Zonotope Domains for Lagrangian Neural Network Verification

Matt Jordan[*]
UT Austin
mjordan@cs.utexas.edu

Jonathan Hayase[*]
University of Washington
jhayase@cs.washington.edu

Alexandros G. Dimakis
UT Austin
dimakis@austin.utexas.edu

Sewoong Oh
University of Washington
sewoong@cs.washington.edu

Abstract

Neural network verification aims to provide provable bounds for the output of a neural network for a given input range. Notable prior works in this domain have either generated bounds using abstract domains, which preserve some dependency between intermediate neurons in the network; or framed verification as an optimization problem and solved a relaxation using Lagrangian methods. A key drawback of the latter technique is that each neuron is treated independently, thereby ignoring important neuron interactions. We provide an approach that merges these two threads and uses zonotopes within a Lagrangian decomposition. Crucially, we can decompose the problem of verifying a deep neural network into the verification of many 2-layer neural networks. While each of these problems is provably hard, we provide efficient relaxation methods that are amenable to efficient dual ascent procedures. Our technique yields bounds that improve upon both linear programming and Lagrangian-based verification techniques in both time and bound tightness.

1 Introduction

With the growing prevalence of machine learning in real-world applications, the brittleness of deep learning systems poses an even greater threat. It is well-known that deep neural networks are vulnerable to adversarial examples, where a minor change in the input to a network can cause a major change in the output [1]. There is a long history of defense techniques being proposed to improve the robustness of a network, only to be completely broken shortly thereafter [2]. This has inspired researchers to focus instead on *neural network verification*, which, for a given network and a range of input, aims to verify whether a certain property is satisfied for every input in that range. A typical adversarial attack seeks to minimize the output of a scalar-valued network subject to certain input constraints. Verification provides lower bounds on the minimum output value achievable by such an adversary. Concretely, for a scalar-valued network f and input range $\mathcal{X}$, verification provides lower bounds to the problem $\min_{x \in \mathcal{X}} f(x)$.

As observed in [3], prior works in verification can be broadly categorized into primal and dual views. In the primal view, convex relaxations are applied to attain, for every intermediate layer of the network, a convex superset of the true attainable range. For example, if $f = f_L \circ \cdots \circ f_1$, convex sets $\mathcal{Z}_k$ are obtained such that $\{f_k \circ \cdots \circ f_1(x) \mid x \in \mathcal{X}\} \subseteq \mathcal{Z}_k$ for $k \in \{1, \ldots, L\}$.

[*]Equal contribution
[†]Github Repo: `https://github.com/revbucket/dual-verification`

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

Under the dual lens, verification is treated as a stagewise optimization problem and Lagrangian relaxation is applied to yield a dual function that always provides valid lower bounds. Often, this dual function is decomposable into a sum of minimization problems which are efficiently computable. One hallmark of all existing dual verification techniques is that intermediate bounds $\mathcal{Z}_k$ are required. In this case, either an efficient primal verification algorithm must first be applied or, for example, the dual verifier can be run iteratively on each neuron to provide upper and lower bounds for each intermediate layer.

Our approach is an attempt to combine the primal and dual threads of verification. We first apply a primal verification algorithm that generates bounds on the attainable range of each intermediate layer. To do this we leverage zonotopes, a more expressive class of polytopes than axis-aligned hyperboxes. Notably, we offer an improvement to existing zonotope bounds when we are also provided with incomparable hyperbox bounds. These zonotopic intermediate bounds are then applied to a dual verification framework, for which dual ascent can be performed. Our formulation may be viewed as taking the original nonconvex verification problem and decomposing it into many subproblems, where each subproblem is a verification problem for a 2-layer neural network. However, as even verifying a 2-layer network is hard in general, we further develop efficient relaxation techniques that allow for tractable dual ascent.

We introduce a novel algorithm, which we call ZonoDual, which *i*) is highly scalable and amenable to GPU acceleration, *ii*) provides tighter bounds than both the prior primal and dual techniques upon which our approach is built, *iii*) is highly tunable and able to effectively balance the competing objectives of bound tightness and computation speed, and *iv*) is applicable as an add-on to existing dual verification frameworks to further boost their performance. We apply ZonoDual to a variety of networks trained on MNIST and CIFAR-10 and demonstrate that ZonoDual outperforms the linear programming relaxation in both tightness and runtime, and yields a tighter bounding algorithm than the prior dual approaches.

We first discuss prior works in the verification domain. Then, we examine the existing dual framework that serves as the backbone for our algorithm, paying particular attention to the areas in which this may be tightened. Then, we discuss how we attain zonotope-based intermediate bounds and introduce the fundamental sub-problem required by our dual ascent algorithm. Ultimately, we combine these components into our final algorithm and demonstrate the scalability and tightness of our approach on networks trained on the MNIST and CIFAR-10 datasets.

2 Related Work

Neural network verification is a well-studied problem and has been approached from several angles. Exactly solving the verification problem is known as *complete verification*, and is known to be NP-hard even for 2-layer ReLU networks [4]. Complete verification approaches include mixed-integer programs, satisfiability modulo theories (SMT), geometric procedures, and branch-and-bound techniques [5, 6, 7, 4, 8, 9, 10]. In particular, branch-and-bound procedures generate custom branching rules to decompose verification into many smaller subproblems, which can be relaxed or further branched, allowing the algorithm to focus its efforts on the areas for greatest improvement [11, 12]. We stress that this current work focuses only on the 'bound' phase, and our approach may be applied to branched subproblems.

Often it is not necessary to exactly solve the verification problem, and only providing a lower bound to the minimization problem can suffice. This is known as *incomplete verification*. We closely examine the primal and dual verification threads, but first mention a few lines of work that do not fit neatly into this paradigm: notable approaches here include semidefinite programming relaxations [13, 14], or verification via provable upper bounds of the Lipschitz constant [15, 16, 17, 18].

Primal verification: A general theme in primal verification techniques is to generate convex sets that bound the attainable range of each intermediate layer. The simplest such approach is interval bound propagation (IBP) which computes a bounding hyperbox for each layer [19]. A more complex approach is to bound each intermediate layer with a polytope. Several different polyhedral relaxations are available, with DeepPoly being on the lesser-complexity side, PLANET being the canonical triangle relaxation, and the formulation by Anderson et. al being tightest albeit with exponential complexity [20, 21, 7]. Zonotopes provide a happy medium, being both highly expressive and computationally easy to work with [22, 23]. We will discuss zonotope relaxations more thoroughly in section 4.

Dual verification: Orthogonal to the primal approaches exist several works which consider verification by relaxations in a dual space [24, 25, 12, 26, 27]. Here, verification is viewed as an optimization problem with constraints enforced by layers of the network. The general scheme here is to introduce dual variables penalizing constraint violation, and then leverage weak duality to provide valid lower bounds. The standard Lagrangian relaxation was first proposed in [24], and then several improvements have been made to provide provably tighter bounds, faster convergence, or generalizations of the augmented lagrangian. These techniques are highly scalable and notably, one recent work has been able to provide bounds tighter than even the standard LP relaxation [9]. Our approach will follow a similar scheme, but differs fundamentally from prior works in that we operate over a tighter primal domain and can therefore provide tighter bounds.

3 Lagrangian Decomposition

Problem Definition: In a canonical neural network verification problem, we are given a *scalar-valued* feedforward neural network, $f(\cdot)$, composed of L alternating compositions of affine layers and elementwise ReLU layers. We name the intermediate representations as $z_0 \dots z_L$, and are supplied with a subset, $\mathcal{X}$, of the domain of $f(\cdot)$, which we assume to be an axis-aligned hyperbox. Verification is then written as an optimization problem

$$\underset{x_0 \in \mathcal{X}, z_0, \dots, z_L}{\text{minimize}} \quad z_L \tag{1a}$$

$$\text{subject to} \quad z_0 = W_0 x_0 \tag{1b}$$

$$z_{k+1} = W_{k+1}\sigma(z_k) \quad (0 \le k \le L-1) \tag{1c}$$

where σ denotes the ReLU operator applied elementwise, and each W_k is the weight of an affine or convolutional layer. Our formulation is also amenable to bias terms everywhere, though we omit these throughout for simplicity.

Dual verification: While there are several verification approaches leveraging Lagrangian duality, we consider the Lagrangian decomposition formulation introduced in [25]. Here, each primal variable z_k of the verification problem is replaced by a pair of primal variables z_k^A, z_k^B, along with accompanying equality constraints, yielding an equivalent optimization problem:

$$\underset{x_0 \in \mathcal{X}, z_0^A, \dots, z_{L-1}^B, z_L^A}{\text{minimize}} \quad z_L^A \tag{2a}$$

$$\text{subject to} \quad z_0^A = W_0 x_0 \tag{2b}$$

$$z_{k+1}^A = W_{k+1}\sigma(z_k^B) \quad (0 \le k \le L-1) \tag{2c}$$

$$z_k^A = z_k^B \quad (0 \le k \le L-1) \tag{2d}$$

By introducing unconstrained dual variables ρ_k for each constraint in Equation (2d) and maximizing over ρ_k, we attain an optimization problem that is equivalent to the original. To tractably solve this, two relaxations are performed. First, weak duality is applied by swapping the max and min, which introduces the dual function $g(\rho)$, for which every value of ρ provides a lower bound:

$$g(\rho) \coloneqq \underset{x_0 \in \mathcal{X}, z_0^A, \dots, z_{L-1}^B, z_L^A}{\text{minimize}} \quad z_L^A + \sum_{k=0}^{L-1} \rho_k^\top (z_k^A - z_k^B) \tag{3a}$$

$$\text{subject to} \quad z_0^A = W_0 x_0 \tag{3b}$$

$$z_{k+1}^A = W_{k+1}\sigma(z_k^B) \quad (0 \le k \le L-1), \tag{3c}$$

The goal now is to solve $\max_\rho g(\rho)$. The dual function can be decomposed into L subproblems by substituting equality constraints and rearranging the objective function:

$$g(\rho) \coloneqq \left(\min_{x_0 \in \mathcal{X}} \rho_0^\top W_0 x_0 \right) + \sum_{k=1}^{L-2} \left(\min_{z_k^B} \rho_{k+1}^\top W_{k+1}\sigma(z_k^B) - \rho_k^\top z_k^B \right) + \tag{4a}$$

$$\left(\min_{z_{L-1}^B} W_L \sigma(z_{L-1}^B) - \rho_{L-1}^\top z_{L-1}^B \right) \tag{4b}$$

In its current state, it is unlikely that $g(\rho)$ for nonzero ρ will lead to any finite lower bound as each z_k^B is unconstrained. In other words, only subproblem, (4a), has any information about the input

constraint ($x_0 \in \mathcal{X}$). To remedy this, one can tighten the dual optimization by imposing *intermediate bounds* on z_k^B's, so long as those bounds include all feasible values. Specifically, for each z_k^B, we can impose a bounding set $\mathcal{Z}_k$, as long as the following implication holds:

$$x_0 \in \mathcal{X} \implies z_k^B \in \mathcal{Z}_k \qquad 1 \leq k \leq L-1. \tag{5}$$

All prior dual approaches have chosen the sets $\mathcal{Z}_k$ to be axis-aligned hyperboxes for two reasons. First, it is extremely efficient to attain hyperbox bounds at every layer. Second, when $\mathcal{Z}_k$ is a hyperbox, the dual function can be evaluated efficiently by further decomposing each component of the dual along its coordinates. Equipped with intermediate bounds $\mathcal{Z}_k$, and noting that $g(\rho)$ is a concave function of ρ, the standard procedure is to perform dual ascent on $g(\rho)$. This requires gradients $\nabla_\rho g(\rho)$, which can easily be computed as the primal residuals. Letting z_k^{A*} and z_k^{B*} be the argmin of the dual function, then $\nabla_{\rho_k} g(\rho) = z_k^{A*} - z_k^{B*}$.

The key innovation in this work is to replace the intermediate bounds $\mathcal{Z}_k$ with zonotopes everywhere. In this case, intermediate bounds may easily be computed as we will see in the next section. Zonotopes have a distinct advantage over hyperboxes in that every coordinate is no longer independent, and therefore neuron dependencies are encoded. This further constrains the feasible set of the dual function and leads to larger dual values. However, this comes with the cost that the dual function is more computationally difficult to evaluate.

4 Zonotopes

We start with a review of zonotopes and how they may be used to attain intermediate bounds. Then we introduce the problem of ReLU programming, tie it to the dual decomposition formulation, and discuss our relaxations for efficient dual evaluation. Proofs of all claims are contained in the appendix.

Zonotope Properties: Zonotopes are a class of polytopes, which we formally define using the notation $Z(c, E)$ to refer to the set

$$Z(c, E) := \{c + Ey \mid y \in [-1, 1]^m\}$$

where $c \in \mathbb{R}^d$ is called the *center*, and $E \in \mathbb{R}^{d \times m}$ is called the *generator matrix*. Each column of the generator matrix is a line segment in $\mathbb{R}^d$, and $Z(c, E)$ can be equivalently be viewed as the Minkowski sum of each generator column offset by c; or as the affine map $y \to c + Ey$ applied to the ℓ_∞ ball in $\mathbb{R}^m$. In particular, note that a hyperbox is a zonotope with a diagonal generator matrix. Zonotopes have several convenient properties:

Efficient Linear Programs: Linear programs can be solved in closed form over a zonotope:

$$\min_{z \in Z(c,E)} a^\top z = a^\top c + |a^\top E|\vec{1}.$$

Closure under affine operators and Minkowski sums: given an affine map $x \to Wx + b$, the set $\{Wz + b \mid z \in Z(c, E)\}$ is equivalent to the zonotope $Z(Wc + b, WE)$. The Minkowski sum of two sets A, B is defined as $A \oplus B := \{x + y \mid x \in A, \quad y \in B\}$. Letting $E_1||E_2$ denotes concatenation of columns, the Minkowski sum of two zonotopes $Z(c_1, E_1)$, $Z(c_2, E_2)$ is also a zonotope:

$$Z(c_1, E_1) \oplus Z(c_2, E_2) = Z(c1 + c2, E_1||E_2).$$

Intermediate Bounds with Zonotopes: Prior work has developed efficient techniques to generate intermediate layer bounds using zonotopes for feedforward neural networks. The details are deferred to the appendix, and we can assume that we have access to a black box that, given a zonotope $\mathcal{Z}$, is able to generate a zonotope $\mathcal{Y}$ such that $\{\sigma(z) \mid z \in \mathcal{Z}\} \subseteq \mathcal{Y}$. Such an operation is called a *sound pushforward operator*. One such pushforward operator is known as `DeepZ` and can be applied repeatedly to provide valid zonotopic bounds for every intermediate layer of the neural network [22]. We make two observations regarding these intermediate bounds here:

Proposition 4.1. *Let $\mathcal{Z}_k$ be the k^{th} zonotope bound provided by the `DeepZ` procedure, and let $\mathcal{H}_k$ be the k^{th} intermediate bound provided by the Kolter-Wong dual procedure [20]. Then the smallest hyperbox containing $\mathcal{Z}_k$ is exactly $\mathcal{H}_k$.*

Proposition 4.2. *Suppose $\mathcal{Z}$ is a zonotope and $\mathcal{H}$ is a hyperbox, such that $\mathcal{Z} \not\subseteq \mathcal{H}$. Then we can develop a sound pushforward operator for the ReLU operator that outputs a zonotope $\mathcal{Z}'$ containing the ReLU of every element of $\mathcal{Z} \cap \mathcal{H}$, with the property that $\mathcal{Z}' \subsetneq$ `DeepZ`$(\mathcal{Z})$.*

The first proposition states that a common technique for generating intermediate hyperbox bounds is never any tighter than the procedure we use to generate intermediate zonotope bounds. The second proposition describes an improved pushforward operator that offers improvements against `DeepZ` but requires extra information provided by a hyperbox.

ReLU Programming: We return our attention to the dual function introduced in equations Equations (4a) and (4b). Note that the first term of Equation 4a is a linear program of a zonotope which can be solved extremely efficiently, while all other terms adopt a form we refer to as a *ReLU program*

Definition 4.1. *Given a set $\mathcal{Z} \subseteq \mathbb{R}^d$, and two objective vectors $c_1, c_2 \in \mathbb{R}^d$, we say the **ReLU Program** is the optimization problem*

$$\min_{z \in \mathcal{Z}} c_1^\top z + c_2^\top \sigma(z). \tag{6}$$

ReLU programming is nonconvex for general c_2. When $\mathcal{Z}$ is an interval, only the endpoints, or 0 if it is contained in the interval, are candidate optima. In this case, the ReLU program may be solved by evaluating the objective at each candidate. When $\mathcal{Z}$ is a hyperbox in $\mathbb{R}^d$, each coordinate can be considered independently, amounting to solving d ReLU programs over intervals. The story quickly becomes more complicated if $\mathcal{Z}$ is a zonotope:

Theorem 4.1. *When $\mathcal{Z}$ is a zonotope, solving a ReLU program over $\mathcal{Z}$ is equivalent to a neural network verification problem for a 2-layer network, which is known to be NP-hard.*

Remarks: This theorem implies that solving ReLU programs over more complex sets, such as polytopes in general, is also NP-hard. However it remains an open question whether solving ReLU programs over other simpler sets such as parallepipeds, i.e. zonotopes with an invertible generator matrix, is hard. Note that ReLU programs can be solved exactly via a Mixed-Integer Program (MIP) using the standard form for encoding a ReLU [5].

We pause here to observe the current situation. Lagrangian methods for verification first make a relaxation by applying weak duality. When using zonotopes for the intermediate bounds, one is able to decompose the very large non-convex optimization problem of deep network verification into many smaller subproblems which are equivalent to two-layer network verification problems. Unfortunately, each of these subproblems is also NP-hard, and thus must be further relaxed to attain tractability.

Zonotope Partitioning: The primary relaxation technique we consider comes from the following observation. Consider a d-dimensional zonotope $\mathcal{Z}$ and ReLU programming objectives (c_1, c_2). Then, for any partition $S_1 \cup \cdots \cup S_n$ of $\{1, \ldots, d\}$, the inequality

$$\min_{z \in \mathcal{Z}} \sum_{i=1}^{d} \left(c_{1,i} z_i + c_{2,i} \sigma(z_i)\right) \geq \sum_{j=1}^{n} \min_{z \in \mathcal{Z}} \sum_{i \in S_j} \left(c_{1,i} z_i + c_{2,i} \sigma(z_i)\right)$$

holds. This approach has a nice geometric interpretation as well. The decomposed minimization $\min_{z \in \mathcal{Z}} \sum_{i \in S_j} \left(c_{1,i} z_i + c_{2,i} \sigma(z_i)\right)$ can be viewed as optimization over $\mathcal{Z}$ when it has been projected onto the linear subspace spanned by the elements of S_j. Projection onto a linear subspace is an affine operator, so each projection of $\mathcal{Z}$ is itself a zonotope. Indeed, the projection onto the linear subspace spanned by S_j is simply the center and generator rows indexed by S_j.

Observe that if each S_j were a singleton set, then decomposing the ReLU program according to these S_j's is equivalent to relaxing the ReLU program by casting $\mathcal{Z}$ to the smallest containing hyperbox. If the partition were the trivial partition $\{1, \ldots, d\}$, then $\mathcal{Z}$ remains unchanged. Finally, notice that the Minkowski sum of every projection of $\mathcal{Z}$ is itself a zonotope.

In this sense, by choosing any partitioning of coordinates, we are able to relax $\mathcal{Z}$ into a zonotope that is simultaneously a superset of $\mathcal{Z}$ and a subset of the hyperbox-relaxation of $\mathcal{Z}$. This is optimistic for our dual ascent procedure, as any partitioning restricts the primal feasible space for the dual function evaluation. It is in this way that we are able to interpolate the coarseness of our relaxation, which is directly controlled by the coarseness of our partitioning. Because we expect the complexity of solving a ReLU program over a zonotope to be superlinear in the dimension, shattering a zonotope in this fashion can drastically improve the complexity of solving a relaxation of a ReLU program.

Two-dimensional Zonotopes: A special case we consider is when we partition a zonotope into 2-dimensional groups. This is because a 2-dimensional zonotope can be significantly more descriptive

than a 2-dimensional rectangle, but the solution of a ReLU program can still be computed efficiently. Observe that the argmin of a ReLU program over a 2-dimensional zonotope must occur at either i) a vertex of the zonotope, ii) a point where the zonotope crosses a coordinate-axis, iii) the origin. This follows from the fact that a ReLU program over 2-dimensions may be viewed as 4 independent linear programs over the intersection of the zonotope and each orthant. For the purposes of ReLU programs, it suffices to enumerate all vertices of the zonotope and compute any axis crossings or origin-containments. It turns out that this may be done in nearly linear time:

Theorem 4.2. *If $Z(c, E)$ is a 2-dimensional zonotope with m generators, Z has $2m$ vertices and the set of all vertices, axis crossings, and the containment of the origin can all be computed in $O(m \log m)$ time.*

5 The ZonoDual Algorithm

With both preliminaries of dual verification techniques and zonotopic primal verification techniques in hand, we can now fully describe the algorithm we employ in practice. We first describe the algorithm as it is used to bound a single neuron's value, i.e. the output neuron. After we describe how we adapt our approach to the stagewise setting where our algorithm is applied to each intermediate neuron to yield tighter box bounds. Our algorithm can be described in three main phases: an *initialization phase*, where primal bounds are generated to define the feasible set for each subproblem in the dual function; an *iteration phase*, where the argmin of the dual function is computed many times to provide informative gradients for updates to the dual variables; and an *evaluation phase*, where the primal bounds are tightened to yield a more computationally intensive dual function, which only needs to be evaluated a single time.

Initialization phase: Naively, we can apply the `DeepZ` procedure here to attain a zonotope bounding the output range of the network at every layer. Both this procedure and our box-improved variant can be done extremely efficiently. We adapt two techniques from [25] in the initialization phase. First we notice that some very minor improvements to `DeepZ` can be made using an application of Proposition 4.2 when provided with the boxes from interval bound propagation. Second, we initialize the dual variables to the KW dual variables from [20], which by 4.1 are equivalent to the scale factors applied internally in the zonotope pushforward operators.

Iteration Phase: In the iteration phase, each intermediate zonotope $\mathcal{Z}_k$ is partitioned into components that are amenable to repeated evaluations of the dual function. We always initially partition each zonotope into 2-d chunks for which the dual function may be evaluated on a GPU without requiring any MIP calls. Recall that for a d-dimensional zonotope with m generators, after the initial $O(dm \log m)$ candidate optima computation, each ReLU program can be evaluated in $O(dm)$ time. These initial 2-d partitions are computed using one of several simple heuristics, described in the appendix, with the aim to generate 2-d zonotopes which are as 'un-box-like' as possible. Optionally, after an initial 2-d iteration phase, these partitions may be combined and dual ascent can be applied, where each subproblem now requires several calls to a MIP solver. At any point during this phase, the dual function evaluations still provide valid lower bounds to the verification problem.

Evaluation Phase: In this phase, we can presume that effective dual variables have been found during the previous phase. The objective here is to tighten the feasible range of each subproblem, which will necessarily improve the value of the dual function. Because we only need to evaluate a bound on the dual function once here, we find that it is often worth it to spend more time solving each subproblem. We do this by simply merging several partition elements together, converting our 2-dimensional zonotopes into higher-dimensional ones that require calls to a MIP solver to compute. Despite the theoretical intractibility of MIPs, we find that this procedure is remarkably efficient. In practice, it is often sufficient to only merge zonotopes at the later layers in a network where the dual function subproblems are the most negative. Finally, we note that as only a *bound* upon the dual function is required here, and each subproblem is itself a 2-layer verification problem, the MIPs may be terminated early. Any other incomplete verification approach can be applied here as well.

Guarantees: Ultimately, our algorithm will always provide a valid lower bound to the verification problem so long as the dual function is evaluated appropriately. The provable correctness of our algorithm stems from i) the fact that weak duality always holds and any evaluation of the dual function with feasible dual variables provides a valid lower bound tot the primal problem; and ii) by leveraging zonotopes and a sound relaxation of them, we only ever provide lower bounds to the evaluation of the dual function. Further, we can also guarantee that, for any fixed set of dual variables ρ, the bounds

we provide will be tighter than those provided by the approach in [25]. This follows because we have reduced the feasible set within each dual subproblem, relative to the hyperbox approaches in prior works. Similarly, because $\rho = 0$ is always a valid choice, when the dual variables are optimized appropriately, we will provide tighter bounds than whichever primal verification method is applied at the initialization phase.

Stagewise computation: When the network of interest is of sufficiently small size, any verification algorithm may be applied stagewise, where the algorithm computes upper and lower bounds to each neuron, starting from those in the first layer. This is often effective for generating tighter intermediate box bounds, which in turn yields much tighter output bounds. However this comes the cost of running many calls to the verification algorithm. This can be parallellized for verification procedures amenable to GPU acceleration, but demands a large memory footprint.

At first glance, it may seem that such a stagewise procedure obviates the improvements we attain by leveraging zonotopes: indeed, each stagewise bound is a *box bound*. However, we show that we can apply this stagewise trick to our approach to generate tighter box bounds, which can then be employed by Proposition 4.2 to yield tighter zonotopic bounds. Additionally, in the case that these box bounds are incomparable to the improved zonotope bounds, these can be employed in imposing further constraints upon each subproblem. Instead of solving each ReLU program over a zonotope, we can instead solve each ReLU program over the intersection of a zonotope and a hyperbox. In the higher-dimensional case, it is trivial to apply elementwise constraints to each variable in the MIP — in fact, this often improves the running time of each MIP. In the 2-d case, we observe that we are able to solve ReLU programs over the intersection of a 2-d zonotope and rectangle as efficiently as solving over a 2-d zonotope alone. This is the content of the following corollary to Theorem 4.2.

Corollary 5.1. *Given a 2-d zonotope with m generators, the set of candidate optimal points to a ReLU program has cardinality no greater than $(2m + 9)$ and is computable in time $O(m \log m)$.*

Ablation study: To clarify the bound improvements and their corresponding runtime cost imposed by each step of our approach, we perform an ablation study on an MNIST fully connected network (MNIST FFNet). We report the bounds relative to the bound attained by the LP relaxation in Table **??**. We consider three initialization settings separately: the standard `DeepZ` zonotopes, the `DeepZ` zonotopes augmented with IBP bounds, and zonotopes informed by the the box bounds from BDD+ [25]. Then we perform dual ascent over 2-d zonotopes, before merging the zonotopes of the final layer only and evaluating the MIP. We report these numbers in Table **??**. Here we see that a tight initialization does indeed help, but this gap shrinks after our dual ascent procedure. We also see a substantial gain in bound improvement by using MIPs, even in the last layer's zonotopes alone.

6 Experiments

We evaluate the effectiveness of our approach on several networks trained on the MNIST and CIFAR-10 datasets. While many recent verification techniques employ a branch-and-bound strategy, we only provide innovations on the bounding component of this pipeline. Strong verifiers should employ both effective bounding algorithms and clever branching strategies, which are both necessary but fairly independent components. Like other dual-based verification techniques, our approach can be applied using any of a number of branching strategies. We focus our efforts primarily on the improvements in the *bound* provided by a single run of each considered method. We argue this is a reasonable choice of metric because most prior benchmarks focus on ϵ-values far lower than typically used in adversarial attack scenarios. This artifically decreases the depth to which branching strategies must search, skewing the results. By focusing on the bound of the verification, we directly measure the metric that bounding algorithms such as ours seek to optimize. However, for the sake of completeness, we do also evaluate verified robustness percentage upon several benchmark MNIST networks. For bound evaluations, we separate our experiments into two cases: we first examine the setting where only the output neuron is optimized, which we denote the 'single-stage' setting; and the case where every neuron is optimized to provide tighter intermediate box bounds, denoted the 'stagewise' setting. We compare the bound and running time against the following approaches: **BDD+**, the Lagrangian Decomposition procedure proposed in [25], where the optimization is performed using optimal proximal step sizes; **LP**, the original linear programming relaxation using the PLANET relaxation on each ReLU neuron [20]; **Anderson**, the linear programming relaxation, where 1-cut is applied per neuron as in [7]; **AS**, a recent Lagrangian technique which iteratively chooses cuts to employ from the Anderson relaxation [9].

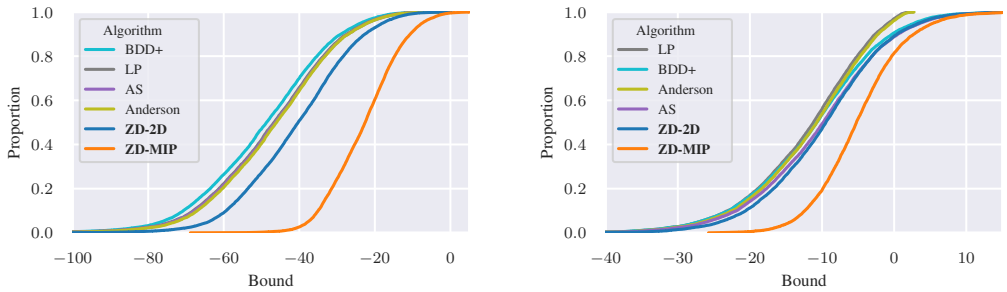


Figure 1: Cumulative density function plots of the bounds obtained by each algorithm for a deep convolutional network trained on MNIST (left) and a deep convolutional network trained on CIFAR10 (right). Lines further to the right indicate tighter bounds. Our algorithms are bolded in the legend.

All baselines are run implemented using the code and hyperparameter choices from prior works, and network weights are taken from existing works where applicable. Architecture descriptions are contained in the supplementary. We follow [25] to construct scalar-valued verification problems: for examples with label i, we compare the difference between the i^{th} and $(i+1)^{th}$ logit of a network. We can extend this procedure to a multi-class task by applying the verification $n-1$ times for each incorrect logit.

Table 1: Timing data for the single-stage bounds.

Alg\Net	MNIST Deep	CIFAR SGD
BDD+	2.5 ± 0.2	1.5 ± 0.1
LP	4.8 ± 0.5	6.2 ± 0.6
ZD-2D	10.9 ± 0.5	7.6 ± 0.3
AS	16.9 ± 1.2	10.2 ± 0.8
ZD-MIP	23.1 ± 12.4	11.0 ± 3.1
Anderson	52.4 ± 27.5	21.3 ± 4.6

Single-stage bound evaluation: In the single-stage setting we make the following hyperparameter choices for our approach. We initialize the intermediate bounds using zonotopes informed by the boxes attained by the best of the `DeepZ` bounds and the IBP bounds. We then decompose each zonotope into 2-dimensional partitions, using the 'similarity' heuristic for fully-connected networks, and the 'spatial' heuristic for convolutional networks. Then we perform 1000 iterations of the Adam optimizer during the dual ascent procedure. We report the value at the last iteration as **ZD-2D**. Then we merge the partitions of only the last layer of each network into 20-dimensional zonotopes, and evaluate the dual function under this new partitioning. We report these values as **ZD-MIP**.

We present results of our approach on the MNIST-Deep network, which has 5196 neurons and 6 layers, evaluating over a domain of ℓ_∞ boxes with $\epsilon = 0.1$. Specifically we present cumulative density function plots: a point (x, y) on any curve in these plots indicates that that particular algorithm attains a bound tighter than x for $(1-y)\%$ of the examples. Curves further to the right indicate tighter bounds in aggregate. Figure 1 (left) contains the distribution of reported bounds over every correctly-classified example in the MNIST validation set. In Figure 1 (right), we present a similar CDF plot for a CIFAR-10 network with 6244 ReLU neurons and 4 layers. For this CIFAR network, we set $\epsilon = 5/255$, as in [9]. The average and standard deviation runtimes for these networks is provided in Table 1.

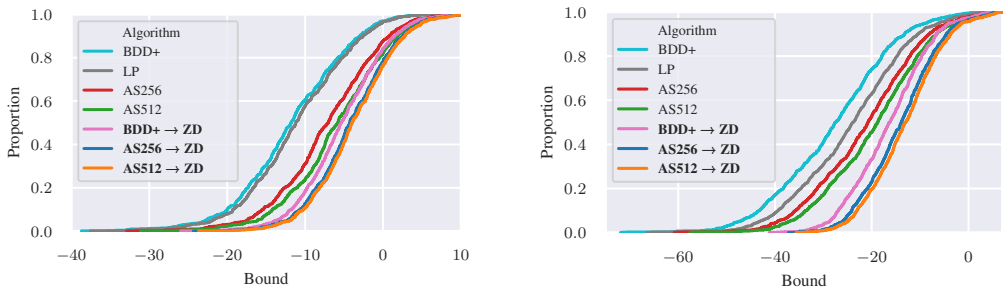


Figure 2: Stagewise bounds on the MNIST-FFNet (left) and MNIST-Deep (right) networks.

Stagewise bound evaluation: Now we turn our attention to the setting where enough computation time is allowed to perform the optimization procedure on every neuron in turn, which is able to

Table 3: Verified Robustness on MNIST Networks. Methods denoted with an asterisk had numbers taken from [28].

Model	ϵ	**ZD**	BDD+	CROWN*	PRIMA*	β-CROWN*
MLP 5x100	0.026	28.5% (24s)	19.8% (7s)	16.0% (1s)	51.0% (159s)	69.9% (102s)
MLP 8x100	0.026	21.8% (19s)	19.1% (15s)	18.2% (1s)	42.8% (301s)	62.0% (103s)
MLP 5x200	0.015	41.8% (28s)	33.2% (8s)	29.2% (2s)	69.0% (224s)	77.4% (86s)
MLP 8x200	0.015	29.7% (26s)	27.4% (17s)	25.9% (6s)	62.4% (395s)	73.5% (95s)
ConvSmall	0.12	52.6% (38s)	16.6% (6s)	15.8% (3s)	59.8% (42s)	72.7% (7s)

generate tighter box bounds. Here we notice an interesting observation: our method can be improved by leveraging tighter intermediate box bounds, but is agnostic as to where these bounds come from. Indeed, our approach can take as input any collection of tighter box bounds and generate tighter zonotopes via Proposition 4.2. However, our current implementation of this tighter primal bound is not vectorized to be amenable, in the way that others are, to stagewise bounding procedures. This is purely an implementation detail, and we instead use existing primal bounding techniques.

As baselines here, we compare against the LP approach, BDD+, and the active set approach with 256 and 512 iterations, denoted AS256, AS512. We leverage box bounds provided by each of these dual approaches to inform our zonotope bounds from which we can run our method, denoted by a $\rightarrow$ ZD. Note that BDD+ can never surpass bounds provided by the LP approach, whereas ours and the active set procedure can both improved even further with a longer runtime. Hence it is not sufficient to provide a better bound, but to also do so more efficiently. In this case, our approach performs 2000 iterations of Adam on 2-D zonotopes and increases the partition size of the final two layers of each network to sizes of 16 and 20 respectively.

We present our results on the MNIST-FFNet and MNIST-Deep network which have 960 and 5196 neurons, and 5 and 6 layers respectively, again using an $\epsilon = 0.1$. CDF plots of bounds for these two networks are provided in 2, with timing data in 2. We observe that the BDD+ bound completes very quickly, but attains a bound only slightly worse than LP. Our approach, when initialized with the BDD+ bounds runs more efficiently than AS256, providing bounds that are comparable or better than the slower AS512 approach.

Table 2: Timing data for stagewise bounds

Alg\Net	MNIST Deep	MNIST Wide
BDD+	15.4 ± 0.4	8.3 ± 0.3
BDD+ $\rightarrow$ ZD	51.7 ± 14.0	36.4 ± 6.8
AS256	138.9 ± 0.9	45.4 ± 0.6
AS256 $\rightarrow$ ZD	174.1 ± 13.6	73.3 ± 6.6
AS512	289.3 ± 1.7	91.3 ± 0.9
AS512 $\rightarrow$ ZD	324.1 ± 13.5	119.2 ± 6.9
LP	2132.2 ± 494.4	387.1 ± 65.5

%-Verified Robustness: Finally we evaluate our approach comparing against all incorrect logits on MNIST trained networks, mirroring the setup of [28]. We report numbers from an unbranched version of ZD and BDD+ alongside several other verification techniques [29, 30, 31]. Compared to other efficient, branch-friendly procedures such as BDD+ and CROWN, our approach provides tighter bounds at a fair cost to efficiency. Our approach provides weaker bounds than PRIMA, but is significantly faster. β-CROWN, in particular, leverages a dual-based branching strategy which could be applied to our method. However due to its extremely efficient bounding procedure and the need to solve millions of subproblems for verification, it is unclear if our approach can surpass this method.

7 Conclusion

In this work, we demonstrated how primal and dual verification techniques can be combined to yield efficient bounds that are tighter than either of them alone. Primal methods capture complex neuron dependencies, but do not optimize for a verification instance; while dual methods are highly scalable, but rely on unnecessarily loose primal bounds. This work is a first step in combining these two approaches. We believe that this combination of approaches is a necessary direction for providing scalable and tight robustness guarantees for safety-critical applications of machine learning.

Acknowledgements: This work is supported in part by NSF grants CNS-2002664, DMS-2134012, CCF-2019844 as a part of NSF Institute for Foundations of Machine Learning (IFML), CNS-2112471 as a part of NSF AI Institute for Future Edge Networks and Distributed Intelligence (AI-EDGE), CCF-1763702, AF-1901292, CNS-2148141, Tripods CCF-1934932, and research gifts by Western

Digital, WNCG IAP, UT Austin Machine Learning Lab (MLL), Cisco and the Archie Straiton Endowed Faculty Fellowship.

References

[1] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. December 2013.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 274–283. PMLR, 2018.

[3] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. February 2019.

[4] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. February 2017.

[5] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. November 2017.

[6] Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, July 2018.

[7] Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pages 1–37, 2020.

[8] Matt Jordan, Justin Lewis, and Alexandros G Dimakis. Provable certificates for adversarial examples: Fitting a ball in the union of polytopes. In *Advances in Neural Information Processing Systems 32*, pages 14059–14069. 2019.

[9] Alessandro De Palma, Rudy Bunel, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip H S Torr, and M Pawan Kumar. Improved branch and bound for neural network verification via lagrangian decomposition. April 2021.

[10] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. November 2020.

[11] Jingyue Lu and M Pawan Kumar. Neural network branching for neural network verification. *arXiv preprint arXiv:1912.01329*, 2019.

[12] Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip H S Torr, and M Pawan Kumar. Scaling the convex barrier with active sets. `https://openreview.net/pdf?id=uQf0y7LrlTR`. Accessed: 2021-9-18.

[13] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Semidefinite relaxations for certifying robustness to adversarial examples. November 2018.

[14] Sumanth Dathathri, Krishnamurthy Dvijotham, Alexey Kurakin, Aditi Raghunathan, Jonathan Uesato, Rudy Bunel, Shreya Shankar, Jacob Steinhardt, Ian Goodfellow, Percy Liang, et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *arXiv preprint arXiv:2010.11645*, 2020.

[15] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems 32*, pages 11423–11434. 2019.

[16] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems 31*, pages 3835–3844. 2018.

[17] Navid Hashemi, Justin Ruths, and Mahyar Fazlyab. Certifying incremental quadratic constraints for neural networks via convex optimization. December 2020.

[18] Matt Jordan and Alexandros G Dimakis. Exactly computing the local lipschitz constant of ReLU networks. March 2020.

[19] Rüdiger Ehlers. Formal verification of Piece-Wise linear Feed-Forward neural networks. In *Automated Technology for Verification and Analysis*, pages 269–286. Springer International Publishing, 2017.

[20] J Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. November 2017.

[21] Singh, Gagandeep, Gehr, Timon, Püschel, Markus, and Vechev, Martin. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, January 2019.

[22] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin T Vechev. Fast and effective robustness certification. *NeurIPS*, 1(4):6, 2018.

[23] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International conference on learning representations*, 2018.

[24] Krishnamurthy, Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. March 2018.

[25] Rudy Bunel, Alessandro De Palma, Alban Desmaison, Krishnamurthy Dvijotham, Pushmeet Kohli, Philip Torr, and M Pawan Kumar. Lagrangian decomposition for neural network verification. In Jonas Peters and David Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 370–379. PMLR, 2020.

[26] Leonard Berrada, Sumanth Dathathri, Krishnamurthy, Dvijotham, Robert Stanforth, Rudy Bunel, Jonathan Uesato, Sven Gowal, and M Pawan Kumar. Verifying probabilistic specifications with functional lagrangians. February 2021.

[27] Shaoru Chen, Eric Wong, J Zico Kolter, and Mahyar Fazlyab. Deepsplit: Scalable verification of deep neural networks via operator splitting. *arXiv preprint arXiv:2106.09117*, 2021.

[28] 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 complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*, 2021.

[29] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

[30] Mark Niklas Müller, Gleb Makarchuk, Gagandeep Singh, Markus Püschel, and Martin Vechev. Precise multi-neuron abstractions for neural network certification. *arXiv e-prints*, pages arXiv–2103, 2021.

[31] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. September 2018.

[32] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3578–3586, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

[33] Matt Jordan and Alex Dimakis. Provable lipschitz certification for generative models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5118–5126. PMLR, 2021.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[36] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL `https://www.gurobi.com`.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

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] See Appendix
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix
   (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] See Appendices
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] Zipped in appendix
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Described in Appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Standard deviations for times, but n/a for CDF plots
   (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] Compute environment delineated in appendix
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] Models and choices of ϵ given proper credit
   (b) Did you mention the license of the assets? [No]
   (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]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] MNIST/CIFAR only

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 Broader Impacts

Our work focuses on proving robustness and security for machine learning applications. While any new technology may be used maliciously, we believe the technical advances proposed in this work are as benign as possible and will only serve to offer certification of existing technologies.

B Limitations and Future Work

While ZonoDual has some benefits over prior works, it is not without limitations. We enumerate these here, as well as potential ways to overcome these:

- **Reliance on ReLU nonlinearities:** ZonoDual relies on the ability to solve ReLU programs over zonotopes using MIPs. Unfortunately, this approach cannot generalize to nonlinearites that are not piecewise linear, such as the sigmoid activation. While it might be possible to solve 'sigmoid programs' over 2-dimensional zonotopes, the formulation becomes significantly more complicated.
- **MIPs are not conducive to GPU acceleration:** A crucial component of our procedure is to evaluate the dual using a tighter relaxation during the final phase of ZonoDual. Prior works that leverage box bounds do not include such a tightening step, but we note that MIPs can not benefit from GPU computations. In practice, small MIPs can solve very quickly, but for extremely large nets, such as SOTA image models, it is unlikely that this approach can scale as nicely as other techniques. This presents a need for fast and tight lower bounds for the reduced verification problem when the network is known to only have one hidden layer. Unfortunately, such methods, such as SDP's cannot currently scale to the case where this would be helpful for us.
- **Intermediate zonotopes may have very large order:** One drawback of using zonotopic intermediate bounds is that the number of generators can grow to be quite large for deeper layers in the network. This can slow down the dual computation procedure, even in the 2-dimensional case for which dual evaluation time is linear in the number of generators. There is room for improvement here in what is known as 'order reduction' techniques for zonotopes, which soundly compresses the number of generators of a zonotope. Naive approaches are very quick and can allow for a 2x compression factor with only marginal degradations to dual function values, but significantly tighter compressions require much more work. We have not included any of these accelerations in our presentation for the sake of simplicity.
- **Exact evaluations of the dual seem to be necessary for informative gradients:** One might hope that the dual need not be evaluated exactly during the iteration phase of ZonoDual. Such a procedure could allow for much faster and tighter dual ascent iterations when only informative gradients are required. Unfortunately the approaches we have tried in this domain (including Frank-Wolfe, L-BFGBS, and a modified simplex algorithm) do not provide gradients that are informative enough to yield good dual values for the final evaluation phase. It is possible there are techniques here that could work, but we have not found them.

C Zonotope Pushforward Operators

Here we describe the sound zonotope pushforward operators we use to generate intermediate bounds. Since zonotopes are closed under affine transformation, all that's required is to describe the pushforward operator for the ReLU operator. Specifically, given a zonotope $Z(c, E)$, we seek to generate a zonotope $Z(c', E')$ such that $\{\sigma(z) \mid z \in Z(c, E)\} \subseteq Z(c', E')$. First we describe the approach when no extra information is known. This was first presented in [32], but we adapt the analysis of [33]. Next we consider the case when we also have the extra information in the form of a hyperbox $H(l, u) := \{x \mid l \leq x \leq u\}$, where the goal is to construct $Z(c', E')$ such that $\{\sigma(z) \mid z \in Z(c, E) \cap H(l, u)\} \subseteq Z(c', E')$.

`DeepZ`**:** The primary strategy is to compress each coordinate of $Z(c, E)$ by a scale factor $\lambda_i \in [0, 1]$, which will in general, incur some error along each coordinate, b_i. These errors are accumulated into a hyperbox, which will then be incorporated into $Z(c', E')$ in the form of a Minkowski sum. The goal is to minimize the amount of error that is accumulated. In other words, each coordinate can be

considered independently, where we wish to solve the minmax procedure

$$\min_{\lambda_i, b_i} \left(\max_{z \in Z(c,E)} |\sigma(z_i) - \lambda_i z_i - b_i| \right)$$

where the choice of the compression factor λ_i is chosen to minimize the error that needs to be added in at the end between $\sigma(z_i)$ and $\lambda_i z_i$ over all feasible points in the zonotope. Noticing that projecting $Z(c, E)$ only onto the i^{th} coordinate necessarily yields an interval, so the $\max$ is performed over an interval, $[l_i, u_i]$. In the case that $l_i \geq 0$, the ReLU is always on, and λ_i can be chosen to be 1, with $b_i = 0$. In the case that $u_i \leq 0$, then the ReLU is always off, and λ_i can be chosen to be 0, again with $b_i = 0$. In the case that $l_i < 0 < u_i$, it is not difficult to see that the optimum value is attained when λ_i is chosen to be $\frac{u_i}{u_i - l_i}$, such that the optimal value is $\frac{-l_i u_i}{u_i - l_i}$. Hence we can accumulate λ_i for each coordinate i into a diagonal matrix Λ. The optimal offsets, $(b_1, \ldots, b_d)$, can be divided by 2, and accumulated into a diagonal matrix B whose columns may be appended to the compressed zonotope. Put concisely,

$$Z(c', E') := Z(\Lambda c, \Lambda E | B)$$

where

$$\Lambda_{i,i} = \begin{cases} 0 & \text{if } u_i \leq 0 \\ 1 & \text{if } l_i > 0 \\ \frac{u_i}{u_i - l_i} & \text{otherwise} \end{cases} \qquad B_{i,i} = \begin{cases} 0 & \text{if } u_i \cdot l_i \geq 0 \\ \frac{-l_i \cdot u_i}{2(u_i - l_i)} & \text{otherwise} \end{cases}$$

Improvements using Hyperboxes: Now we restate our proposition that allows for an improvement upon this pushforward operator when we have an incomparable hyperbox bound.

Proposition 4.2. *Suppose $\mathcal{Z}$ is a zonotope and $\mathcal{H}$ is a hyperbox, such that $\mathcal{Z} \not\subseteq \mathcal{H}$. Then we can develop a sound pushforward operator for the ReLU operator that outputs a zonotope $\mathcal{Z}'$ containing the ReLU of every element of $\mathcal{Z} \cap \mathcal{H}$, with the property that $\mathcal{Z}' \subsetneq \texttt{DeepZ}(\mathcal{Z})$.*

Proof. Let $Z(c, E)$ be the zonotope $\mathcal{Z}$, and let $H(l^h, u^h)$ be the hyperbox $\mathcal{H}$. Similar to DeepZ, the strategy we employ to develop a sound pushforward operator is to scale each coordinate independently, and Minkwoski sum the errors, accumulated as a hyperbox. Again, this decomposes into a minmax procedure as

$$\min_{\lambda_i, b_i} \left(\max_{z \in Z(c,E) \cap H(l^h, u^h)} |\sigma(z_i) - \lambda_i z_i - b_i| \right).$$

In a single dimension, both a zonotope and hyperbox are intervals, so their intersection is also an interval. Letting (l_i^z, u_i^z) be the i^{th} coordinates lower and upper bounds for $\mathcal{Z}$, then the intersection between the i^{th} coordinate projection of $\mathcal{H}$ and $\mathcal{Z}$ is the interval $[l_i, u_i] := [\max(\{l_i^z, l_i^h\}, \min(\{u_i^z, u_i^h\})]$. From here the choice of Λ and B is exactly the same as in DeepZ, where the tightened coordinate-wise bounds are used. This is guaranteed to be as tighter as long as at least one coordinate exists for which the interval is reduced, because both the compression factor λ_i and error b_i are reduced by tightening the interval. By our assumption that $\mathcal{Z} \not\subseteq \mathcal{H}$, at least one coordinate interval must be tightened, so the inclusion $\mathcal{Z}' \subset \texttt{DeepZ}(\mathcal{Z})$ is strict. □

Kolter-Wong Bounds Here we present a geometric proof of Proposition 4.1. We will only briefly discuss the derivation of this bound in the context of this proof. For a more thorough discussion on the exact structure and derivation of the Kolter/Wong dual relaxation, we point to the original paper [20] and a self-contained derivation in the appendix of [25].

Proposition 4.1. *Let $\mathcal{Z}_k$ be the k^{th} zonotope bound provided by the DeepZ procedure, and let $\mathcal{H}_k$ be the k^{th} intermediate bound provided by the Kolter-Wong dual procedure [20]. Then the smallest hyperbox containing $\mathcal{Z}_k$ is exactly $\mathcal{H}_k$.*

Proof. In [20], the authors derive the original linear programming relaxation and dual procedure for verifying ReLU networks. This technique can be interpreted in the primal sense as maintaining a polyhedral primal bound for every intermediate layer. Assuming a polyhedral input range, such a hyperbox, this procedure works by incrementally lifting the dimensionality of the polytope. In other words, an H-polytope is maintained, where new dimensions/variables are introduced through equality constraints defined by each layer. For affine layers, this lifting procedure is exact. For ReLU layers,

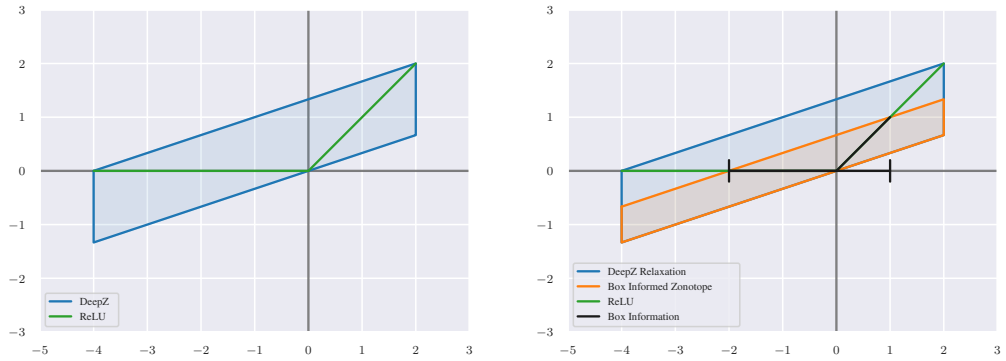


Figure 3: Graphical representation of the coordinatewise bounding procedure of both DeepZ and our improved version when box bounds are available. On the left, the blue zonotope is the one introduced by DeepZ. On the right, we are also aware than our interval is bounded by the blue-interval, and the blue zonotope is the improved bound.

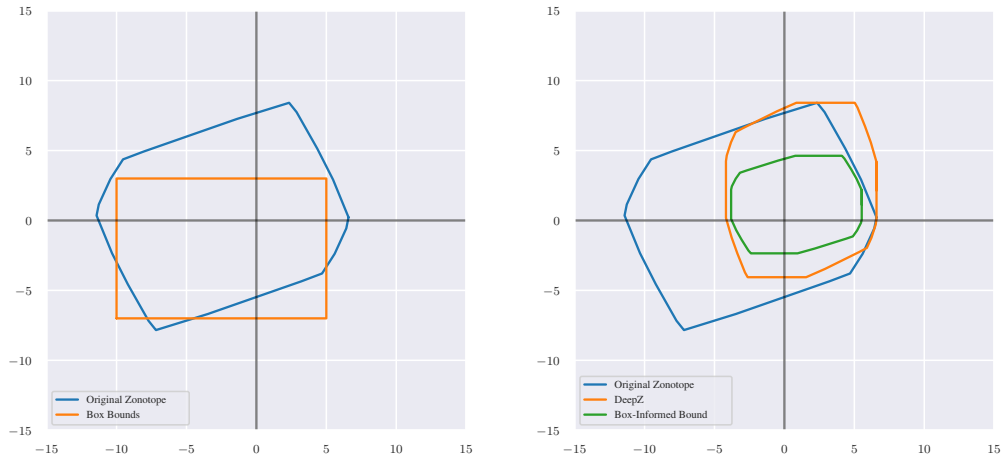


Figure 4: An application of the improved zonotope pushforward operator on a 2-d example. On the left, the original zonotope is outlined in blue, and the extra hyperbox information is drawn in orange. On the right, the output of DeepZ is drawn in orange. The output of the improved pushforward operator, that has access to the box, is drawn in green.

this lifting procedure introduces some error. This follows because the graph of the ReLU function, viewed as a 2-d set, is not convex. In this case the canonical triangle relaxation is applied, which is simply the convex hull of the graph of the ReLU function. This triangle relaxation, applied over an interval $[l, u]$ with $l < 0 < u$, has as its upper convex hull, a line with slope $\frac{u}{u-l}$ and intercept $\frac{-ul}{u-l}$. This lifting procedure is iterated over all layers, ultimately yielding a polytope of dimension $dim(\mathcal{X}) + \sum_{i=1}^{L} 2n_i$, where n_i refers to the number of neurons of the i^{th} layer. Optimization over this lifted polytope was framed as a linear program, where the objective was set to minimize the variable corresponding to the final output neuron.

In the original KW formulation, the dual of this linear program, a dual variable was introduced that geometrically corresponds to a lower-bounding linear functional on the triangle relaxation. This is similar to the observation made in $\alpha - \beta - CROWN$ [28]. In the original linear programming paper, an initial choice of $\alpha = \frac{u}{u-l}$ was chosen, which in this case may be interpreted as lower bounding the triangle relaxation by a line that is parallel to the upper convex hull, essentially providing an initial relaxation as a parallelogram with vertical sides.

On the other hand, the zonotope propagation procedure may also be viewed as a polyhedral lifting procedure. Recall in the DeepZ formulation, the sound pushforward operator for the ReLU operator works by scaling the i^{th} coordinate by the factor $\frac{u_i}{u_i - l_i}$ and incorporating the error in terms of a

Minkowski sum. This may equivalently be viewed as lifting a d-dimensional zonotope, to a $(d+1)$-dimensional zonotope where the newly added dimension has range that is a function of only the i^{th} coordinate. In particular, it is constrained to lie inside the vertical parallelogram defined on the vertical edges by l, u, with the slanted edge having slope $\frac{u_i}{u_i - l_i}$. The upper edge is exactly the convex upper hull of the graph of the ReLU function. In practice, because it is convenient and efficient to project zonotopes onto lower dimensional subspaces, when mapping a d-dimensional through a ReLU layer, instead of lifting to a $2d$-dimensional zonotope, the original dimensions are simply dropped to retain a d-dimensional zonotope.

It is in this sense that the convex outer adversarial polytope described by Kolter and Wong, with the choice of $\alpha = \frac{u}{u-l}$ is exactly equivalent to the zonotope attained by `DeepZ` if no dimensions were dropped in the zonotope. Hence, each coordinate-wise upper and lower bound attained by each is exactly equivalent. □

D Proof of Hardness of ReLU Programming

Theorem 4.1. *When $\mathcal{Z}$ is a zonotope, solving a ReLU program over $\mathcal{Z}$ is equivalent to a neural network verification problem for a 2-layer network, which is known to be NP-hard.*

Proof. Let $\mathcal{Z} = Z(c, E)$ be a zonotope in $\mathbb{R}^d$ with m columns in the generator matrix. Then we can construct an equivalent scalar-valued ReLU network, $f(y) = w_2^\top \sigma(W_1 y + b)$. We define w_2, W_1, b as follows:

$$W_1 = \begin{bmatrix} E \\ -E \end{bmatrix} \qquad b = \begin{bmatrix} c \\ -c \end{bmatrix} \qquad w_2 = \begin{bmatrix} c_1 + c_2 \\ -c1 \end{bmatrix}$$

such that $f(y) = c_2^\top \sigma(c + Ey) + c_1^\top (\sigma(c + Ey) - \sigma(-c - Ey))T$, where the ReLU programming objective is recovered by $x = \sigma(x) - \sigma(-x)$. Then we can set the input domain to $[-1, 1]^m$ to equate this verification problem to ReLU programming over $\mathcal{Z}$. The hardness result comes directly from Katz et al [4]. □

E 2-Dimensional Zonotopes

Now we turn our attention to 2-dimensional zonotopes, ultimately, we will arrive at a proof of Theorem 4.2 and Corollary 5.1. Throughout, we consider a 2-dimensional zonotope $Z(c, E)$, where E is assumed to have m generators. Further we will assume that the generators are in general position, i.e. two generators are colinear. If this is the case they can be combined into a single generator without changing the set represented by the zonotope. The strategy of this proof will follow in several parts. First we start with a preliminary theorem that holds for zonotopes of all dimension, which describes a relationship between the vertices of the zonotope and the faces of the generating hypercube. Then we show how every two-dimensional zonotope has $2m$ vertices and therefore $2m$ edges. Next we prove that each edge corresponds to exactly one generator, and conversely, every generator corresponds to exactly 2 edges. Finally we describe the procedure we use to enumerate these vertices in a sorted fashion, and compute axis-crossings, origin-containment, and extend this to the intersection of a zonotope and rectangle. We will commonly make use of the notion of the *upper and lower convex hulls* of a 2-dimensional shape. The upper hull of a convex set is the smallest concave function that is larger than the convex hull, and the lower convex hull of a convex set is the largest convex function that is smaller than the convex hull; it is helpful to think of this as the 'upper half' and 'lower half' of the convex hull.

Lemma E.1. *Every vertex v of a zonotope with m generators has a corresponding face of the m-dimensional ℓ_∞ ball that maps to that vertex.*

Proof. Consider any zonotope $\mathcal{Z} = Z(c, E)$, which can be equivalently viewed as the affine mapping $y \to c + Ey$ applied to every element in the set $\{y \mid y \in [-1, 1]^m\}$. Consider any vertex v of $\mathcal{Z}$, which by definition must have some vector a (in the polar cone of that vertex), such that $a^\top v < a^\top z$ for all $z \neq v$ in $\mathcal{Z}$. In other words, v is the argmin of some linear program with feasible set $\mathcal{Z}$. Equivalently, the set of y such that $c + Ey = v$ are the argmin of some linear program over the ℓ_∞ ball in $\mathbb{R}^m$. This implies that the set $\{y \mid c + Ey = v\}$ is a face of the ℓ_∞ ball. Note that face here includes 0-faces such as vertices of the ℓ_∞ ball, as well as higher dimensional faces. □

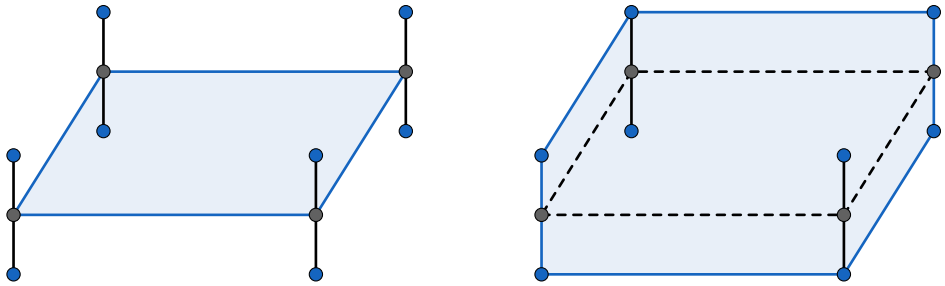


Figure 5: Pictorial aid for Lemma E.2. We consider the following 2-dimensional shapes. The parallelogram on the left is the base case – a zonotope with 2 generators, denoted by the convex hull of the gray vertices only. By incorporating a new vertical generator, the new candidate vertices are colored blue. Taking the convex hull of the blue vertices, we arrive at the 2-dimensional irregular hexagon on the right. Here we see that only the vertices on the left and right endpoints of the upper convex hull yield 2 vertices, whereas the other original vertices only generate 1 new vertex.

Lemma E.2. *A 2-dimensional zonotope with m generators in general position has exactly $2m$ vertices and edges.*

Proof. We proceed by induction. First note that a zonotope with a two generators is a parallelogram in $2d$ and has 4 vertices and edges. Now assume we have a zonotope $\mathcal{Z}_k$ with k generators, and by the induction hypothesis, $2k$ vertices. Now assume without loss of generality that we add the $k+1^{th}$ generator which is the vector $g := [0, 1]$: we are always able to rotate and scale the zonotope such that this is the case, and invert this operation afterwards. If $V_k = \{v_1, \ldots, v_{2k}\}$ is the set of vertices of $\mathcal{Z}_k$, then $\mathcal{Z}_k$ can be equivalently represented as $\text{conv}(V_k)$. Because adding a generator to a zonotope is the minkowski sum of a segment, $\mathcal{Z}_{k+1}$ can equivalently be written as $\text{conv}(\tilde{V_{k+1}})$, where $\tilde{V_{k+1}}$ is the set of points $\{v_1 + g, v_1 - g, \ldots v_{2k} + g, v_{2k} - g\}$. We now show that only $2k + 2$ of these $4k$ points are vertices. To see this, consider just the vertices of the convex upper hull of $\mathcal{Z}_k$. Because zonotopes are centrally symmetric, there are exactly $k + 1$ of these points. By choosing g to be $[0, 1]$, we see that for every v_i in the convex upper hull, $v_i + g$ is a vertex of $\mathcal{Z}_{k+1}$. We also see that unless v_i is also a point of the convex lower hull of $\mathcal{Z}_k$, $v_i - g$ is not a vertex of $\mathcal{Z}_{k+1}$, because by definition there is an interval vertically beneath v_i such that the entire interval, with g subtracted, is also in $\mathcal{Z}_{k+1}$. Hence for $k - 1$ points of the convex upper hull, we can eliminate $v_i - g$ from the candidate vertex set. We can repeat a similar procedure for the convex lower hull. Hence we are able to eliminate $2k - 2$ points from the $4k$ candidate vertices, leaving exactly $2k + 2$ vertices in $\mathcal{Z}_k$ as desired. Finally notice that every 2-d polytope with $2k + 2$ vertices has exactly $2k + 2$ edges. □

Lemma E.3. *If $\mathcal{Z}$ is a 2-dimensional zonotope in general position, then every edge corresponds to exactly one generator. Conversely, every generator corresponds to exactly two edges.*

Proof. Pick any edge of $\mathcal{Z}$ and without loss of generality, assume that zonotope is oriented such that the left endpoint of this edge, v_l, is the left-most point of the zonotope, neither edge connected to that endpoint is vertical, and the right endpoint of this edge, v_r is above v_l. Note that this can always be done and then undone later. Next assume that all generators are oriented such that the x-coordinate of each generator is nonnegative. This follows because any generator can be negated without changing the zonotope. From the previous lemma, it can be shown that for zonotope $\mathcal{Z} = Z(c, E)$, the vector $y = -\vec{1}$ satisfies $c + Ey = v_l$. Then, by moving in y-space by swapping any coordinate of y from -1 to $+1$, we may end up at a new vertex. We can now sort the generators by slope, in terms of $\frac{g(2)}{g(1)}$, where $g(i)$ refers to the i^{th} coordinate of generator g. Suppose $g^{(j)}$ is the generator with the largest slope. Then by interpolating from y to $y + 2e_j$ we traverse along an edge of the zonotope. Indeed, there exists no other direction in y-space in which we could move that corresponds to a larger slope-increase in z-space. This is essentially a gift-wrapping procedure. Thus, the edge spanned by v_l, v_r corresponds exactly to $g^{(j)}$. To show that every generator corresponds to at least two edges, this follows easily from the fact that zonotopes are centrally symmetric. To show that every generator corresponds to exactly two edges, we can apply Lemma E.2 to show by the pigeonhole principle, no generator can correspond to more than two edges. □

Now with these lemmas in hand, we are prepared to solve the main 2-d vertex enumeration theorem.

Theorem 4.2. *If $Z(c, E)$ is a 2-dimensional zonotope with m generators, Z has $2m$ vertices and the set of all vertices, axis crossings, and the containment of the origin can all be computed in $O(m \log m)$ time.*

Proof. First we describe how to enumerate all vertices of $Z(c, E)$ in $O(m \log m)$ time. This procedure is a simple extension of Lemma E.3. Recall, in that lemma, we were able to find the left-most vertex by solving a linear program minimizing the $e_1^\top z$ over $Z(c, E)$. Then we were able to find the next vertex, starting from the left and going clockwise, by finding the generator with the largest slope, once generators had been appropriately sign-normalized. We can simply continue this procedure, and from the right-endpoint of the edge enumerated in Lemma E.3, the next clockwise vertex is attained by traversing the generator with the next highest slope. In this way, we can sort the entire set of m generators by their slope in decreasing order and iteratively enumerate the vertices, where the $(i+1)^{th}$ vertex v_{i+1} is simply $v_i + 2g_i$, where v_i is the i^{th} vertex and g_i is the i^{th} generator, in the sorted order. This process generates the $m+1$ vertices of the entire convex upper hull of $Z(c, E)$ in left-right order. To attain the vertices of the lower convex hull, we rely on the centrally symmetric property of zonotopes: negate each vertex, add $2 \cdot c$, and reverse the order of this list to generate the $m+1$ vertices of the lower convex hull in right-left order. By concatenating these two lists, and removing duplicates at the endpoints of each convex hull, we are able to generate all $2m$ vertices of the zonotope in clockwise order. The runtime of this algorithm is the time to compute the first vertex and then the slope of each generator, each of which is $O(m)$ time. Then the generator slopes must be sorted in $O(m \log m)$ time. Iterating through this list and adding each generator to the current vertex requires $O(m)$ constant time steps. Computing the lower convex hull only requires $O(m)$ time as well, so the whole procedure requires $O(m \log m)$ time, dominated by the cost of sorting the generators by their slopes.

Now we demonstrate how it takes an additional $O(m)$ time to compute the points on the zonotope where either $x = 0$ or $y = 0$. We consider only the $x = 0$ case, without loss of generality. Since we are given the vertices in clockwise order, it is trivial to walk through the vertex list and report any pair of vertices for which $v_i(1) \cdot v_{i+1}(1) < 0$. If this is the case, then the edge between v_i and v_{i+1} crosses the y-axis. Since we have both endpoints on this edge, it requires constant time to compute the point along that line segment for which $x = 0$. A similar procedure can be applied for the $x = 0$ case. Finally, notice that the zonotope only contains the origin if it crosses both the x-axis and y-axis in two places. □

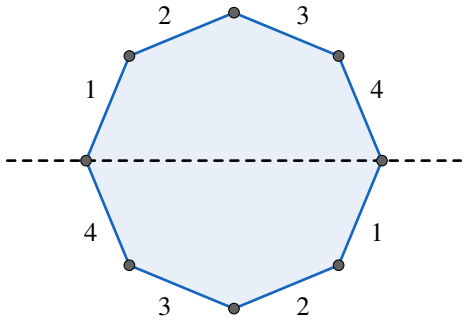


Figure 6: Pictorial aid for Theorem 4.2. By only considering the edges on the upper convex hull (above the dotted line), we sort the edges according to slope and step along them until we reach the end of the upper convex hull. The convex lower hull can be computed easily by central symmetry of zonotopes.

From this procedure, it is now trivial to consider the intersection of a 2-d zonotope and a rectangle.

Corollary 5.1. *Given a 2-d zonotope with m generators, the set of candidate optimal points to a ReLU program has cardinality no greater than $(2m + 9)$ and is computable in time $O(m \log m)$.*

Proof. First we provide a bound on the cardinality of the set of vertices of the intersection of a 2-d zonotope and a rectangle. The set of candidate vertices is exactly: the vertices of the zonotope, the

vertices of the rectangle, and any intersections of edges of the rectangle and zonotope. Here it is easiest to refer the pictorial aid in Fig. 7. For any rectangle vertex contained in the zonotope, there is necessarily a zonotope vertex not contained in their intersection. For any rectangle vertex not contained in the zonotope, it is possible that up to two new vertices are introduced in the intersection. This means that each vertex of the rectangle can add a net $+1$ to the number of vertices of their intersection. In general, the number of vertices of the intersection of two 2-d polygons is bounded by the sum of the number of vertices of each. Then the candidate optima for a ReLU program over this set is exactly this set of vertices, with cardinality at most $2m + 4$, and any extra axis crossings, and the origin. This brings the set of candidate optima to no more than $2m + 9$.

Next, we describe how to compute this set of candidate optima. We assume that we have obtained the $2m$ vertices of the zonotope in $O(m \log m)$ time. We now check containment of each of the candidate vertices. It requires constant time to compute point-containment within a rectangle, so we can eliminate any zonotope vertices not contained in the box in $O(2m)$ time. It is more complicated to compute point-containment within a zonotope. In general this requires a linear program, but since we already have the vertex list oriented clockwise, we are able to leverage this to compute the set of rectangle vertices contained in the zonotope as well as any edge-edge crossings in $O(m)$ time. The idea is similar to the idea used to compute axis crossings in Theorem 4.2.

The general procedure is to fix an axis and value and shoot a line through that value. For example, we consider the $x = a$ line and compute the intersection of the set $\{(x, y) \mid x = a\}$ with $Z(c, E)$. We can do this by finding the y-coordinates of the edges of $Z(c, E)$ that cross this vertical line. This amounts to a single scan through the oriented vertex list of the zonotope, and yields an interval $[l_z, u_z]$ or `None`. We perform this line-shooting procedure at most 6 times. Once for the left-bound, right-bound, top-bound, and bottom-bound of the rectangle, and if the rectangle itself crosses the x or y axis, we line-shoot the $x = 0$ line or $y = 0$ lines accordingly. For each line-shoot, we are able to compare the interval defined by the box $[l_b, u_b]$ and the line-shoot interval $[l_z, u_z]$. The intersection of this interval can be used to determine which box vertices and edge-edge intersections are contained within the intersection of the zonotope and rectangle. Overall this requires a constant number of calls to a subroutine running in $O(m)$ time, so the overall runtime of this procedure is still dominated by the sort time from the zonotope vertex enumeration, i.e. $O(m \log m)$.

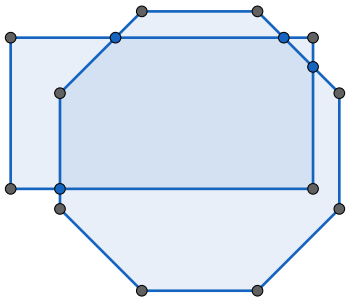


Figure 7: Pictorial aid for the proof of Lemma 5.1. A zonotope (octagon) is intersected with a rectangle. Some rectangle vertices are not contained within the zonotope and can be eliminated, and vice versa. The rectangle vertex on the upper right introduces a net $+1$ vertices to the intersection.

□

E.1 Partitioning Heuristics

Here we describe the heuristics used to fragment a higher dimensional zonotope into two-dimnesional pieces. We certainly could do this randomly, but we have found some minor performance gains can be achieved by cleverly choosing this partitioning. The key idea for each of these is to choose pairs of dimensions yielding zonotopes that would suffer greatly from being relaxed into a rectangle. Recall that for a two-dimensional zonotope, a rectangle is attained when each generator is colinear with an elementary basis vector. Our two partitioning styles are as follows:

- **Similarity heuristic:** Here we assign each pair of dimensions a scalar-valued score that roughly resembles how far each generator column is from an elementary basis vector. These pairwise scores are encoded in a $d \times d$ matrix. Each row of this matrix is sorted in descending

order, and the partitions are formed by looping over the rows of the matrix and choosing the first available index, that has not already been chosen by a prior pair. The scoring function here, for dimensions i, j is the dot product of the absolute values of the *rows* of the generator matrix. Hence the scoring matrix can be computed by the matrix-matrix multiplication $|E| \cdot |E|^\top$. For zonotopes of very large dimension, this method requires the multiplication of two $(d \times m)$ matrices, and then the sort-and-choose method takes $O(d^2)$ time in the worst case. However since we only have to do this one time, it is often worth it to apply this heuristic on networks with intermediate networks with moderate dimension, or those without spatial information imbued by convolutional layers.

- **Spatial or depthwise heuristic:** For convolutional networks, the structure of convolutional operators can be applied to choose partitionings. In the spatial partitioning heuristic, we assign each coordinate its pair by choosing a coordinate adjacent to it in the feature map. In the depthwise partitioning heuristic, we assign each coordinate its pair by choosing a coordinate in the same location on the feature map but in a different channel. The intuition here is that random partitioning here would often choose coordinates that have disjoint receptive fields and would yield zonotopes that are rectangles. These heuristics much more quickly than the similarity heuristic and are amenable to larger convolutional nets.

F Experimental Details

F.1 Computation Environment

Computations for this paper were performed using the idle resource queue of computing cluster. The cluster uses Intel Xeon Gold 6230R and 6230R CPUs and mixture of NVIDIA 2080 Ti, Quadro RTX 6000, and Telsa A40 GPUs. Due to our usage of the idle resource queue, we ran on a variety of different CPUs and GPUs depending on which resources were being underutilized on the cluster. Our job requirements were 5 CPU cores, 20 GB of RAM, and 1 GPU of any kind per job. Despite our timing information not being representative of a single hardware configuration, we believe we can still make valid comparisons using it, as we ensured that we ran all methods for a particular image on the same node.

F.2 Network Architectures

F.2.1 MNIST Networks:

We consider three different architectures trained on the MNIST dataset. Each is randomly initialized, and trained adversarially for 25 epochs using the Adam optimizer and standard hyperparameters to update network weights [34]. The adversarial training was perfromed using 10 iterations of PGD with an ℓ_∞ adversarial radius of $\epsilon = 0.1$ [35]. The architectures are as follows, where $\texttt{Conv2D}(in, out, k, s, p)$ refers to a convolutional layer with in input channels, out output channels, a square kernel of size k, a stride of s, and padding p. The Wide and Deep network architecture choices were taken from [11], though the FFNet was our choice for a fully connected network. We trained these networks ourselves.

- `MNIST FFNet:` (960 ReLU neurons)
  1. $\texttt{Linear}(784, 512) \to \texttt{ReLU}$
  2. $\texttt{Linear}(512, 256) \to \texttt{ReLU}$
  3. $\texttt{Linear}(256, 128) \to \texttt{ReLU}$
  4. $\texttt{Linear}(128, 64) \to \texttt{ReLU}$
  5. $\texttt{Linear}(64, 10)$
- `MNIST Wide:` (4804 ReLU neurons)
  1. $\texttt{Conv2D}(1, 16, 4, 2, 1) \to \texttt{ReLU}$
  2. $\texttt{Conv2D}(16, 32, 4, 2, 1) \to \texttt{ReLU} \to \texttt{Flatten}$
  3. $\texttt{Linear}(1568, 100) \to \texttt{ReLU}$
  4. $\texttt{Linear}(100, 10)$
- `MNIST Deep:` (5196 ReLU neurons)
  1. $\texttt{Conv2D}(1, 8, 4, 2, 1) \to \texttt{ReLU}$
  2. $\texttt{Conv2D}(8, 8, 3, 1, 1) \to \texttt{ReLU}$

3. `Conv2D`$(8, 8, 3, 1, 1) \rightarrow$ `ReLU`
4. `Conv2D`$(8, 8, 4, 2, 1) \rightarrow$ `ReLU` $\rightarrow$ `Flatten`
5. `Linear`$(392, 100) \rightarrow$ `ReLU`
6. `Linear`$(100, 10)$

F.2.2 CIFAR-10 Networks

For the CIFAR-10 networks, we directly utilized the models and weights from prior work [9][†]. Specifically, we considered the models trained both using SGD and adversarially using PGD with the architecture:

- `CIFAR WIDE:` (6244 ReLU neurons)
  1. `Conv2D`$(3, 16, 4, 2, 1) \rightarrow$ `ReLU`
  2. `Conv2D`$(16, 32, 4, 2, 1) \rightarrow$ `ReLU` $\rightarrow$ `Flatten`
  3. `Linear`$(2048, 100) \rightarrow$ `ReLU`
  4. `Linear`$(100, 10)$

F.3 Hyperparameter Choices

Here we describe the hyperparameter choices for both our model and competing methods. Wherever possible, code from prior works was utilized, with minimal tuning applied. We found the codebase associated with the Active Set paper to be immensely helpful here [9]. Because we changed our partitioning hyperparameters slightly between networks, we describe each hyperparameter choice explicitly. Scripts to run each experiment are contained with the attached codebase. For all experiments, we employ Gurobi as MIP-solver [36].

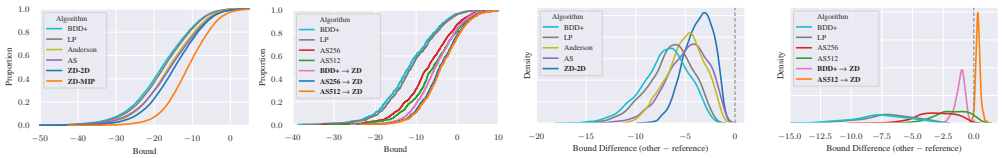
Single-stage setting:

- **BDD+**: For the optimal proximal method, we apply the default parameters where 400 iterations were applied with 2 inner steps and η ranging from 1 to 5.
- **AS**: 1000 iterations of AS were increased to 1000 to match our approach, where all other hyperparameters were kept at their default values.
- **ZD (MNIST)**: For the MNIST networks, we initialized with the KW dual variables, used the 'similarity' partitioning heuristic for the 2-d zonotopes, and used 1000 iterations of Adam with default β parameters. The learning rate was initialized at $\eta = 0.01$, and decayed by a multiplicative 0.75 factor every 100 iterations.
- **ZD (CIFAR)**: For the CIFAR networks, we initialized with the KW dual variables, used the 'spatial' partitioning heuristic for the 2-d zonotopes, and used 1000 iterations of Adam with default β parameters. The learning rate was initialized at $\eta = 0.0025$ and decayed by a multiplicative 0.5 factor every 200 iterations.
- **ZD-MIP**: For all single stage experiments, for every network, we ran the exact same setting as ZD, but merged partitions of the final layer only into zonotopes of dimension 20 and evaluated the dual function. The only exception here is in MNIST FFNET, where only the final layer was merged to zonotopes of dimension 16.

Stagewise Setting: In this setting, we kept most hyperparameters the same as in the single-stage setting. The notable exception is a change in the number of iterations in the AS method. We found that 1000 iterations would frequently cause CUDA memory errors, and to the best of our understanding, these methods weren't run in a stagewise setting in [9]. Because these methods are allowed to run for a longer time, we increased the number of iterations dual ascent was performed for, as well as the partitioning at the end. Essentially, these were tuned slightly in order to reliably prove tighter bounds such that AS256-ZD was both faster and tighter than AS512. We summarize the iteration and partitioning changes here :

- **MNIST FFNET**: We initialize as before, using the specified box bounds, and then run 2000 iterations of gradient ascent, with an initial learning rate of 0.0025, decaying by a factor of 0.75 every 400 epochs. Because this method is relatively smaller, we increase the final

[†] https://github.com/oval-group/scaling-the-convex-barrier



(a) Single-stage CDF (b) All-stage CDF (c) Single-stage difference (d) All-stage difference

Figure 8: Full plots for the MNIST FFNet network (zoom in for detail). (a) The CDF plot for the single-stage setting. (b) The CDF plot for the multi-stage setting. (c) The difference plot for the single-stage setting, where the reference method is ZD-MIP. (d) The difference plot for the multi-stage setting, where the reference method is AS256→ZD.

layer partition size to 32 and run 20 iterations of dual ascent. Finally, we expand all other 2-dimensional partitions to zonotopes of dimensin 16 and evaluate the dual.

- **MNIST WIDE/DEEP**: We initialize and run dual ascent over the 2-d zonotopes as in MNIST FFNet. We then increase partition sizes to 16 for the penultimate layer and 20 for the final layer and evaluate the dual function once.

Ablation Study Details: For the ablation study, we only evaluate on the MNIST FFNet. We initialize either with the `DeepZ` bounds, the `DeepZ` bounds augmented via IBP, or the zonotopes augmented by the stagewise BDD+ bounds. Then we partition the zonotopes using the 'similarity' heuristic and run 1000 iterations of gradient ascent with an initial learning rate of 0.01, decaying by a factor of 0.75 every 100 iterations. We increase the partition size to 32 on the final layer only and evaluate the dual. We give results complete with error bars in Table 4.

Table 4: Ablation study for each of phase of ZonoDual. We evaluate the bound and runtime after completion of the Initialization phase (init), Iteration phase (iter), and Evaluation (eval) phases, averaging over the first 1000 examples from MNIST. We report three initialization techniques: DeepZ preactivation bounds, and DeepZ augmented with IBP or BDD+ hyperbox bounds. The computed lower bounds are reported relative to the LP relaxation (a larger number means a tighter bound).

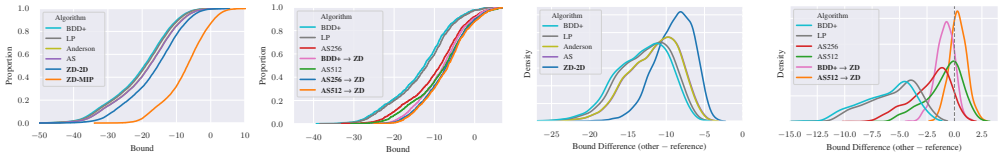
Init\Phase	Init	Iter	Eval
DeepZ	-3.9 ± 1.1	2.9 ± 1.0	7.7 ± 2.6
Runtime (s)	0.0047 ± 0.0004	5.6 ± 0.3	7.0 ± 1.5
DeepZ + IBP	-3.9 ± 1.1	2.9 ± 1.0	7.7 ± 2.6
Runtime (s)	0.024 ± 0.002	5.6 ± 0.2	6.9 ± 1.4
DeepZ + BDD+	5.3 ± 1.8	6.4 ± 1.9	10.1 ± 3.2
Runtime (s)	4.9 ± 0.3	10.5 ± 0.5	11.5 ± 1.4

G Additional experimental results

Here we have included data from experiments that did not fit in the main paper. Unless otherwise stated, the setup and hyperparameters are chosen as in the main paper and in Appendix F.3.

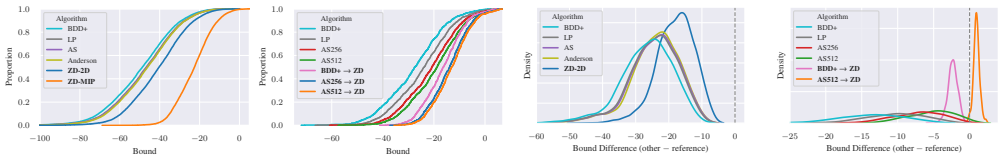
We present two styles of plots in Figs. 8 to 11. The cumulative distribution plots present the quantity of examples considered that yield a bound less than a specified value. Curves further to the right are better here. The difference plots present the distribution of bound differences relative to a reference method, which is represented by the dotted vertical line. Curves further to the right provide better bounds, and portions of the curve greater than 0 indicate the proportion of examples that the method beats the reference. For example, in the single-stage setting, the curves represent the difference between the technique in the plot and the ZD-MIP method. In the multi-stage setting, we use AS256→ZD as the reference.

We also report numerical timing results in Tables 5 and 6 and bound statistics in Tables 7 and 8.



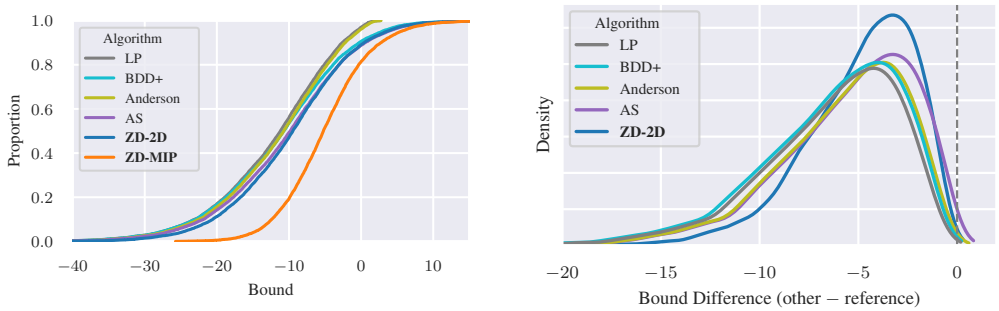
(a) Single-stage CDF (b) All-stage CDF (c) Single-stage difference (d) All-stage difference

Figure 9: Full plots for the MNIST Wide network (zoom in for detail). (a) The CDF plot for the single-stage setting. (b) The CDF plot for the multi-stage setting. (c) The difference plot for the single-stage setting, where the reference method is ZD-MIP. (d) The difference plot for the multi-stage setting, where the reference method is AS256→ZD.



(a) Single-stage CDF (b) All-stage CDF (c) Single-stage difference (d) All-stage difference

Figure 10: Full plots for the MNIST Deep network (zoom in for detail). (a) The CDF plot for the single-stage setting. (b) The CDF plot for the multi-stage setting. (c) The difference plot for the single-stage setting, where the reference method is ZD-MIP. (d) The difference plot for the multi-stage setting, where the reference method is AS256→ZD.



(a) Single-stage CDF (b) Single-stage difference

Figure 11: Plots for the CIFAR-10 network: (a) CDF plots repeated from the main paper. (b) the difference plots where the reference method is the ZD-MIP method.

Table 5: Time required by each algorithm across a set of four networks in the single-stage setting. Numbers reported are mean and standard deviations in seconds.

Alg\Net	MNIST FFNet	MNIST Deep	MNIST Wide	CIFAR SGD
BDD+	1.7 ± 0.1	2.5 ± 0.2	1.5 ± 0.1	1.5 ± 0.1
LP	6.2 ± 1.1	4.8 ± 0.5	5.5 ± 0.5	6.2 ± 0.6
ZD-2D	5.7 ± 0.3	10.9 ± 0.5	10.2 ± 0.5	7.6 ± 0.3
AS	10.7 ± 0.9	16.9 ± 1.2	10.4 ± 0.7	10.2 ± 0.8
ZD-MIP	6.0 ± 0.3	23.1 ± 12.4	14.7 ± 4.6	11.0 ± 3.1
Anderson	13.1 ± 2.5	52.4 ± 27.5	52.4 ± 19.2	21.3 ± 4.6

Table 6: Time required by each algorithm across a set of three networks in the all-stage setting. Numbers reported are mean and standard deviations in seconds.

Alg\Net	MNIST FFNet	MNIST Deep	MNIST Wide
BDD+	5.2 ± 0.5	15.4 ± 0.4	8.3 ± 0.3
BDD+ → ZD	52.0 ± 31.2	51.7 ± 14.0	36.4 ± 6.8
AS256	29.5 ± 0.9	138.9 ± 0.9	45.4 ± 0.6
AS256 → ZD	70.0 ± 27.5	174.1 ± 13.6	73.3 ± 6.6
AS512	61.6 ± 1.8	289.3 ± 1.7	91.3 ± 0.9
AS512 → ZD	100.3 ± 26.6	324.1 ± 13.5	119.2 ± 6.9
LP	90.0 ± 16.9	2132.2 ± 494.4	387.1 ± 65.5

Table 7: Bounds obtained by each algorithm across a set of four networks in the single-stage setting. Numbers reported are average and standard deviation of the bound relative to **ZD-2D** over the first 1000 images from the respective validation set.

Alg\Net	MNIST FFNet	MNIST Deep	MNIST Wide	CIFAR SGD
BDD+	-3.4 ± 1.1	-8.6 ± 2.7	-4.2 ± 1.3	-1.5 ± 0.9
LP	-2.8 ± 1.0	-6.6 ± 2.4	-3.7 ± 1.3	-1.2 ± 0.9
Anderson	-1.3 ± 0.7	-5.9 ± 2.3	-2.6 ± 1.2	-0.7 ± 0.8
AS	-1.2 ± 0.9	-6.3 ± 2.4	-2.6 ± 1.2	-0.5 ± 0.8
ZD-2D	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
ZD-MIP	3.7 ± 1.4	18.1 ± 5.8	9.0 ± 2.6	4.9 ± 2.8

Table 8: Bounds obtained by each algorithm across a set of three networks in the all-stage setting. Numbers reported are average and standard deviation of the bound relative to **BDD+ → ZD** over the first 1000 images from the MNIST validation set.

Alg\Net	MNIST FFNet	MNIST Deep	MNIST Wide
BDD+	-6.5 ± 2.1	-11.5 ± 4.5	-5.4 ± 2.2
LP	-5.9 ± 2.0	-8.2 ± 4.0	-4.7 ± 2.1
AS256	-2.0 ± 1.5	-5.0 ± 3.4	-1.4 ± 1.7
AS512	-0.7 ± 1.3	-3.1 ± 3.1	-0.1 ± 1.5
BDD+ → ZD	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
AS256 → ZD	1.1 ± 0.4	2.3 ± 0.7	0.9 ± 0.8
AS512 → ZD	1.5 ± 0.5	3.3 ± 0.9	1.2 ± 0.8