianle Cai, Zhou Lu, Di He, and Liwei Wang. Towards certifying L-infinity robustness using neural networks with L-inf-dist neurons. In *International Conference on Machine Learning*, pages 12368–12379. PMLR, 2021.

[74] Bohang Zhang, Du Jiang, Di He, and Liwei Wang. Boosting the certified robustness of L-infinity distance nets. In *International Conference on Learning Representations*, 2022.

[75] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. In *International Conference on Learning Representations*, 2020.

[76] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, volume 31, pages 4939–4948, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We have discussed the limitations of this work in detail.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Our work is only for academic research purpose, without any foreseeable negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Complete proofs are given either in the main text or in Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We give complete implementation details in Appendix E. Our code and models will be released once the paper is published. We use public benchmark datasets which can be downloaded freely.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix E.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Appendix F.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix E.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We use public benchmark datasets which can be downloaded freely.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We use public benchmark datasets which can be downloaded freely.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Boolean Functions

In this section, we review some basic concepts in Boolean algebra, which will be frequently used in our subsequent analysis. We only present the content that is most relevant to this paper, in particular, the concept of *disjunctive normal form* and *symmetric Boolean functions*. Throughout this paper, we use $g^{\mathrm{B}} : \{0,1\}^d \to \{0,1\}$ to denote a Boolean function.

A.1 Disjunctive normal form

Three most elementary Boolean functions in Boolean algebra are the logical AND, logical OR and logical NOT, denoted as $\wedge$, $\vee$, and $\neg$, respectively. The value of $\wedge_{i=1}^d x_i = x_1 \wedge \cdots \wedge x_d$ is 1 if and only if $x_i = 1\ \forall i \in [d]$. Similarly, $\vee_{i=1}^d x_i = x_1 \vee \cdots \vee x_d$ is 0 if and only if $x_i = 0\ \forall i \in [d]$. The logical NOT is a unary operation which outputs 1 if and only if the input is 0.

All Boolean functions can be expressed using the above three elementary operations as building blocks. Among all possible forms of expressions, the disjunctive normal form (DNF) is a canonical form used in Boolean algebra theory. To define a DNF, we will need the notion of conjunction/disjunction.

Definition A.1. Let $\boldsymbol{x} = (x_1, \cdots, x_d) \in \{0,1\}^d$ be a Boolean vector.

- A *literal* is an atomic formula consisting of a single element or its negation, i.e. x_i or $\neg x_i$.
- A *literal conjunction* is a formula consisting of a set of literals linked by the logical AND operation, i.e. $x_{i_1} \wedge \cdots \wedge x_{i_r} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_s}$, $1 \le i_1 \le \cdots \le i_r \le d, 1 \le j_1 \le \cdots \le j_s \le d$, where $r, s \in \mathbb{N}$. Typically, the definition further requires that $i_1, \cdots, i_r, j_1, \cdots, j_s$ are different indices, otherwise the literal conjunction is *unsatisfiable* (i.e. always outputs 0). Similarly, a *literal disjunction* is a formula consisting of a set of literals linked by the logical OR operation.
- A logical formula is considered to be in its disjunctive normal form if it is a disjunction of one or more conjunctions of one or more literals.

We list some examples of DNF: x_1, $x_1 \wedge x_2$, $x_1 \vee x_2$, $(x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge \neg x_2) \vee \neg x_4$, etc. A fundamental result in Boolean algebra is shown in the following:

Fact A.2. *Any satisfiable Boolean function can be written in its disjunctive normal form.*

A.2 Symmetric Boolean functions

Symmetric Boolean functions are an important class of Boolean functions defined in Definition 3.3. In this subsection, we will delve into this concept and give more concrete examples. First note that due to the symmetry property, the output of such functions can only depend on the number of ones (or zeros) in the Boolean input. For this reason, they are also known as *Boolean counting functions*.

As simple examples, the logical AND and logical OR are symmetric Boolean functions. Similarly, the NAND/NOR operations (i.e. $\neg(\wedge_i x_i)$ and $\neg(\vee_i x_i)$) are also symmetric. Another important example is the XOR function, denoted as $\oplus$. XOR outputs 1 iff the number of ones in the input is odd, i.e.

$$x_1 \oplus \cdots \oplus x_d = 1 \quad \text{iff } (\textstyle\sum_i x_i) \bmod 2 = 1.$$

Threshold functions. If a symmetric Boolean function g^{B} further satisfies a property called monotonicity, then we call it a threshold function. Formally, a function f is *monotonically increasing* if for all vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ such that $x_i \le x_i'$ ($\forall i \in [d]$), one has $f(\boldsymbol{x}) \le f(\boldsymbol{y})$. It is easy to see that symmetric monotonic Boolean function must have the form: $g^{\mathrm{B}}(\boldsymbol{x}) = \mathbb{I}(\sum_i x_i \ge k)$ with integer k, which gives the name "threshold function". Depending on k, there are a total of $d+2$ different d-dimensional threshold functions (including two constant functions). In particular, for $k = 1$ and $k = d$, we can recover the logical OR/AND functions, respectively. When $k = \lceil d/2 \rceil$, the resulting function is called the *majority function*.

B Proof of Theorems in Section 3

This section provides all the missing proofs in Section 3. For the convenience of reading, we will restate each theorem before giving a proof.

B.1 Proof of Proposition 3.1

Proposition B.1. (Certified Robustness of Lipschitz networks) *For a neural network $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^K$ with Lipschitz constant L under ℓ_p-norm $\|\cdot\|_p$, define the resulting classifier g as $g(\boldsymbol{x}) := \arg\max_{k\in[K]} f_k(\boldsymbol{x})$ for an input $\boldsymbol{x}$. Then g is provably robust under perturbations $\|\boldsymbol{\delta}\|_p < \frac{c}{L}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$, i.e.*

$$g(\boldsymbol{x}+\boldsymbol{\delta}) = g(\boldsymbol{x}) \quad \textit{for all } \|\boldsymbol{\delta}\|_p < \frac{c}{L}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x})). \tag{9}$$

Here c is a constant depending only on the norm $\|\cdot\|_p$, which is $1/2$ for ℓ_∞-norm, and $\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$ is the margin between the largest and second largest output logits.

Proof. Let $g(\boldsymbol{x}) = y$. Suppose there exists a $\widehat{\boldsymbol{\delta}}$ such that $g(\boldsymbol{x}+\widehat{\boldsymbol{\delta}}) \neq g(\boldsymbol{x})$, and $f_j(\boldsymbol{x}+\widehat{\boldsymbol{\delta}}) \geq f_y(\boldsymbol{x}+\widehat{\boldsymbol{\delta}})$ for some $j \neq y$. We will prove that $\|\widehat{\boldsymbol{\delta}}\|_p \geq \frac{c}{L}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$ where c only depends on the norm.

Define $\widehat{\boldsymbol{z}} = f(\boldsymbol{x}+\widehat{\boldsymbol{\delta}})$, then $\widehat{z}_y \leq \widehat{z}_j$. We first bound the difference between outputs $\boldsymbol{z}$ and $\boldsymbol{f}(\boldsymbol{x})$ as follows:

$$\|\widehat{\boldsymbol{z}} - \boldsymbol{f}(\boldsymbol{x})\|_p \geq \|(\widehat{z}_y, \widehat{z}_j)^{\mathrm{T}} - ([f_y(\boldsymbol{x}), f_j(\boldsymbol{x}))^{\mathrm{T}}\|_p \tag{10}$$

$$= (|\widehat{z}_y - f_y(\boldsymbol{x})|^p + |\widehat{z}_j - f_j(\boldsymbol{x})|^p)^{1/p}. \tag{11}$$

In (10) we use the fact that zero out elements for a vector can only decrease its ℓ_p-norm. Now consider the following optimization problem:

$$\min_{\widehat{\boldsymbol{z}}} |\widehat{z}_y - f_y(\boldsymbol{x})|^p + |\widehat{z}_j - f_j(\boldsymbol{x})|^p \quad \text{s.t. } \widehat{z}_y \leq \widehat{z}_j. \tag{12}$$

It is easy to prove that the minimum is attained when $\widehat{z}_y = \widehat{z}_j = (f_y(\boldsymbol{x}) + f_j(\boldsymbol{x}))/2$. Substituting the assignment into (11) yields

$$\|\widehat{\boldsymbol{z}} - \boldsymbol{f}(\boldsymbol{x})\|_p \geq \frac{\sqrt[p]{2}}{2}(f_y(\boldsymbol{x}) - f_j(\boldsymbol{x})) \tag{13}$$

which can be further lower bounded by $\frac{\sqrt[p]{2}}{2}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$ based on the definition of margin. Finally, due to the Lipschitz property,

$$\|\widehat{\boldsymbol{z}} - \boldsymbol{f}(\boldsymbol{x})\|_p \leq L\|\boldsymbol{\delta}\|_p. \tag{14}$$

Therefore $\|\boldsymbol{\delta}\|_p \geq \frac{\sqrt[p]{2}}{2L}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$, which concludes the proof. □

Remark B.2. All inequalities in the above proof is tight by choosing a $\widehat{\boldsymbol{\delta}}$ with $\|\widehat{\boldsymbol{\delta}}\|_p = \frac{\sqrt[p]{2}}{2L}\operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))$ and a Lipschitz function $\boldsymbol{f}$ such that $\widehat{z}_y = f_y(\boldsymbol{x}) - \operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))/2$, $\widehat{z}_i = f_i(\boldsymbol{x}) + \operatorname{margin}(\boldsymbol{f}(\boldsymbol{x}))/2$ and $\widehat{z}_k = f_k(\boldsymbol{x})$ for all $k \neq y$ and $k \neq i$. This implies that the certified guarantee in the above proposition is tight if only the Lipschitz property is known.

B.2 Proof of Proposition 3.2

Proposition B.3. *For any Boolean dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^n$, there exists a classifier $\hat{g} : \mathbb{R}^d \to \{0,1\}$ induced by a 1-Lipschitz mapping $\hat{\boldsymbol{f}} : \mathbb{R}^d \to \mathbb{R}^2$ under ℓ_∞-norm, such that $\hat{g}$ can fit the whole dataset with $\operatorname{margin}(\hat{\boldsymbol{f}}(\boldsymbol{x}^{(i)})) = 1\ \forall i \in [n]$, thus achieving an optimal certified ℓ_∞ robust radius of $1/2$.*

Proof. Without loss of generality, assume for each class (either 0 or 1), there is at least one sample. Define the nearest neighbor classifier $\hat{g} : \mathbb{R}^d \to \{0,1\}$ as $\hat{g}(x) = \arg\max_{k\in\{0,1\}} \hat{f}_k(\boldsymbol{x})$ where

$$\hat{f}_k(\boldsymbol{x}) = -\min\{\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|_\infty : (\boldsymbol{x}^{(i)}, k) \in \mathcal{D}\}, \quad k \in \{0,1\}.$$

Then for any $(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D}$, $f_{y^{(i)}}(\boldsymbol{x}) = 0$ and $f_{1-y^{(i)}}(\boldsymbol{x}) = 1$ because $\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_\infty = 1$ for all $i \neq j$. Therefore, the classifier g can correctly classify the whole dataset with margin $\operatorname{margin}(\hat{\boldsymbol{f}}(\boldsymbol{x}^{(i)})) = 1$ ($\forall i \in [n]$). The left thing is to prove the Lipschitzness of $\boldsymbol{f}$, which is equivalent to proving that each function $\hat{f}_k(\boldsymbol{x})$ is 1-Lipschitz w.r.t. ℓ_∞-norm. This simply follows from the fact that $\hat{f}_k$ is the composition of two 1-Lipschitz functions: the ℓ_∞-distance function $\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|_\infty$ and the minimum function. Since the composition of 1-Lipschitz functions is still 1-Lipschitz, we have concluded the proof. □

B.3 Proof of Theorem 3.5

We first present a core lemma which will be used to prove Theorem 3.5.

Lemma B.4. *Let $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)$ be a d-dimensional function where $\|\boldsymbol{w}\|_1 \leq 1$ and σ is a 1-Lipschitz scalar function, and let $\{(\boldsymbol{u}^{(i)}, \boldsymbol{v}^{(i)}) \in \mathbb{R}^d \times \mathbb{R}^d\}_{i=1}^n$ be n pairs of inputs. Then*

$$\sum_{i=1}^n \left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| \leq \left\| \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right\|_\infty .$$

Proof. First note that

$$\begin{aligned} \left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| &= \left| \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{u}^{(i)} + b) - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{v}^{(i)} + b) \right| \\ &\leq \left| \boldsymbol{w}^{\mathrm{T}}(\boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)}) \right| \qquad (15) \\ &\leq |\boldsymbol{w}|^{\mathrm{T}} \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| . \qquad (16) \end{aligned}$$

where (15) uses the Lipschitz property of σ. Therefore

$$\sum_{i=1}^n \left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| \leq |\boldsymbol{w}|^{\mathrm{T}} \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \leq \|\boldsymbol{w}\|_1 \left\| \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right\|_\infty , \qquad (17)$$

where the last inequiality in (17) follows by using the Hölder's inequality. Finally, observing $\|\boldsymbol{w}\|_1 \leq 1$, we thus conclude the proof. □

Corollary B.5. *Let $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^K$ be a standard Lipschitz network defined in (3), and let $\{(\boldsymbol{u}^{(i)}, \boldsymbol{v}^{(i)}) \in \mathbb{R}^d \times \mathbb{R}^d\}_{i=1}^n$ be n pairs of inputs. Then*

$$\left\| \sum_{i=1}^n \left| \boldsymbol{f}(\boldsymbol{u}^{(i)}) - \boldsymbol{f}(\boldsymbol{v}^{(i)}) \right| \right\|_\infty \leq \left\| \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right\|_\infty .$$

Proof. Denote $\boldsymbol{f}^{(l)}(\boldsymbol{z})$ be the output of the l-th layer for a standard Lipschitz network $\boldsymbol{f}$ given input $\boldsymbol{z}$. Based on the definition in (3), for the l-th layer, $\sigma^{(l)}$ is 1-Lipschitz and $\|\mathbf{W}^{(l)}\|_\infty \leq 1$, which is equivalent to $\|\mathbf{W}^{(l)}_{j,:}\|_1 \leq 1$ for the weight vector of each neuron j in layer l. One can thus apply Lemma B.4 for the j-th neuron, given arbitrary inputs $\{\boldsymbol{f}^{(l-1)}(\boldsymbol{u}^{(i)})\}_{i=1}^n$ and $\{\boldsymbol{f}^{(l-1)}(\boldsymbol{v}^{(i)})\}_{i=1}^n$:

$$\sum_{i=1}^n \left| f_j^{(l)}(\boldsymbol{u}^{(i)}) - f_j^{(l)}(\boldsymbol{v}^{(i)}) \right| \leq \left\| \sum_{i=1}^n \left| \boldsymbol{f}^{(l-1)}(\boldsymbol{u}^{(i)}) - \boldsymbol{f}^{(l-1)}(\boldsymbol{v}^{(i)}) \right| \right\|_\infty . \qquad (18)$$

This is equivalence to the following inequality due to the definition of ℓ_∞-norm:

$$\left\| \sum_{i=1}^n \left| \boldsymbol{f}^{(l)}(\boldsymbol{u}^{(i)}) - \boldsymbol{f}^{(l)}(\boldsymbol{v}^{(i)}) \right| \right\|_\infty \leq \left\| \sum_{i=1}^n \left| \boldsymbol{f}^{(l-1)}(\boldsymbol{u}^{(i)}) - \boldsymbol{f}^{(l-1)}(\boldsymbol{v}^{(i)}) \right| \right\|_\infty . \qquad (19)$$

which is a recursive formula. Applying (19) recursively from the last layer to the first yields the desired result. □

We are now ready for the proof of Theorem 3.5, which is restated below.

Theorem B.6. *For any non-constant symmetric Boolean function $g^B : \{0,1\}^d \to \{0,1\}$, there exists a Boolean dataset with labels $y^{(i)} = g^B(\boldsymbol{x}^{(i)})$, such that no standard Lipschitz network can achieve a certified ℓ_∞ robust radius larger than $1/2d$ on the dataset.*

Proof. We first introduce some notations. Denote $\mathcal{S}_k = \left\{\boldsymbol{x} \in \{0,1\}^d : \sum_i x_i = k\right\}$ as the set containing all Boolean vectors with k ones. It follows that $\{\mathcal{S}_k\}_{k=0}^d$ is a partition of the set of d-dimensional Boolean vectors $\{0,1\}^d$. For a Boolean function g, define the positive set and the negative set as $\mathcal{S}^+ = \left\{\boldsymbol{x} \in \{0,1\}^d : g^{\mathrm{B}}(\boldsymbol{x}) = 1\right\}$ and $\mathcal{S}^- = \left\{\boldsymbol{x} \in \{0,1\}^d : g^{\mathrm{B}}(\boldsymbol{x}) = 0\right\}$, respectively.

Recall that the output of a symmetric Boolean function depends only on the number of ones in the input (Appendix A.2). Therefore, instead of using the positive/negative set, we can use the following two sets to equivalently characterize a symmetric Boolean function g^{B}:

$$\mathcal{N}^+ = \left\{k \in \{0, \cdots, k\} : \mathcal{S}_k \cup \mathcal{S}^+ \neq \emptyset\right\}, \quad \mathcal{N}^- = \left\{k \in \{0, \cdots, k\} : \mathcal{S}_k \cup \mathcal{S}^- \neq \emptyset\right\}.$$

It turns out that $\mathcal{S}^+ = \bigcup_{k \in \mathcal{N}^+} \mathcal{S}_k$, $\mathcal{S}^- = \bigcup_{k \in \mathcal{N}^-} \mathcal{S}_k$, and $\mathcal{N}^+ \cup \mathcal{N}^- = \{0, \cdots, d\}$.

Since g^{B} is non-constant, both $\mathcal{N}^+$ and $\mathcal{N}^-$ are non-empty. Therefore, there must exist two adjacent integers $p, q \in \{0, \cdots, d\}$, $p - q = 1$, such that either ($p \in \mathcal{N}^+$, $q \in \mathcal{N}^-$) or ($p \in \mathcal{N}^-$, $q \in \mathcal{N}^+$). Consider the following set of Boolean vector pairs:

$$\mathcal{T} = \{(\boldsymbol{u}, \boldsymbol{v}) : \boldsymbol{u} \in \mathcal{S}_p, \boldsymbol{v} \in \mathcal{S}_q, \|\boldsymbol{u} - \boldsymbol{v}\|_1 = 1\}. \tag{20}$$

The set $\mathcal{T}$ satisfies the following three properties:

- $g^{\mathrm{B}}(\boldsymbol{u}) \neq g^{\mathrm{B}}(\boldsymbol{v})$ for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{T}$;
- The size of the set can be calculated by

  $$|\mathcal{T}| = |\mathcal{S}_p| p = |\mathcal{S}_q|(d - q) = \tfrac{d!}{q!(d-p)!},$$

  by using $|\mathcal{S}_k| = \binom{d}{k}$;
- $\left\|\sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{T}} |\boldsymbol{u} - \boldsymbol{v}|\right\|_\infty = \frac{(d-1)!}{q!(d-p)!} = \frac{|\mathcal{T}|}{d}$. This can be seen from the symmetry of the vector $\sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{T}} |\boldsymbol{u} - \boldsymbol{v}|$ and the fact that all vectors $|\boldsymbol{u} - \boldsymbol{v}|$ are unit (one-hot).

Based on the last property, given any standard Lischitz network $\boldsymbol{f}$, one can apply Corollary B.5 on $\mathcal{T}$, which yields

$$\left\|\sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{T}} |\boldsymbol{f}(\boldsymbol{u}) - \boldsymbol{f}(\boldsymbol{v})|\right\|_\infty \leq \frac{|\mathcal{T}|}{d}. \tag{21}$$

Using the definition of ℓ_∞-norm and noting that the output of function $\boldsymbol{f}$ is 2-dimensional, (21) implies that

$$\sum_{k \in \{0,1\}} \sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{T}} |f_k(\boldsymbol{u}) - f_k(\boldsymbol{v})| \leq \frac{2|\mathcal{T}|}{d}. \tag{22}$$

By the Pigeon Hole principle, there must exist a pair of points $(\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{T}$, such that

$$\sum_{k \in \{0,1\}} |f_k(\boldsymbol{u}) - f_k(\boldsymbol{v})| \leq \frac{2}{d}. \tag{23}$$

Now it is time to construct the dataset. Define a Boolean dataset $\mathcal{D} = \{(\boldsymbol{z}, g^{\mathrm{B}}(\boldsymbol{z})) : \boldsymbol{z} \in \mathcal{S}_p \cup \mathcal{S}_q\}$. Then there must exist two data $(\boldsymbol{u}, g^{\mathrm{B}}(\boldsymbol{u}))$ and $(\boldsymbol{v}, g^{\mathrm{B}}(\boldsymbol{v}))$ with different labels (i.e. $g^{\mathrm{B}}(\boldsymbol{u}) \neq g^{\mathrm{B}}(\boldsymbol{v})$) such that (23) holds. Let us calculate the output margin for the two inputs $\boldsymbol{u}$ and $\boldsymbol{v}$. It can be obtained that

$$\begin{aligned} |(f_0(\boldsymbol{u}) - f_1(\boldsymbol{u})) + (f_1(\boldsymbol{v}) - f_0(\boldsymbol{v}))| &= |(f_0(\boldsymbol{u}) - f_0(\boldsymbol{v})) + (f_1(\boldsymbol{v}) - f_1(\boldsymbol{u}))| \\ &\leq \sum_{k \in \{0,1\}} |f_k(\boldsymbol{v}) - f_k(\boldsymbol{u})| \leq \frac{2}{d}. \end{aligned} \tag{24}$$

We will prove that it is impossible to classify both $\boldsymbol{u}$ and $\boldsymbol{v}$ with a margin greater than $1/d$. Otherwise, since $\boldsymbol{u}$ and $\boldsymbol{v}$ are in different classes, $f_0(\boldsymbol{u}) - f_1(\boldsymbol{u})$ and $f_1(\boldsymbol{v}) - f_0(\boldsymbol{v})$ will have the same sign, and both $|f_0(\boldsymbol{u}) - f_1(\boldsymbol{u})|$ and $|f_1(\boldsymbol{v}) - f_0(\boldsymbol{v})|$ will be greater than $1/d$, which contradicts (24). Therefore, the output margin must be no greater than $1/d$ for either input $\boldsymbol{u}$ or $\boldsymbol{v}$, implying that the certified ℓ_∞ robust radius cannot exceed $1/2d$ (based on Proposition 3.1) and concluding the proof. □

Remark B.7. The symmetry assumption of the Boolean function g^{B} may be relaxed, and the conclusion is still correct as long as there exists a set $\mathcal{T} \subset \{(\boldsymbol{u}, \boldsymbol{v}) : \boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^d\}$ satisfying that (i) $g^{\mathrm{B}}(\boldsymbol{u}) \neq g^{\mathrm{B}}(\boldsymbol{v})$ $\forall (\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{T}$ and (ii) $\left\|\sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{T}} |\boldsymbol{u} - \boldsymbol{v}|\right\|_\infty = \mathcal{O}(|\mathcal{T}|/d)$.

We now prove that the bound of certified radius is in fact *tight*, in the sense that there exists a standard Lipschitz network with a suitable activation function that achieves a certified ℓ_∞ radius of $1/2d$.

Proposition B.8. *For any d-dimensional symmetric Boolean dataset, there exists a standard Lipschitz network that correctly classifies the whole dataset and achieves a certified ℓ_∞ robust radius of $1/2d$.*

Proof. Consider a d-dimensional Boolean Dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}\}_{i=1}^n\}$ where $y^{(i)} = g^{\mathrm{B}}(\boldsymbol{x}^{(i)})$ for some symmetric Boolean function g^{B}. Since the output of a symmetric Boolean function depends only on the number of ones in the input (Appendix A.2), we can equivalently express g^{B} by a Boolean vector $(g_0, \cdots, g_d) \in \{0,1\}^{d+1}$ with $g^{\mathrm{B}}(\boldsymbol{x}) = g_k$ if $\sum_i x_i = k$. Consider a special one-layer standard Lipschitz network $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^2$ with activation σ defined as follows:

$$\boldsymbol{f}(\boldsymbol{x}) = (f_0(\boldsymbol{x}), f_1(\boldsymbol{x}))^{\mathrm{T}} = \left(\sigma\left(-\frac{1}{d}\sum_i x_i - \frac{1}{d}\right), \sigma\left(\frac{1}{d}\sum_i x_i + \frac{1}{d}\right)\right)^{\mathrm{T}}.$$

$\boldsymbol{f}$ is clearly 1-Lipschitz w.r.t. ℓ_∞-norm as long as σ is 1-Lipschitz. If we choose a special σ defined on the interval $[-1-1/d, 1+1/d]$ as

$$\sigma(x) = \begin{cases} \frac{(-1)^{g_0+1}}{2} x & \text{if } -1 \le dx \le 1, \\ \frac{\mathrm{sgn}(x)}{2d}(2g_{k-1} - 1 + 2(d|x| - k)(g_k - g_{k-1})) & \text{if } k < d|x| \le k+1, \quad k \in [d], \end{cases}$$

which is a piece-wise linear function, it is easy to see that σ is continuous and 1-Lipschitz. Furthermore, we can prove that the classifier induced by $\boldsymbol{f}$ can correctly classify the dataset with a margin of $1/d$ for any data. Consider an input $\boldsymbol{x}$ with $\sum_i x_i = k$. The output of $\boldsymbol{f}(\boldsymbol{x})$ is thus $\left(\sigma(-\frac{k+1}{d}), \sigma(\frac{k+1}{d})\right)^{\mathrm{T}}$. Noting that σ is an odd function, we have

$$f_1(\boldsymbol{x}) - f_0(\boldsymbol{x}) = 2\sigma\left(\frac{k+1}{d}\right) = \frac{2g_k - 1}{d} = \frac{1}{d}(2g^{\mathrm{B}}(\boldsymbol{x}) - 1) = \begin{cases} 1/d & \text{if } g^{\mathrm{B}}(\boldsymbol{x}) = 1, \\ -1/d & \text{if } g^{\mathrm{B}}(\boldsymbol{x}) = 0. \end{cases}$$

This proves that the margin is always $1/d$ for each data of $\mathcal{D}$, and we have concluded the proof. □

Remark B.9. Let us consider the sample complexity in the proof of Theorem 3.5. The proof constructs the Boolean dataset $\mathcal{D} = \{(\boldsymbol{z}, g^{\mathrm{B}}(\boldsymbol{z})) : \boldsymbol{z} \in \mathcal{S}_p \cup \mathcal{S}_q\}$, which has size $|\mathcal{D}| = \binom{d}{p} + \binom{d}{q}$. In the simple case when g^{B} is the logical AND/OR function, one has $|\mathcal{D}| = d+1$, which means that a dataset of size $\mathcal{O}(d)$ is sufficient to yield the impossibility result. However, in the extreme case when $p = \lceil \frac{d}{2} \rceil$, the size of $\mathcal{D}$ will become exponential in d, which makes the theorem impractical. One may ask whether for all non-constant symmetric Boolean functions, such an impossibility result holds on some dataset with a size that grows linear in d. We will show it is indeed true if the bound of certified radius is relaxed by constant to $1/d$, which is formalized in the following corollary.

Corollary B.10. *For any non-constant symmetric Boolean function $g^B : \{0,1\}^d \to \{0,1\}$, there exists a Boolean dataset of size no more than $d+1$ with labels $y^{(i)} = g^B(\boldsymbol{x}^{(i)})$, such that no standard Lipschitz network can achieve a certified ℓ_∞ robust radius larger than $1/d$ on the dataset.*

Proof. The proof is almost the same as the one in Theorem 3.5, except for a different construction of the set $\mathcal{T}$ than in (20). We use the same notations as before, such as the sets $\mathcal{S}_k, \mathcal{S}^+, \mathcal{S}^-, \mathcal{N}^+, \mathcal{N}^-$, and the integers p, q with $p = q+1$. We separately consider two cases:

- $p \ge \lceil d/2 \rceil$. In this case, pick any vector $\boldsymbol{u} \in \mathcal{S}_p$ and define
$$\mathcal{T} = \{(\boldsymbol{u}, \boldsymbol{v}) : \boldsymbol{v} \in \mathcal{S}_q, \|\boldsymbol{u} - \boldsymbol{v}\|_1 = 1\}. \tag{25}$$
- $p < \lceil d/2 \rceil$. In this case, pick any vector $\boldsymbol{v} \in \mathcal{S}_q$ and define
$$\mathcal{T} = \{(\boldsymbol{u}, \boldsymbol{v}) : \boldsymbol{u} \in \mathcal{S}_p, \|\boldsymbol{u} - \boldsymbol{v}\|_1 = 1\}. \tag{26}$$

In both bases, the set $\mathcal{T}$ satisfies the following three properties:

- $g^{\mathrm{B}}(\boldsymbol{u}) \ne g^{\mathrm{B}}(\boldsymbol{v})$ for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{T}$;
- The size of the set can be bounded by $\lceil d/2 \rceil \le |\mathcal{T}| \le d$;
- $\left\| \sum_{(\boldsymbol{u},\boldsymbol{v}) \in \mathcal{T}} |\boldsymbol{u} - \boldsymbol{v}| \right\|_\infty = 1 \le \frac{2|\mathcal{T}|}{d}$.

The remaining proof directly parallels the proof of Theorem 3.5. □

B.4 Proof of Theorem 3.6

Theorem B.11. *Any standard Lipschitz network $f : \mathbb{R}^d \to \mathbb{R}$ cannot approximate the simple 1-Lipschitz function $\boldsymbol{x} \to x_{(k)}$ for arbitrary $k \in [d]$ on a bounded domain $\mathcal{K} = [0,1]^d$ if $d \geq 2$. Moreover, there exists a point $\widehat{\boldsymbol{x}} \in \mathcal{K}$, such that*

$$|f(\widehat{\boldsymbol{x}}) - \widehat{x}_{(k)}| \geq \frac{1}{2} - \frac{1}{2d}. \tag{27}$$

Proof. Consider the k-threshold function $g^{\mathrm{B},k}(\boldsymbol{x}) = \mathbb{I}(\sum_i x_i \geq k)$. Following the proof of Theorem 3.5, there exists a set $\mathcal{T} \subset \left\{(\boldsymbol{u}, \boldsymbol{v}) : \boldsymbol{u}, \boldsymbol{v} \in \{0,1\}^d\right\}$ satisfying the following two properties:

- $g^{\mathrm{B}}(\boldsymbol{u}) \neq g^{\mathrm{B}}(\boldsymbol{v})$ for all $(\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{T}$;
- $\left\|\sum_{(\boldsymbol{u},\boldsymbol{v})\in\mathcal{T}} |\boldsymbol{u} - \boldsymbol{v}|\right\|_\infty = \frac{|\mathcal{T}|}{d}$.

By applying Corollary B.5 and noting that f outputs a scalar, we obtain

$$\sum_{(\boldsymbol{u},\boldsymbol{v})\in\mathcal{T}} |f(\boldsymbol{u}) - f(\boldsymbol{v})| \leq \frac{|\mathcal{T}|}{d}. \tag{28}$$

By the Pigeon Hole principle, there must exist a pair of points $(\boldsymbol{u}, \boldsymbol{v}) \in \mathcal{T}$, such that

$$|f(\boldsymbol{u}) - f(\boldsymbol{v})| \leq \frac{1}{d}. \tag{29}$$

However, since $g^{\mathrm{B},k}(\boldsymbol{u}) \neq g^{\mathrm{B},k}(\boldsymbol{v})$, we have $\left|g^{\mathrm{B},k}(\boldsymbol{u}) - g^{\mathrm{B},k}(\boldsymbol{v})\right| = 1$. Therefore

$$\begin{aligned}
\left|f(\boldsymbol{v}) - g^{\mathrm{B},k}(\boldsymbol{v})\right| + \left|f(\boldsymbol{u}) - g^{\mathrm{B},k}(\boldsymbol{u})\right| &\geq \left|f(\boldsymbol{v}) - f(\boldsymbol{u}) + g^{\mathrm{B},k}(\boldsymbol{u}) - g^{\mathrm{B},k}(\boldsymbol{v})\right| && (30)\\
&\geq \left|g^{\mathrm{B},k}(\boldsymbol{u}) - g^{\mathrm{B},k}(\boldsymbol{v})\right| - |f(\boldsymbol{v}) - f(\boldsymbol{u})| && (31)\\
&\geq 1 - \frac{1}{d} && (32)
\end{aligned}$$

where (30) and (31) are based on the triangular inequality. Therefore, either $\left|f(\boldsymbol{v}) - g^{\mathrm{B},k}(\boldsymbol{v})\right|$ or $\left|f(\boldsymbol{u}) - g^{\mathrm{B},k}(\boldsymbol{u})\right|$ must be no smaller than $\frac{1}{2} - \frac{1}{2d}$. Proof finishes by noting that $g^{\mathrm{B},k}(\boldsymbol{x}) = x_{(k)}$ for any Boolean input $\boldsymbol{x} \in \{0,1\}^d$. □

Remark B.12. The bound of approximation error $\frac{1}{2} - \frac{1}{2d}$ is *tight*. Indeed, it is easy to prove that the linear function $f(\boldsymbol{x}) = \frac{1}{2} + \frac{1}{d}(\sum_i x_i - k + \frac{1}{2})$ can satisfy the bound, i.e. $|f(\boldsymbol{x}) - x_{(k)}| \leq \frac{1}{2} - \frac{1}{2d}$ for all $\boldsymbol{x} \in \mathcal{K}$.

Corollary B.13. *For any standard Lipschitz network $f : \mathbb{R}^d \to \mathbb{R}$ and any $k \in [d]$ where $d \geq 2$, there exists a hypercube $\mathcal{B}_\infty^r(\widehat{\boldsymbol{x}}) := \{\boldsymbol{x} : \|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_\infty \leq r\} \subset [0,1]^d$ satisfying $r \geq 1/20$, such that for all $\tilde{\boldsymbol{x}} \in \mathcal{B}_\infty^r(\widehat{\boldsymbol{x}})$,*

$$|f(\tilde{\boldsymbol{x}}) - \tilde{x}_{(k)}| \geq \frac{1}{20}. \tag{33}$$

Proof. Let $\boldsymbol{u} \in \{0,1\}^d$ be a point satisfying (4) in Theorem 3.6, i.e. $|f(\boldsymbol{u}) - u_{(k)}| \geq 1/4$. Consider the hypercube $\mathcal{B}_\infty^{1/10}(\boldsymbol{u})$ with a length of $1/5$. For any $\boldsymbol{x} \in \mathcal{B}_\infty^{1/10}(\boldsymbol{u})$, $\|\boldsymbol{x} - \boldsymbol{u}\|_\infty \leq 1/10$. Therefore,

$$\begin{aligned}
|f(\boldsymbol{x}) - x_{(k)}| &= |f(\boldsymbol{x}) - f(\boldsymbol{u}) + f(\boldsymbol{u}) - u_{(k)} + u_{(k)} - x_{(k)}| && (34)\\
&\geq |f(\boldsymbol{u}) - u_{(k)}| - |f(\boldsymbol{u}) - f(\boldsymbol{x})| - |u_{(k)} - x_{(k)}| && (35)\\
&\geq \frac{1}{4} - \|\boldsymbol{x} - \boldsymbol{u}\|_\infty - \|\boldsymbol{x} - \boldsymbol{u}\|_\infty \geq \frac{1}{20} && (36)
\end{aligned}$$

where (35) uses the triangular inequality and (36) is based on the fact that f and $x_{(k)}$ are both 1-Lipschitz w.r.t. ℓ_∞-norm. Finally, consider the intersection $\mathcal{B}_\infty^{1/10}(\boldsymbol{u}) \cup [0,1]^d$, which must contain a hypercube with a raius no smaller than $1/20$. We have thus finished the proof. □

B.5 Proof of Theorem 3.8

To prove Theorem 3.8, we need the following lemma:

Lemma B.14. *A single ℓ_∞-distance neuron with suitable parameters can exactly represent any literal disjunction defined in Definition A.1. Formally, for any Boolean function*

$$g^B(\boldsymbol{x}) = x_{i_1} \vee \cdots \vee x_{i_r} \vee \neg x_{j_1} \vee \cdots \vee \neg x_{j_s},$$

where $1 \leq i_1 \leq \cdots \leq i_r \leq d, 1 \leq j_1 \leq \cdots \leq j_s \leq d$ $(r + s \geq 1)$ are different indices, there exists an ℓ_∞-distance neuron $f(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{w}\|_\infty + b$, such that $f(\boldsymbol{x}) = g^B(\boldsymbol{x})$ for all $\boldsymbol{x} \in \{0,1\}^d$.

Proof. Consider an ℓ_∞-distance neuron f with parameters $\boldsymbol{w}$ and b defined as

$$w_{i_k} = -1,\ k \in [r], \quad w_{j_k} = 2,\ k \in [s], \quad \text{and } w_k = \tfrac{1}{2} \text{ for other } k,$$
$$b = -1.$$

It is easy to see that

- When $x_{i_k} = 0\ \forall k \in [r]$ and $x_{j_k} = 1\ \forall k \in [s]$,
$$\begin{aligned} f(\boldsymbol{x}) &= \|\boldsymbol{x} - \boldsymbol{w}\|_\infty + b \\ &= \max\left(\max_{k\in[r]} |x_{i_k} - w_{i_k}|, \max_{k\in[s]} |x_{j_k} - w_{j_k}|, \max_{k\in[d]/\{i_1,\cdots,i_r,j_1,\cdots,j_s\}} |x_k - w_k|\right) - 1 \\ &= \max\left(1, 1, \max_{k\in[d]/\{i_1,\cdots,i_r,j_1,\cdots,j_s\}} \left|x_k - \frac{1}{2}\right|\right) - 1 \\ &= 1 - 1 = 0 = g^{\mathrm{B}}(\boldsymbol{x}). \end{aligned}$$
- When $x_{i_k} = 1$ for some $k \in [r]$, or $x_{j_k} = 0$ for some $k \in [s]$, a similar calculation yields $f(\boldsymbol{x}) = 2 - 1 = 1 = g^{\mathrm{B}}(\boldsymbol{x})$.

This concludes the proof. □

Theorem B.15. *A two-layer ℓ_∞-distance net can exactly represent any discrete Boolean function as well as any continuous order-statistic function on a compact domain.*

Proof. Based on Fact A.2, any satisfiable Boolean function can be written as a DNF $g^{\mathrm{B}}(\boldsymbol{x}) = \vee_{i=1}^m g_m^{\mathrm{B}}(\boldsymbol{x})$ where $g_m^{\mathrm{B}}(\boldsymbol{x})$ are literal conjunctions. By the De Morgan's law, any literal conjunction can be equivalently written as the negative of a literal disjunction, i.e.

$$x_{i_1} \wedge \cdots \wedge x_{i_r} \wedge \neg x_{j_1} \wedge \cdots \wedge \neg x_{j_s} = \neg(\neg x_{i_1} \vee \cdots \vee \neg x_{i_r} \vee x_{j_1} \vee \cdots \vee x_{j_s}). \tag{37}$$

Therefore, we can write $g^{\mathrm{B}}(\boldsymbol{x}) = \vee_{i=1}^m \neg \hat{g}_i^{\mathrm{B}}(\boldsymbol{x})$ where $\hat{g}_i^{\mathrm{B}}(\boldsymbol{x}) := \neg g_i^{\mathrm{B}}(\boldsymbol{x})$ are all disjunctive literals. Due to Lemma B.14, each $\hat{g}_i^{\mathrm{B}}(\boldsymbol{x})$ can be represented by an ℓ_∞-distance neuron that takes $\boldsymbol{x}$ as input, and $g^{\mathrm{B}}(\boldsymbol{x})$ can be represented by an ℓ_∞-distance neuron that takes the vector $(\hat{g}_1^{\mathrm{B}}(\boldsymbol{x}), \cdots, \hat{g}_m^{\mathrm{B}}(\boldsymbol{x}))^{\mathrm{T}}$ as input. This proves that a two-layer ℓ_∞-distance net suffices to represent any satisfiable discrete Boolean function. Finally, for the unsatisfiable case where the Boolean function simply outputs zero, we can construct the following two-layer ℓ_∞-distance net:

$$f(\boldsymbol{x}) = \left\|(\|\boldsymbol{x}\|_\infty, \|\boldsymbol{x} - \mathbf{1}\|_\infty)^{\mathrm{T}}\right\|_\infty - 1 \tag{38}$$

where $\mathbf{1}$ is the all-one vector. It follows that $(\|\boldsymbol{x}\|, \|\boldsymbol{x} - \mathbf{1}\|)^{\mathrm{T}}$ must be one of the following three cases: $(0,1)^{\mathrm{T}}$ (when $\boldsymbol{x} = \mathbf{0}$), $(1,0)^{\mathrm{T}}$ (when $\boldsymbol{x} = \mathbf{1}$), or $(1,1)^{\mathrm{T}}$ (when $\boldsymbol{x} \neq \mathbf{0}$ and $\boldsymbol{x} \neq \mathbf{1}$). Therefore $f(\boldsymbol{x}) = 0$ always hold $\forall \boldsymbol{x} \in \{0,1\}^d$. This concludes the first part of the proof.

We next turn to the case of representing order statistics. We use the key observation that any order statistic function can be written as a nested max-min function:

$$x_{(k)} = \max_{S\subset[d], |S|=k} \min_{i\in S} x_i. \tag{39}$$

This can be further equivalently written as a nested max-max function:

$$x_{(k)} = \max_{S\subset[d], |S|=k} -f^S(\boldsymbol{x}) \tag{40}$$

where $f^S(\boldsymbol{x}) = \max_{i\in S} -x_i$. This basically concludes the proof, as the max function over a set of elements $\{x_{i_1}, \cdots, x_{i_r}, -x_{j_1}, \cdots, -x_{j_s}\}$ can be exactly represented by an ℓ_∞-distance neuron following the construction in the proof of Lemma B.14. Therefore, each $f^S(\boldsymbol{x})$ can be represented by an ℓ_∞-distance neuron with input $\boldsymbol{x}$, and $x_{(k)}$ can be represented by an ℓ_∞-distance neuron that takes all $f^S(\boldsymbol{x})$ as input. As a result, a two-layer ℓ_∞-distance net suffices to represent the order statistics. □

Remark B.16. (On the representation efficiency of ℓ_∞-distance net) Based on the above proof, the width of the two-layer ℓ_∞-distance net in representing a Boolean function depends on the minimum number of conjunctions in its DNF, which may be exponentially large. Indeed, it is known that no DNF with a polynomial number of conjunctions can approximate the majority function defined in Section A.2 [44]. Similarly, in the continuous setting, the construction in (39, 40) needs a network of size $\binom{d}{\lfloor d/2 \rfloor}$ in order to represent the median function $k = \lfloor d/2 \rfloor$, which is exponential in d. However, when using a *multi-layer* ℓ_∞-distance net, the representation efficiency can be improved dramatically. For example, using the idea of sorting, it is easy to construct an ℓ_∞-distance net with polynomial size that represents the median function.

B.6 Proof of Theorem 3.9

The proof is almost the same as the one in Theorem 3.6. The major difference is to replace Lemma B.4 by the following lemma:

Lemma B.17. *Let $f(\boldsymbol{x}) = \sigma(\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1, \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} + b_2)$ be a d-dimensional function where $\|\boldsymbol{w}_i\|_1 \le 1$ $\forall i \in [2]$ and σ is a two-dimensional 1-Lipschitz function w.r.t. ℓ_∞-norm. Let $\{(\boldsymbol{u}^{(i)}, \boldsymbol{v}^{(i)}) \in \mathbb{R}^d \times \mathbb{R}^d\}_{i=1}^n$ be n pairs of inputs. Then*

$$\sum_{i=1}^n \left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| \le 2 \left\| \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right\|_\infty .$$

Proof. Using the Lipschitz property of σ, we have

$$\begin{aligned}
\left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| &= \left| \sigma(\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{u}^{(i)} + b_1, \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{u}^{(i)} + b_2) - \sigma(\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{v}^{(i)} + b_1, \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{v}^{(i)} + b_2) \right| \\
&\le \max\left(\left| \boldsymbol{w}_1^{\mathrm{T}}(\boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)}) \right|, \left| \boldsymbol{w}_2^{\mathrm{T}}(\boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)}) \right| \right) && (41) \\
&\le \max\left(|\boldsymbol{w}_1|^{\mathrm{T}} \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right|, |\boldsymbol{w}_2|^{\mathrm{T}} \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right) && (42) \\
&\le (|\boldsymbol{w}_1| + |\boldsymbol{w}_2|)^{\mathrm{T}} \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| && (43)
\end{aligned}$$

Therefore

$$\begin{aligned}
\sum_{i=1}^n \left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| &\le (|\boldsymbol{w}_1| + |\boldsymbol{w}_2|)^{\mathrm{T}} \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \\
&\le \||\boldsymbol{w}_1| + |\boldsymbol{w}_2|\|_1 \left\| \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right\|_\infty , && (44)
\end{aligned}$$

where the last inequiality in (44) follows by using the Hölder's inequality. Finally, observing that $\||\boldsymbol{w}_1| + |\boldsymbol{w}_2|\|_1 \le \|\boldsymbol{w}_1\|_1 + \|\boldsymbol{w}_2\|_1 \le 2$, we have arrived at the conclusion. □

Theorem B.18. *Any M-layer MaxMin network $f : \mathbb{R}^d \to \mathbb{R}$ cannot approximate any k-th order statistic function on a bounded domain $\mathcal{K} = [0,1]^d$ if $M \le \lceil \log_2 d \rceil$. Moreover, there exists a point $\widehat{\boldsymbol{x}} \in \mathcal{K}$, such that*

$$|f(\widehat{\boldsymbol{x}}) - \widehat{x}_{(k)}| \ge \frac{1}{2} - \frac{2^{M-2}}{d} \ge \frac{1}{4} \quad \text{if } M \le \lfloor \log_2 d \rfloor. \tag{45}$$

Proof. For an M-layer MaxMin network, there are $M-1$ layers with the MaxMin activation. Using the same proof as in Corollary B.5, we have that for any set $\{(\boldsymbol{u}^{(i)}, \boldsymbol{v}^{(i)}) \in \mathbb{R}^d \times \mathbb{R}^d\}_{i=1}^n$,

$$\left\| \sum_{i=1}^n \left| f(\boldsymbol{u}^{(i)}) - f(\boldsymbol{v}^{(i)}) \right| \right\|_\infty \le 2^{M-1} \left\| \sum_{i=1}^n \left| \boldsymbol{u}^{(i)} - \boldsymbol{v}^{(i)} \right| \right\|_\infty .$$

The remaining proof is the same as the proof of Theorem 3.6, with (28) replaced by

$$\left\| \sum_{(\boldsymbol{u},\boldsymbol{v})\in\mathcal{T}} |f(\boldsymbol{u}) - f(\boldsymbol{v})| \right\|_\infty \leq \frac{2^{M-1}|\mathcal{T}|}{d}. \tag{46}$$

□

We can also give an upper bound of the minimum depth required. The upper bound in the following theorem shows that the lower bound of the depth in Theorem 3.9 is tight up to a constant factor.

Theorem B.19. *An M-layer MaxMin network $f : \mathbb{R}^d \to \mathbb{R}$ with dpeth $M = \mathcal{O}(\log_2 d)$ can exactly represent any d-dimensional order statistic function on a bounded domain $\mathcal{K}$.*

Proof. Our proof leverages the work of Ajtai et al. [1], who constructed a so-called *sorting network* which has depth $\mathcal{O}(\log_2 d)$. The basic component of a sorting network is called the *comparator*. Each comparator connects two inputs and swaps the values if and only if the value of the first input is greater than the second. The comparators have a hierarchical structure (arranged layer by layer), and the execution is parallel for all comparators in the same layer. To avoid conflicts, no input can be simultaneously connected by two comparators in the same layer. We will show that given a sorting network (denoted as sort), one can construct an equivalent MaxMin network $\boldsymbol{f} : \mathbb{R}^d \to \mathbb{R}^d$ that has the same depth, such that for some fixed permutation $\pi \in S_d$, $\mathrm{sort}(\boldsymbol{x}) = (f_{\pi_1}(\boldsymbol{x}), \cdots, f_{\pi_d}(\boldsymbol{x}))^{\mathrm{T}}$ holds for all inputs $\boldsymbol{x} \in \mathcal{K}$.

Intuitively, each pair of MaxMin neurons can perform the same operation as a comparator. This is simply because

$$(\max(x_i, x_j), \min(x_i, x_j)) = \mathrm{MaxMin}(\boldsymbol{e}_i^{\mathrm{T}}\boldsymbol{x}, \boldsymbol{e}_j^{\mathrm{T}}\boldsymbol{x}).$$

For those inputs that are not connected to any comparator, one can construct an identity function to propagate the input without change, by using a pair of MaxMin neurons with properly chosen bias values so that one neuron is always larger than the other:

$$(x_i, b) = \mathrm{MaxMin}(\boldsymbol{e}_i^{\mathrm{T}}\boldsymbol{x}, b) \quad \text{where } b \to -\infty.$$

Note that we waste the neuron that always outputs b. Eventually, we have constructed a MaxMin network that can output the sorting result (differing by up to a permutation) within $\mathcal{O}(\log_2 d)$ depth. Therefore, by adding one more layer that extracts the k-th order statistic, the network then exactly represents the k-th order statistic function. □

B.7 Proof of Theorem 3.10

Proposition B.20. *An M-layer MaxMin network can exactly represent any (discrete) d-dimensional Boolean function if $M > d + \lceil \log_2 d \rceil$.*

Proof. First, it is easy to see that a pair of MaxMin neurons can represent any two-dimensional literal conjunctions/disjunctions, i.e. $r_i \wedge r_j$ and $r_i \vee x_j$, where r_i is either x_i or $\neg x_i$. For example, $\max(x_i, x_j) = x_i \vee x_j$, $\min(x_i, x_j) = x_i \wedge x_j$, $\min(1 - x_i, 1 - x_j) = \neg x_i \wedge \neg x_j$, etc. Next, using reduction, any d-dimensional literal conjunction/disjunction can be represented by a $(\lceil \log_2 d \rceil + 1)$-layer MaxMin network (here $+1$ means that the last layer does not have MaxMin activation). Finally, recall that any Boolean function can be written in its DNF (Appendix A.1) and the number of disjunctions cannot exceed 2^d for any d-dimensional Boolean function, we have proved that a MaxMin network with depth $M = d + \lceil \log_2 d \rceil + 1$ can represent any Boolean function. □

In the following, we will prove that the bound of $M = \mathcal{O}(d)$ is *tight* in order to interpolate d-dimensional Boolean functions. To do this, we first define several notations.

Definition B.21. Let $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathbb{R}^d \times \{0, 1\}\}_{i=1}^n$ be a dataset with real vectors as inputs and Boolean labels, and the inputs have a bounded diameter of 1 w.r.t. ℓ_∞-norm, i.e. $\|\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)}\|_\infty \leq 1$ $\forall j, k \in [n]$.

- Denote $l_i(\mathcal{D}) = \min_{j\in[n]} x_i^{(j)}$ and $u_i(\mathcal{D}) = \max_{j\in[n]} x_i^{(j)}$. We say the i-th input element is *useful* if $u_i(\mathcal{D}) - l_i(\mathcal{D}) = 1$; Correspondingly, the i-th input element is *useless* if $u_i(\mathcal{D}) - l_i(\mathcal{D}) < 1$.

- Given an index set $\mathcal{I} \subset [d]$, define the $\mathcal{I}$-*masked* dataset $\mathcal{D}^{\mathrm{M}}_{\mathcal{I}}$ to be the dataset obtained by zeroing out the i-th input element of each sample for each $i \in \mathcal{I}$. Specifically, $\mathcal{D}^{\mathrm{M}}_{\mathcal{I}} := \{(\tilde{\boldsymbol{x}}^{(j)}, y^{(j)})\}_{j=1}^n$ where
$$\tilde{x}_i^{(j)} = \begin{cases} x_i^{(j)} & i \notin \mathcal{I}, \\ 0 & i \in \mathcal{I}. \end{cases}$$
- Given an index set $\mathcal{I} \subset [d]$, define the $\mathcal{I}$-*thresholded* dataset $\mathcal{D}^{\mathrm{T}}_{\mathcal{I}}$ to be any dataset satisfying that, for each $i \in \mathcal{I}$, change the i-th input element of all samples to the extreme values, either the minimum $l_i(\mathcal{D})$ or the maximum $u_i(\mathcal{D})$, but keep the original minimum and maximum elements unchanged. Specifically, $\mathcal{D}^{\mathrm{T}}_{\mathcal{I}} := \{(\tilde{\boldsymbol{x}}^{(j)}, y^{(j)})\}_{j=1}^n$ where
$$\begin{cases} \tilde{x}_i^{(j)} = x_i^{(j)} & \text{if} \quad i \notin \mathcal{I}, \\ \tilde{x}_i^{(j)} = x_i^{(j)} & \text{if} \quad i \in \mathcal{I},\ x_i^{(j)} = l_i(\mathcal{D}) \text{ or } x_i^{(i)} = u_i(\mathcal{D}), \\ \tilde{x}_i^{(j)} \in \{l_i(\mathcal{D}), u_i(\mathcal{D})\} & \text{if} \quad i \in \mathcal{I},\ l_i(\mathcal{D}) < x_i^{(j)} < u_i(\mathcal{D}). \end{cases}$$
Clearly, the $\mathcal{I}$-thresholded dataset is not unique.
- We say a new dataset $\mathcal{D}^{\mathrm{F}}$ is *formatted* from dataset $\mathcal{D}$, if $\mathcal{D}^{\mathrm{F}}$ can be obtained by *masking* all the *useless* elements in $\mathcal{D}$ and *thresholding* all *useful* elements in $\mathcal{D}$.

Lemma B.22. *Let $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathbb{R}^d \times \{0,1\}\}_{i=1}^n$ be a finite dataset with a bounded diameter of 1 w.r.t. ℓ_∞-norm. If $\mathcal{D}$ can be interpolated by a MaxMin network, then any dataset $\mathcal{D}^F$ formatted from $\mathcal{D}$ can be interpolated by a MaxMin network with the same structure (i.e. depth and width).*

Proof. Our proof is based on induction over the network depth. For the base case, consider the network that performs the identify mapping: $f(x) = x$ where $x \in \mathbb{R}$ is a scalar. If f interpolates the dataset $\mathcal{D}$, then $d = 1$ and $\boldsymbol{x}^{(i)} \in \{0,1\}\ \forall i \in [n]$. Therefore, $\mathcal{D}^{\mathrm{F}} = \mathcal{D}$ and f interpolates $\mathcal{D}^{\mathrm{F}}$.

Now assume the conclusion holds for a MaxMin network $f : \mathbb{R}^{d_f} \to \mathbb{R}$. We would like to prove that it also holds for the MaxMin network $f \circ \boldsymbol{h}$ where $\boldsymbol{h} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_f}$ is either the affine layer or the MaxMin activation layer. This will yield the conclusion, as any MaxMin network can be constructed by starting with the identity function and stacking affine layers and MaxMin activations (from the last layer to the first).

Case 1: $\boldsymbol{h} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_f}$ is an affine layer. We denote $h_i(\boldsymbol{x}) = \boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x} + b_i$ to be the i-th element of $\boldsymbol{h}$ with weight $\boldsymbol{w}_i$ and bias b_i. Assume $f \circ \boldsymbol{h}$ can interpolate dataset $\mathcal{D}$. We will construct a new MaxMin network $\tilde{f} \circ \tilde{\boldsymbol{h}}$ such that $\tilde{f}$ has the same topology as f, $\tilde{\boldsymbol{h}} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_f}$ is an affine layer with weights $\tilde{\boldsymbol{w}}_i$ and biases $\tilde{b}_i$ ($i \in [d_f]$), and $\tilde{f} \circ \tilde{\boldsymbol{h}}$ interpolates $\mathcal{D}^{\mathrm{F}}$. Note that $\|\boldsymbol{w}_i\|_1 \le 1$ and $\|\tilde{\boldsymbol{w}}_i\|_1 \le 1$ must hold ($\forall i \in [d_f]$).

Given dataset $\mathcal{D}$, denote $\boldsymbol{z}^{(j)} = \boldsymbol{h}(\boldsymbol{x}^{(j)})\ \forall j \in [n]$. Applying $\boldsymbol{h}$ to $\mathcal{D}$ yields another dataset denoted as $\boldsymbol{h}(\mathcal{D}) := \{(\boldsymbol{z}^{(j)}, y^{(j)})\}_{j=1}^n$, which also has a bounded diameter of 1 due to the 1-Lipschitz property of $\boldsymbol{h}$. By the assumption, f interpolates $\boldsymbol{h}(\mathcal{D})$. Therefore, one can use induction to prove that there exists $\tilde{f}$ that interpolates $(\boldsymbol{h}(\mathcal{D}))^{\mathrm{F}}$ where $(\boldsymbol{h}(\mathcal{D}))^{\mathrm{F}}$ is any dataset formatted from $\boldsymbol{h}(\mathcal{D})$.

We now construct the weights and biases of $\tilde{\boldsymbol{h}}$ as follows:

- If the i-th element of $\boldsymbol{h}(\mathcal{D})$ is useless (i.e. $u_i(\boldsymbol{h}(\mathcal{D})) - l_i(\boldsymbol{h}(\mathcal{D})) < 1$), then set $\tilde{\boldsymbol{w}}_i = \boldsymbol{0}$ and $\tilde{b}_i = 0$;
- If the i-th element of $\boldsymbol{h}(\mathcal{D})$ is useful, then there exists two samples $\boldsymbol{x}^{(j)}$ and $\boldsymbol{x}^{(k)}$ satisfying $|z_i^{(j)} - z_i^{(k)}| = 1$. Therefore, $|\boldsymbol{w}_i^{\mathrm{T}}(\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)})| = 1$. Note that $\|\boldsymbol{w}_i\|_1 \le 1$ and $\|\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)}\|_\infty \le 1$. Thus by Hölder's inequality, it must be $\|\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)}\|_\infty = 1$ and $[\boldsymbol{w}_i]_s = 0$ for all s satisfying $|x_s^{(j)} - x_s^{(k)}| < 1$. Pick any s with $[\boldsymbol{w}_i]_s \neq 0$ (thus $|x_s^{(j)} - x_s^{(k)}| = 1$). Choose $\tilde{\boldsymbol{w}}_i = \mathrm{sgn}([\boldsymbol{w}_i]_s)\boldsymbol{e}_s$ (the unit vector with the s-th element being 1 or -1 depending on the sign of $[\boldsymbol{w}_i]_s$) and $\tilde{b}_i = (\boldsymbol{w}_i - \tilde{\boldsymbol{w}}_i)^{\mathrm{T}}\boldsymbol{x}^{(j)} + b_i$.

Similarly, denote $\tilde{\boldsymbol{z}}^{(j)} = \tilde{\boldsymbol{h}}(\boldsymbol{x}^{(j)})\ \forall j \in [n]$ and the the corresponding dataset $\tilde{\boldsymbol{h}}(\mathcal{D}) := \{(\tilde{\boldsymbol{z}}^{(j)}, y^{(j)})\}_{j=1}^n$ (which also has a bounded diameter of 1 due to the 1-Lipschitz property of $\tilde{\boldsymbol{h}}$). First observe that if the i-th element is useless in $\boldsymbol{h}(\mathcal{D})$, then it is also useless in $\tilde{\boldsymbol{h}}(\mathcal{D})$. We now prove that

the converse also holds: if the i-th element is useful in $\boldsymbol{h}(\mathcal{D})$, then it is also useful in $\tilde{\boldsymbol{h}}(\mathcal{D})$. Otherwise, there exists $\boldsymbol{x}^{(j)}$ and $\boldsymbol{x}^{(k)}$ such that $|\boldsymbol{w}_i^{\mathrm{T}}(\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)})| = 1$ but $|\tilde{\boldsymbol{w}}_i^{\mathrm{T}}(\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)})| < 1$. However, in this case, by construction $\tilde{\boldsymbol{w}}_i = \pm \boldsymbol{e}_s$ is the unit vector for some s, implying that $|x_s^{(j)} - x_s^{(k)}| < 1$. Since $[\boldsymbol{w}_i]_s \neq 0$, Hölder's inequality implies that $|\boldsymbol{w}_i^{\mathrm{T}}(\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)})| < 1$, a contradiction.

Therefore, the i-th element is useful in $\boldsymbol{h}(\mathcal{D})$ if and only if it is useful in $\tilde{\boldsymbol{h}}(\mathcal{D})$. Now assume that the i-th element is both useful in $\boldsymbol{h}(\mathcal{D})$ and $\tilde{\boldsymbol{h}}(\mathcal{D})$. We next prove that $l_i(\boldsymbol{h}(\mathcal{D})) = l_i(\tilde{\boldsymbol{h}}(\mathcal{D}))$ and $u_i(\boldsymbol{h}(\mathcal{D})) = u_i(\tilde{\boldsymbol{h}}(\mathcal{D}))$. By the assignment of $\tilde{b}_i$, there exists an $\boldsymbol{x}^{(j)}$ such that $\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}^{(j)} + b_i = \tilde{\boldsymbol{w}}_i^{\mathrm{T}}\boldsymbol{x}^{(j)} + \tilde{b}_i$, namely $z_i^{(j)} = \tilde{z}_i^{(j)}$. Since $z_i^{(j)}$ must at the extreme value, without loss of generality, assume $z_i^{(j)} = l_i(\boldsymbol{h}(\mathcal{D}))$. Based on the above paragraph, for any $\boldsymbol{x}^{(k)}$ such that $|\boldsymbol{w}_i^{\mathrm{T}}(\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)})| = 1$, $|\tilde{\boldsymbol{w}}_i^{\mathrm{T}}(\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(k)})| = 1$. Thus for any $z_i^{(k)} = u_i(\boldsymbol{h}(\mathcal{D}))$, $|z_i^{(j)} - z_i^{(k)}| = |\tilde{z}_i^{(j)} - \tilde{z}_i^{(k)}|$. Also, $|z_i^{(k)} - \tilde{z}_i^{(k)}| < 2$ because by construction $\|\tilde{\boldsymbol{w}} - \boldsymbol{w}\|_1 < 2$. This proves that $\tilde{z}_i^{(j)} = l_i(\tilde{\boldsymbol{h}}(\mathcal{D}))$ and $\tilde{z}_i^{(k)} = u_i(\tilde{\boldsymbol{h}}(\mathcal{D}))$, thus $l_i(\boldsymbol{h}(\mathcal{D})) = l_i(\tilde{\boldsymbol{h}}(\mathcal{D}))$ and $u_i(\boldsymbol{h}(\mathcal{D})) = u_i(\tilde{\boldsymbol{h}}(\mathcal{D}))$. Note that we have actually proved a stronger result: given samples $\boldsymbol{z}^{(k)}$ and $\tilde{\boldsymbol{z}}^{(k)}$, $z_i^{(k)} = l_i(\boldsymbol{h}(\mathcal{D})) \iff \tilde{z}_i^{(k)} = l_i(\boldsymbol{h}(\mathcal{D}))$, and $z_i^{(k)} = u_i(\boldsymbol{h}(\mathcal{D})) \iff \tilde{z}_i^{(k)} = u_i(\boldsymbol{h}(\mathcal{D}))$.

Combining the above results, one can conclude that for a dataset $(\boldsymbol{h}(\mathcal{D}))^{\mathrm{F}}$ formatted from $\boldsymbol{h}(\mathcal{D})$, it is also formatted from $\tilde{\boldsymbol{h}}(\mathcal{D})$. (*)

Now let $\tilde{\boldsymbol{h}}(\mathcal{D}^{\mathrm{F}})$ be the dataset obtained by applying $\tilde{\boldsymbol{h}}$ to the formatted dataset $\mathcal{D}^{\mathrm{F}}$. Note that the weights in $\tilde{\boldsymbol{h}}$ are special in that all $\tilde{\boldsymbol{w}}_i$ are either unit or zero vectors by construction. We will use this property to prove that $\tilde{\boldsymbol{h}}(\mathcal{D}^{\mathrm{F}})$ is just the dataset formatted from $\tilde{\boldsymbol{h}}(\mathcal{D})$. The key observation is that

- If $[\tilde{\boldsymbol{w}}_i]_s = 0$, then masking/thresholding the s-th element in $\mathcal{D}$ does not change the network output after applying $\tilde{h}_i$.
- If $[\tilde{\boldsymbol{w}}_i]_s \neq 0$, then the s-th element must be useful in $\mathcal{D}$ (bacause the i-th element is useful in $\boldsymbol{h}(\mathcal{D})$ and $[\boldsymbol{w}_i]_s \neq 0$). Since $\tilde{\boldsymbol{w}}_i$ is unit ($[\tilde{\boldsymbol{w}}_i]_s = \pm 1$), it is easy to see that thresholding the s-th element in $\mathcal{D}$ does not change the minimum and maximum of the output $\tilde{h}_i$.

Using these results as well as the fact that each input element across $\tilde{\boldsymbol{h}}(\mathcal{D}^{\mathrm{F}})$ can have at most two different values (because the weight of $\tilde{\boldsymbol{h}}$ is either unit or zero and $\mathcal{D}^{\mathrm{F}}$ is formatted), we have that $\tilde{\boldsymbol{h}}(\mathcal{D}^{\mathrm{F}})$ is just the dataset formatted from $\tilde{\boldsymbol{h}}(\mathcal{D})$! Finally, by induction, $(\boldsymbol{h}(\mathcal{D}))^{\mathrm{F}}$ can be interpolated by $\tilde{f}$, which yields that $(\tilde{\boldsymbol{h}}(\mathcal{D}))^{\mathrm{F}}$ can be interpolated by $\tilde{f}$ based on (*). Therefore, $\tilde{\boldsymbol{h}}(\mathcal{D}^{\mathrm{F}})$ can be interpolated by $\tilde{f}$, namely $\tilde{f} \circ \tilde{\boldsymbol{h}}$ interpolates $\mathcal{D}^{\mathrm{F}}$, which concludes the proof of Case 1.

Case 2: $\boldsymbol{h} : \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$ is the MaxMin layer. Here we consider the MaxMin layer with learnable bias, i.e. $\boldsymbol{h}(\boldsymbol{x}) = \mathrm{MaxMin}(\boldsymbol{x} + \boldsymbol{b})$ with parameter $\boldsymbol{b} \in \mathbb{R}^{d_h}$. Note that the bias will be absorbed into the previous linear layer so it does not change the resulting architecture, but it facilities the proof here. Assume $f \circ \boldsymbol{h}$ can interpolate dataset $\mathcal{D}$. We will construct a new MaxMin network $\tilde{f} \circ \tilde{\boldsymbol{h}}$ such that $\tilde{f}$ has the same topology as f, $\tilde{\boldsymbol{h}}$ is the MaxMin layer with bias $\tilde{\boldsymbol{b}}$, and $\tilde{f} \circ \tilde{\boldsymbol{h}}$ interpolates $\mathcal{D}^{\mathrm{F}}$. As in Case 1, we use the same notations of $\boldsymbol{h}(\mathcal{D})$, $\tilde{\boldsymbol{h}}(\mathcal{D})$, $(\boldsymbol{h}(\mathcal{D}))^{\mathrm{F}}$, $\boldsymbol{h}(\mathcal{D}^{\mathrm{F}})$, etc.

Given dataset $\mathcal{D}$, denote $\boldsymbol{z}^{(j)} = \boldsymbol{h}(\boldsymbol{x}^{(j)})\ \forall j \in [n]$. It follows that $z_{2i-1}^{(j)} = \max(x_{2i-1}^{(j)} + b_{2i-1}, x_{2i}^{(j)} + b_{2i})$ and $z_{2i}^{(j)} = \min(x_{2i-1}^{(j)} + b_{2i-1}, x_{2i}^{(j)} + b_{2i})$. We can separately focus on each pair. Consider the following three sub-cases:

- Both the $(2i-1)$-th and the $2i$-th elements in $\boldsymbol{h}(\mathcal{D})$ are useful. In this sub-case, we can prove that both the $(2i-1)$-th and the $2i$-th elements in $\mathcal{D}$ are useful. This is because

$$
\begin{aligned}
2 &= u_{2i-1}(\boldsymbol{h}(\mathcal{D})) - l_{2i-1}(\boldsymbol{h}(\mathcal{D})) + u_{2i}(\boldsymbol{h}(\mathcal{D})) - l_{2i}(\boldsymbol{h}(\mathcal{D})) \\
&= \left(\max_{j\in[n]} z_{2i-1}^{(j)} + \max_{j\in[n]} z_{2i}^{(j)}\right) - \left(\min_{j\in[n]} z_{2i-1}^{(j)} + \min_{j\in[n]} z_{2i}^{(j)}\right) \\
&\leq \left(\max_{j\in[n]} x_{2i-1}^{(j)} + \max_{j\in[n]} x_{2i}^{(j)} + b_{2i-1} + +b_{2i}\right) - \left(\min_{j\in[n]} x_{2i-1}^{(j)} + \min_{j\in[n]} x_{2i}^{(j)} + b_{2i-1} + +b_{2i}\right) \\
&= u_{2i-1}(\mathcal{D}) - l_{2i-1}(\mathcal{D}) + u_{2i}(\mathcal{D}) - l_{2i}(\mathcal{D}) \leq 2.
\end{aligned}
$$

The equality holds when $u_{2i-1}(\mathcal{D}) - l_{2i-1}(\mathcal{D}) = 1$ and $u_{2i}(\mathcal{D}) - l_{2i}(\mathcal{D}) = 1$. Also, it is easy to see that thresholding the $(2i-1)$-th and the $2i$-th elements in $\mathcal{D}$ is equivalent to thresholding them in $\boldsymbol{h}(\mathcal{D})$. Formally, $l_{2i-1}(\boldsymbol{h}(\mathcal{D}^{\mathrm{F}})) = l_{2i-1}(\boldsymbol{h}(\mathcal{D}))$ and $u_{2i-1}(\boldsymbol{h}(\mathcal{D}^{\mathrm{F}})) = u_{2i-1}(\boldsymbol{h}(\mathcal{D}))$ (similar for l_{2i} and u_{2i}). Therefore, we just set $\tilde{b}_{2i-1} = b_{2i-1}$ and $\tilde{b}_{2i} = b_{2i}$.

- Both the $(2i-1)$-th and the $2i$-th elements in $\boldsymbol{h}(\mathcal{D})$ are useless. This sub-case is trivial: these useless elements can be handled in $\tilde{f}$, because the first layer in $\tilde{f}$ is linear and the corresponding weights are zero for the useless input elements (see Case 1).
- One of the $(2i-1)$-th and the $2i$-th elements in $\boldsymbol{h}(\mathcal{D})$ is useful and the other is useless. Without loss of generality, assume the $(2i-1)$-th element in $\boldsymbol{h}(\mathcal{D})$ is useful and the $2i$-th element in $\boldsymbol{h}(\mathcal{D})$ is useless. Then we can prove that either ($l_{2i-1}(\boldsymbol{h}(\mathcal{D})) = l_{2i-1}(\mathcal{D})$ and $u_{2i-1}(\boldsymbol{h}(\mathcal{D})) = u_{2i-1}(\mathcal{D})$) or ($l_{2i-1}(\boldsymbol{h}(\mathcal{D})) = l_{2i}(\mathcal{D})$ and $u_{2i-1}(\boldsymbol{h}(\mathcal{D})) = u_{2i}(\mathcal{D})$). Otherwise, it is easy to see that both the $(2i-1)$-th and the $2i$-th elements in $\mathcal{D}$ are useless and thus the $(2i-1)$-th element in $\boldsymbol{h}(\mathcal{D})$ cannot be useful.

  Again without loss of generality, assume the $l_{2i-1}(\boldsymbol{h}(\mathcal{D})) = l_{2i-1}(\mathcal{D})$ and $u_{2i-1}(\boldsymbol{h}(\mathcal{D})) = u_{2i-1}(\mathcal{D})$. Then by assigning a sufficiently small bias $\tilde{b}_{2i} \to -\infty$ and setting $\tilde{b}_{2i-1} = b_{2i-1}$, one can ensure that even after thresholding/masking $\mathcal{D}$, the following relation still holds: $l_{2i-1}(\boldsymbol{h}(\mathcal{D}^{\mathrm{F}})) = l_{2i-1}(\mathcal{D}^{\mathrm{F}})$ and $u_{2i-1}(\boldsymbol{h}(\mathcal{D}^{\mathrm{F}})) = u_{2i-1}(\mathcal{D}^{\mathrm{F}})$. On the other hand, the $2i$-th element in $\boldsymbol{h}(\mathcal{D})$ is useless so it can be handled in $\tilde{f}$ by the corresponding zero weights in the first layer of $\tilde{f}$ (see Case 1).

Combining the three sub-cases, we have concluded the proof of Case 2. □

Note that for a Boolean dataset $\mathcal{D}$, the formatted version is itself, i.e. $\mathcal{D}^{\mathrm{F}} = \mathcal{D}$. Thus the proof of Lemma B.22 directly leads to the following corollary:

Corollary B.23. *Let $\mathcal{D}$ be a Boolean dataset that can be interpolated by a MaxMin network f. Then there exists a network $\tilde{f}$ with the same architecture (i.e. depth and width) as f, such that:*

- *Denote $\mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ as the weight matrix of the l-th layer of $\tilde{f}$. Then the following holds for all $i \in [d_l]$:*

$$[\mathbf{W}^{(l)}]_{i,:} \in \{s \cdot \boldsymbol{e}_r : s \in \{1, -1\}, r \in [d_{l-1}]\} \cup \{\mathbf{0}\}. \tag{47}$$

- *Denote $f_i^{(l)}(\boldsymbol{x})$ as the i-th neuron output of the l-th layer given input $\boldsymbol{x}$, and denote $\mathcal{S}_i^{(l)} = \{f_i^{(l)}(\boldsymbol{x}) : (\boldsymbol{x}, y) \in \mathcal{D}\}$. Then either $|\mathcal{S}_i^{(l)}| = 1$, or $|\mathcal{S}_i^{(l)}| = 2$ and $\max \mathcal{S}_i^{(l)} - \min \mathcal{S}_i^{(l)} = 1$.*

We are now ready for the proof of Theorem 3.10, which is restated below.

Theorem B.24. *Let M_d be the minimum depth such that an M_d-layer MaxMin network can exactly represent any (discrete) d-dimensional Boolean function. Then $M_d = \Omega(d)$.*

Proof. Let $\tilde{f}$ be an M_d-layer MaxMin network that interpolates the d-dimensional Boolean function, satisfying the condition in Corollary B.23. Using the notations in Corollary B.23, one has

$$f_{2i-1}^{(l)}(\boldsymbol{x}) = \max\left([\mathbf{W}^{(l)}]_{2i-1,:}^{\mathrm{T}} \boldsymbol{f}^{(l-1)}(\boldsymbol{x}) + b_{2i-1}^{(l)}, [\mathbf{W}^{(l)}]_{2i,:}^{\mathrm{T}} \boldsymbol{f}^{(l-1)}(\boldsymbol{x}) + b_{2i}^{(l)}\right) \tag{48}$$

which must have one of the following forms due to the choice of weight $[\mathbf{W}^{(l)}]$:

- $f_{2i-1}^{(l)}(\boldsymbol{x}) = c$, where $c \in \mathbb{R}$ is a constant;
- $f_{2i-1}^{(l)}(\boldsymbol{x}) = \pm f_r^{(l-1)}(\boldsymbol{x}) + c$, where $r \in [d_{l-1}]$ and $c \in \mathbb{R}$ are constants;
- $f_{2i-1}^{(l)}(\boldsymbol{x}) = \max\left(s_1 \cdot f_{r_1}^{(l-1)}(\boldsymbol{x}), s_2 \cdot f_{r_2}^{(l-1)}(\boldsymbol{x})\right) + c$, where $s_1, s_2 \in \{1, -1\}$, $r_1, r_2 \in [d_{l-1}]$ and $c \in \mathbb{R}$ are constants.

In the last case, we further have $\max_{(\boldsymbol{x},y)\in\mathcal{D}} s_1 \cdot f_{r_1}^{(l-1)}(\boldsymbol{x}) = \max_{(\boldsymbol{x},y)\in\mathcal{D}} s_2 \cdot f_{r_2}^{(l-1)}(\boldsymbol{x})$ and $\min_{(\boldsymbol{x},y)\in\mathcal{D}} s_1 \cdot f_{r_1}^{(l-1)}(\boldsymbol{x}) = \min_{(\boldsymbol{x},y)\in\mathcal{D}} s_2 \cdot f_{r_2}^{(l-1)}(\boldsymbol{x})$ due to the second property in Corollary B.23. Therefore, $f_{2i-1}^{(l)}(\boldsymbol{x})$ can calculate a constant, the identity function, or the 2-ary logical OR of *literals*

(up to an additional bias). Similarly, $f_{2i}^{(l)}(\boldsymbol{x})$ can calculate a constant, the identity function, or the logical AND of *literals*.

Therefore, when considering the whole MaxMin network, it is no powerful than a 2-ary Boolean circuit. Here a 2-ary Boolean circuit refers to a directional acyclic computation graph where each internal node connects to at most two nodes by incoming edges and can only calculate the 2-ary logical AND, logical OR, and (unary) logical NOT. Define the depth of a 2-ary Boolean circuit to be the length of the longest path from the input to the output. It then follows that the depth correponds to the depth of the MaxMin netwotk, and it suffices to prove that the depth of a 2-ary Boolean circuit must be $\Omega(d)$ to represent certain Boolean functions of d variables.

Note that for a 2-ary Boolean circuit that has M layers and outputs a scalar, the number of nodes will not exceed $2^{M+1}-1$, where the maximum size is achieved by a complete binary tree. However, the classic result in Boolean circuit theory (Shannon 1942) showed that for most Boolean functions of d variables, a lower bound on the minimum size of 2-ary Boolean circuits is $\Omega(2^d/d)$, which thus yields $M=\Omega(d)$ and concludes the proof. □

C Special SortNet

C.1 GroupSort Network

In this section, we formally prove that any GroupSort network can be exactly represented by a SortNet with the same topological structure. An M-layer GroupSort network with group size G and hidden size Gd can be defined as:

$$\begin{cases} \widetilde{\boldsymbol{z}}^{(l)} = \widetilde{\mathbf{W}}^{(l)}\widetilde{\boldsymbol{x}}^{(l-1)} + \widetilde{\boldsymbol{b}}^{(l)} & l \in [M], \\ \widetilde{x}_{iG+j}^{(l)} = \left(\{\widetilde{z}_{iG+t}^{(l)}\}_{t=1}^{G}\right)_{(j)} & l \in [M-1], i \in \{0,\cdots,d-1\}, j \in [G], \end{cases} \tag{49}$$
$$s.t.\ \|\widetilde{\mathbf{W}}^{(l)}\|_\infty \le 1 \text{ for } l \in [M].$$

In (49), $(\cdot)_{(j)}$ denotes the j-th largest element in a sequence. The GroupSort network takes $\widetilde{\boldsymbol{x}}^{(0)}$ as input and outputs $\widetilde{\boldsymbol{z}}^{(M)}$. We will construct a SortNet such that the output is equal to $\widetilde{\boldsymbol{z}}^{(M)}$.

The key observation is that for any vector $\boldsymbol{z}$ in a bounded domain $\mathbb{K}$, the GroupSort activation with arbitrary group size G can be represented by a sort operation plus a bias term. Concretely, we prove

$$\mathrm{GroupSort}_G(\boldsymbol{z}) = \mathrm{sort}(\boldsymbol{z}+\boldsymbol{b}) + \boldsymbol{c} \tag{50}$$

by assigning

$$b_{iG+j} = -iC, \quad c_{iG+j} = iC \quad \text{for } j \in [G], i \in \{0,\cdots,d-1\} \tag{51}$$

where C is a sufficiently large positive constant. This is because a large C dominates the relative order, so the sorting result is divided into groups of size G. Then $\boldsymbol{b}+\boldsymbol{c}=0$ ensures that the introduced bias is finally eliminated.

Equipped with (50), we are now ready to construct the SortNet. We use the notations in Definition 4.1. Denote vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ as

$$\begin{array}{ll} a_j = -jC & \text{for } j \in [Gd], \\ b_{iG+j} = -iC & \text{for } j \in [G], i \in [d]. \end{array}$$

The parameters of SortNet are assigned below:

$$\begin{array}{ll} \boldsymbol{w}^{(l,k)} = [\widetilde{\mathbf{W}}^{(l)}]_{k,:} \text{ (the } k\text{th row)} & k \in [Gd] \\ \boldsymbol{b}^{(1,k)} = \boldsymbol{a} & k \in [Gd] \\ \boldsymbol{b}^{(2,k)} = \boldsymbol{b} + \widetilde{\boldsymbol{b}}^{(1)} - \widetilde{\mathbf{W}}^{(1)}\boldsymbol{a} & k \in [Gd] \\ \boldsymbol{b}^{(l,k)} = \boldsymbol{b} + \widetilde{\boldsymbol{b}}^{(l-1)} - \widetilde{\mathbf{W}}^{(l-1)}\boldsymbol{b} & k \in [Gd], 3 \le l \le M \\ \boldsymbol{b}^{\text{out}} = \widetilde{\boldsymbol{b}}^{(M)} - \widetilde{\mathbf{W}}^{(M)}\boldsymbol{b} & \end{array}$$

The activation is simply chosen to be the identity function $\sigma(x)=x$. This leads to the following calculation which can be proved straightforwardly:

$$\begin{aligned} \boldsymbol{x}^{(1)} &= \widetilde{\boldsymbol{z}}^{(1)} - \widetilde{\boldsymbol{b}}^{(1)} + \widetilde{\mathbf{W}}^{(1)}\boldsymbol{a} \\ \boldsymbol{x}^{(l)} &= \widetilde{\boldsymbol{z}}^{(l)} - \widetilde{\boldsymbol{b}}^{(l)} + \widetilde{\mathbf{W}}^{(l)}\boldsymbol{b} \quad \text{for } 2 \le l \le M \end{aligned}$$

Finally, the SortNet outputs $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^{(M)} + \boldsymbol{b}^{\text{out}} = \widetilde{\boldsymbol{z}}^{(M)}$, which matches the output of the GroupSort network. Proof finishes.

Remark C.1. In the above construction, the same value is assigned for all biases $\boldsymbol{b}^{(l,k)}$ of each SortNet layer. It implies that SortNet is more powerful than GroupSort, since SortNet allows diverse biases for different neurons in the same layer. This is very useful because the value of bias plays an important role in controlling the sorting behavior. For example, in a SortNet neuron (7), setting $b_i^{(l,k)}$ to a sufficiently large value will dominant $x_i^{(l-1)}$ to rank the first, which can extract the i-th coordinate of the input vector if $\boldsymbol{w}^{(l,k)} = \boldsymbol{e}_1$. Similarly, setting $b_i^{(l,k)}$ to a large negative value will cause $x_i^{(l-1)}$ to rank the last, and when combined with $w_i^{(l,k)} = 0$, will preclude $x_i^{(l-1)}$ in the computation of the neuron output. However, for a GroupSort layer, performing these operations will affect the output of other neurons as all neurons in the group share the same bias vector and sorting operation. The interference between neurons may lead to some neurons being wasted and not being able to express the required operations, which is undesirable. In contrast, SortNet does not face the above problems.

C.2 ℓ_∞-distance Net

In this section we formally prove that any ℓ_∞-distance net can be exactly represented by a special SortNet with the same topological architecture. An M-layer ℓ_∞-distance net with a hidden size of d can be formally defined as:

$$\widetilde{x}_k^{(l)} = \|\widetilde{\boldsymbol{x}}^{(l-1)} - \widetilde{\boldsymbol{w}}^{(l,k)}\|_\infty + \widetilde{b}_k^{(l)}, \quad l \in [M], k \in [d]. \tag{52}$$

The network takes $\widetilde{\boldsymbol{x}}^{(0)}$ as input and outputs $\widetilde{\boldsymbol{x}}^{(M)}$. We will construct a SortNet such that the output exactly matches $\widetilde{\boldsymbol{x}}^{(M)}$.

The key observation is that for any input vector $\boldsymbol{z}$ and parameter $\boldsymbol{w}$,

$$\|\boldsymbol{z} - \boldsymbol{w}\|_\infty = \max_i |z_i - w_i| = \boldsymbol{e}_1^{\mathrm{T}} \operatorname{sort}(|\boldsymbol{z} - \boldsymbol{w}|) \tag{53}$$

where $\boldsymbol{e}_1$ is the unit vector with the first element being one. Based on the above equation, we now construct a SortNet as follows. We use the notations in Definition 4.1 and set all weights $\boldsymbol{w}^{(l,k)} = \boldsymbol{e}_1$, set biases

$$\begin{aligned}
&\boldsymbol{b}^{(1,k)} = -\widetilde{\boldsymbol{w}}^{(1,k)}, \\
&\boldsymbol{b}^{(l,k)} = \widetilde{\boldsymbol{b}}^{(l-1)} - \widetilde{\boldsymbol{w}}^{(l,k)} \quad \text{for } 2 \le l \le M, \\
&\boldsymbol{b}^{\text{out}} = \widetilde{\boldsymbol{b}}^{(M)},
\end{aligned}$$

and set the activation $\sigma(x) = |x|$. Then by a simple induction over layer l we can prove

$$\boldsymbol{x}^{(l)} = \widetilde{\boldsymbol{x}}^{(l)} - \widetilde{\boldsymbol{b}}^{(l)}. \quad \forall\, l \in [M] \tag{54}$$

Finally, the SortNet outputs $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^{(M)} + \boldsymbol{b}^{\text{out}} = \widetilde{\boldsymbol{x}}^{(M)}$, which matches the output of the ℓ_∞-distance net. Proof finishes.

D An Efficient GPU Implementation of SortNet

In this section we describe an efficient GPU implementation of Section 4 for training and inference, which is used in this paper. Some basic tensor operations are described in the context of Pytorch framework.

D.1 Training

Consider a fully-connected SortNet layer $\boldsymbol{f}$ with parameters $\mathbf{W}, \mathbf{B} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, and $W_{ij} = (1-\rho)\rho^{j-1}$. For an input batch $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{in}}}$ consisting of n samples, denote the output in a training iteration as $\mathbf{Z} = \boldsymbol{f}(\mathbf{X}) \in \mathbb{R}^{n \times d_{\text{out}}}$ which is stochastic and depends on the mask. The generated stochastic mask is also a matrix denoted as $\mathbf{S} \in \{0,1\}^{n \times d_{\text{in}}}$. According to the formula (8),

$$Z_{ik} = \max_{j \in [d_{\text{in}}]} S_{ij} \sigma(X_{ij} + B_{kj}) \tag{55}$$

Note that S_{ij} can only be 0 or 1, therefore the multiplication between S_{ij} and $\sigma(X_{ij} - B_{kj})$ is unnecessary. We can enumerate the indices such that $S_{ij} = 1$, resulting in the following calculation:

$$Z_{ik} = \max_{j,\ \text{s.t. } S_{ij}=1} \sigma(X_{ij} + B_{kj}) \tag{56}$$

With hyper-parameter ρ, (56) will give a computational complexity of $\Theta((1-\rho)nd_{\text{in}}d_{\text{out}})$.

In fact, the above computation can be further accelerated if two adjacent SortNet layers are cascaded. In this case, the output of the first layer becomes the input of the second layer, which will also be dropped out, so the unused neurons does not need to be computed. In other words, it suffices to only calculate the output neurons which are involved in the next layer of computation. This results in a computational complexity of $\Theta((1-\rho)^2nd_{\text{in}}d_{\text{out}})$.

Things become more complicated when it turns to a GPU implementation. Since GPU is highly parallel, it is best to share the same instructions for different samples in a batch for ease of parallelization. However, the number of ones in the mask may vary for different samples. In other words, in (56) enumerating the indices j subject to $S_{ij} = 1$ cannot be efficiently executed in a parallel way.

To address the problem, we introduce a notion called sub-batch. In a sub-batch, all samples have the same mask for ease of parallelization, and different sub-batches have different masks. Typically, a sub-batch contains 32 samples (due to the GPU underlying architecture). A batch size of 512 then contains 16 sub-batches. For each sub-batch, we first generate two index sequences $\boldsymbol{s}^{\text{in}} \in [d_{\text{in}}]^{d_1}$ and $\boldsymbol{s}^{\text{out}} \in [d_{\text{out}}]^{d_2}$ (the input mask and output mask respectively). We then fetch the input neurons using mask $\boldsymbol{s}^{\text{in}}$, which corresponds to a gather operation (e.g. `torch.gather` in Pytorch). Denote the gathered input as $\widetilde{\mathbf{X}} \in \mathbb{R}^{32\times d_1}$. Then the output $\widetilde{\mathbf{Z}} \in \mathbb{R}^{32\times d_2}$ of this sub-batch can be calculated by

$$\widetilde{Z}_{ik} = \max_{j\in[d_1]} \sigma(\widetilde{X}_{ij} + B_{s_k^{\text{out}}, s_j^{\text{in}}}) \quad i \in [32], k \in [d_2]. \tag{57}$$

We finally point out that the introduced sub-batch is only used to accelerate GPU implementation. If one does not care about the speed, simply using (56) for training leads to almost the same test performance.

D.2 Inference

For inference, the sorting operations must be calculated exactly. To speed up the computation, note that the weight $\boldsymbol{w}$ in Proposition 4.4 is exponentially decayed, implying that a lot of elements are close to zero. This leads to the following results:

Proposition D.1. *Let $\boldsymbol{w} \in \mathbb{R}^d$ be a vector with $w_i = (1-\rho)\rho^{i-1}$. Then for any vector $\boldsymbol{x} \in \mathbb{R}_+^d$ with non-negative elements and any integer $k > 0$,*

$$0 \le \boldsymbol{w}^{\mathrm{T}} \operatorname{sort}(\boldsymbol{x}) - \sum_{i=1}^{k} w_i x_{(i)} \le \rho^k \|\boldsymbol{x}\|_\infty. \tag{58}$$

The proof of Proposition D.1 is trivial. Based on the above proposition, picking a k much smaller than d already suffices to control the error below the numerical precision. In this way, only a top-k operation is needed. We choose $k = 10$ for all experiments. Also note that zeroing out some elements of a weight vector does not contradict the Lipschitz property of the sort neuron, and thus margin-based certification still gives a correct certified accuracy.

E Training Details

In this section we provide complete train details for readers to reproduce our results. Our experiments are implemented using the Pytorch framework.

E.1 Datasets

We consider four benchmark datasets: MNIST, CIFAR-10, TinyImageNet and ImageNet (64×64). The statistics of these datasets are listed below, including the number of classes, the dataset size, and the image size.

Table 4: Datasets used in this paper.

Dataset	# Classes	# Images (train)	# Images (test)	Image size
MNIST	10	60K	10K	28×28
CIFAR-10	10	50K	10K	32×32
TinyImageNet	200	100K	10K	64×64
ImageNet	1000	1.28M	50K	64×64

For all datasets, we pre-process each training image using the same pipeline as previous works. Concretely, we use random horizontal flip (except for MNIST) and random crop as the data augmentation. Each image is padded to a larger size with `padding_mode=edge` and then randomly crop to the original size. The number of pixels padded is 1,3,5,4 for MNIST, CIFAR-10, TinyImageNet, and ImageNet, respectively. Finally, the images are normalized to have zero-mean and unit variance across the dataset. For testing, no data augmentation is performed.

E.2 Models

This paper considers 3 types of models in total. The first is a simple fully-connected SortNet. The second is the combination of a Lipschitz SortNet backbone and a non-Lipschitz two-layer perceptron. In this way SortNet serves as a robust feature extractor and the top perceptron is used for classification. The last is a larger model similar to the second model, except we try to use a partially convolutional SortNet backbone for better performance. Details of these models are presented in Table 5. For the ImageNet dataset, we increase the number of hidden neurons of the top perceptron to 2048, since the number of classes is 1000 (larger than the original hidden size which is only 512).

Table 5: Network architectures used in this paper. In the table, SortFC(n) denotes a fully-connected SortNet layer with n output neurons, and SortConv(n, ker=k) denotes a convolutional SortNet layer with n output channels and a kernel size of k. Padding is not used. We use s=s to denote strided convolutions with step s, and by default s=1. FC(n) is the fully-connected linear layer with n output neurons. K denotes the number of classes.

SortNet	SortNet+MLP	SortNet+MLP (2x)
	SortFC(5120)	SortConv(5120,ker=60,s=4)
SortFC(5120)	SortFC(5120)	SortConv(5120, ker=1)
SortFC(5120)	SortFC(5120)	SortConv(5120, ker=1)
SortFC(5120)	SortFC(5120)	SortConv(5120, ker=1)
SortFC(5120)	SortFC(5120)	SortConv(5120, ker=1)
SortFC(5120)	FC(512, bias=True)	FC(512, bias=True)
SortFC(K)	Tanh	Tanh
	FC(K, bias=True)	FC(K, bias=True)

We also list information of the model size in Table 6. Roughly speaking, SortNet and SortNet+MLP have similar computational costs, and SortNet+MLP (2x) is about 4 times larger than SortNet+MLP. We also point out that the models used in this paper have exactly the same topology and number of parameters as ℓ_∞-distance Net (or ℓ_∞-distance Net+MLP) in [73], which enables a fair comparison between our approach and theirs.

E.3 Loss function

Let $(\boldsymbol{x}, y)$ be a training sample and $\boldsymbol{f}$ be the mapping represented by the network. For a SortNet model, since it is strictly 1-Lipschitz, we can directly apply hinge loss to increase the margin of output logits (see Proposition 3.1). To make the training more effective, we adopt the following loss function:

$$\ell(\boldsymbol{f}, \boldsymbol{x}, y) = \lambda \cdot \ell_{\text{CE}}(s \cdot \boldsymbol{f}(\boldsymbol{x}), y) + \mathbb{I}\left(\arg\max_i [\boldsymbol{f}(\boldsymbol{x})]_i = y\right) \ell_{\text{hinge}}(\boldsymbol{f}(\boldsymbol{x})/\theta, y) \tag{59}$$

where θ is the hinge threshold hyper-parameter, s is a learnable scalar, λ is the mixing hyper-parameter, $\mathbb{I}(\cdot)$ is the indicator function, and ℓ_{CE} and ℓ_{hinge} are the cross-entropy loss and standard hinge loss,

Table 6: Statistics of model size on different datasets. In the table, "FLOPs" measures the number of floating-point operations in forward propagation, and "# Neurons" denotes the number of hidden neurons.

	Model	CIFAR -10	Tiny ImageNet	ImageNet (64×64)
FLOPs	SortNet	121M	–	–
	SortNet+MLP	123M	171M	180M
	SortNet+MLP (2x)	–	651M	684M
# Parameters	SortNet	121M	–	–
	SortNet+MLP	123M	170M	180M
	SortNet+MLP (2x)	–	171M	204M
# Neurons	SortNet	25.6K	–	–
	SortNet+MLP	26.1K	26.1K	27.6K
	SortNet+MLP (2x)	–	103K	104K

respectively:

$$\ell_{\text{CE}}(\boldsymbol{z}, y) = \log\left(\sum_i \exp(z_i)\right) - z_y, \tag{60}$$

$$\ell_{\text{hinge}}(\boldsymbol{z}, y) = \max\{\max_{i \neq y} z_i - z_y + 1, 0\}. \tag{61}$$

Intuitively speaking, this loss first uses cross entropy to improve the clean accuracy, then boost the certified robustness for samples that have been correctly classified. Such a loss design is quite common and is also used in prior works [55, 74, 14]. The coefficient λ balances the two loss terms and controls the trade-off between accuracy and robustness. We find good results can be achieved by starting with an initial value λ_0 and slowly reducing it throughout training until it approaches zero (similar to [75]).

Now consider the composite SortNet+MLP model. The whole model is non-Lipschitz, so we have to use relaxation-based methods to calculate a margin vector for the top MLP. We adopt the simplest method called *interval bound propagation* (IBP) [21], similar to [73]. After obtaining the margin vector, we use the following loss function

$$\ell(\boldsymbol{f}, \boldsymbol{x}, y) = \lambda \cdot \ell_{\text{CE}}(s \cdot \boldsymbol{f}(\boldsymbol{x}), y) + \mathbb{I}\left(\arg\max_i [\boldsymbol{f}(\boldsymbol{x})]_i = y\right) \ell_{\text{hinge}}(\text{IBP}(\boldsymbol{f}, \boldsymbol{x}, y, \epsilon), y) \tag{62}$$

which is almost the same as (59). Here $\text{IBP}(\boldsymbol{f}, \boldsymbol{x}, y, \epsilon)$ denotes the margin calculated by interval bound propagation for sample $(\boldsymbol{x}, y)$ and network $\boldsymbol{f}$ under perturbation radius ϵ. Similarly, the coefficient λ decays from an initial value λ_0 to zero throughout training, and the value ϵ follows a warmup schedule in the training process. Such a strategy is broadly applied in previous literature [21, 75, 69]. See Appendix E.5 for details of the λ and ϵ schedule.

E.4 Training strategy

Parameter initialization. All parameters (in both the SortNet and the top MLP) are initialized using the standard Gaussian distribution. The learnable scalar in the loss function is initialized to be 1. To keep the output scale of the linear layer the same as the input scale, the output of each linear neuron is further multiplied by $1/\sqrt{d}$, where d is the input dimension. Such an initialization strategy is often called the NTK initialization [26].

Training pipeline. At each iteration, we first generate masks in each *hidden* layer and remove neurons with zero masks, resulting in a sub-network. This is similar to applying dropout in each hidden layer with a dropout rate of ρ (we do not dropout the input). We then perform forward and backward propagation on the sub-network (corresponding to an ℓ_∞-distance net) following Section D.1 and calculate the gradients of parameters. We use two additional tricks that is adopted in [73]: batch normalization and ℓ_p-relaxation.

Batch normalization. Since the output of the sort neuron (7) is always non-negative due to the absolute-value activation function, a mean-shift version of batch normalization [25] is applied after each SortNet layers. Concretely, it normalizes each hidden neuron to have zero mean for a batch of

samples in each iteration. We do not use scaling since it destroys the Lipschitz property. Unlike the original batch normalization, we do not calculate the running mean statistics in our training procedure since the model is also randomly sampled (with hidden neurons dropped out). The running mean can be calculated later after the training finishes, by conducting one pass of forward propagation on the whole training dataset for the inference model.

ℓ_p-relaxation. The calculation of maximum in (8) leads to sparse gradient for input $\boldsymbol{x}$, which makes optimization difficult. To alleviate the problem, we adopt ℓ_p-relaxation proposed in [73] for training, which gives a smooth approximation of the maximum operation. Concretely, for a vector $\boldsymbol{x}$ with non-negative elements, the following holds:

$$\max_i x_i = \lim_{p \to +\infty} \left(\sum_i x_i^p\right)^{1/p}. \tag{63}$$

At the initial phase, p is set to a small value so that the gradients are non-sparse. As training progresses, p gradually increases until reaching a large number. For the last few epochs, p is set to infinity. In this paper, we exponentially increase p from 8 to 1000 without tuning for all experiments, following [73].

E.5 Details and hyper-parameters

All the experiments regarding SortNet models are run on NVIDIA Tesla V100S-PCIe GPUs. As for speed comparison to prior works, we further test time metrics for all prior works as well as our approach on the latest NVIDIA RTX-3090 GPUs (24GB memory). The CUDA version is 11.1.

Hyper-parameters of the optimizer. Following [73, 74], we adopt the Adam optimizer with hyper-parameters $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-10}$, and use a batch size of 512 for all datasets and experiments. The learning rate is initialized to be 0.02 and is decayed using a simple cosine annealing throughout the whole training process. The only exception is for ImageNet, where we use 2 GPUs in parallel due to the large dataset size. This results in a total batch size of 1024, and we change the initial learning rate to be 0.01. In all experiments, the weight decay is set to 0.02 for all but SortNet parameters (e.g., for the top MLP), and we do not use weight decay for SortNet parameters.

ϵ Schedule. For the SortNet+MLP model, a warmup over ϵ is required due to the use of interval bound propagation, and we adopt the same warmup schedule as in [69] (with the same hyper-parameters), which increases ϵ_{train} first-exponentially-then-linearly from 0 to $1.1\epsilon_{\text{test}}$.

Epoch-related hyperparameters. Such hyper-parameters are listed below, which depends on the dataset.

- **MNIST**. We train all models for 1500 epochs. The ℓ_p schedule starts at the 100th epoch and ends at the 1450th epoch.
- **CIFAR-10**. We train all models for 3000 epochs. The ℓ_p schedule starts at the 200th epoch and ends at 2950th epoch. The warmup of ϵ is in the first 500 epochs.
- **TinyImageNet**. We train all models for 1000 epochs. The ℓ_p schedule starts at the 100th epoch and ends at the 980th epoch. The warmup of ϵ is in the first 500 epochs.
- **ImageNet** (64×64). We train all models for 300 epochs. The ℓ_p schedule starts at the 50th epoch and ends at the 290th epoch. The warmup of ϵ is in the first 200 epochs.

Hyper-parameters θ, ρ and λ. For the SortNet model, there is a hinge threshold hyper-parameter θ that has to be tuned, and we choose θ using a course grid search. Concertely, we pick $\theta = 0.6$ for $\epsilon = 0.1$ and $\theta = 0.9$ for $\epsilon = 0.3$ on MNIST, and pick $\theta = 16/255$ for $\epsilon = 2/255$ and $\theta = 48/255$ for $\epsilon = 8/255$ on CIFAR-10. As for hyper-parameter ρ, we find $\rho = 0.3$ typically works well on most settings, while a slight adjustment may achieve the best performance. On MNIST, we pick $\rho = 0.3$ for $\epsilon = 0.1$ and $\rho = 0.25$ for $\epsilon = 0.3$. On CIFAR-10, we pick $\rho = 0.25$ for $\epsilon = 2/255$ and $\rho = 0.15$ for $\epsilon = 8/255$ in SortNet, and pick $\rho = 0.3$ for $\epsilon = 2/255$ and $\rho = 0.4$ for $\epsilon = 8/255$ in SortNet+MLP. On Tiny-ImageNet and ImageNet 64×64, we do not tune the hyper-parameter ρ and simply pick the value 0.3. Sensitivity analysis of hyper-parameter ρ is shown in Section F.

Finally, as for hyper-parameter λ in loss (59), we find choosing its value proportional to $1/\epsilon$ works well in all experiments. Such a choice is quite reasonable in balancing the cross-entropy term and the hinge term in the loss (59) because the magnitude of the gradient of the hinge term $\ell_{\text{hinge}}(\boldsymbol{f}(\boldsymbol{x})/\theta, y)$ is $\mathcal{O}(1/\theta)$ which is (roughly) inversely proportional to ϵ. To make the value more regular, we just pick λ_0 from the set $\{0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1.0\}$, while maintaining the property of being

inversely proportional to ϵ. Throughout training, λ is decayed from λ_0 to a vanishing small value $0.01\lambda_0$ exponentially, and is performed simultaneously with the ℓ_p relaxation process. The concrete values in different settings are listed below.

Table 7: The value of hyper-parameter λ in loss (59) used in this paper.

Setting	λ_0	λ_{end}
MNIST ($\epsilon = 0.1$)	0.1	0.001
MNIST ($\epsilon = 0.3$)	0.02	0.0002
CIFAR-10 ($\epsilon = 2/255$)	1.0	0.01
CIFAR-10 ($\epsilon = 8/255$)	0.2	0.002

As for λ in loss (62), we find the value do not depend on ϵ, and in all settings we simply pick $\lambda_0 = 1.0$ and $\lambda_{\text{end}} = 0.01$ without tuning.

E.6 Reproducing baseline methods

In this paper we also reproduce baseline methods by measuring the wall-clock time in training and certification. For all methods in Tables 1, 2 and 3, we test the computational cost on a single NVIDIA RTX-3090 GPU. We use the official Github codes and commands for each method whenever possible. In some situations, the batch size used in the original command may be too large to fit the 24GB GPU memory, so we reduce the batch size to match the memory capacity.

- **IBP** [21]. We use the official code of `CROWN-IBP` on CIFAR-10 dataset since the original code may not be reproducible according to [75]. Note that the corresponding numbers in Table 2 is also obtained from the CROWN-IBP paper. We use the official code of `auto_LiRPA` on TinyImageNet and ImageNet (64×64) datasets. The corresponding numbers of TinyImageNet dataset in Table 3 are obtained from [69]. Due to limited GPU memory, for TinyImageNet and ImageNet (64×64), we reduce the batch size to 64.
- **CROWN-IBP** [75]. We use the official code of `CROWN-IBP` on CIFAR-10 dataset. Due to limited GPU memory, we reduce the batch size to 256. The training speed is measured in the ϵ-warmup phase, and the certification speed is measured at $\epsilon = 2/255$ where both IBP and linear relaxation is used in certification.
- **CROWN-IBP** [69]. We use the official code of `auto_LiRPA` on CIFAR-10, TinyImageNet, and ImageNet (64×64) datasets. Due to limited GPU memory, we reduce the batch size to 256 on CIFAR-10 and 32 on TinyImageNet and ImageNet. The training speed is measured in the ϵ-warmup phase, and the certification is the simple IBP for these settings.
- **IBP** [52]. We use the official code of `Fast-Certified-Robust-Training`. Due to limited GPU memory, for TinyImageNet dataset we reduce the batch size to 64.
- **CAP** [66]. We use the official code of `convex_adversarial`.
- **COLT** [3]. We use the official code of `COLT`. The training has 4 stages, and the per-epoch training time (seconds) for each stage is 51, 240, 332, 385, respectively. Therefore the average time listed in Table 2 is 252.0 seconds.
- **ℓ_∞-distance Net** [73, 74]. We use the official code of `L_inf-dist-net-v2`. The training speed is measured in the ℓ_p-relaxation phase which is the slowest.

We also reproduce 100-PGD attack for the ℓ_∞-distance net in the original paper [73], because their paper only reported the robust accuracy under the (*weaker*) 20-step PGD attack. Furthermore, we report two additional settings: $\epsilon = 0.1$ on MNIST and $\epsilon = 2/255$ on CIFAR-10 based on their Github repo `L_inf-dist-net`, which are not presented in [73].

We also try to reproduce the result of ℓ_∞-distance net on TinyImageNet using the training approach proposed by [74]. We try our best to grid search over several hyper-parameters, including λ_0, λ_{end} and θ, while keeping other hyper-parameters the same as in [74]. The values are chosen from $\lambda_0 \in \{0.05, 0.1, 0.2, 0.5, 1.0\}$, $\lambda_{\text{end}} \in \{0.0005, 0.001, 0.002, 0.005, 0.01\}$, and $\theta \in \{12/255, 16/255, 20/255, 24/255\}$. We use 1000 training epochs, and the ℓ_p-relaxation starts at the 20th epoch. Results are present in Table 3, where we can only achieve 11.04% certified accuracy, which is much lower than other methods. We hypothesis that their approach may not suit for large-scale datasets with a huge number of classes.

F Ablation Studies

We finally make a further investigation of the performance of SortNet models by varying the value of ρ and comparing it with ℓ_∞-distance net (corresponding to $\rho = 0$), using the training approach proposed in Appendix E. Figure 1 presents the performance of trained models, where we plot both clean accuracy and certified accuracy w.r.t. different choices of hyper-parameters ρ ranging from 0 to 0.5. It can be seen that for all four cases in Figure 1, a mid-range value consistently gives the best results. In particular, choosing $\rho = 0.3$ typically attains the peak certified accuracy, and the improvement is most significant when comparing to the the extreme case of $\rho = 0$ (often more than 5 points). Considering that the training in each figure is under the same configuration and hyper-parameters, the improvements justify that SortNet is a better Lipschitz model than ℓ_∞-distance net for certified ℓ_∞ robustness, due to the introduced full order statistics.

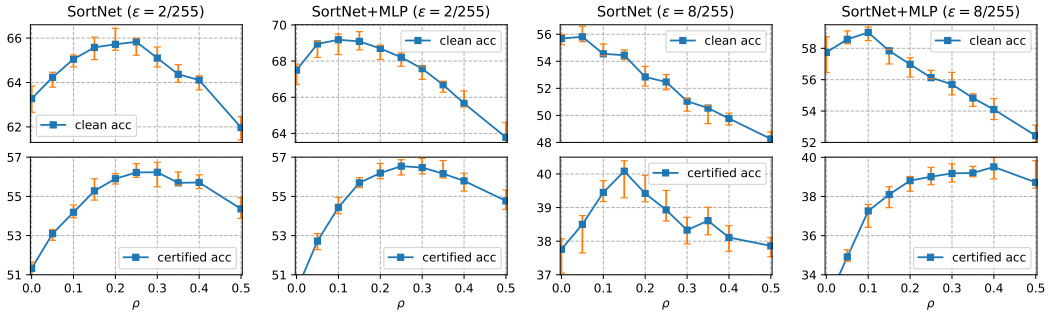

Figure 1: Performance of SortNet/SortNet+MLP trained with different hyper-paramters ρ on CIFAR-10 dataset. For each ρ we independently run 8 experiments and show the median value (blue square) as well as the best/worst performance (orange bar).

G Randomized Smoothing

Different from other approaches in Table 2, randomized smoothing is a *probabilistic* method that provides certified guarantees mainly for ℓ_2 perturbations. To apply these methods in the ℓ_∞ perturbation case, a conversion of perturbation radius is performed by using norm inequalities. Specifically, to certify the robustness under ℓ_∞ perturbations with radius ϵ, one can certify the robustness under ℓ_2 with radius $\epsilon\sqrt{d}$ where d is the input dimension. This clearly produces a lower bound estimate of the certified accuracy.

In the following table, we present the best-known results on MNIST and CIFAR-10. See [49, 5, 27] as well as [74, Appendix D] for references. In summary, randomized smoothing is competitive for the $\epsilon = 2/255$ case on CIFAR-10. However, when ϵ is relatively large, randomized smoothing cannot achieve non-trivial performance.

Table 8: Results of randomized smoothing in different settings.

Dataset	ϵ	Clean	Certified	Reference
MNIST	0.1	≈92	≈12	[27]
	0.3	–	10.0	–
CIFAR-10	2/255	78.8	62.6	[5]
	8/255	52.3	25.2	[27]

H Full Results

As is pointed out above, for all SortNet models we run 8 set of experiments independently and report the median of the accuracy. In this section, we release full results of clean accuracy and certified accuracy in Tables 9 and 10. The best achieved numbers are indicated in boldface.

Table 9: Full results of clean accuracy over 8 independent runs, sorted in descending order.

Dataset	ϵ	Model	Clean								Median
MNIST	0.1	SortNet	**99.08**	99.07	99.04	99.01	99.01	98.97	98.95	98.94	99.01
	0.3	SortNet	**98.68**	98.67	98.59	98.59	98.59	98.57	98.53	98.46	98.59
CIFAR-10	2/255	SortNet	**66.00**	65.98	65.96	65.86	65.79	65.78	65.69	65.23	65.83
		SortNet+MLP	**67.72**	67.72	67.64	67.59	67.57	67.11	66.99	66.99	67.58
	8/255	SortNet	**54.84**	54.77	54.70	54.59	54.30	54.30	54.28	54.12	54.45
		SortNet+MLP	**54.80**	54.39	54.33	54.13	54.07	54.02	53.71	53.46	54.10
Tiny ImageNet	1/255	SortNet+MLP	**24.17**	24.07	24.01	23.97	23.95	23.85	23.75	23.52	23.96
		SortNet+MLP(2x)	**25.77**	25.69	25.55	25.43	25.33	25.30	25.26	25.02	25.38
ImageNet 64×64	1/255	SortNet+MLP	**13.48**	13.37	13.35	13.33	13.30	13.26	13.22	13.21	13.32
		SortNet+MLP(2x)	**14.96**	14.87	14.86	14.84	14.82	14.79	14.76	14.71	14.83

Table 10: Full results of certified accuracy over 8 independent runs, sorted in descending order.

Dataset	ϵ	Model	Certified								Median
MNIST	0.1	SortNet	**98.14**	98.13	98.11	98.07	98.06	98.05	98.04	97.93	98.07
	0.3	SortNet	**93.40**	93.40	93.39	93.39	93.36	93.25	93.22	93.19	93.38
CIFAR-10	2/255	SortNet	**56.67**	56.65	56.29	56.23	56.20	56.07	56.06	55.97	56.22
		SortNet+MLP	**56.94**	56.80	56.70	56.63	56.30	56.25	56.16	56.14	56.47
	8/255	SortNet	**40.39**	40.14	40.13	40.10	40.05	39.96	39.61	39.29	40.08
		SortNet+MLP	**39.99**	39.76	39.70	39.56	39.45	39.40	39.25	38.90	39.51
Tiny ImageNet	1/255	SortNet+MLP	**17.92**	17.52	17.51	17.48	17.43	17.33	17.25	17.21	17.46
		SortNet+MLP(2x)	**18.18**	17.74	17.68	17.64	17.64	17.62	17.59	17.50	17.64
ImageNet 64×64	1/255	SortNet+MLP	**9.02**	9.00	8.97	8.93	8.92	8.89	8.89	8.89	8.93
		SortNet+MLP(2x)	**9.54**	9.45	9.43	9.41	9.41	9.41	9.35	9.35	9.41

I Training SortNet with Shorter Epochs

In our main results, we use quite long training epochs in order to achieve the best performance. For example, the number of epochs is 3000 on CIFAR-10, which is larger than several prior works, in particular, the work of [74]. In this section, we consider reducing the training budget so that the total training time is less than [74]. A direct calculation shows that using 1800 epochs, the total training cost of SortNet will be less than [74]. So we simply change the number of epochs to 1800, *without turning any other hyper-parameters*. Results show that SortNet can achieve 56.05% certified accuracy for $\epsilon = 2/255$, which is still significantly higher than [74] (54.12% certified accuracy). For the $\epsilon = 8/255$ case, SortNet can achieve 39.81% certified accuracy, which is slightly lower than [74] (with a gap of -0.25%). However, note that we do not use several training tricks, such as the special initialization strategy or the special weight decay adopted in [74, 73]. On the other hand, we find that increasing the training budget of [74] to 3000 epochs does not help or even leads to a worse certified accuracy possibly due to overfitting.