Unifying Syntactic and Semantic Abstractions for Deep Neural Networks

Sanaa Siddiqui[1], Diganta Mukhopadhyay[2], Mohammad Afzal[2,3], Hrishikesh Karmarkar[2], and Kumar Madhukar[1]

[1] Indian Institute of Technology Delhi, New Delhi, India
sanaasiddiqui@cse.iitd.ac.in madhukar@cse.iitd.ac.in
[2] TCS Research, Pune, India diganta.m@tcs.com afzal.2@tcs.com
hrishi.karmarkar@tcs.com
[3] Indian Institute of Bombay, Mumbai, India

Abstract. Deep Neural Networks (DNNs) are being trained and trusted for performing fairly complex tasks, even in safety-critical applications such as automated driving, medical diagnosis, and air traffic control. However, these real-world applications tend to rely on very large DNNs to achieve the desired accuracy, making it a challenge for them to be executed in resource-constrained and real-time settings. The size of these networks is also a bottleneck in proving their trustworthiness through formal verification or explanation, limiting the deployability of these networks in safety-critical domains. Therefore, it is imperative to be able to compress these networks while maintaining a strong formal connection while preserving desirable safety properties. Several *syntactic* abstraction techniques have been proposed that produce an abstract network with a formal guarantee that safety properties will be preserved. These, however, do not take the *semantic* behaviour of the network and thus produce sub-optimally large networks. On the other hand, compression and *semantic* abstraction techniques have been proposed which achieve a significant reduction in network size but only weakly preserve a limited set of safety properties. In this paper, we propose to combine the semantic and syntactic approaches into a single framework to get the best of both worlds. This allows us to guide the abstraction using global semantic information while still providing concrete soundness guarantees based on syntactic constraints. Our experiments on standard neural network benchmarks show that this can produce smaller abstract networks than existing methods while preserving safety properties.

Keywords: DNN Abstraction · Formal Verification · DNN Compression

1 Introduction

Advances in Deep Neural Networks (DNNs) have enabled the scalable solution of several previously intractable problems such as image recognition and natural language processing. Due to this, DNNs have increasingly assumed a central role across various domains. These include several safety-critical domains like

healthcare [15], where they contribute significantly to medical diagnosis and predictive analysis [31], and autonomous vehicles, where DNNs serve as the backbone for sophisticated perception systems, supporting tasks such as object recognition and decision making [3].

However, given the enormous size and the substantial resource requirements of these DNNs in terms of CPU, memory and power, their implementation on embedded devices and real-time systems is often infeasible. This issue becomes especially apparent when integrating DNNs into embedded and real-time systems for performing safety-critical tasks, such as obstacle recognition in autonomous vehicles. Furthermore, DNNs are well known to be vulnerable to adversarial [45,23,33,27,5,6] and backdoor attacks [9], and are also difficult to interpret. While a number of formal analysis techniques have been proposed to build trust on the reliability of DNNs in safety-critical settings, including verification [25,43,52,46,20] and formal explainability [32,2], the size of the DNNs continues to be the limiting factor for the scalability of these techniques. To tackle both these issues, it is imperative to reduce the size of the DNNs involved while maintaining a strong formal connection to the original network, and preserving desirable safety properties.

A typical approach within formal methods to reduce the complexity of any object while maintaining a strong formal connection with the original network is *abstraction*. For DNNs, structural abstraction based on the *syntax* (the local weights and biases at each neuron of the DNN) forms the basis of several techniques [20,13,53,35] that work by converting a large *concrete* DNN $\mathcal{N}$ into a smaller *abstract* DNN $\mathcal{N}'$ via *merging* groups of neurons in $\mathcal{N}$ into single neurons in $\mathcal{N}'$. Each such merge is done in a way that ensures that there are *concrete*, formal soundness guarantees linking the behaviour of $\mathcal{N}$ and $\mathcal{N}'$, thus maintaining safety-critical properties. In particular, one can verify properties on $\mathcal{N}'$, and using the concrete soundness guarantees, lift the result to $\mathcal{N}$ and argue about its reliability. However, while these techniques have been shown to extend the scalability of neural network verification techniques, they do not take the *semantic* behaviour of the network into account, thereby producing sub-optimally large $\mathcal{N}'$. [20]. In fact, in [20], the $\mathcal{N}'$ produced was sometimes observed to be larger than the original $\mathcal{N}$. This sub-optimality with respect to size prevents these techniques from being useful for compressing DNNs for deployment in safety-critical resource-constrained environments.

On the other hand, neural network compression techniques [11] and semantic abstraction techniques [1,7] take into account the global *semantic* behavior of the network, and are able to achieve a significant reduction in size. However, heuristic based compression techniques [11] do not formally maintain any connection with $\mathcal{N}$, while semantic abstraction [1,7] techniques only provide some limited kinds of formal connections with $\mathcal{N}$. In particular, the guarantees provided by clustering based methods like [1] are limited to lifting specific bound propagation based proofs from $\mathcal{N}'$ to $\mathcal{N}$, and those provided by linear combination based methods like [7] only bound the difference in behavior of $\mathcal{N}'$ and $\mathcal{N}$ on a finite subset of the

input space. Thus, without strong formal guarantees connecting the behaviors of $\mathcal{N}'$ and $\mathcal{N}$, the $\mathcal{N}'$ may not preserve desired safety properties.

In this work we combine the syntactic and semantic approaches into a single framework for generating an abstract network. By splitting and labeling the neurons *inc* and *dec*, similar to [20], and restricting ourselves to merges involving only similarly labeled neurons, we provide a concrete formal link the behavior of $\mathcal{N}$ and $\mathcal{N}'$ via syntactic constraints. At the same time, to take into account the global semantic behavior, we introduce a semantic closeness metric between two neurons in the same layer. Using this metric, we construct a tree of merge operations that captures the relative contribution of each merge to the quality of the abstraction. We propose a refinement procedure that uses this tree as a guide to undo the merge operations that contribute most to the poor quality of $\mathcal{N}'$. We show that in the resulting refined network, groups of neurons that remain merged are semantically closer than neurons that get un-merged. Thus, we are able to refine $\mathcal{N}'$ in a way that is optimal with respect to the semantic behavior. Thus, combining both syntactic and semantic approaches, our framework is able to find an $\mathcal{N}'$ that preserves desired safety properties of a smaller size.

We assemble these pieces into an abstraction-refinement framework that can generate a small size $\mathcal{N}'$ by starting with a fully merged network and iteratively refining until a strong enough network is obtained. To demonstrate the usefulness of this framework we set up a CEGAR [20,12] loop to verify the ACASXu [36,25] networks. In these experiments we find that we are able to produce smaller networks than existing works that are still strong enough to verify the property.

2 Background

2.1 Notation

We restrict ourselves to fully-connected, feed-forward neural networks with $ReLU$ activation function. Neurons in our network are denoted as $n_{(i,l)}$, where i signifies the neuron number in layer l. The function taking a given input $\boldsymbol{x}$ to the value of $n_{(i,l)}$ is represented by $v_{(i,l)}(\boldsymbol{x})$, The function $o_{(i,l)}$ takes a list of input vectors $[\boldsymbol{x_1} \cdots \boldsymbol{x_N}]$ and returns a vector with $[v_{(i,l)}(\boldsymbol{x_1}) \cdots v_{(i,l)}(\boldsymbol{x_N})]$ for a particular neuron $n_{(i,l)}$.

2.2 Formal Analysis of Neural Networks

Several techniques and methods have been studied to improve the reliability and trustworthiness of DNNs deployed in safety-critical settings via formal analysis. This includes verifying DNNs with respect to a given safety property [25,20,43,13,53,35,1,7], providing formal explanations of the behavior of DNNs [2,32], and defending against backdoor attacks [38]. To provide the formal guarantees behind the analyses performed, all of these techniques rely on making *neural network queries*.

These neural network queries are of the form $(P, \mathcal{N}, Q)$, and asks: if for all inputs $\boldsymbol{x}$ to $\mathcal{N}$ for which the formula P holds, the formula Q also hold on the

output $\mathcal{N}(\boldsymbol{x})$. While there are several tools that can handle such queries, like Marabou [25,26,47], $\alpha, \beta - CROWN$ [52,48,40,49,46,51,50,41] and NeuralSAT [16], scalability remains an issue, and so reducing the size of $\mathcal{N}$ is desirable.

2.3 Semantic Compressions and Abstractions with Empirical Guarantees

Several techniques utilize semantic information, typically extracted via simulation of the DNNs, to obtain a smaller $\mathcal{N}'$. Neural network compression techniques [11], produce small $\mathcal{N}'$, but the behavior of $\mathcal{N}'$ in connection to $\mathcal{N}$ is only characterized empirically. Similarly, some semantic abstraction techniques [7] provide bi-simulation guarantees bounding the difference in the behavior of $\mathcal{N}'$ and $\mathcal{N}$ on a finite set of input points, typically a subset of the training dataset. Since these techniques characterize the behavior of $\mathcal{N}'$ only on a finite set of input points, the trust obtained on the connection between $\mathcal{N}$ and $\mathcal{N}'$ is only of a empirical nature. Other techniques like [1] use bi-simulation to lift interval bound propagation performed on $\mathcal{N}'$ to get sound bounds on $\mathcal{N}$. While this does provide a sound proof, interval bounds are typically not strong enough to prove many interesting and practically relevant neural network queries.

2.4 Strong Formal Connections between $\mathcal{N}$ and $\mathcal{N}'$

Structural abstraction techniques including [20] provide the following strong formal connection between the behavior of $\mathcal{N}$ and $\mathcal{N}'$: $\forall x, \mathcal{N}'(x) \geq \mathcal{N}(x)$. Any general neural network query can be converted to a query of the form $(P, \mathcal{N}, y < c)$ for some c by encoding the postcondition as extra layers in $\mathcal{N}$ [20,25,39]. Therefore this notion of formal connection allows one to abstract the larger $\mathcal{N}$ to a smaller $\mathcal{N}'$, dispatch the easier query $(P, \mathcal{N}', y < c)$ using a solver call, and argue that the original query holds [20,13,53]. This immediately makes such an $\mathcal{N}'$ useful for accelerating several formal analysis techniques (Section 2.2). Therefore, in this work we focus on developing a framework that produces abstract networks that maintains this formal connection.

2.5 Syntactic Neural Network Splitting and Merging

In order to ensure the soundness guarantees when transforming $\mathcal{N}$ into $\mathcal{N}'$, instead of adopting the four-way split approach proposed in [20], we opt for a two-way split method [8,28,29]. This involves partitioning neurons in $\mathcal{N}$ into duplicate copies labeled with either *inc* or *dec*, ensuring that any increase (decrease) in the value of a neuron labeled *inc* (*dec*) only results in an increase in the output value. This can be done because it is not necessary to perform *pos-neg* splitting to soundly obtain an *inc-dec* split, as seen in [8,28,29]. Then, following [20], a sound abstraction can be obtained by *merging* all similarly labeled neurons as follows: if the neurons have the label *inc* (*dec*), replace incoming edges from the same previous layer neuron with a single edge with the maximum (minimum)

of the original edge weights. Outgoing edges to the same next layer neuron are replaced with a single edge with the sum of the weights for both the *inc* and *dec* case.

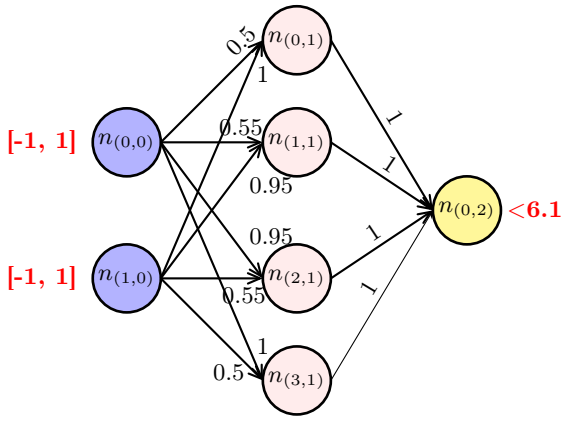

Fig. 1: Original Network and Property

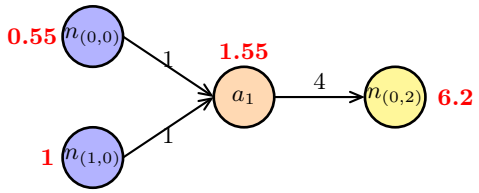

Fig. 2: Fully Abstracted Network

For example, consider the network and property in Figure 1. For this example, all the neurons in the output and middle layer get classified as *inc*. Thus, they are all merged together, leading to the network in Figure 2.

Note that this process only considers the syntactic structure of the network, no semantic information is used.

2.6 Syntactic Refinement

The fully merged network obtained in Section 2.5 may not be sufficiently strong to be able to dispatch, that is, there may be spurious counterexamples. In such situations, a common approach to obtain a better quality $\mathcal{N}'$ is to perform refinement steps based on a spurious counterexamples $\boldsymbol{\beta}$ [20,13,53]. In existing techniques, this is typically done by restoring a single neuron coming from $\mathcal{N}$

that had been merged with other neurons in $\mathcal{N}'$. The neuron chosen is typically one whose contribution to $\boldsymbol{\beta}$ is estimated to be the highest.

These techniques, however, do not consider any semantic behavior to guide their refinement. As such, the refinement process tends to produce a large number of restored neurons that are not merged with any other neurons. A proliferation of these *singleton* neurons leads to $\mathcal{N}'$ having a larger size. At the same time, a large group of neurons remains merged in $\mathcal{N}'$, which affects the quality of $\mathcal{N}'$.

We can see this in our example. Say the fully merged network in Figure 2 we get a $\boldsymbol{\beta}$ given by the values in **bold**. Then, in the next refinement step, the neuron $n_{(3,1)}$ gets restored, giving us the network in Figure 3. Again, the $\boldsymbol{\beta}$ obtained is shown in **bold**, and the next refinement step restores $n_{(0,1)}$ leading to Figure 4. We see that in the resultant network, two semantically dis-similar neurons $n_{(1,1)}$ and $n_{(2,1)}$ remain merged, while the merges between the similar pairs of neurons $n_{(3,1)}$, $n_{(2,1)}$ and $n_{(1,1)}$, $n_{(0,1)}$ have been un-done. Indeed, the network in Figure 4 still admits spurious counterexamples, as seen by the values in **bold**, and we end up refining all the way to the original network.

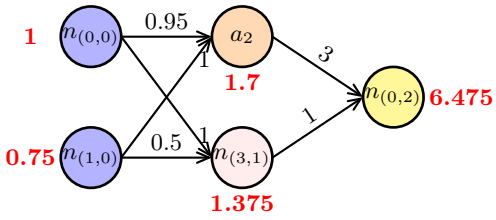

Fig. 3: Refine Step 1: Culprit Neuron is 3

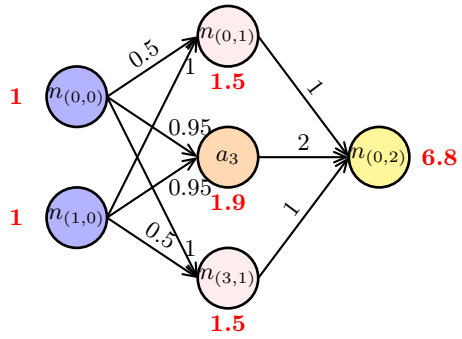

Fig. 4: Refine Step 2: Culprit Neuron is 0

3 Methodology

In this work, we aim to utilize information from the semantic behavior of $\mathcal{N}$ to better control and guide the refinement process, producing a smaller $\mathcal{N}'$ while retaining the strong formal connection to $\mathcal{N}'$.

In particular, we define a *semantic closeness metric* that captures how close the semantic behavior of two neurons is (Section 3.1). We utilize this semantic closeness metric to arrange the merge operations into a tree where the lower quality merges involving semantically far neurons appear higher and can be refined with higher priority (Section 3.2).

Using this tree, we build a framework where refinement can be done by making cuts of this tree while still providing concrete soundness guarantees (Section 3.3). We show that the merges retained in this refinement process are optimal with respect to the semantic information (Section 3.4). This allows us to avoid restoring a large number of singleton neurons (see Section 2.5) and lets us retain merge operations of higher quality.

Using these components, we propose a general CEGAR [20,12] loop based framework (Section 3.5) that combines syntactic merge operations with semantic behavior. This framework is able produce an $\mathcal{N}'$ strong enough to not have any spurious counterexamples and with a much smaller size.

3.1 Semantic Closeness Factor

To guide the semantic abstraction process, we define a *semantic closeness metric* $\mathcal{C}$: $\mathcal{C}(n_{(i_1,l)}, n_{(i_2,l)})$ is a function that takes two neurons $n_{(i_1,l)}$ and $n_{(i_2,l)}$ in the same layer l, and returns a real number that captures how close the behaviors of $n_{(i_1,l)}$ and $n_{(i_2,l)}$ are from a semantic point of view. The number returned by $\mathcal{C}$ should be smaller for neurons whose semantic behavior is closer. Intuitively, this metric would characterize the semantic behavior of the neurons in layer l relative to each other, and prioritize certain merges over others.

Depending on the application, the precise definition of this metric may be chosen in various ways. We note that our framework is agnostic to the particular choice of semantic metric, and the concrete soundness guarantee holds for any such choice. Inspired by [1], we choose the semantic closeness metric to be the difference between the functions computed by the two neurons: $||v_{(i_1,l)} - v_{(i_2,l)}||$.

However, since $v_{(i_1,l)}$ and $v_{(i_2,l)}$ are functions, computing $||v_{(i_1,l)} - v_{(i_2,l)}||$ precisely is not feasible. Therefore, we estimate it using a sample set of inputs X: $||o_{(i_1,l)}(\mathrm{X}) - o_{(i_2,l)}(\mathrm{X})||_2$. In general this X may be chosen in different ways, and our framework is agnostic to the sampling strategy used. In our experiments, we use a uniform sampling of the input region where P holds.

3.2 Tree of Merges

We use $\mathcal{C}$ to create a tree structure to prioritize merges where leaf nodes represent the original neurons and non-leaf nodes represent merge operations. In

particular, each non-leaf node m_i represents an operation merging all the neurons corresponding to the leaf nodes that are descendants of m_i. For instance, the top half of Figure 5 shows a tree where m_4 merges $n_{(0,1)}$ and $n_{(1,1)}$, while m_6 merges $n_{(0,1)}$, $n_{(1,1)}$, $n_{(2,1)}$ and $n_{(3,1)}$.

The construction of the tree follows a bottom-up approach, where we start from individual neurons, and greedily perform the merge operation involving the most similar groups of neurons, delaying the merging of dissimilar ones. This is detailed in Algorithm 1 [4]:

We start with an initial structure consisting only of leaf nodes corresponding to original neurons (line 1). At this stage, no merge operation has yet been done, and each node is a potential candidate for a merge, stored in *Cand* at line 2.

Now, as long as we have at-least two candidates that we can merge, we keep greedily merging them in the loop starting at line 11. To do this, we find which two candidate nodes m_i and m_j are most semantically similar in line 12, and introduce an operation merging them in lines 13-15.

We measure how semantically close a pair of candidates m_1 and m_2 are by looking at the maximum semantic distance between a neuron merged as a part of m_1's process and a that merged as a part of m_2's process. This is done recursively by the *PairwiseMax* function in lines 3-10.

The resulting tree captures an optimal ordering of merge operations. That is, as we progress up the tree, the maximum value of $\mathcal{C}$ between any two neurons involved in a merge increases and the imprecision introduced by the merge operation increases. This is formalized in Section 3.4).

As an example, consider the middle layer in Figure 1. Here, $n_{(0,1)}$ and $n_{(1,1)}$ are semantically closest. Thus, in the tree (top half of Figure 5), we first merge these two first to get the node m_4, representing the merge group $\{n_{(0,1)}, n_{(1,1)}\}$. At this point, we have three choices for the next merge operation: m_4 and $n_{(2,1)}$, m_4 and $n_{(3,1)}$, or $n_{(2,1)}$ and $n_{(3,1)}$. Since $n_{(2,1)}$ and $n_{(3,1)}$ are semantically closer to each other than to $n_{(0,1)}$ or $n_{(1,1)}$, the algorithm merges $n_{(2,1)}$ and $n_{(3,1)}$ to get m_5. This produces the merge group $\{n_{(2,1)}, n_{(3,1)}\}$, which has elements that are semantically closer than the groups $\{n_{(0,1)}, n_{(1,1)}\, n_{(2,1)}\}$ or $\{n_{(0,1)}, n_{(1,1)}, n_{(3,1)}\}$ obtained from following the other two choices. Finally, m_4 and m_5 get merged to m_6, giving us the complete tree.

3.3 Tree-cuts and Refinement

In our abstraction refinement loop, we start with the fully merged network. Then, whenever we get a spurious counterexamples $\boldsymbol{\beta}$, we wish to refine the network. That is, we wish to choose which neurons should remain merged. Intuitively, this choice should be guided two factors: optimizing with respect to the semantic behavior of the network, and attempting to eliminate $\boldsymbol{\beta}$.

[4] For our choice of $\mathcal{C}$, (Section 3.1), Algorithm 1 reduces to hierarchial clustering [24], allowing us to leverage existing efficient implementations. Nonetheless, the general algorithm presented here will work for any choice of $\mathcal{C}$.

Algorithm 1 Building the Tree

Input: Neurons $\{n_{(i_1,l)}, \cdots, n_{(i_r,l)}\}$ with *inc-dec* label, Closeness metric $\mathcal{C}$

Initialize G with nodes $\{n_{(i_1,l)}, \cdots, n_{(i_r,l)}\}$ and no edges.
Initialize $Cand = \{n_{(i_1,l)}, \cdots, n_{(i_r,l)}\}$
function PairwiseMax(m_1, m_2)
    if m_1 or m_2 has children **then**
        Without loss of generality, say m_1 has children c_1 and c_2.
        return $\max(PairwiseMax(c_1, m_2), PairwiseMax(c_2, m_2))$
    else
        m_1 and m_2 are neurons, **return** $\mathcal{C}(m_1, m_2)$.
    end if
end function
while $|Q| > 1$ **do**
    $m_{j_1}, m_{j_2} = \arg\min_{m_1,m_2 \in Q}$ PairwiseMax(m_1, m_2)
    Add new node m_{j_3} to T
    Make m_{j_1}, m_{j_2} children of m_{j_3} in T
    Remove m_{j_1}, m_{j_2} from Q and add m_{j_3} to Q.
end while
G now is a tree, call it T

Output: Tree of merges T

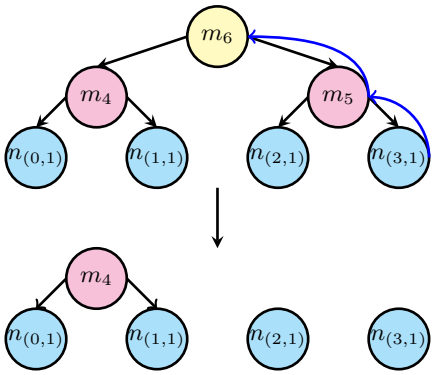

Fig. 5: Trees and Cuts

The tree produced in the previous Section 3.2 captures the semantic behavior, and we use it to guide the refinement process as follows: Any cut of the tree produces a set of trees. Then, the groups of neurons that we choose to keep merged correspond to the leaf nodes of these trees. Therefore, finding a refinement reduces to finding cuts in the tree.

To attempt to eliminate $\boldsymbol{\beta}$, we identify a *culprit neuron* γ that contributes most to the spurious output on $\boldsymbol{\beta}$. The intuition is that γ should not be merged with any other neuron, as any over-approximation of the behavior of γ has a high chance of introducing $\boldsymbol{\beta}$.

Thus, we do refinement in two steps. Firstly, we find the culprit neuron γ. Then, we find a cut in the tree that ensures that γ is not merged with any other neuron.

Finding $\boldsymbol{\gamma}$ Many possible strategies may be used to identify the culprit neuron γ, and our framework is agnostic to the specific strategy chosen. In our experiments, the strategy we chose is based on the *gradient-guided refinement* described in [7]. For each potential γ We calculate the following score, and chose γ with the largest score:

$$\|v_\gamma^*(\boldsymbol{\beta}) - v_\gamma(\boldsymbol{\beta})\|_2 \cdot \left|\frac{\delta y(\boldsymbol{\beta})}{\delta v_\gamma}\right|$$

Here, $v_\gamma(\boldsymbol{\beta})$ is the value at the neuron γ for input $\boldsymbol{\beta}$ in the original $\mathcal{N}$, while $v_\gamma^*(\boldsymbol{\beta})$ is the value of the neuron that γ has been merged into within our current $\mathcal{N}'$. $\frac{\delta y(\boldsymbol{\beta})}{\delta v_\gamma}$ is the partial derivative of the output y of $\mathcal{N}$ with respect to the value at γ for the input $\boldsymbol{\beta}$.

Cutting the Tree We wish to find a cut in the tree where γ is not merged with any other neuron, while also making sure that as many neurons remain merged as possible (therefore minimizing the increase in size of $\mathcal{N}'$). To do this, we delete precisely those nodes that are dependent on γ, starting from the parent of γ and moving up the tree following the parent links.

In our example, the culprit neuron is $n_{(3,1)}$. Thus, we traverse the tree following the blue edges in Figure 5, undoing m_5 and m_6. This produces three trees, corresponding to leaving $n_{(0,1)}$ and $n_{(1,1)}$ merged, while undoing the merge of $n_{(2,1)}$ and $n_{(3,1)}$. Therefore, we get the $\mathcal{N}'$ shown in Figure 6. Note that in contrast to the refinement process followed by [20] (Section 2.5), we retain merges of neurons that are semantically close ($n_{(0,1)}$ and $n_{(1,1)}$), avoid proliferation of singletons, and achieve a smaller $\mathcal{N}'$ that is sufficient to prove the property in fewer iterations.

Once we have cut the tree and decided on which neurons to leave merged, the actual merge operation is the exact same as that followed by [20] (Section 2.5). Therefore, we are able to retain concrete soundness guarantees.

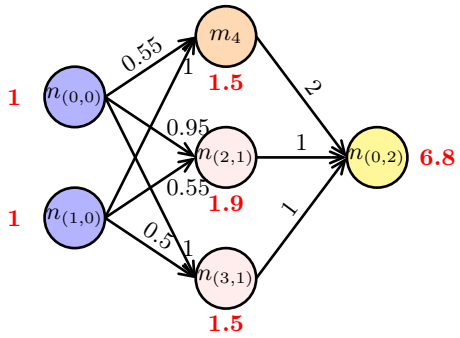

Fig. 6: Refining by our method: Culprit Neuron is 3

3.4 Optimality of Tree

Notation: Given a (sub)tree T of merge operations, we say a neuron $n_{(i,l)} \in T$ if and only if T has a leaf node corresponding to $n_{(i,l)}$. That is, $n_{(i,l)} \in T$ if and only if the merge operations forming T involve $n_{(i,l)}$ at any point.

The tree T produced in Section 3.2 captures an optimal ordering of merge operations with respect to the semantic information in the following sense:

Lemma 1. *Let T_1 and T_2 be two sub-trees of T. Then, we have:*

$$\max_{\substack{n_{(i_1,l)} \in T_1 \\ n_{(i'_1,l)} \in T_1}} \mathcal{C}(n_{(i_1,l)}, n_{(i'_1,l)}) \leq \max_{\substack{n_{(i_1,l)} \in T_1 \\ n_{(i_2,l)} \in T_2}} \mathcal{C}(n_{(i_1,l)}, n_{(i_2,l)})$$

Proof. This fact can be easily proved via induction on the combined size of T_1 and T_2. If a violation to the inequality exists, there may be two cases. In the first case, we have $n_{(i_1,l)}, n_{(i'_1,l)} \in T'_1$ where T'_1 is a strict sub-tree of T_1. ButT'_1 and T_2 would then form a violation of the induction hypothesis. The other case directly violates the pairwise maximum condition used in the construction of the tree in Algorithm 1 line 3 in Section 3.2.

Intuitively, the lemma shows that for any cut in the tree, the maximum difference in the semantic behavior of neurons that have been left merged is less than the maximum difference in the semantic behavior of neurons that have been un-merged. In particular, this implies that after each refinement step, the groups of neurons that remain merged together are optimal with respect to the semantic behavior of the network.

However, note that our semantic closeness metric fails to say anything about the value produced at the output layer of the network for any given input. Thus, although we have optimality with respect to semantic behavior, we are unable to predict the result making a cut would have on the output for the given spurious counterexamples. Attempting to make such a prediction, or provide some guarantees on the output of the network for a given spurious input, would nonetheless be an interesting direction for future work.

3.5 CEGAR [20,12] Loop Framework

We combine the pieces discussed so far into a CEGAR [20,12] loop. We start with the fully merged network. Then, utilizing spurious counterexamples, we iteratively refine the network until we have obtained an $\mathcal{N}'$ where there is no spurious counterexamples. This loop is parametrized by:

- The definition of the semantic closeness factor $\mathcal{C}$.
- The strategy for selecting the culprit neuron γ.

We note that while we have provided specific strategies for each of these, our framework is flexible enough so that, depending on the application, these pieces may be swapped out or modified to achieve better performance. We intend to study other strategies for each of these pieces in the future.

4 Experiments

We have implemented our method in python[5], utilizing the NumPy library for linear algebra operations and the SciPy library for an implementation of hierarchial clustering [24]. We have used a linkage-matrix based data structure similar to the one used in SciPy to store the tree, and have precomputed and cached several operations that may need to be repeated every refinement iteration. This allows us to quickly perform the merge and split operations and calculate the scores (Section 3.3), without having to do (relatively) expensive tree traversal operations in each iteration of the abstraction refinement loop.

Using this implementation, we have performed a set of experiments demonstrating the effectiveness of our abstraction technique for verification of neural network queries on the ACASXu [36,25] set of networks, using both the original safety properties from [25] and the ϵ-robustness properties introduced in [20]. To do so, we set up a CEGAR [20,12] loop (Section 3.5) using our abstraction technique, using the NeuralSAT [16] solver as the underlying solver to dispatch the verification queries on $\mathcal{N}'$. We compare our abstraction framework with the existing CEGAR [20,12] framework proposed in [20] [6] setting a timeout of 200 seconds for each instance in the benchmark and for both the our technique and the existing work. The experiments were run on a machine running on Intel(R) Core(TM) i7-6700 CPU with 8 CPUs running at 3.40GHz, having 16 GB RAM and running running Ubuntu 22.04 LTS.

[5] The entirety of the code, networks, datasets, and properties utilized in our evaluation will be made available via a publicly hosted code repository in the camera ready version.

[6] We have used a faithful re-implementation of this framework that follows exactly the procedure in the paper, with the only two distinctions being that we are using a two-class classification as seen in [8,28,29], and that the call to verify the $\mathcal{N}'$ obtained in each iteration is sent to an instance of the NeuralSAT [16] solver as opposed to Marabou [25,26,47].

If the $\mathcal{N}'$ produced has multiple neurons with the exact same set of incoming edges in the same layer, these neurons compute the same function and are redundant. Therefore, as an added optimization step, we safely *re-merge* them by taking the sum of the outgoing edges. Note that this does not change the behavior of $\mathcal{N}'$.

Method	No. Safe	No. Unsafe	No. Timeout	Average Size
Ours	121	43	16	335.3
Existing [20]	118	43	19	536.0

Table 1: Summary of ACASXu [36,25] on original safety properties

Method	Percentage Verified	Average Size
Ours	100%	27.9
Existing [20]	100%	31.5

Table 2: Summary of ACASXu [36,25] on robustness properties

Tables 1 and summarizes the results on these benchmarks. We find that using our framework, we are able to perform better than the existing CEGAR [20,12] approach [20] on the original safety properties, verifying more networks to be safe, while we do not loose performance on the robustness properties.

For each framework, we collected the final $\mathcal{N}'$ at the end of the CEGAR [20,12] iterations, for which either the property can be proved to be safe, or the solver is able to find an actual counterexample, or the solver times out. Figure 7 shows a scatter plot comparing the sizes of these final $\mathcal{N}'$ obtained by our framework and by existing work [20] for each instance in the benchmark. A point below the red diagonal line represents an instance for which we obtain a smaller final $\mathcal{N}'$ than the existing work, therefore points below the line represent instances for which we perform better. The average sizes of these $\mathcal{N}'$ over all instances are reported in the 'Average Size' columns in tables 1 and .

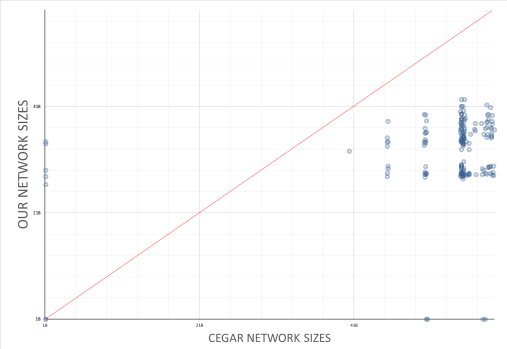

Fig. 7: Scatter plot of network sizes produced by our framework vs existing work [20]

It is apparent from the Figure 7, table 1 and table 4 that compared to the existing techniques, we are explore smaller $\mathcal{N}'$ that are effective at proving or disproving the property in question. This shows that using semantic information to guide the CEGAR [20,12] process can effectively find more efficient abstractions than the existing technique.

Note that in our experiments, we found that the time taken by both our CEGAR [20,12] approach and the existing CEGAR [20,12] approach [20] was more than what the NeuralSAT [16] solver takes for the ACASXu [36,25] benchmarks.

However, while we would expect the solver call times to exponentially scale with network size, the overheads from the abstraction procedure will not scale exponentially. Thus, for larger and larger benchmarks, being able to find smaller $\mathcal{N}'$ will produce a significant difference in times. Furthermore, we believe that a verified $\mathcal{N}'$ is useful beyond verifying a single property - it may be used for other related queries, or may be useful as a safely deployable compressed network.

Additionally, we find that the final solver times on the $\mathcal{N}'$ are actually comparable with the times obtained on the original un-abstracted $\mathcal{N}$. In general it has may observed, both by our experiments and in [20], that the effort needed to verify a network is dependent on more than just network size. In fact, in [20], they are able to achieve smaller solver times on larger networks. While it is true that in general the worst-case performance of neural network solvers will almost certainly remain exponential in the size of the network [25], other factors on which the performance of neural network solvers may depend on remains an interesting direction of future work.

5 Related Work

In response to the increasing use of neural networks in safety-critical settings, various techniques have been developed to analyze their behavior, including verification, explainability, etc.

Our work closely relates to the syntactic structural abstraction technique that proposes to reduce the network size by merging similar neurons [20,13,53], and employs a counterexample-guided refinement. They have used syntactic constraints to achieve concrete soundness guarantees. But unlike our approach, their work does not take into account the global *semantic* behavior of the network, thus producing potentially suboptimal abstractions. At the same time, the approach proposed in [1] performs a semantic analysis to decrease the network size, but it leads to an *approximation* instead of an abstraction. The approximation allows lifting certain bound propagation based proofs, but not the concrete gurantee that one gets from [20]. Also, in [7] the authors have used linear combinations of neurons to compress the networks, but their method only provides guarantees on a finite dataset. In comparison, we provide concrete soundness guarantees that allows us to lift any proofs from the abstract network back to the original network.

In general, methodologies for verifying neural networks generally fall into two main categories: sound and complete methods [25,19,30,21,17,10,26,34,46,49,50], and sound and incomplete methods [43,52,18,22,42,44]. Sound and complete methods aim to explore the entire state space to verify the properties of neural networks. In contrast, sound and incomplete methods employ an overapproximation of the state space, sacrificing completeness for scalability and efficiency.

An instance of a sound and complete methodology is Reluplex, which extends the simplex algorithm [14] to handle ReLU constraints. Initially, it focuses on finding an assignment that satisfies the linear constraints, subsequently incorporating non-linear constraints to validate their satisfaction. In [19] the authors

introduce triangular over-approximation, infer neuron phase fixtures, learn conflict clauses and safe neuron phase fixtures to aid in pruning the search space, which is similar to classical SMT solving approaches. Another complete technique is NeuralSAT [16], which performs exhaustive theory propagation and conflict clause learning similar to DPLL(T) used in classical SMT solving. However, these methods often encounter scalability issues due to their exhaustive exploration of the entire state space.

On the other hand, alternative techniques like [52,43], which propagates linear upper and lower bound constraints, exhibit better scalability at the cost of over-approximation. [49] optimizes the bounds of [52] using gradient descent. [46] incorporates ReLU split constraints into the CROWN bound propagation process, allowing integration into the branch and bound (BaB) framework [51,4,37]. This combined implementation makes the $\alpha, \beta - CROWN$ [52,48,40,49,46,51,50,41] framework sound and complete.

DNN compression has been explored [11] in general to obtain a network of a smaller size, however, as opposed to abstraction, they do not provide any guarantees connecting the original network to the smaller compressed network.

6 Conclusion and Future Work

This paper puts forth a framework to combine syntactic and semantic approaches for DNN abstraction. While this opens several directions for future work, an immediate question that can be asked is about achieving "optimal refinement". In the abstract network, it would be ideal to not have neurons that can be merged back without introducing spurious counterexamples. This implies that we should have added the minimum number of neurons necessary to prevent spurious counterexamples. This seems to depend on the direction in which one starts refining. Of the multiple refinement paths that are possible, is there one that is guaranteed to do a *minimal* refinement?

Exploring alternative measures for the semantic closeness factor $\mathcal{C}$ is another intriguing prospect. Note that our framework allows one to seamlessly experiment with any $\mathcal{C}$. Our current $\mathcal{C}$ does capture optimality with respect to the I/O behavior of the neurons, but we are not able to guarantee minimization of the over-approximation at the output layer. It would help to identify better metrics for the semantic closeness factor that minimize over-approximation at the output neuron.

In our experiments, we noticed that the time needed to verify a specific property of a network is not solely determined by the network's size. It is possible that a larger network can be verified quickly, while a smaller one may take longer or even fail to be verified altogether. Obtaining a more accurate measure of the effort required to verify a network would provide a better optimization target for abstraction used in the context of verification.

Acknowledgments.

References

1. Ashok, P., Hashemi, V., Kretínský, J., Mohr, S.: Deepabstract: Neural network abstraction for accelerating verification. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12302, pp. 92–107. Springer (2020)
2. Bassan, S., Katz, G.: Towards formal XAI: formally approximate minimal explanations of neural networks. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13993, pp. 187–207. Springer (2023)
3. Bojarski, M., Testa, D.D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., Zieba, K.: End to end learning for self-driving cars. CoRR **abs/1604.07316** (2016)
4. Bunel, R., Lu, J., Turkaslan, I., Torr, P.H.S., Kohli, P., Kumar, M.P.: Branch and bound for piecewise linear neural network verification. J. Mach. Learn. Res. **21**, 42:1–42:39 (2020)
5. Carlini, N., Katz, G., Barrett, C., Dill, D.L.: Provably minimally-distorted adversarial examples (2018)
6. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 39–57. IEEE Computer Society (2017)
7. Chau, C., Kretínský, J., Mohr, S.: Syntactic vs semantic linear abstraction and refinement of neural networks. In: André, É., Sun, J. (eds.) Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14215, pp. 401–421. Springer (2023)
8. Chauhan, A., Afzal, M., Karmarkar, H., Elboher, Y., Madhukar, K., Katz, G.: Efficiently finding adversarial examples with dnn preprocessing (2022)
9. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. CoRR **abs/1712.05526** (2017)
10. Cheng, C., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. CoRR **abs/1705.01040** (2017)
11. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. CoRR **abs/1710.09282** (2017)
12. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003)
13. Cohen, E., Elboher, Y.Y., Barrett, C.W., Katz, G.: Tighter abstract queries in neural network verification. In: Piskac, R., Voronkov, A. (eds.) LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023. EPiC Series in Computing, vol. 94, pp. 124–143. EasyChair (2023)
14. Dantzig, G.B.: Linear Programming and Extensions. RAND Corporation, Santa Monica, CA (1963). https://doi.org/10.7249/R366
15. Djavanshir, G.R., Chen, X., Yang, W.: A review of artificial intelligence's neural networks (deep learning) applications in medical diagnosis and prediction. IT Professional **23**(3), 58–62 (2021)

16. Duong, H., Nguyen, T., Dwyer, M.: A dpll(t) framework for verifying deep neural networks (2024)
17. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: Dutle, A., Muñoz, C.A., Narkawicz, A. (eds.) NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10811, pp. 121–138. Springer (2018)
18. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: Globerson, A., Silva, R. (eds.) Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018. pp. 550–559. AUAI Press (2018)
19. Ehlers, R.: Formal verification of piece-wise linear feed-forward neural networks. In: D'Souza, D., Kumar, K.N. (eds.) Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10482, pp. 269–286. Springer (2017)
20. Elboher, Y.Y., Gottschlich, J., Katz, G.: An abstraction-based framework for neural network verification. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12224, pp. 43–65. Springer (2020)
21. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints An Int. J. **23**(3), 296–309 (2018)
22. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.T.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 3–18. IEEE Computer Society (2018)
23. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
24. Jr., J.H.W.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association **58**(301), 236–244 (1963)
25. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kuncak, V. (eds.) Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10426, pp. 97–117. Springer (2017)
26. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 443–452. Springer (2019)
27. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings. OpenReview.net (2017)

28. Liu, J., Xing, Y., Shi, X., Song, F., Xu, Z., Ming, Z.: Abstraction and refinement: Towards scalable and exact verification of neural networks (2022)
29. Liu, J., Xing, Y., Shi, X., Song, F., Xu, Z., Ming, Z.: Abstraction and refinement: Towards scalable and exact verification of neural networks. ACM Trans. Softw. Eng. Methodol. (feb 2024). https://doi.org/10.1145/3644387, https://doi.org/10.1145/3644387, just Accepted
30. Lomuscio, A., Maganti, L.: An approach to reachability analysis for feed-forward relu neural networks. CoRR **abs/1706.07351** (2017), http://arxiv.org/abs/1706.07351
31. Mangal, A., Kalia, S., Rajgopal, H., Rangarajan, K., Namboodiri, V.P., Banerjee, S., Arora, C.: Covidaid: COVID-19 detection using chest x-ray. CoRR **abs/2004.09803** (2020)
32. Marques-Silva, J., Ignatiev, A.: Delivering trustworthy AI through formal XAI. In: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022. pp. 12342–12350. AAAI Press (2022)
33. Moosavi-Dezfooli, S., Fawzi, A., Frossard, P.: Deepfool: A simple and accurate method to fool deep neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 2574–2582. IEEE Computer Society (2016)
34. Neider, D., Johnson, T.T.: Track C1: safety verification of deep neural networks (dnns). In: Steffen, B. (ed.) Bridging the Gap Between AI and Reality - First International Conference, AISoLA 2023, Crete, Greece, October 23-28, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14380, pp. 217–224. Springer (2023)
35. Ostrovsky, M., Barrett, C.W., Katz, G.: An abstraction-refinement approach to verifying convolutional neural networks. In: Bouajjani, A., Holík, L., Wu, Z. (eds.) Automated Technology for Verification and Analysis - 20th International Symposium, ATVA 2022, Virtual Event, October 25-28, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13505, pp. 391–396. Springer (2022)
36. Owen, M.P., Panken, A., Moss, R., Alvarez, L., Leeper, C.: Acas xu: Integrated collision avoidance and detect and avoid capability for uas. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). pp. 1–10 (2019). https://doi.org/10.1109/DASC43569.2019.9081758
37. Palma, A.D., Bunel, R., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P.H.S., Kumar, M.P.: Improved branch and bound for neural network verification via lagrangian decomposition. CoRR **abs/2104.06718** (2021)
38. Pham, L.H., Sun, J.: Verifying neural networks against backdoor attacks. In: Shoham, S., Vizel, Y. (eds.) Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13371, pp. 171–192. Springer (2022)
39. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden. pp. 2651–2659. ijcai.org (2018)
40. Salman, H., Yang, G., Zhang, H., Hsieh, C.J., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. Advances in Neural Information Processing Systems **32**, 9835–9846 (2019)

41. Shi, Z., Jin, Q., Kolter, J.Z., Jana, S., Hsieh, C.J., Zhang, H.: Formal verification for neural networks with general nonlinearities via branch-and-bound. 2nd Workshop on Formal Verification of Machine Learning (WFVML 2023) (2023)
42. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.T.: Fast and effective robustness certification. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 10825–10836 (2018)
43. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 41:1–41:30 (2019)
44. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: Boosting robustness certification of neural networks. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019)
45. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014)
46. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 29909–29921 (2021)
47. Wu, H., Isac, O., Zeljic, A., Tagomori, T., Daggitt, M.L., Kokke, W., Refaeli, I., Amir, G., Julian, K., Bassan, S., Huang, P., Lahav, O., Wu, M., Zhang, M., Komendantskaya, E., Katz, G., Barrett, C.W.: Marabou 2.0: A versatile formal analyzer of neural networks. CoRR **abs/2401.14461** (2024)
48. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.: Automatic perturbation analysis for scalable certified robustness and beyond. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)
49. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. CoRR **abs/2011.13824** (2020)
50. Zhang, H., Wang, S., Xu, K., Li, L., Li, B., Jana, S., Hsieh, C.J., Kolter, J.Z.: General cutting planes for bound-propagation-based neural network verification. Advances in Neural Information Processing Systems (2022)
51. Zhang, H., Wang, S., Xu, K., Wang, Y., Jana, S., Hsieh, C.J., Kolter, Z.: A branch and bound framework for stronger adversarial attacks of ReLU networks. In: Proceedings of the 39th International Conference on Machine Learning. vol. 162, pp. 26591–26604 (2022)
52. Zhang, H., Weng, T., Chen, P., Hsieh, C., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada. pp. 4944–4953 (2018)

53. Zhao, Z., Zhang, Y., Chen, G., Song, F., Chen, T., Liu, J.: CLEVEREST: accelerating cegar-based neural network verification via adversarial attacks. In: Singh, G., Urban, C. (eds.) Static Analysis - 29th International Symposium, SAS 2022, Auckland, New Zealand, December 5-7, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13790, pp. 449–473. Springer (2022)