Efficient Neural Network Analysis with Sum-of-Infeasibilities

Haoze Wu[1]✉, Aleksandar Zeljić[1], Guy Katz[2], and Clark Barrett[1]

[1] Stanford University, Stanford, USA
[2] The Hebrew University of Jerusalem, Jerusalem, Israel

Abstract. Inspired by sum-of-infeasibilities methods in convex optimization, we propose a novel procedure for analyzing verification queries on neural networks with piecewise-linear activation functions. Given a convex relaxation which over-approximates the non-convex activation functions, we encode the violations of activation functions as a cost function and optimize it with respect to the convex relaxation. The cost function, referred to as the Sum-of-Infeasibilities (SoI), is designed so that its minimum is zero and achieved only if all the activation functions are satisfied. We propose a stochastic procedure, `DeepSoI`, to efficiently minimize the SoI. An extension to a canonical case-analysis-based complete search procedure can be achieved by replacing the convex procedure executed at each search state with `DeepSoI`. Extending the complete search with `DeepSoI` achieves multiple simultaneous goals: 1) it guides the search towards a counter-example; 2) it enables more informed branching decisions; and 3) it creates additional opportunities for bound derivation. An extensive evaluation across different benchmarks and solvers demonstrates the benefit of the proposed techniques. In particular, we demonstrate that SoI significantly improves the performance of an existing complete search procedure. Moreover, the SoI-based implementation outperforms other state-of-the-art complete verifiers. We also show that our technique can efficiently improve upon the perturbation bound derived by a recent adversarial attack algorithm.

Keywords: neural networks · sum of infeasibilities · convex optimization · stochastic local search.

1 Introduction

Neural networks have become state-of-the-art solutions in various application domains, e.g., face recognition, voice recognition, game-playing, and automated piloting [47,30,55,7]. While generally successful, neural networks are known to be susceptible to input perturbations that humans are naturally invariant to [61,41]. This calls the trustworthiness of neural networks into question, particularly in safety-critical domains.

In recent years, there has been a growing interest in applying formal methods to neural networks to analyze certain robustness or safety *specifications* [43]. Such specifications are often defined by a collection of partial input/output relations:

e.g., the network uniformly and correctly classifies inputs within a certain distance (in some l_p norm) of a selection of input points. The goal of formal verification is to either prove that the network meets the specification or to disprove it by constructing a counter-example.

Most standard activation functions in neural networks are non-linear, making them challenging to reason about. Consider the rectified linear unit (ReLU): if a ReLU can take both positive and negative inputs, a verifier will typically need to consider, separately, each of these two activation phases. Naive case analysis requires exploring a number of combinations that is exponential in the number of ReLUs, which quickly becomes computationally infeasible for large networks. To mitigate this complexity, neural network verifiers typically operate on convex relaxations of the activation functions. The relaxed problem can often be solved with an efficient convex procedure, such as Simplex [35,23] or (sub-)gradient methods [51,21]. Due to the relaxation, however, a solution may be inconsistent with the true activation functions. When this happens, the convex procedure cannot make further progress on its own. For this reason, to ensure completeness, the convex procedure is typically embedded in an exhaustive search shell, which encodes the activation functions explicitly and branches on them when needed. While the exhaustive search ensures progress, it also brings back the problem of combinatorial explosion. This raises the key question: **can we guide the convex procedure to satisfy the activation functions without explicitly encoding them?**

In convex optimization, the sum-of-infeasibilities (SoI) [10] function measures the error (with respect to variable bounds) of a variable assignment. Minimizing the SoI naturally guides the procedure to a satisfying assignment. In this paper, we extend this idea to instead represent the error in the non-linear activation functions. The goal is to "softly" guide the search over the relaxed problem using information about the precise activation functions. If an assignment is found for which the SoI is zero, then not only is the assignment a solution for the relaxation, but it also solves the precise problem. Encoding the SoI w.r.t. the piecewise-linear activation functions yields a concave piecewise-linear function, which is challenging to minimize directly. Instead, we propose to minimize the SoI for individual *activation patterns* and reduce the SoI minimization to a *stochastic* search for the activation pattern where the SoI is minimal. The advantage is that for each activation pattern, the SoI collapses into a *linear* cost function, which can be easily handled by a convex solver. We introduce a specialized procedure, `DeepSoI`, which uses Markov chain Monte Carlo (MCMC) search to efficiently navigate towards activation patterns at the global minimum of the SoI. If the minimal SoI is ever zero for an activation pattern, then a solution has been found.

An extension to a canonical complete search procedure can be achieved by replacing the convex procedure call at each search state with the `DeepSoI` procedure. Since the SoI contains additional information about the problem, we propose a novel SoI-aware branching heuristic based on the estimated impact of each activation function on the SoI. Finally, `DeepSoI` naturally preserves new bounds derived during the execution of the underlying convex procedure (e.g.,

Simplex), which further prunes the search space in the complete search. For simplicity, we focus on ReLU activation functions in this paper, though the proposed approach can be applied to any piecewise-linear activation function.

We implemented the proposed techniques in the Marabou framework for Neural Network Analysis [36] and performed an extensive performance evaluation on a wide range of benchmarks. We compare against multiple baselines and show that extending a complete search procedure with our SoI-based techniques results in significant overall speed-ups. Finally, we present an interesting use case for our procedure — efficiently improving the perturbation bounds found by AutoAttack [17], a state-of-the-art adversarial attack algorithm.

To summarize, the contributions of the paper are: (i) a technique for guiding a convex solver with an SoI function w.r.t. the activation functions; (ii) `DeepSoI`— a procedure for minimizing the non-linear SoI via the interleaving use of an MCMC sampler and a convex solver; (iii) an SoI-aware branching heuristic, which complements the integration of `DeepSoI` into a case-analysis based search shell; and (iv) a thorough evaluation of the proposed techniques.

The rest of the paper is organized as follows. Section 2 presents an overview of related work. Section 3 introduces preliminaries. Section 4 introduces the SoI and proposes a solution for its minimization. Section 5 presents the analysis procedure `DeepSoI`, its use in the complete verification setting, and an SoI-aware branching heuristic. Section 6 presents an extensive experimental evaluation. Conclusions and future work are in Section 7.

2 Related Work

Approaches to complete analysis of neural networks can be divided into SMT-based [35,36,23], reachability-analysis based [5,64,65,29,25], and the more general branch-and-bound approaches [1,63,24,44,13,37,9]. As mentioned in [14], these approaches are related, and differ primarily in their techniques for bounding and branching. Given the computational complexity of neural network verification, a diverse set of research directions aims to improve performance in practice. Many approaches prune the search space using tighter convex relaxations and bound inference techniques [64,23,31,58,56,45,76,70,67,66,20,63,69,52,62,51,73,26,68,59,8,57]. Another direction leverages parallelism by exploiting independent structures in the search space [48,75,71]. Different encodings of the neural network verification problems have also been studied: e.g., as MILP problems that can be tackled by off-the-shelf solvers [63,2], or as dual problems admitting efficient GPU-based algorithms [12,21,22,19]. `DeepSoI` can be instantiated with any sound convex relaxations and matching convex procedures. It can also be installed in any case-analysis-based complete search shell, therefore integrating easily with existing parallelization techniques, bound-tightening passes, and branching heuristics.

Two approaches most relevant to our work are Reluplex [35] and PeregriNN [37]. Reluplex invokes an LP solver to solve the relaxed problem, and then updates its solution to satisfy the violated activation functions — with the hope of nudging the produced solutions towards a satisfying assignment. However,

the updated solution by Reluplex could violate the linear relaxation, leading to non-convergent cycling between solution updates and LP solver calls, which can only be broken by branching. In contrast, our approach uses information about the precise activation functions to *actively* guide the convex solver. Furthermore, in the limit `DeepSoI` converges to a solution (if one exists). PeregriNN also uses an objective function to guide the solving of the convex relaxation. However, their objective function *approximates* the ReLU violation and does not guarantee a real counter-example when the minimum is reached. In contrast, the SoI function captures the *exact* ReLU violation, and if a zero-valued point is found, it is *guaranteed* to be a real counter-example. We compare our techniques to PeregriNN in Section 6.

We use MCMC-sampling combined with a convex procedure to minimize the concave piecewise-linear SoI function. MCMC-sampling is a common approach for stochastically minimizing irregular cost functions that are not amenable to exact optimization techniques [32,53,3]. Other stochastic local search techniques [54,27] could also be used for this task. However, we chose MCMC because it is adept at escaping local optima, and in the limit, it samples more frequently the region around the optimum value. As one point of comparison, in Section 6, we compare MCMC-sampling with a Walksat-based [54] local search strategy.

3 Preliminaries

Neural Networks. We define a feed-forward, convolutional, or residual neural network with $k+1$ layers as a set of *neurons* N, topologically ordered into *layers* $L_0, ..., L_k$, where L_0 is the input layer and L_k is the output layer. Given $n_i, n_j \in N$, we use $n_i \prec n_j$ to denote that the layer of n_i precedes the layer of n_j. The value of a neuron $n_i \in N \backslash L_0$ is computed as $act_i(b_i + \sum_{n_j \prec n_i} w_{ij} * n_j)$, an affine transformation of the preceding neurons followed by an activation function act_i. We use n_i^b and n_i^a to represent the pre- and post-activation values of such a neuron: $n_i^a = act_i(n_i^b)$. For $n_i \in L_0$, n_i^b is undefined and we assume n_i^a can take any value. In this paper, we focus on ReLU neural networks. That is, act_i is the ReLU function ($ReLU(x) = \max(0, x)$) unless n_i belongs to the output layer L_k, in which case act_i is the identity function. We use $R(N)$ to denote the set of ReLU neurons in N. An *activation pattern* is defined by choosing a particular phase (either active or inactive) for every $n \in R(N)$ (i.e., choosing either $n_i^b < 0$ or $n_i^b \geq 0$ for each $n_i \in R(N)$).

Neural Network Verification as Satisfiability. Consider the verification of a property P over a neural network N. The property P has the form $P_{in} \Rightarrow P_{out}$, where P_{in} and P_{out} constrain the input and output layers, respectively. P states that for each input point satisfying P_{in}, the output layer satisfies P_{out}. To formalize the verification problem, we first define the set of *variables* in a neural network N, denoted as $Var(N)$, to be $\cup_{n_i \in N \backslash L_k} \{n_i^a\} \cup \cup_{n_i \in N \backslash L_0} \{n_i^b\}$. We define a *variable assignment*, $\alpha : Var(N) \rightarrow \mathbb{R}$, to be a mapping from variables in N to real values. The verification task thus can be formally stated as finding a variable assignment α that satisfies the following set of *constraints* over $Var(N)$ (denoted

as ϕ):[3]

$$\forall n_i \in N \backslash L_0, n_i^b = b_i + \sum_{n_j \prec n_i} w_{ij} * n_j^a \tag{1a}$$

$$\forall n_i \in R(N), n_i^a = act_i(n_i^b) \tag{1b}$$

$$P_{in} \wedge \neg P_{out} \tag{1c}$$

If such an assignment α exists, we say that ϕ is *satisfiable* and can conclude that P does not hold, as from α we can retrieve an input $x \in P_{in}$, such that the neural network's output violates P_{out}. If such an α does *not* exist, we say ϕ is *unsatisfiable* and can conclude that P holds. We use $\alpha \models \phi$ to denote that an assignment α satisfies ϕ. In short, verifying whether P holds on a neural network N boils down to deciding the satisfiability of ϕ. We refer to ϕ also as the *verification query* in this paper.

Convex Relaxation of Neural Networks. Deciding whether P holds on a ReLU network N is NP-complete [35]. To curb intractability, many verifiers consider the convex (e.g., linear, semi-definite) relaxation of the verification problem, sacrificing completeness in exchange for a reduction in the computational complexity. We use $\widetilde{\phi}$ to denote the convex relaxation of the exact problem ϕ. If $\widetilde{\phi}$ is unsatisfiable, then ϕ is also unsatisfiable, and property P holds. If the convex relaxation is satisfiable with satisfying assignment α and α also satisfies ϕ, then P does not hold.

In this paper, we use the *Planet relaxation* introduced in [23]. It is a linear relaxation, illustrated in Figure 1. Each ReLU constraint $\text{ReLU}(n^b) = n^a$ is over-approximated by three linear constraints: $n^a \geq 0$, $n^a \geq n^b$, and $n^a \leq \frac{u}{u-l} n^b - \frac{u*l}{u-l}$, where u and l are the upper and lower bounds of n^b, respectively (which can be derived using bound-tightening techniques such as those in [67,58,76]). If Constraint 1c is also linear, the convex relaxation $\widetilde{\phi}$ is a Linear Program, whose satisfiability can be decided efficiently (e.g., using the *Simplex* algorithm [18]).

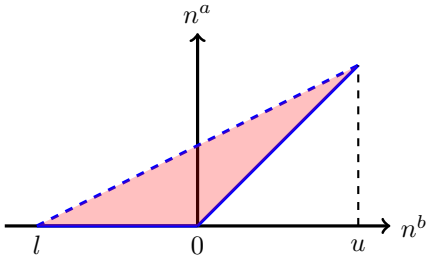

Fig. 1: The Planet relaxation.

Sum-of-Infeasibilities. In convex optimization [10,39], the sum-of-infeasibilities (SOI) method can be used to direct the feasibility search. The satisfiability of a formula ϕ is cast as an optimization problem, with an objective function representing the total *error* (i.e., the sum of the distances from each out-of-bounds variable to its closest bound). The lower bound of f is 0 and is achieved only if ϕ is satisfiable. In our context, we use a similar function f_{soi}, but with the difference that it represents the total error of the ReLU constraints in ϕ. In our case, f_{soi} is non-convex, and thus a more sophisticated approach is needed to minimize it efficiently.

[3] The verification can also be equivalently viewed as an optimization problem [14].

Complete Analysis via Exhaustive Search. One common approach for complete verification involves constructing a search tree and calling a *convex procedure* solveConv at each tree node, as shown in Algorithm 1. solveConv solves the convex relaxation $\widetilde{\phi}$ and returns a pair r, α where either: 1) $r = \texttt{SAT}$ and $\alpha \models \widetilde{\phi}$; or 2) $r = \texttt{UNSAT}$ and $\widetilde{\phi}$ is unsatisfiable. If $\widetilde{\phi}$ is unsatisfiable or α also satisfies ϕ, then the result for $\widetilde{\phi}$ also holds for ϕ and is returned. Otherwise, the search space is divided further using branch, which returns a set Ψ of sub-problems such that ϕ and $\bigvee \Psi$ are equisatisfiable.

Before invoking solveConv to solve $\widetilde{\phi}$, it is common to first call an efficient bound-tightening procedure (tightenBounds) to prune the search space or even derive UNSAT preemptively. This tightenBounds procedure can be instantiated in various ways, including with analyses based on LiPRA [74,76,58,70], kReLU [56], or PRIMA [49]. In addition to the dedicated bound-tightening pass, some convex procedures (e.g., Simplex) also naturally lend themselves to bound inference during their executions [38,35]. The overall performance of Algorithm 1 depends on the efficacy of bound-tightening, the branching heuristics, and the underlying convex procedure.

Algorithm 1 Complete search.

```
1: Input: a verification query φ.
2: Output: SAT/UNSAT
3: function completeSearch(φ)
4:     φ ← tightenBounds(φ)
5:     r, α ← solveConv(φ̃)
6:     if r = UNSAT ∨ α ⊨ φ then
7:         return r
8:     for φ_i ∈ branch(φ) do
9:         if completeSearch(φ_i) = SAT then
10:            return SAT
11:    return UNSAT
```

Adversarial attacks. Adversarial attacks [61,46,28,15] are another approach for assessing neural network robustness. While verification uses exhaustive search to either prove or disprove a particular specification, adversarial attacks focus on efficient heuristic algorithms for the latter. From another perspective, they can demonstrate *upper bounds* on neural network robustness. In Section 6, we show that our analysis procedure can improve the bounds found by AutoAttack [17].

4 Sum of Infeasibilities in Neural Network Analysis

In this section, we introduce our SoI function, consider the challenge of its minimization, and present a stochastic local search solution.

4.1 The Sum of Infeasibilities

As mentioned above, in convex optimization, an SoI function represents the sum of errors in a candidate variable assignment. Here, we build on this idea by introducing a cost function f_{soi}, which computes the sum of errors introduced by a convex relaxation of a verification query. We aim to use f_{soi} to reduce the satisfiability problem for ϕ to a simpler optimization problem. We will need the following property to hold.

Condition 1. *For an assignment* α, $\alpha \models \phi$ *iff* $\alpha \models \widetilde{\phi} \wedge f_{soi} \leq 0$.

If Condition 1 is met, then satisfiability of ϕ reduces to the following minimization problem:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & f_{soi} \\ \text{subject to} \quad & \alpha \models \widetilde{\phi} \end{aligned} \tag{2}$$

To formulate the SoI for ReLU networks, we first define the error in a ReLU constraint n as:

$$\mathbf{E}(n) = \min(n^a - n^b, n^a) \tag{3}$$

The two arguments correspond to the error when the ReLU is in the active and inactive phase, respectively. Recall that the Planet relaxation constrains (n^b, n^a) in the triangular area in Figure 1, where $n^a \geq n^b$ and $n^a \geq 0$. Thus, the minimum of $\mathbf{E}(n)$ subject to $\widetilde{\phi}$ is non-negative, and furthermore, $\mathbf{E}(n) = 0$ iff the ReLU constraint n is satisfied (this is also true for any relaxation at least as tight as the Planet relaxation). We now define f_{soi} as the sum of errors in individual ReLUs:

$$f_{soi} = \sum_{n \in R(N)} \mathbf{E}(n) \tag{4}$$

Theorem 1. *Let N be a set of neurons for a neural network, ϕ a verification query (an instance of (1)), and $\widetilde{\phi}$ the planet relaxation of ϕ. Then f_{soi} as given by (4) satisfies Condition 1.*

Proof. It is straightforward to show that f_{soi} subject to $\widetilde{\phi}$ is non-negative and is zero if and only if each $\mathbf{E}(n_i)$ is zero. That is, $\min f_{soi}$ subject to $\widetilde{\phi}$ is zero if and only if all ReLUs are satisfied. Therefore, if α satisfies ϕ, then $\alpha \models f_{soi} = 0$. On the other hand, since an assignment α that satisfies $\widetilde{\phi}$ can only violate the ReLU constraints in ϕ, if $\alpha \models f_{soi} = 0$, then all the constraints in ϕ must be satisfied, i.e., $\alpha \models \phi$. □

Note that the error $\mathbf{E}$, and its extension to SoI, can easily be defined for other piecewise-linear functions besides ReLU. We now turn to the question of minimizing f_{soi}. Observe that

$$\min f_{soi} = \min \sum_{n \in R(N)} \mathbf{E}(n) = \min \left(\left\{ f \mid f = \sum_{n_i \in R(N)} t_i, \quad t_i \in \{n_i^a - n_i^b, n_i^a\} \right\} \right). \tag{5}$$

Thus, f_{soi} is the minimum over a set, which we will denote S_{soi}, of linear functions. Although $\min f_{soi}$ cannot be used directly as an objective in a convex procedure, we could minimize each individual linear function $f \in S_{soi}$ with a convex procedure and then keep the minimum over all functions. We refer to the functions in S_{soi} as *phase patterns* of f_{soi}. For notational convenience, we define $cost(f, \phi)$ to be the minimum of f subject to ϕ. The minimization problem (2) can thus be restated as searching for the phase pattern $f \in S_{soi}$, where $cost(f, \widetilde{\phi})$ is minimal. Note that for a particular *activation pattern*, $f_{soi} = f$ for some $f \in S_{soi}$. From this perspective, searching for the $f \in S_{soi}$ where $cost(f, \widetilde{\phi})$ is minimal can also be viewed as searching for the activation pattern where the global minimum of f_{soi} is achieved.

4.2 Stochastically Minimizing the SoI with MCMC Sampling

In the worst case, finding the minimal value of $cost(f, \widetilde{\phi})$ requires enumerating and minimizing each f in S_{soi} (or equivalently, minimizing f_{soi} for each activation pattern), which has size $2^{|R(N)|}$. However, importantly, the search can terminate immediately if a phase pattern f is found such that $cost(f, \widetilde{\phi}) = 0$. We leverage this fact below. Note that each phase pattern has $|R(N)|$ adjacent phase patterns, each differing in only one linear subexpression. The space of phase patterns is thus fairly dense, making it amenable to traversal using stochastic local search methods. In particular, intelligent hill-climbing algorithms, which can be made robust against local optima, are well suited for this task.

Markov chain Monte Carlo (MCMC) [11] methods are such an approach. In our context, MCMC methods can be used to generate a sequence of phase patterns $f_0, f_1, f_2 ... \in S_{soi}$, with the desirable property that in the limit, the phase patterns are more frequently from the minimum region of $cost(f, \widetilde{\phi})$.

We use the Metropolis-Hastings (M-H) algorithm [16], a widely applicable MCMC method, to construct the sequence. The algorithm maintains a current phase pattern f and proposes to replace f with a new phase pattern f'. The proposal comes from a *proposal distribution* $q(f'|f)$ and is accepted with a certain *acceptance probability* $m(f \rightarrow f')$. If the proposal is accepted, f' becomes the new current phase pattern. Otherwise, another proposal is considered. This process is repeated until one of the following scenarios happen: 1) a phase pattern f is chosen with $cost(f, \widetilde{\phi}) = 0$; 2) a predetermined computational budget is exhausted; or 3) all possible phase patterns have been considered. The last scenario is generally infeasible for non-trivial networks. In order to employ the algorithm, we transform $cost(f, \widetilde{\phi})$ into a probability distribution $p(f)$ using a common method [34]:

$$p(f) \propto \exp(-\beta \cdot cost(f, \widetilde{\phi}))$$

where β is a configurable parameter. If the proposal distribution is symmetric (i.e., $q(f|f') = q(f'|f)$), the acceptance probability is the following (often referred to as the *Metropolis ratio*) [34]:

$$m(f \rightarrow f') = \min(1, \frac{p(f')}{p(f)}) = \min\left(1, \exp\left(-\beta \cdot \left(cost(f', \widetilde{\phi}) - cost(f, \widetilde{\phi})\right)\right)\right)$$

Importantly, under this acceptance probability, *a proposal reducing the value of the cost function is always accepted, while a proposal that does not may still be accepted* (albeit with a probability that is inversely correlated with the increase in the cost). This means that the algorithm always greedily moves to a lower cost phase pattern whenever it can, but it also has an effective means for escaping local minima. Note that since the sample space is finite, as long as the proposal strategy is *ergodic*,[4] in the limit, the probability of sampling *every* phase pattern (therefore deciding the satisfiability of ϕ) converges to 1. However, we do not

[4] A proposal strategy is ergodic if it is capable of transforming any phase pattern to any other through a sequence of applications. We use a symmetric and ergodic proposal distribution as explained in Section 5.1.

have formal guarantees about the convergence rate, and it is usually impractical to prove unsatisfiability this way. Instead, as we shall see in the next section, we enable complete verification by embedding the M-H algorithm in an exhaustive search shell.

5 The DeepSoI Algorithm

In this section, we introduce `DeepSoI`, a novel verification algorithm that leverages the SoI function, and show how to integrate it with a complete verification procedure. We also discuss the impact of `DeepSoI` on complete verification and propose an SoI-aware branching heuristic.

5.1 DeepSoI

Our procedure `DeepSoI`, shown in Algorithm 2, takes an input verification query ϕ and tries to determine its satisfiability. `DeepSoI` follows the standard two-phase convex optimization approach. Phase I finds *some* assignment α_0 satisfying $\widetilde{\phi}$, and phase II attempts to optimize the assignment using the M-H algorithm. Phase II uses a convex optimization procedure OPTIMIZECONV which takes an objective function f and a formula ϕ as inputs and returns a pair α, c, where $\alpha \models \phi$ and $c = cost(f, \phi)$ is the optimal value of f. Phase II chooses an initial phase pattern f based on α_0 (Line 6) and computes its optimal value c. The M-H algorithm repeatedly proposes a new phase pattern f' (Line 9), computes its optimal value c', and decides whether to accept f' as the current phase pattern f. The procedure returns `SAT` when a phase pattern f is found such that $cost(f, \widetilde{\phi}) = 0$ and `UNSAT` if all phase patterns have been considered (EXHAUSTED returns true) before a threshold of T rejections is exceeded. Otherwise, the analysis is inconclusive (`UNKNOWN`).

Algorithm 2 Analyzing ϕ with `DeepSoI`.

```
1:  Input: A verification query φ.
2:  Output: SAT/UNSAT/UNKNOWN
3:  function DeepSoI(φ)
4:      r, α_0 ← solveConv(φ̃)
5:      if r = UNSAT ∨ α_0 ⊨ φ then return r, α_0          } Phs. I
6:      k, f ← 0, initPhasePattern(α_0)
7:      α, c ← optimizeConv(f, φ̃)
8:      while c > 0 ∧ ¬exhausted() ∧ k < T do
9:          f' ← propose(f)
10:         α', c' ← optimizeConv(f', φ̃)                  } Phs. II
11:         if accept(c, c') then f, c, α ← f', c', α'
12:         else k ← k + 1
13:     if c = 0 then return SAT, α
14:     else return exhausted() ? UNSAT : UNKNOWN
```

The ACCEPT method decides whether a proposal is accepted based on the Metropolis ratio (see Section 4). Function INITPHASEPATTERN proposes the initial phase pattern f induced by the activation pattern corresponding to assignment α_0. Our proposal strategy (PROPOSE) is also simple: pick a ReLU n at random and flip its cost component in the current phase pattern f (either from $n^a - n^b$ to n^a, or vice-versa). This proposal strategy is symmetric, ergodic,

and performs well in practice. Both the initialization strategy and the proposal strategy are crucial to the performance of the M-H Algorithm, and exploring more sophisticated strategies is a promising avenue for future work. Importantly, the same convex procedure is used in both phases. Therefore, from the perspective of the convex procedure, `DeepSoI` solves a sequence of convex optimization problems that differ only in the objective functions, and each problem can be solved incrementally by updating the phase pattern without the need for a restart.

5.2 Complete Analysis and Pseudo-impact Branching

To extend a canonical complete verification procedure (i.e., Algorithm 1), its SOLVECONV call is replaced with `DeepSoI`. Note that the implementation of BRANCH in this algorithm has a significant influence on its performance. Here, we consider an SoI-aware implementation of BRANCH, which makes decisions by selecting a particular ReLU to be active or inactive. The choice of *which* ReLU is crucial. Intuitively, we want to branch on the ReLU with the most impact on the value of f_{soi}. After branching, `DeepSoI` should be closer to either: finding a satisfying assignment (if f_{soi} is decreased), or determining unsatisfiability (if f_{soi} is increased). Computing the exact impact of each ReLU n on f_{soi} would be expensive; however, we can estimate it by recording changes in f_{soi} during the execution of `DeepSoI`.

Concretely, for each ReLU n, we maintain its *pseudo-impact*,[5] $\mathbf{PI}(n)$, which represents the estimated impact of n on f_{soi}. For each n, $\mathbf{PI}(n)$ is initialized to 0. Then during the M-H algorithm, whenever the next proposal flips the cost component of ReLU n, we calculate the local impact on f_{soi}: $\Delta = |cost(f, \widetilde{\phi}) - cost(f', \widetilde{\phi})|$. We use Δ to update the value of $\mathbf{PI}(n)$ according to the *exponential moving average* (EMA): $\mathbf{PI}(n) = \gamma * \mathbf{PI}(n) + (1 - \gamma) \cdot \Delta$, where γ attenuates previous estimates of n's impact. We use EMA because recent estimates are more likely to be relevant to the current phase pattern. At branching time, the *pseudo-impact heuristic* picks $\arg\max_n \mathbf{PI}(n)$ as the ReLU to split on. The heuristic is inaccurate early in the search, so we use a static branching order (e.g., [71,13]) while the depth of the search tree is shallow (e.g., < 3).

6 Experimental Evaluation

In this section, we present an experimental evaluation of the proposed techniques. Our experiments include: 1. an ablation study to examine the effect of each proposed technique; 2. a run-time comparison of our prototype with other complete analyzers; 3. an empirical study of the choice of the rejection threshold T in Algorithm 2; and 4. an experiment in using our analysis procedure to improve the perturbation bounds found by AutoAttack [17], an adversarial attack algorithm. An artifact with which the results can be replicated is available on Zenodo [72].

[5] The name is in analogy to pseudo-cost branching heuristics in MILP, where the integer variable with the largest impact on the objective function is chosen [6].

6.1 Implementation.

We implemented our techniques in Marabou [36], an open-source toolbox for analyzing Neural Networks. It features a user-friendly python API for defining properties and loading networks, and a native implementation of the Simplex algorithm. Besides the Markov chain Monte Carlo stochastic local search algorithm presented in Section 5.1 and the pseudo-impact branching heuristic presented in Section 5.2, we also implemented a Walksat-inspired [54] stochastic local search strategy to evaluate the effectiveness of MCMC-sampling as a local minimization strategy. Concretely, from a phase pattern f, the strategy greedily moves to a neighbor f' of f, with $cost(f', \widetilde{\phi}) < cost(f, \widetilde{\phi})$. If no such f' exists (i.e., a local minimum has been reached), the strategy moves to a random neighbor.

The SOLVECONV and OPTIMIZECONV methods in Algorithm 2 can be instantiated with either the native Simplex engine of Marabou or with Gurobi, an off-the-shelf (MI)LP-solver. The TIGHTENBOUNDS method is instantiated with the DeepPoly analysis from [58], an effective and light-weight bound-tightening pass, which is also implemented in Marabou.

6.2 Benchmarks.

We evaluate on networks from four different applications: `MNIST`, `CIFAR10`, `TaxiNet`, and `GTSR`. The network architectures are shown in Table 2.

The `MNIST` [42] and `CIFAR10` [40] networks are established benchmarks used in previous literature (e.g., [19,37,71,75]) as well as in the 2021 VNN Competition [4]. Notably, the same `MNIST` networks are used to evaluate the original PeregriNN work.

Bench.	Model	Type	ReLUs	Hid. Layers
MNIST	$\texttt{MNIST}_1$	FC	512	2
	$\texttt{MNIST}_2$	FC	1024	4
	$\texttt{MNIST}_3$	FC	1536	6
TaxiNet	Taxi1	Conv	688	6
	Taxi2	Conv	2048	4
	Taxi3	Conv	2752	6
CIFAR10	$\texttt{CIFAR10}_\texttt{b}$	Conv	1226	4
	$\texttt{CIFAR10}_\texttt{w}$	Conv	4804	4
	$\texttt{CIFAR10}_\texttt{d}$	Conv	5196	6
GTSR	$\texttt{GTSR}_1$	FC	600	3
	$\texttt{GTSR}_2$	Conv	2784	4

Fig. 2: Architecture overview.

For `MNIST` and `CIFAR10` networks, we check robustness against targeted l_∞ attacks on randomly selected images from the test sets. The target labels are chosen randomly from the incorrect labels, and the perturbation bound is sampled uniformly from $\{0.01, 0.03, 0.06, 0.09, 0.12, 0.15\}$. The `TaxiNet` [33] benchmark set comprises robustness queries over regression models used for vision-based autonomous taxiing. Given an image of the taxiway captured by the aircraft, the model predicts its displacement (in meters) from the center of the taxiway. A controller uses the output to adjust the heading of the aircraft. Robustness is parametrized by input perturbation δ and output perturbation ϵ; we sample (δ, ϵ) uniformly from $\{0.01, 0.03, 0.06\} \times \{2, 6\}$. The `GTSR` benchmark set comprises robustness queries on image classifiers trained on a subset of the German Traffic Sign Recognition benchmark set [60]. Given a 32×32 RGB image the networks classify it as one of seven different kinds of traffic signs. A hazing perturbation [50] drains color from the image to create a veil of colored mist. Given an image I, a

Bench. (#)	$\texttt{MILP}_{\texttt{MIPVerify}}$		$\texttt{LP}^{\texttt{snc}}$		$\texttt{SOI}^{\texttt{snc}}_{\texttt{mcmc}}$		$\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$		$\texttt{SOI}^{\texttt{pi}}_{\texttt{wsat}}$	
	Solv.	Time	Solv.	Time	Solv.	Time	Solv.	Time	Solv.	Time
$\texttt{MNIST}_1$ (90)	**77**	19791	47	6892	66	5635	70	5976	68	5388
$\texttt{MNIST}_2$ (90)	29	6125	24	514	**36**	4356	31	757	31	909
$\texttt{MNIST}_3$ (90)	23	957	21	1609	34	9519	**35**	8327	33	5270
$\texttt{Taxi}_1$ (90)	**90**	786	61	9054	80	4257	89	1390	90	1489
$\texttt{Taxi}_2$ (90)	40	17093	2	891	70	5503	**71**	6889	71	7407
$\texttt{Taxi}_3$ (90)	**89**	5058	64	69715	87	1034	88	2164	87	997
$\texttt{CIFAR10}_\texttt{b}$ (90)	**76**	4316	26	7425	69	6286	73	16469	69	5200
$\texttt{CIFAR10}_\texttt{w}$ (90)	38	9879	18	845	41	4619	42	8129	**42**	6415
$\texttt{CIFAR10}_\texttt{d}$ (90)	30	4198	21	3395	51	17679	51	15056	51	15015
$\texttt{GTSR}_1$ (90)	90	2541	**90**	2435	89	4900	90	15238	90	4805
$\texttt{GTSR}_2$ (90)	90	23613	**90**	4456	90	7507	90	10426	90	6180
Total (990)	673	94354	463	107230	711	71294	**730**	90822	721	59073

Table 1: Instances solved by different configurations and their runtime (in seconds) on solved instances.

perturbation parameter ϵ, and a haze color C^f, the perturbed image I' is equal to $(1-\epsilon)\cdot I + \epsilon \cdot C^f$. The robustness queries check whether the bound yielded by the test-based method in [50] is minimal.All pixel values are normalized to $[0, 1]$, and the chosen perturbation values yield a mix of non-trivial `SAT` and `UNSAT` instances.

6.3 Experimental Setup.

Experiments are run on a cluster equipped with Intel Xeon E5-2637 v4 CPUs running Ubuntu 16.04. Unless specified otherwise, each job is run with 1 thread, 8GB memory, and a 1-hour CPU timeout. By default, the SOLVECONV and OPTIMIZECONV methods use Gurobi. The following hyper-parameters are used: the rejection threshold T in Algorithm 2 is 2; the discount factor γ in the EMA is 0.5; and the probability density parameter β in the Metropolis ratio is 10. These parameters are empirically optimal on a subset of MNIST benchmarks. In practice, the performance is most sensitive to the rejection threshold T, and below (Section 6.6), we conduct experiments to study its effect.

6.4 Ablation study of the proposed techniques.

To evaluate each individual component of our proposed techniques, we run several configurations across the full set of benchmarks described above.

We first consider two baselines that do not minimize the SoI: 1. $\texttt{LP}^{\texttt{snc}}$— runs Algorithm 1 with the Split-and-Conquer (SnC) branching heuristic [71], which estimates the number of tightened bounds from a ReLU split; 2. $\texttt{MILP}_{\texttt{MIPVerify}}$— encodes the query in Gurobi using MIPVerify's MILP encoding [63].[6]

We then evaluate three configurations of SoI-based complete analysis parameterized by the branching heuristic and the SoI-minimization algorithm: 1. $\texttt{SOI}^{\texttt{snc}}_{\texttt{mcmc}}$—

[6] This configuration does not use the LP/MILP-based preprocessing passes from MIPVerify [63] because they degrade performance on our benchmarks.

Bench. (#)	$\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$		PeregriNN		$\texttt{ERAN}_1$		$\texttt{ERAN}_2$	
	Solv.	Time	Solv.	Time	Solv.	Time	Solv.	Time
$\texttt{MNIST}_1$ (90)	70	5976	64	11117	**76**	18679	75	19520
$\texttt{MNIST}_2$ (90)	**31**	757	31	2287	28	1910	28	3126
$\texttt{MNIST}_3$ (90)	**35**	8327	26	2344	24	1538	24	3292
$\texttt{Taxi}_1$ (90)	89	1390	-	-	**90**	1653	90	3262
$\texttt{Taxi}_2$ (90)	**71**	6889	-	-	40	16460	35	31778
$\texttt{Taxi}_3$ (90)	88	2164	-	-	**88**	1389	88	4581
$\texttt{CIFAR10}_\texttt{b}$ (90)	73	16469	-	-	**77**	4604	77	14269
$\texttt{CIFAR10}_\texttt{w}$ (90)	**42**	8129	-	-	41	14403	37	14453
$\texttt{CIFAR10}_\texttt{d}$ (90)	**51**	15056	-	-	31	7587	26	5245
$\texttt{GTSR}_1$ (90)	90	15238	-	-	**90**	2023	90	32585
$\texttt{GTSR}_2$ (90)	**90**	10426	-	-	78	77829	75	81232
Total (990)	**730**	90822	-	-	663	148075	645	213343

Table 2: Instances solved by different complete verifiers and their runtime (in seconds) on solved instances.

runs `DeepSoI` with the SnC branching heuristic; 2. $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$— runs `DeepSoI` with the pseudo-impact (PI) heuristic; 3. $\texttt{SOI}^{\texttt{pi}}_{\texttt{wsat}}$— runs the Walksat-based algorithm with the PI heuristic. Each SoI configuration differs in one parameter w.r.t. the previous, so that pair-wise comparison highlights the effect of that parameter.

Table 1 summarizes the runtime performance of different configurations on the four benchmark sets. The three configurations that minimize the SoI, namely $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$, $\texttt{SOI}^{\texttt{pi}}_{\texttt{wsat}}$ and $\texttt{SOI}^{\texttt{snc}}_{\texttt{mcmc}}$, all solve significantly more instances than the two baseline configurations. In particular, $\texttt{SOI}^{\texttt{snc}}_{\texttt{mcmc}}$ solves 248 (53.4%) more instances than $\texttt{LP}^{\texttt{snc}}$. Since all configurations start with the same variable bounds derived by the DeepPoly analysis, the performance gain is mainly due to the use of SoI.

Among the three SoI configurations, the one with both `pi` and `mcmc` solves the most instances. In particular, it solves 8 more instances than $\texttt{SOI}^{\texttt{pi}}_{\texttt{wsat}}$, suggesting that MCMC sampling is, overall, a better approach than the Walksat-based strategy. On the other hand, $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$ and $\texttt{SOI}^{\texttt{snc}}_{\texttt{mcmc}}$ show complementary behaviors. For instance, the latter solves 5 more instances on $\texttt{MNIST}_1$, and the former solves 11 more on the `Taxi` benchmarks. This motivates a portfolio configuration $\texttt{SOI}_{\texttt{portfolio}}$,which runs $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$ and $\texttt{SOI}^{\texttt{snc}}_{\texttt{mcmc}}$ in parallel. This strategy is able to solve 742 instances overall with a 1-hour wall-clock timeout, yielding a gain of at least 12 more solved instances compared with any single-threaded configuration.

6.5 Comparison with other complete analyzers.

In this section, we compare our implementation with other complete analyzers. We first compare with PeregriNN, which as described in Section 2 introduces a heuristic cost function to guide the search. We evaluate PeregriNN on the `MNIST` networks, the same set of networks used in its original evaluation. We did not run PeregriNN on the other benchmarks because it only supports `.nnet` format, which is designed for fully connected feed-forward ReLU networks.

In addition, we also compare with ERAN, a state-of-the-art complete analyzer based on abstract interpretation, on the full set of benchmarks. ERAN is often used as a strong baseline in recent neural network verification literature and was

among the top performers in the past VNN Competition 2021. We compare with two ERAN configurations: 1. $\texttt{ERAN}_1$ — ERAN using the DeepPoly analysis [58] for abstract interpretation and Gurobi for solving; 2. $\texttt{ERAN}_2$ — same as above except using the k-ReLU analysis [56] for abstract interpretation. We choose to compare with ERAN instead of other state-of-the-art neural network analyzers, e.g., alpha-beta crown [76,68], OVAL [19], and fast-and-complete [75], mainly because the latter tools are GPU-based, while ERAN supports execution on CPU, where our prototype is designed to run. This makes a fair comparison possible. Note that our goal in this section is not to claim superiority over all state-of-the-art solvers. Rather, the goal is to provide assurance that our implementation is reasonable. As explained earlier, our approach can be integrated into other complete search shells with different search heuristics, and is orthogonal to techniques such as GPU-acceleration, parallelization, and tighter convex relaxation (e.g., beyond the Planet relaxation), which are all future development directions for Marabou.

Table 2 summarizes the runtime performance of different solvers. We include again our best configuration, $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$, for ease of comparison. On the three `MNIST` benchmark sets, PeregriNN either solves fewer instances than $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$ or takes longer time to solve the same number of instances. We note that PeregriNN's heuristic objective function could be employed during the feasibility check of `DeepSoI` (Line 4, Algorithm 2). Exploring this complementarity between PeregriNN and our approach is left as future work.

Compared with $\texttt{ERAN}_1$ and $\texttt{ERAN}_2$, $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$ also solves significantly more instances overall, with a performance gain of at least 10.1% more solved instances. Taking a closer look at the performance breakdown on individual benchmarks, we observe complementary behaviors between $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$ and $\texttt{ERAN}_1$, with the latter solving more instances than $\texttt{SOI}^{\texttt{pi}}_{\texttt{mcmc}}$ on 3 of the 11 benchmark sets. Figure 3 shows the cactus plot of configurations that run on all benchmarks. $\texttt{ERAN}_1$ is able to solve more instances than all the other configurations when the time limit is short, but is overtaken by the three SoI-based configurations once the time limit exceeds 30s. One explanation for this is that the SoI-enabled configurations spend more time probing at each search state, and for easier instances, it might be more beneficial to branch eagerly.

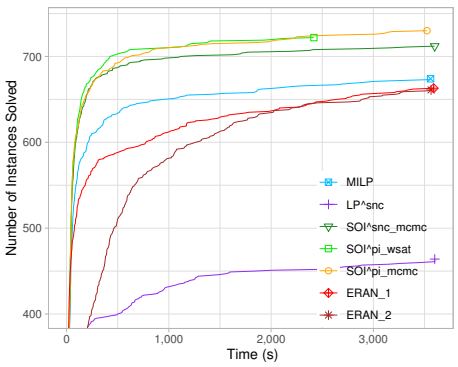

Fig. 3: Cactus plot on all benchmarks.

Finally, we compare the portfolio strategy $\texttt{SOI}_{\texttt{portfolio}}$ described in the previous subsection to $\texttt{ERAN}_1$ running 2 threads. The latter solves 10.3% fewer instances (673 overall). Figure 4 shows a scatter plot of the runtime performance of these two configurations. For unsatisfiable instances, most can be resolved efficiently by both solvers, and each solver has a few unique solves. On the other hand, $\texttt{SOI}_{\texttt{portfolio}}$ is able to solve significantly more satisfiable benchmarks.

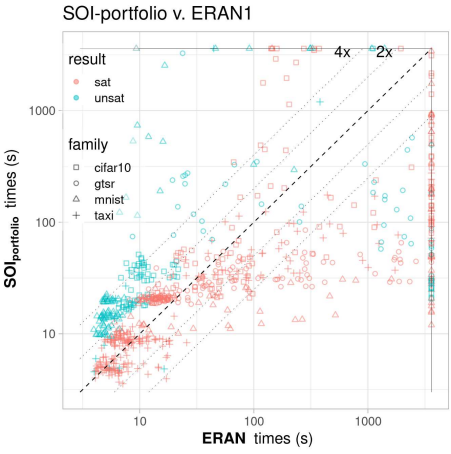

Fig. 4: Runtime of $\texttt{SOI}_{\texttt{portfolio}}$ and $\texttt{ERAN}_1$ running with 2 threads.

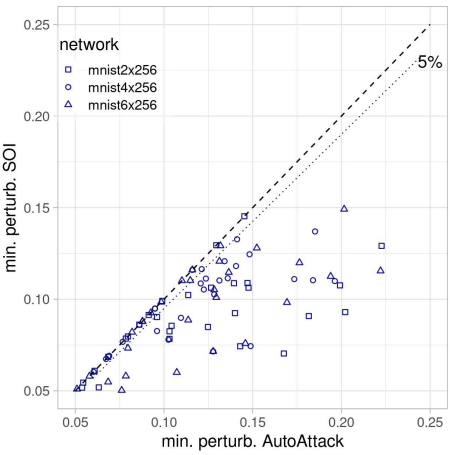

Fig. 5: Improvements over the perturbation bounds found by AutoAttack.

6.6 Incremental Solving and the Rejection Threshold T

The rejection threshold T in Algorithm 2 controls the number of rejected proposals allowed before returning `UNKNOWN`. An *incremental* solver is one that can accept a sequence of queries, accumulating and reusing relevant bounds derived by each query. To investigate the interplay of T and incrementality, we perform an experiment using the incremental simplex engine in Marabou while varying the value of T. We additionally control the branching order (by using a fixed topological order). We conduct the experiment on 180 $\texttt{MNIST}_1$ and 180 $\texttt{Taxi}_1$ benchmarks from the aforementioned distributions.

Table 3 shows the number of solved instances, as well as the average time (in seconds) and number of search states on the 95 commonly solved `UNSAT` instances. As T increases, more satisfiable benchmarks are solved.

Rejection threshold T	1	2	3	4	5	6
SAT Solv.	192	199	196	204	203	**207**
UNSAT Solv.	**91**	90	90	89	90	89
Avg. time (common)	97.75	129.0	**83.6**	108.1	137.0	187.8
Avg. states (common)	12948	12712	6122	**5586**	6404	8948

Table 3: Effect of the rejection threshold.

Increasing T can also result in improvement on unsatisfiable instances—either the average time decreases, or fewer search states are required to solve the same instance. We believe this improvement is due to the reuse of bounds derived during the execution of `DeepSoI`. This suggests that adding incrementality to the convex solver (like Gurobi) could be highly beneficial for verification applications. It also suggests that the bounds derived during the simplex execution cannot be subsumed by bound-tightening analyses such as DeepPoly.

6.7 Improving the perturbation bounds found by AutoAttack

Our proposed techniques result in significant performance gain on satisfiable instances. It is natural to ask whether the satisfiable instances solvable by the SoI-enabled analysis can also be easily handled by adversarial attack algorithms, which as mentioned in Section 2, focus solely on finding satisfying assignments. In this section, we show that this is not the case by presenting an experiment where we use our procedure in combination with AutoAttack [17], a state-of-the-art adversarial attack algorithm, to find higher-quality adversarial examples.

Concereteley, we first use AutoAttack to find an upper bound on the minimal perturbation required for a successful l_∞ attack.We then use our procedure to search for smaller perturbation bounds, repeatedly decreasing the bound by 2% until either UNSAT is proven or a timeout (30 minutes) is reached. We use the adversarial label of the last successful attack found by AutoAttack as the target label. We do this for the first 40 correctly classified test images for the three MNIST architectures, which yields 120 instances. Figure 5 shows the improvement of the perturbation bounds. Reduction of the bound is obtained for 53.3% of the instances, with an average reduction of 26.3%, a median reduction of 22%, and a maximum reduction of 58%. This suggests that our procedure can help obtain a more precise robustness estimation.

7 Conclusions and Future Work

In this paper, we introduced a procedure, `DeepSoI`, for efficiently minimizing the sum of infeasibilities in activation function constraints with respect to the convex relaxation of a neural network verification query. We showed how `DeepSoI` can be integrated into a complete verification procedure, and we introduced a novel SoI-enabled branching heuristic. Extensive experimental results suggest that our approach is a useful contribution towards scalable analysis of neural networks. Our work also opens up multiple promising future directions, including: 1) improving the scalability of `DeepSoI` by using heuristically chosen subsets of activation functions in the cost function instead of all unfixed activation functions; 2) leveraging parallelism by using GPU-friendly convex procedures or minimizing the SoI in a distributed manner; 3) devising more sophisticated initialization and proposal strategies for the Metropolis-Hastings algorithm; 4) understanding the effects of the proposed branching heuristics on different types of benchmarks; 5) investigating the use of `DeepSoI` as a stand-alone adversarial attack algorithm.

Acknowledgements We thank Gagandeep Singh for providing useful feedback and Haitham Khedr for help with running PeregriNN. This work was partially supported by DARPA (grant FA8750-18-C-0099), a Ford Alliance Project (199909), NSF (grant 1814369), and the US-Israel Binational Science Foundation (grant 2020250).

References

1. Anderson, G., Pailoor, S., Dillig, I., Chaudhuri, S.: Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In: Proc. Programming Language Design and Implementation (PLDI). p. 731–744 (2019)
2. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming pp. 1–37 (2020)
3. Andrieu, C., De Freitas, N., Doucet, A., Jordan, M.I.: An introduction to mcmc for machine learning. Machine learning **50**(1), 5–43 (2003)
4. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (vnn-comp 2021): Summary and results. arXiv preprint arXiv:2109.00498 (2021)
5. Bak, S., Tran, H.D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying relu neural networks. In: International Conference on Computer Aided Verification. pp. 66–96. Springer (2020)
6. Bénichou, M., Gauthier, J.M., Girodet, P., Hentges, G., Ribière, G., Vincent, O.: Experiments in mixed-integer linear programming. Mathematical Programming **1**(1), 76–94 (1971)
7. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
8. Boopathy, A., Weng, T.W., Chen, P.Y., Liu, S., Daniel, L.: Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 3240–3247 (2019)
9. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 3291–3299 (2020)
10. Boyd, S., Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge university press (2004)
11. Brooks, S., Gelman, A., Jones, G., Meng, X.L.: Handbook of markov chain monte carlo. CRC press (2011)
12. Bunel, R., De Palma, A., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P., Kumar, M.P.: Lagrangian decomposition for neural network verification. In: Conference on Uncertainty in Artificial Intelligence. pp. 370–379. PMLR (2020)
13. Bunel, R., Lu, J., Turkaslan, I., Kohli, P., Torr, P., Mudigonda, P.: Branch and bound for piecewise linear neural network verification. Journal of Machine Learning Research **21**(2020) (2020)
14. Bunel, R.R., Turkaslan, I., Torr, P., Kohli, P., Mudigonda, P.K.: A unified view of piecewise linear neural network verification. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper/2018/file/be53d253d6bc3258a8160556dda3e9b2-Paper.pdf
15. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 ieee symposium on security and privacy (sp). pp. 39–57. IEEE (2017)
16. Chib, S., Greenberg, E.: Understanding the metropolis-hastings algorithm. The american statistician **49**(4), 327–335 (1995)
17. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International conference on machine learning. pp. 2206–2216. PMLR (2020)

18. Dantzig, G.B., Orden, A., Wolfe, P., et al.: The generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific Journal of Mathematics **5**(2), 183–195 (1955)
19. De Palma, A., Behl, H., Bunel, R.R., Torr, P., Kumar, M.P.: Scaling the convex barrier with active sets. In: International Conference on Learning Representations (2020)
20. Dutta, S., Jha, S., Sankaranarayanan, S., Tiwari, A.: Output range analysis for deep feedforward neural networks. In: NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings (2018)
21. Dvijotham, K., Stanforth, R., Gowal, S., Mann, T.A., Kohli, P.: A dual approach to scalable verification of deep networks. In: UAI. vol. 1, p. 3 (2018)
22. Dvijotham, K.D., Stanforth, R., Gowal, S., Qin, C., De, S., Kohli, P.: Efficient neural network verification with exactness characterization. In: Uncertainty in Artificial Intelligence. pp. 497–507. PMLR (2020)
23. 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)
24. Fischetti, M., Jo, J.: Deep neural networks as 0-1 mixed integer linear programs: A feasibility study. CoRR **abs/1712.06174** (2017)
25. Fromherz, A., Leino, K., Fredrikson, M., Parno, B., Păsăreanu, C.: Fast geometric projections for local robustness certification. arXiv preprint arXiv:2002.04742 (2020)
26. 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 (2018). https://doi.org/10.1109/SP.2018.00058, https://doi.org/10.1109/SP.2018.00058
27. Gent, I.P., IRST, T.: Hybrid problems, hybrid solutions 73 j. hallam et al.(eds.) ios press, 1995 unsatisfied variables in local search. Hybrid problems, hybrid solutions **27**, 73 (1995)
28. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572 (2014)
29. Henriksen, P., Lomuscio, A.: Deepsplit: An efficient splitting method for neural network verification via indirect effect analysis. In: Zhou, Z.H. (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21. pp. 2549–2555. International Joint Conferences on Artificial Intelligence Organization (8 2021). https://doi.org/10.24963/ijcai.2021/351, https://doi.org/10.24963/ijcai.2021/351, main Track
30. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine **29**(6), 82–97 (2012)
31. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV (2017)
32. Jia, Z., Zaharia, M., Aiken, A.: Beyond data and model parallelism for deep neural networks. In: Talwalkar, A., Smith, V., Zaharia, M. (eds.) Proceedings of Machine Learning and Systems. vol. 1, pp. 1–13 (2019), https://proceedings.mlsys.org/paper/2019/file/c74d97b01eae257e44aa9d5bade97baf-Paper.pdf
33. Julian, K.D., Lee, R., Kochenderfer, M.J.: Validation of image-based neural network controllers through adaptive stress testing. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). pp. 1–7. IEEE (2020)

34. Kass, R.E., Carlin, B.P., Gelman, A., Neal, R.M.: Markov chain monte carlo in practice: a roundtable discussion. The American Statistician **52**(2), 93–100 (1998)
35. Katz, G., Barrett, C., Dill, D., Julian, K., Kochenderfer, M.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: Proc. 29th Int. Conf. on Computer Aided Verification (CAV). pp. 97–117 (2017)
36. 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 (2019)
37. Khedr, H., Ferlez, J., Shoukry, Y.: Peregrinn: Penalized-relaxation greedy neural network verifier. arXiv preprint arXiv:2006.10864 (2020)
38. King, T.: Effective algorithms for the satisfiability of quantifier-free formulas over linear real and integer arithmetic. Ph.D. thesis, Citeseer (2014)
39. King, T., Barrett, C., Dutertre, B.: Simplex with sum of infeasibilities for smt. In: 2013 Formal Methods in Computer-Aided Design. pp. 189–196. IEEE (2013)
40. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research). URL http://www. cs. toronto. edu/kriz/cifar. html **5**(4), 1 (2010)
41. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: ICLR (Workshop). OpenReview.net (2017)
42. LeCun, Y., Cortes, C.: MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/ (2010), http://yann.lecun.com/exdb/mnist/
43. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. arXiv preprint arXiv:1903.06758 (2019)
44. Lu, J., Kumar, M.P.: Neural network branching for neural network verification. arXiv preprint arXiv:1912.01329 (2019)
45. Lyu, Z., Ko, C.Y., Kong, Z., Wong, N., Lin, D., Daniel, L.: Fastened crown: Tightened neural network robustness certificates. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 5037–5044 (2020)
46. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083 (2017)
47. Masi, I., Wu, Y., Hassner, T., Natarajan, P.: Deep face recognition: A survey. In: 2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI). pp. 471–478. IEEE (2018)
48. Müller, C., Serre, F., Singh, G., Püschel, M., Vechev, M.: Scaling polyhedral neural network verification on gpus. Proceedings of Machine Learning and Systems **3** (2021)
49. Müller, M.N., Makarchuk, G., Singh, G., Püschel, M., Vechev, M.: Precise multi-neuron abstractions for neural network certification. arXiv preprint arXiv:2103.03638 (2021)
50. Paterson, C., Wu, H., Grese, J., Calinescu, R., Pasareanu, C.S., Barrett, C.: Deepcert: Verification of contextually relevant robustness for neural network image classifiers. arXiv preprint arXiv:2103.01629 (2021)
51. Raghunathan, A., Steinhardt, J., Liang, P.: Semidefinite relaxations for certifying robustness to adversarial examples. arXiv preprint arXiv:1811.01057 (2018)
52. Salman, H., Yang, G., Zhang, H., Hsieh, C.J., Zhang, P.: A convex relaxation barrier to tight robustness verification of neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper/2019/file/246a3c5544feb054f3ea718f61adfa16-Paper.pdf

53. Schkufza, E., Sharma, R., Aiken, A.: Stochastic superoptimization. ACM SIGARCH Computer Architecture News **41**(1), 305–316 (2013)
54. Selman, B., Kautz, H.A., Cohen, B.: Noise strategies for improving local search. In: AAAI. vol. 94, pp. 337–343 (1994)
55. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484 (2016)
56. Singh, G., Ganvir, R., Püschel, M., Vechev, M.: Beyond the single neuron convex barrier for neural network certification. Advances in Neural Information Processing Systems **32**, 15098–15109 (2019)
57. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. Advances in Neural Information Processing Systems **31**, 10802–10813 (2018)
58. 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)
59. Singh, G., Gehr, T., Püschel, M., Vechev, M.: Boosting robustness certification of neural networks. In: International Conference on Learning Representations (2019)
60. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The german traffic sign recognition benchmark: a multi-class classification competition. In: The 2011 international joint conference on neural networks. pp. 1453–1460. IEEE (2011)
61. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199 (2013)
62. Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., PATEL, K.K., Vielma, J.P.: The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 21675–21686. Curran Associates, Inc. (2020), https://proceedings.neurips.cc/paper/2020/file/f6c2a0c4b566bc99d596e58638e342b0-Paper.pdf
63. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=HyGIdiRqtm
64. Tran, H.D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using imagestars. In: International Conference on Computer Aided Verification. pp. 18–42. Springer (2020)
65. Vincent, J.A., Schwager, M.: Reachable polyhedral marching (rpm): A safety verification algorithm for robotic systems with deep neural network components. arXiv preprint arXiv:2011.11609 (2020)
66. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 6369–6379 (2018), http://papers.nips.cc/paper/7873-efficient-formal-safety-analysis-of-neural-networks
67. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. pp. 1599–1614 (2018), https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi

68. Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.J., Kolter, J.Z.: Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. arXiv preprint arXiv:2103.06624 (2021)
69. Weng, L., Zhang, H., Chen, H., Song, Z., Hsieh, C.J., Daniel, L., Boning, D., Dhillon, I.: Towards fast computation of certified robustness for relu networks. In: International Conference on Machine Learning. pp. 5276–5285. PMLR (2018)
70. Wong, E., Kolter, Z.: Provable defenses against adversarial examples via the convex outer adversarial polytope. In: International Conference on Machine Learning. pp. 5286–5295. PMLR (2018)
71. Wu, H., Ozdemir, A., Zeljić, A., Julian, K., Irfan, A., Gopinath, D., Fouladi, S., Katz, G., Pasareanu, C., Barrett, C.: Parallelization techniques for verifying neural networks. In: 2020 Formal Methods in Computer Aided Design (FMCAD). pp. 128–137. IEEE (2020)
72. Wu, H., Zeljić, A., Katz, G., Barrett, C.: Artifact for Paper Efficient Neural Network Analysis with Sum-of-Infeasibilities (Feb 2022), https://doi.org/10.5281/zenodo.6109456
73. 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)
74. Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.W., Huang, M., Kailkhura, B., Lin, X., Hsieh, C.J.: Automatic perturbation analysis for scalable certified robustness and beyond. Advances in Neural Information Processing Systems **33** (2020)
75. Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., Hsieh, C.J.: Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. arXiv preprint arXiv:2011.13824 (2020)
76. Zhang, H., Weng, T.W., Chen, P.Y., Hsieh, C.J., Daniel, L.: Efficient neural network robustness certification with general activation functions. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018), https://proceedings.neurips.cc/paper/2018/file/d04863f100d59b3eb688a11f95b0ae60-Paper.pdf