nnenum: Verification of ReLU Neural Networks with Optimized Abstraction Refinement

Stanley Bak[0000−0003−4947−9553]

Stony Brook University
Stony Brook NY 11794, USA
stanley.bak@stonybrook.edu

Abstract. The surge of interest in applications of deep neural networks has led to a surge of interest in verification methods for such architectures. In summer 2020, the first international competition on neural network verification was held. This paper presents and evaluates the main optimizations used in the `nnenum` tool, which outperformed all other tools in the ACAS Xu benchmark category, sometimes by orders of magnitude. The method uses fast abstractions for speed, combined with refinement through ReLU splitting to increase accuracy when properties cannot be proven. Although the abstraction refinement process is a classic approach in formal methods, directly applying it to the neural network verification problem actually reduces performance, due to a cascade of overapproximation error when using abstraction. This makes optimizations and their systematic evaluation essential for high performance.

Keywords: Neural Network Verification · ReLU · ACAS Xu.

1 Introduction

Deep neural networks are powerful machine learning methods that can provide accurate approximations to functions learned from data. One downside of neural networks is that they are often unexpectedly sensitive to small targeted changes in the inputs. This is most well-known in the context of perception systems, where changes that a human cannot see can sometimes cause the systems to misclassify an image [13]. Such adversarial examples attacks [24] can also be applied to decision-making networks, where the system can fail due to what could essentially be sensor noise [15].

In order to apply neural networks to safety-critical and even mission-critical systems, stronger assurances are usually desired. One approach to do this involves developing algorithms to reason formally over the function computed by a neural network. The *open-loop neural network verification problem* tries to prove properties over the inputs and outputs of a network. For example, given interval bounds on each input, can you prove the maximum output of the network does not change?

The most studied version involves neural networks with ReLU activation functions, for which many algorithms tools have been proposed. The biggest

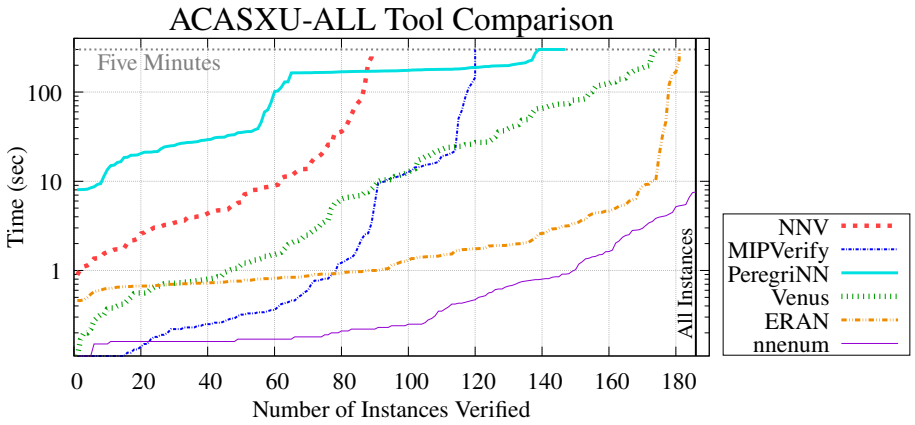

Fig. 1. At VNN-COMP 2020, the `nnenum` tool verified each ACAS Xu benchmark in under 10 seconds. (image from competition report [16]).

problem with these methods is often scalability, where networks cannot be verified in a timely manner. Analysis speed also affects the size of networks that can be analyzed in a reasonable amount of time. Speed improvements to a verification algorithm will mean that analysis of larger networks becomes more feasible.

This last summer in July 2020, the first international competition on neural network verification, VNN-COMP 2020, was held [16]. There were two categories of benchmarks evaluated: (i) image classification benchmarks that have generally larger numbers of inputs and network sizes, and (ii) control benchmarks which have less inputs and are generally smaller. The second category consisted of 184 benchmarks taken from the well-studied ACAS Xu system [18].

The results from the control category are shown in the cactus plot in Figure 1. The `nnenum` tool was the fastest for this category, sometimes by orders of magnitude (note the y-axis is log scale). Although the comparison is imperfect—the participants each ran their own tool on their own hardware—the performance difference cannot be explained by hardware alone. Several new algorithmic optimizations were necessary to achieve `nnenum`'s performance.

This paper outlines and evaluates the main optimizations used in `nnenum`. Although `nnenum` also performed well on the image classification category, we focus our measurements in this paper on the ACAS Xu benchmarks, as a different set of optimizations was used for the larger perception networks. The next section provides a brief description of the high-level algorithm used for neural network verification of ReLU networks and the ACAS Xu benchmarks. A presentation, evaluation, and tool implementation of several optimizations is the main contribution of this work, and is presented in Section 3. The paper finishes with related work and a conclusion.

2 Overview

In this section, we provide an overview of the verification problem, benchmarks, and algorithm used in our evaluation in the next section.

2.1 Verification Problem

Our goal in this work is to efficiently solve the open-loop neural network verification problem. In this problem, we assume we are given a set defined with linear constraints over the network inputs $\mathcal{I}$ and a second set of unsafe states defined with linear constraints over the outputs $\mathcal{U}$. The network we analyze consists of fully connected layers with rectified linear unit (ReLU) activation functions, although the method has also been extended to work with convolutional layers, max pooling layers, and others [33,35]. ReLU activation functions are defined as $ReLU(x) = \max(x, 0)$, so that the neural networks compute a piecewise linear function. The verification problem is to find an input $i \in \mathcal{I}$ which when executed on the neural network produces an unsafe output $u \in \mathcal{U}$, or prove no such input exists. The execution semantics of fully-connected neural networks is well-known and reviewed in many papers [3], so we do not restate it here.

2.2 ACAS Xu Benchmarks

In this work, we focus our evaluation on the well-known ACAS Xu benchmarks [18]. These benchmarks provide several open-loop specifications for neural networks that are intended to compute a lossy compression of a large lookup table containing actions to prevent collisions among aircraft [17]. Each of the 45 neural networks has 300 neurons arranged in six fully-connected layers with ReLU activation functions. There are five inputs corresponding to the aircraft states, and five outputs corresponding to possible commands of the ownship aircraft to avoid collisions. Each property is defined using linear constraints over the inputs and outputs, matching the problem description in Section 2.1. The first four of these properties are applicable to all 45 networks, and we use these 180 benchmarks for our evaluation throughout this paper. In the original work with Reluplex [18], analysis times for these properties ranged from seconds to days, with some unsolved instances that presumably ran for days without producing a verification result. The ACAS Xu benchmarks are thus a mix of easy and difficult problems, some of which are `SAT` and some are `UNSAT`. Although the networks are small compared with image classification neural networks, they are similar enough to neural networks used in decision making and control to make them good benchmarks for verification algorithms.

2.3 Set Representations

The basis of our implementation is a reachability algorithm [34] based on the linear star set data structure [10], which we just call *star sets* in this paper.

Star sets are way to represent a set of states in some Euclidean space $\mathbb{R}^n$. They are defined using two parts: (1) a half-space polytope ($\mathcal{H}$-polytope) in some m-dimensional space, and (2) an affine transformation that takes the m-dimensional $\mathcal{H}$-polytope to the n-dimensional space. Star sets are efficient for (i) computing high-dimensional affine transformations of sets, (ii) taking intersections with arbitrary linear constraints and (iii) optimizing using linear programming (LP). These are all the operations needed to analyze neural networks with fully-connected ReLU layers [3].

The algorithm we describe in the next subsection also makes use of zonotope overapproximations to improve efficiency. Zonotopes are a representation of a set of states in some Euclidean space $\mathbb{R}^n$ that encode an affine transformation of a *box* from some k-dimensional space. Zonotopes are efficient for affine transformations and very fast for linear optimization using a simple loop, but unlike star sets they cannot encode general intersections with linear constraints. Note that star sets are also efficient for linear optimization, but this requires invoking an LP solver so is often orders of magnitude slower than optimization with zonotopes.

2.4 Verification Algorithm Overview

We next provide a brief overview of the verification algorithm. A more complete description of the problem and algorithm are provided in an earlier work [3], where systematic analysis of optimization for *exact* analysis was performed. In the next section we will continue this approach of systematically analyzing optimizations, but instead for a modified version of the algorithm that uses abstractions to compute overapproximations of the set of possible outputs of a neural network. When the abstract system fails to verify the property, refinement is performed and the process repeats with a finer abstraction, all the way down to exact analysis if necessary.

The algorithm first represents the input set $\mathcal{I}$ using *both* a star set and a zonotope (called a prefilter zonotope in the earlier work). Each layer is then iterated over in the network, with an inner loop that iterates over each neuron in the layer. To go from the n_i values (dimensions) from one layer to the n_{i+1} values (dimensions) at the next layer, an affine transformation is performed on the sets, defined by the $n_{i+1} \times n_i$ weights matrix and n_{i+1} bias vector in the neural network at layer i. At each neuron within a layer, the sets are optimized over to check the lower and upper bound of the possible inputs to the neuron. If the lower bound is greater than zero or the upper bound is less than zero—for which we say the input or neuron is *one-sided*—then the nonlinear ReLU activation function is equivalent to a linear transformation and can be directly performed on the star set and zonotope. For efficiency, the optimization is first done using the zonotope rather than the star set, which can sometimes prove the input is one-sided without using LP. If the input is not one-sided, for exact analysis, the set can be split in two along the boundary where the input to the neuron is equal to zero, using an intersection operation. For the zonotope, since intersections are not supported, the operation is generally ignored resulting in a strictly larger set (an overapproximation), although some accuracy control can be performed

through *zonotope domain contraction*. Zonotope domain contraction is the process of reducing the size of the box domain in the zonotope definition, so that it is a tighter overapproximation of the exact set represented by the star set. Since the affine transformations in the star set and zonotope are identical, zonotope contraction can be done by computing box bounds on the star set whenever an intersection is taken. When the sets are split in two, the algorithm proceeds recursively, performing the appropriate linear operation on the sets for the current neuron and proceeding to the next one. In the worst case, if splitting is done at every neuron, this can result in an exponential number of sets, which is not surprising as exact analysis for ReLU neural networks is NP-complete [18]. In practice, the problem is sometimes tractable, which means the choice of benchmarks is essential to evaluate an algorithm's practicality. This is a bit similar to how SAT solving is an NP-complete problem, but annual competitions since 1992 have pushed the limits of what is achievable in practice [6].

In the abstraction refinement version of the algorithm developed here, rather than immediately splitting the star set when neurons are not one-sided, we first overapproximate the ReLU region using the best convex relaxation in the neuron's input-output plane, sometimes called the triangle relaxation [12], which can be efficiently represented using a star set. This overapproximation is shown later in Figure 3 (top-left), and has been formally described in other work [34]. Use overapproximation rather than splitting has the advantage of avoiding worst-case exponential splitting, but also has trade offs. The overapproximation adds a dimension to the domain of the star set as well as extra constraints, which makes future optimizations using LP slightly slower computationally. Also, the result is an overapproximation, which means that future neurons analyzed may look like they split whereas in reality they are one-sided, leading to even more error that takes even longer to analyze. Error can cascade through the network in this way leading to what we call an *error snowball*. Further, once the set is propagated through all the layers, it is possible that the output set intersects the unsafe set $\mathcal{U}$, whereas the real set does not; there can be spurious counter-examples due to the overapproximation in the abstraction.

In the previous work with star sets [34], cases where the abstraction intersected the unsafe set $\mathcal{U}$ produced a verification result of `UNKNOWN`. Here, we instead propose to go back and perform a split on the first neuron where overapproximation was used—a refinement step. The process then proceeds recursively, again trying overapproximation at each remaining neuron that is not one-sided and then checking for intersection with the unsafe set $\mathcal{U}$. If there is still an intersection, refinement is performed again on a second neuron, and so on, until either the property is proved, an unsafe counter-example is found, or no remaining neurons exist where an overapproximation was done. Since the algorithm eventually reverts to exact analysis, it is sound and complete. It also has the potential to be faster, when abstractions can prove there is no intersection with $\mathcal{U}$. However, as will be shown in the next section, a direct implementation can actually be slower in many cases.

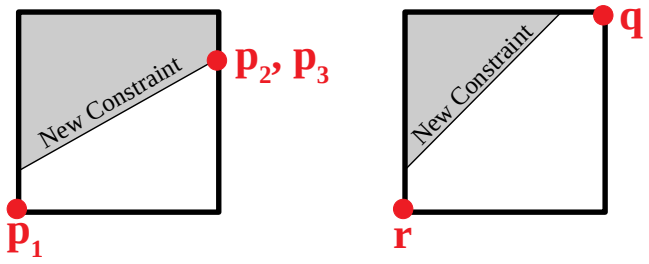

Fig. 2. The new zonotope domain contraction algorithm computes new box bounds by finding points p_1, p_2, and then p_3, requiring only three LPs instead of four (left). In the best case, computing bounds with the new approach only requires two LPs, regardless of the number of dimensions n, instead of $2n$ with the old approach (right).

3 Optimizations

We now present and evaluate several key optimizations that are essential to the performance of `nnenum`. We run each optimization on all the 180 ACAS Xu benchmarks described in the Section 2.2. We use a timeout of one minute for each benchmark and report the number of timeouts encountered as well as the total runtime. For lack of a better option, we count a timeout as one minute of total runtime, but keep in mind that the number of timeouts should weigh much more heavily when comparing approaches; some ACAS Xu benchmarks have been known to run for hours or days in other work. All measurements in this paper were done on a laptop running Ubuntu 20.04 with an Intel Xeon E-2176M CPU running at 2.70GHz containing 32 GB RAM. Our implementation of the algorithm is available online as part of the `nnenum` tool[1], including Dockerfile and scripts to run all the ACAS Xu benchmarks.

3.1 Zonotope Domain Contraction

As mentioned in the algorithm overview in Section 2, zonotopes are used to provide quick outer approximation of the bounds of the possible inputs to each neuron. If these are accurate, many neurons can be proven to be one-sided without needing to do linear programming in the star sets, which is significantly more efficient. For abstract analysis later, we will also consider using zonotopes, so the tightness of the zonotope is even more important.

Although zonotopes do not support intersections when splitting sets, their box domains can be reduced to contain the set after intersection which improves accuracy and reduces runtime. The problem of determining the tightest box domain can be solved using linear programming on the exact set represented by the star set. The most direct algorithm is to minimize and maximize along every input, so that a neural network with n inputs will require solving $2n$ LPs. In our

[1] https://github.com/stanleybak/nnenum

earlier work on optimizing exact analysis [3], we compared this direct approach, which we call `Old LP` here, with a `Single Loop` approach for zonotope domain contraction. In the `Single Loop` method, a single constraint was analyzed to see if it reduced the box bounds of the zonotope's domain. This does not produce tight box bounds, but it was shown to be faster overall as it did not require LP solving.

Here we re-examine domain contraction using a few further optimizations. First, rather than solving $2n$ LPs to determine box bounds, we leverage additional information available to the problem. Namely, we take advantage of the box bounds on the zonotope domain that are available prior to taking the intersection. The problem becomes to determine if adding one additional constraint to the star set's domain (an $\mathcal{H}$-polytope) changes its bounding box, given the original bounding box.

Our new zonotope contraction approach, called `New LP`, uses LP to optimize over the star set's $\mathcal{H}$-polytope with an optimization direction vector of all -1 values, to try to simultaneously find the lower bounds of all the variables in a single LP. In the best case, this point will have a value equal to the minimum value of each dimension in the old bounding box, and we can proceed to find upper bounds in the same fashion but using a vector of all 1 values. If only some of the variables achieved their earlier lower bound, the process repeats, setting to 0 in the optimization direction vector all of the variables that matched their earlier lower bound. To process stops when no new variables in the result get to their earlier lower bound, after which each variable that did not achieve their earlier lower bound is minimized individually using LP.

An example of this algorithm is shown in the 2-D case in Figure 2. In the figure, the box domain is intersected with a linear constraint, so that the upper grey region should be excluded from the resultant set. On the left side of the figure, the algorithm would first maximize in the direction of $\langle -1, -1 \rangle$ (minimizing the sum of x and y) finding point p_1, which matches the previous bounding box, so that we know that the lower bounds of both x and y are unchanged. Next, we maximize $\langle 1, 1 \rangle$ finding point p_2, which matches the old upper bound of x. The next maximization direction is $\langle 0, 1 \rangle$, which gives point p_3, which proves that the upper bound of y should be reduced, because the vector we optimized over only had a single non-zero entry. In this case, we found the box bounds using three LPs, whereas the original approach needed four.

In the best case, the algorithm can reduce the number of LPs to solve in an n-dimensional problem, corresponding to a neural network verification problem with n inputs, from $2n$ to 2. Such as case is shown in Figure 2 on the right, where one LP would first find point r and then a second LP would find point q, which is sufficient to show the box bounds have not been updated due to the new constraint. For this reason we expect the benefit to be greatest for this new zonotope domain contraction algorithm when the input space is high dimensional, such as in image recognition adversarial example verification benchmarks. Even in ACAS Xu, however, there are five inputs and so the savings can be beneficial.

Table 1. Domain Contraction Optimizations with Exact analysis.

Optimization	Timeouts	Runtime [sec]
`No Contraction`	50	3471.2
`Single Loop`	32	2778.8
`Old LP`	44	3220.9
`Old LP + Witnesses`	31	2680.1
`New LP`	31	2743.8
`New LP + Witnesses`	28	2549.0

A second optimization we consider for zonotope domain contraction is to track the witnesses of the box bounds. Whenever we compute box bounds, we can store the witness points in the set that exhibit the lower and upper bounds. When a new constraint is added, if the new constraint does not exclude the witness point from the set, which can be checked with simple dot product, then the bound for that variable/direction is unchanged. This witness approach can be applied to both the old contraction algorithm (`Old LP + Witnesses`) and new proposed contraction algorithm (`New LP + Witnesses`). For example, consider again the right side of Figure 2. If p and r are the witness points exhibiting the box bounds, then we can quickly check that the new constraint does not eliminate p or r from the set. Thus, we can prove the box bounds have not changed, without any need for LP solving.

The results for the various zonotope domain contraction options, in addition to a `No Contraction` approach, are shown in Table 1. Both the newer algorithm and tracking witnesses improve performance. Although the `Single Loop` approach is faster than the `Old LP` method, using the `New LP` method, especially when tracking witnesses is even faster. All zonotope contraction methods outperform `No Contraction`. For the remaining optimization in this paper, we use zonotope domain contraction with the `New LP + Witnesses` approach.

3.2 Direct Abstraction Refinement

We next examine direct abstraction refinement approaches. As described earlier, these algorithms first use an abstraction to try to prove the property by overapproximating all the ReLU functions when individual neurons are not one-sided. If the property cannot be proven, the computation backtracks to the first ReLU that was an overapproximation and the set is instead split. Each of the two sets is then processed recursively, again trying overapproximations at the next neuron which could split.

If the ith neuron in a layer has an input with lower bound $l_i < 0$ and upper bound $u_i > 0$, various overapproximations are possible for the output of the neuron which do not require splitting. Figure 3 shows four possibilities, where the input to the neuron is on the x axis and the output is on the y axis. The triangle overapproximation (top-left) is the best convex relaxation on the

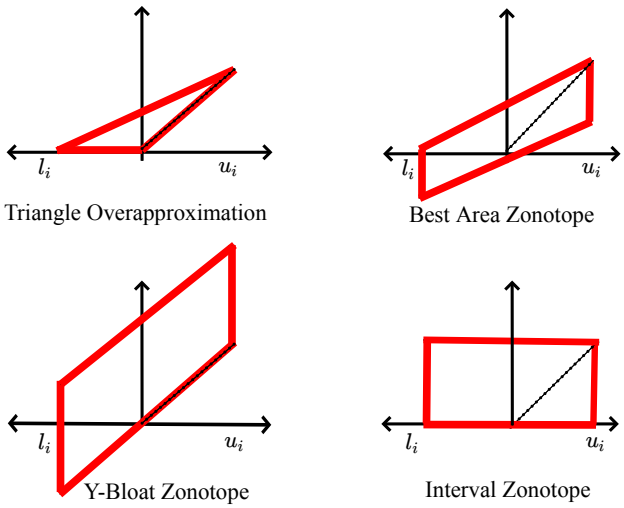

Fig. 3. When the inputs to some neuron i are bounded between l_i and u_i, many overapproximations can provide efficient abstractions of the nonlinear ReLU activation function, $ReLU(x) = \max(x, 0)$. In each plot, the x axis is the input to the neuron, and the y axis is the output.

single neuron input-output plane and is possible to represent with star sets [34]. Although better linear overapproximations are possible by bounding multiple neurons at once [28], it is unclear so far if this improves overall performance, so we focus on the single neuron case in `nnenum`. Although the star set triangle overapproximation approach, which we call `Star`, is the most accurate, it can sometimes be slow as optimization over star sets uses linear programming. It can be actually beneficial to use a less accurate zonotope abstraction, which can compute bounds of subsequent neurons more quickly. With zonotopes, actually multiple abstractions are possible, which can be parameterized by the slope of the zonotope [29]. Three choices are shown in Figure 3 (top-right and bottom row). While the best-area zonotope generally has less overapproximation error, all the zonotopes are actually incomparable from a set perspective. Each zonotope type has points in the neuron input-output plane that are excluded in one of the other zonotope abstractions, and so none is better in all cases. The `Zonotope` approach uses the best-area zonotope abstraction for each neuron that can split.

In addition to using individual abstractions, `nnenum` also makes use of two new ideas in the context of neural network verification: (i) *multi-round abstractions* and (ii) *multi-abstraction analysis within each round.*

Multi-round analysis proceeds by first trying one abstraction and then trying another one before proceeding to split. For example, multi-round analysis with two rounds could first try to prove the property using the best-area zonotope abstraction, which is fast but less accurate. If that does not succeed, analysis would then try to use the triangle overapproximation with star sets, which is

Table 2. Direct Abstraction Refinement Analysis.

Optimization	Timeouts	Runtime [sec]
`Zonotope`	29	3517.1
`Star`	64	4174.3
`Zono-Star`	63	4186.8
`Three Rounds`	65	4190.0

more accurate but slower. If that also fails, then splitting would occur and the process would proceed recursively on the split sets.

Multi-abstraction within each round is a little bit similar to the use prefilter zonotopes during exact analysis. There, the idea was to first use a quick abstraction to compute if neurons can split, and only if inputs cannot be proven to be one-sided do we use the slower, more accurate star set to compute bounds. We call this combined abstraction method `Zono-Star`. This process can be extended to an arbitrary list of abstractions, and towards the problem of computing the lower and upper bounds of each neuron rather than just rejecting splits. The approach works by using each abstraction to compute the lower and upper bounds of the next neuron's input, stopping if any of the abstractions shows it is one-sided. Since all abstractions are overapproximations, even when splits cannot be rejected we can still use the greatest lower bound and the least upper bound from all of the abstractions. Then, for constructing the overapproximating zonotopes or star sets for the next neuron, these tighter bounds are used for all of the abstractions. For example, we could consider all three zonotope abstractions *at the same time*. Since zonotope analysis is quick, this multi-abstraction method offers a way to improve accuracy with little cost. A similar concurrent multi-abstraction approach has been used in the context of parallelotope bundles [8] for reachability analysis of dynamical systems [9,20] where it was called "all-for-one" analysis.

Multi-round and multi-abstraction can also be combined. For example, in the first round could try to prove the property using just the best-area zonotope abstraction, and if that fails a second round could use all three zonotopes together, and then if that fails we could do a third round with all three zonotopes as well as the triangle overapproximation with star sets. This approach is called `Three Rounds`.

The measurements for the various abstractions are shown in Table 2. From the analysis, the `Zonotope` methods looks better than the more accurate abstractions, even with multi-round and multi-abstraction analysis. The main reason is that the other three levels of abstraction have star sets that use LP solving to determine bounds, which turns out to be a bottleneck.

However, even the `Zonotope` abstraction refinement method is slower than the `New LP + Witnesses` exact analysis method described earlier in Section 3.1. This would seem to imply that abstraction refinement is a losing strategy for high-performance neural network verification, unless we can find further opti-

mizations to improve the approach. In the next subsection, we discuss the cause for this unintuitively poor performance, and propose and evaluate further optimizations to the abstraction refinement algorithm.

3.3 Abstraction Refinement Improvements

In the worst case, the abstraction refinement approach will revert to exact analysis. However, performance in these cases is much worse than directly doing exact analysis, as time must be spent at each abstraction phase to propagate abstract sets and check if the final sets are unsafe. Upon examining the performance of various phases of the algorithm, it turned out that propagating abstract sets can sometimes be very slow. The cause of this was determined to be the *error snowball* effect described in Section 2.4. Basically, an overapproximation is less accurate, and so it is more likely that neurons analyzed will look like they may split, causing a further reduction in accuracy. The splitting processes adds variables to the LPs needed to compute neuron bounds using star sets, and adds generators to the zonotope abstractions, slowing down analysis of neurons further in the network. Worse, after all the slow computation completes, the set at the end of the network has lots of error and very rarely can prove the safety specification. The computation has taken a long time and its result was useless. To speed up performance, then, we need methods that prevent error snowballs, reduce their impact, or detect them before they get out of hand.

We consider different ways to adjust the computation time / error trade off when using star sets. If we can increase the analysis speed for a slight reduction in accuracy, this may reduce the computation time when error snowballs occur during abstract analysis. The first method we look at to do this we call `Quick Star`. Here, the bounds for each neuron are computed only using zonotopes, like with a multi-abstraction round, but these bounds are also used to construct a star set overapproximation. This star set is only used to check for intersection with the unsafe set, rather than for computing the bounds on the neuron inputs. Only a single LP is used at the end to check for intersection with the unsafe set, as the bounds computations are all done with zonotopes. The approach we use with `Quick Star` uses the `Three Rounds` abstraction which uses three zonotopes in the final round to get the tightest bounds possible without LP.

A second way to speed up star set analysis with a slight accuracy reduction deals with the bounds computation process. Rather than computing two bounds for each neuron, we consider only computing a single bound in order to try to prove the neuron is one-sided, and use the zonotope bounds for the other side if this fails. The side to choose (lower or upper bound), is selected by leveraging a quick concrete execution of the neural network. In this way, for example, we will never compute the upper bound for a neuron's input if the concrete execution has a positive input value for that neuron, as the upper bound could not be less than concrete input value. This method, which uses a single LP per bounds computation, is a bit of a compromise between direct star set abstraction, which use two LPs to compute the lower and upper bounds, and `Quick Star`, which

uses zero LPs. We evaluate this approach using the combined zonotope-star abstraction and refer to it as `One Sided Zono-Star`.

One way we investigate to try to prevent error snowballs, rather than just reducing their effect, is called Execution-Guided Overapproximation (`EGO`). `EGO` works by changing the order in which abstractions are constructed. Rather than first trying abstract analysis and then splitting if the abstraction fails, `EGO` first splits as much as possible essentially proceeding as if it were exact analysis. Upon succeeding to verify one branch of the search space after many splits, the method backtracks like a normal depth-first search, but now starts to try abstract analysis. This continues further and further up the search tree, until abstract analysis no longer succeeds in proving the property, causing the method to again switch to exact analysis and repeat. Essentially, rather than starting from an abstract system and performing refinement, `EGO` analysis starts with a concrete set and iteratively constructs more and more abstract systems. This avoids costly abstraction analysis near the root of the search tree that often result in error snowballs. More details on `EGO` are available in a online report [1]. For the `EGO` method, we use the `Zono-Star` abstraction, as well as the `Three Rounds` abstraction which we call `EGO Three Rounds`.

Instead of `EGO`, there are other methods we try to use to prevent and detect error snowballs. One method, called `Split Limit`, tracks the number of splits that occurred whenever an abstraction successfully verified a portion of the search tree. When backtracking and continuing abstract analysis, if the number of splits using zonotopes exceeds some factor multiplied by the previous number of splits, this method directly splits without trying abstract analysis. The intuition for this approach comes from the observation that error snowballs often have a huge number of neurons that split, much more than previous successful analysis. We analyze different multiplication factors in the context of the `Split Limit` method. For example `Split Limit 1.5` would mean that if the last successfully verified abstraction had 10 neurons that split, and the current zonotope-only abstraction has more than 15 splits, abstraction analysis with star sets would not even be attempted.

A second idea to reduce the impact of error snowballs is to directly use timeouts for the abstraction analysis (`Abs Timeout`). These methods again have a parameter that we tune which is the number of seconds abstraction analysis runs before it is stopped.

The third way to improve performance creates a threshold for when `Split Limit` should be used. We noticed when the number of splits is very small in successful abstract analysis, using a simple multiplicative factor may give up on abstract analysis too quickly. For example, if an abstraction with two splits succeeds, having a `Split Limit` of 2 would reject any system with more than four splits in zonotope analysis. In the `Split Min` methods, we enforce a minimum on the number of splits before we consider the `Split Limit` multiplicative factor. For example, we could set a `Split Min` threshold of 30, where any abstraction where the zonotope analysis split on less than 30 neurons will always be analyzed

Table 3. Optimized Abstraction Refinement Analysis.

Optimization	Timeouts	Runtime [sec]
`Quick Star`	43	3802.8
`One Sided`	26	2998.5
`EGO`	2	1469.2
`EGO 3 Rounds`	2	1467.1
`Split Limit 1.1`	31	3090.2
`Split Limit 1.2`	32	3288.4
`Split Limit 1.3`	31	3247.7
`Split Limit 1.4`	31	3215.4
`Split Limit 1.5`	28	3229.9
`Split Limit 1.6`	32	3291.1
`Split Limit 1.7`	30	3184.8
`Abs Timeout 0.02`	0	274.8
`Abs Timeout 0.04`	0	371.7
`Abs Timeout 0.06`	0	451.5
`Abs Timeout 0.08`	0	520.7
`Abs Timeout 0.1`	0	577.7
`Split Min 10`	3	1236.4
`Split Min 20`	0	485.5
`Split Min 30`	0	298.6
`Split Min 40`	0	224.7
`Split Min 50`	0	191.5
`Split Min 60`	0	178.4
`Split Min 70`	0	173.6
`Split Min 80`	0	175.7
`Split Min 90`	0	188.7

abstractly with star sets. Again the minimum value is a parameter that we tune through measurements.

The result of each of the optimizations run on all the ACAS Xu benchmarks is shown in Table 3. Using `Quick Star` reduced the number of timeouts from 63 with `Zono-Star` down to 43. The `One Sided` approach made a bigger difference, reducing the number of timeouts from 63 with `Zono-Star` to 26. For the rest of the optimizations we continue to use the one-sided optimization. `EGO` was even faster, where analysis now only had 2 benchmarks that exceeded the one minute timeout. The additional abstraction rounds in `EGO 3 Rounds` no longer slowed the method down as with the direct abstraction refinement approaches from Table 2. We noticed this was generally the case when there were few error snowball cases, so we continue to use the three-round abstraction in the remaining measurements.

Although `EGO` and `Quick Star` methods both improved performance by reducing the effect of error snowballs, it was difficult to think of ways to further improve their speed. The goal of the remaining three optimizations, `Split Limit`,

`Abs Timeout`, and `Split Min`, was to be able to get closer to the accuracy of using star set overapproximation without the large runtime.

Using `Split Limit`, we improve upon the `Three Rounds` result which had 65 timeouts, although the method is still slower than `EGO`. The method is not too sensitive to the parameter used, and we use `Split Limit 1.5` for the remaining measurements.

Adding in `Abs Timeout`, we finally achieve a method where every benchmark finishes within one minute (no timeouts occur). Generally, smaller values of the abstraction timeout parameter seem to be faster. We used an `Abs Timeout 0.04` when evaluating the `Split Min` parameter (we also tried 0.02 and 0.06, but they was slightly slower when used with `Split Min`).

Finally, adding the `Split Min` optimization further reduces the total computation time to run all the ACAS Xu benchmarks. The `Split Min 70` method analyzed all 180 benchmarks using a runtime sum of 173 seconds.

We also tried many optimization options individually that were not reported in detail in the tables. For example only using `Abs Timeout` without `Split Limit` was also fairly fast, but still had a few timeouts. Doing things like turning off zonotope domain contraction also severely hurt performance of the overapproximation methods. Overall, the best performance we found was achieved using a combination the presented optimizations, with `Split Limit`, `Abs Timeout`, and `Split Min` used in combination to prevent error snowballs with star set analysis.

4 Related Work

Many additional methods for verification of neural networks have been proposed [21,38], including methods based on mixed integer-linear programming (MILP) [22,32], symbolic interval propagation [37,36], SMT-based approaches based on modifications to the Simplex linear programming algorithm [18,19], and MILP methods with local search [11].

Linear star sets have been explored in the context of other problems. In particular, efficient high-dimensional reachability analysis of hybrid systems with linear differential equations is possible with star sets [2,4], where the primary operations needed are the same as in the neural network verification case: affine transformation, intersection with linear constraints, and optimization. Other names for essentially the same data structure as linear star sets include affine forms [14], constrained zonotopes [27,23] and $\mathcal{AH}$-Polytopes [26].

Other optimizations may provide further improvements to performance, such as using the spurious region to guide refinement [40], similar to counter-example guided abstraction refinement [7]. We did not find a way to use this in our algorithm without hurting overall performance, although it may be an avenue for further investigation.

Another critical choice in the algorithm is the ordering of neurons when performing splitting. We tried several heuristic orderings within each layer, with only minor impact on performance. In `nnenum`, we always visit the neurons layer

by layer, but rearrange the order of neurons so that they are sorted in decreasing order of their L_∞ norm distance of the zonotope bounds estimates, which only slightly outperforms just visiting the neurons in the original order. More computationally advanced methods, such as those that track gradient information [37], compute output sensitivity [39], perform dependency analysis [5] or use information from LP shadow prices [25] could further improve efficiency.

In the future, we may also consider other abstractions such as symbolic intervals [37,36] or using single upper and lower bounds (called DeepPoly [30]) which could offer different accuracy / performance trade offs.

5 Conclusion

We presented an abstraction refinement algorithm for verification of neural networks based on the star set data structure. Importantly, we showed that several optimizations are possible and necessary with the approach in order to create a highly efficient algorithm—abstraction refinement is actually slower than directly using exact analysis without the optimizations presented. While optimizing an algorithm might be seen as an engineering problem, we believe such optimization is necessary to guide the appropriate place to develop new theory. It is difficult to really evaluate an algorithm and know which bottlenecks are worth improving without an optimized implementation.

In this paper, we showed that the fully optimized version of `nnenum` verified all 180 ACAS Xu benchmarks from properties 1 to 4 using a sum total runtime of 178 seconds. During the VNN-COMP 2020 competition, the *single* ACAS Xu benchmark consisting of property 2 and network 4-2 required 240 seconds for ERAN [31] and 648 seconds for Venus [5], the two next fastest tools. The performance of `nnenum` would not be possible without the methods presented in this paper.

References

1. Bak, S.: Execution-guided overapproximation (EGO) for improving scalability of neural network verification (2020), http://stanleybak.com/papers/bak2020vnn.pdf
2. Bak, S., Duggirala, P.S.: Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In: Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control. HSCC '17 (2017)
3. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: Proceedings of the 32nd International Conference on Computer Aided Verification. Springer (2020)
4. Bak, S., Tran, H.D., Johnson, T.T.: Numerical verification of affine systems with up to a billion dimensions. In: Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 23–32. HSCC '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3302504.3311792, http://doi.acm.org/10.1145/3302504.3311792

5. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 3291–3299 (Apr 2020). https://doi.org/10.1609/aaai.v34i04.5729, https://ojs.aaai.org/index.php/AAAI/article/view/5729
6. Buro, M., Büning, H.K.: Report on a SAT competition. Fachbereich Math.-Informatik, Univ. Gesamthochschule (1992)
7. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: International Conference on Computer Aided Verification. pp. 154–169. Springer (2000)
8. Dreossi, T., Dang, T., Piazza, C.: Parallelotope bundles for polynomial reachability. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control. pp. 297–306 (2016)
9. Dreossi, T., Dang, T., Piazza, C.: Reachability computation for polynomial dynamical systems. Formal Methods in System Design **50**(1), 1–38 (2017)
10. Duggirala, P.S., Viswanathan, M.: Parsimonious, simulation based verification of linear systems. In: International Conference on Computer Aided Verification. pp. 477–494. Springer (2016)
11. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods Symposium. pp. 121–138. Springer (2018)
12. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 269–286. Springer (2017)
13. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
14. Han, Z., Krogh, B.H.: Reachability analysis of large-scale affine systems using low-dimensional polytopes. In: International Workshop on Hybrid Systems: Computation and Control. pp. 287–301. Springer (2006)
15. Huang, S., Papernot, N., Goodfellow, I., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies. arXiv preprint arXiv:1702.02284 (2017)
16. Johnson, T.T.: International verification of neural networks com- petition (vnn-comp) (2020)
17. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2016)
18. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: International Conference on Computer Aided Verification. pp. 97–117. Springer (2017)
19. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The Marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
20. Kim, E., Duggirala, P.S.: Kaa: A python implementation of reachable set computation using bernstein polynomials. EPiC Series in Computing **74**, 184–196 (2020)
21. Liu, C., Arnon, T., Lazarus, C., Barrett, C., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
22. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. arXiv preprint arXiv:1706.07351 (2017)
23. Raghuraman, V., Koeln, J.P.: Set operations and order reductions for constrained zonotopes. arXiv preprint arXiv:2009.06039 (2020)

24. Rauber, J., Brendel, W., Bethge, M.: Foolbox: A python toolbox to benchmark the robustness of machine learning models. arXiv preprint arXiv:1707.04131 (2017)
25. Royo, V.R., Calandra, R., Stipanovic, D.M., Tomlin, C.: Fast neural network verification via shadow prices. arXiv preprint arXiv:1902.07247 (2019)
26. Sadraddini, S., Tedrake, R.: Linear encodings for polytope containment problems. In: 2019 IEEE 58th Conference on Decision and Control (CDC). pp. 4367–4372. IEEE (2019)
27. Scott, J.K., Raimondo, D.M., Marseglia, G.R., Braatz, R.D.: Constrained zonotopes: A new tool for set-based estimation and fault detection. Automatica **69**, 126–136 (2016)
28. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. In: Advances in Neural Information Processing Systems. pp. 15098–15109 (2019)
29. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. In: Advances in Neural Information Processing Systems. pp. 10802–10813 (2018)
30. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 1–30 (2019)
31. Singh, G., Gehr, T., Püschel, M., Vechev, M.: Boosting robustness certification of neural networks. International Conference on Learning Representations (ICLR (2019)
32. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. arXiv preprint arXiv:1711.07356 (2017)
33. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: Proceedings of the 32nd International Conference on Computer Aided Verification. Springer (2020)
34. Tran, H.D., Lopez, D.M., Musau, P., Yang, X., Nguyen, L.V., Xiang, W., Johnson, T.T.: Star-based reachability analysis of deep neural networks. In: International Symposium on Formal Methods. pp. 670–686. Springer (2019)
35. Tran, H.D., Yang, X., Manzanas, D., Musau, P., Nguyen, L., Xiang, W., Bak, S., Johnson, T.T.: Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Proceedings of the 32nd International Conference on Computer Aided Verification. Springer (2020)
36. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems. pp. 6367–6377 (2018)
37. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium. pp. 1599–1614 (2018)
38. Xiang, W., Musau, P., Wild, A.A., Lopez, D.M., Hamilton, N., Yang, X., Rosenfeld, J., Johnson, T.T.: Verification for machine learning, autonomy, and neural networks survey. arXiv preprint arXiv:1810.01989 (2018)
39. Xiang, W., Tran, H.D., Johnson, T.T.: Output reachable set estimation and verification for multilayer neural networks. IEEE transactions on neural networks and learning systems **29**(11), 5777–5783 (2018)
40. Yang, P., Li, R., Li, J., Huang, C.C., Wang, J., Sun, J., Xue, B., Zhang, L.: Improving neural network verification through spurious region guided refinement. arXiv preprint arXiv:2010.07722 (2020)