A Branch and Bound Framework for Stronger Adversarial Attacks of ReLU Networks

Huan Zhang [*1] **Shiqi Wang** [*2] **Kaidi Xu** [3] **Yihan Wang** [4] **Suman Jana** [2] **Cho-Jui Hsieh** [4] **Zico Kolter** [1]

Abstract

Strong adversarial attacks are important for evaluating the true robustness of deep neural networks. Most existing attacks search in the input space, e.g., using gradient descent, and may miss adversarial examples due to non-convexity. In this work, we *systematically* search adversarial examples in the *activation space* of ReLU networks to tackle hard instances where none of the existing adversarial attacks succeed. Unfortunately, searching the activation space typically relies on generic mixed integer programming (MIP) solvers and is limited to small networks and easy problem instances. To improve scalability and practicability, we use branch and bound (BaB) with specialized GPU-based bound propagation methods, and propose a *top-down beam-search* approach to quickly identify the subspace that may contain adversarial examples. Moreover, we build an *adversarial candidates pool* using cheap attacks to further assist the search in activation space via *diving* techniques and a *bottom-up* large neighborhood search. Our adversarial attack framework, BaB-Attack, opens up a new opportunity for designing novel adversarial attacks not limited to searching the input space, and enables us to borrow techniques from integer programming theory and neural network verification. In experiments, we can successfully generate adversarial examples when existing attacks on input space fail. Compared to off-the-shelf MIP solver based attacks that requires significant computations, we outperform in both success rates and efficiency.

[*]Equal contribution [1]Carnegie Mellon University [2]Columbia University [3]Drexel University [4]UCLA. Correspondence to: Huan Zhang <huan@huan-zhang.com>, Shiqi Wang <sw3215@columbia.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

1. Introduction

Adversarial attacks aim to find adversarial examples (Szegedy et al., 2013), which are close to benign inputs in certain distance metrics yet trigger wrong behavior of neural networks (Carlini & Wagner, 2017; Madry et al., 2018; Athalye et al., 2018; Croce & Hein, 2020b). Adversarial attacks are important tools to gauge the empirical robustness of neural networks (NNs). Finding an adversarial example can be generally formulated as a constrained optimization:

$$x_{\text{adv}} = \arg\min_{x \in \mathcal{C}} f(x) \tag{1}$$

where $\mathcal{C}$ is often an ℓ_∞ or ℓ_2 norm ball around the original input x_0, and $f(x)$ is an attack success criterion involving a neural network (such as the margin between the groundtruth class and another class): $f(x) < 0$ indicates a successful attack. A straightforward way of solving Eq. (1) is via first-order constrained optimization methods, such as projected gradient descent (PGD) (Madry et al., 2018) and its variants (Croce & Hein, 2020b; Tashiro et al., 2020; Croce & Hein, 2020a; Xie et al., 2019; Dong et al., 2018; Zheng et al., 2019). Additionally, some gradient-free attacks were proposed (Brendel et al., 2018; Cheng et al., 2018; Alzantot et al., 2019; Andriushchenko et al., 2020), mostly based on certain heuristic search on the input space x.

Are adversarial attacks a solved problem? Although it is often easy to attack a model, the attack problem is far from being completely solved. To precisely characterize the robustness of a model, we must prove one of the following:

- There exists a $x^* \in \mathcal{C}$ such that $f(x^*) < 0$ (attack)
- $f(x) \geq 0$ for all $x \in \mathcal{C}$ (verification)

Unfortunately, *even for small models*, it can be hard to prove either one of the two cases, leading to "unknown" robustness for some examples and a large gap between attack accuracy and verified accuracy. The NN verification community has been working valiantly to close this gap, and developed several strong verification tools recently (Bak et al., 2021), allowing a precise characterization of NNs in mission-critical tasks such as aircraft control (Katz et al., 2017; 2019) and cyber-physical systems (Tran et al., 2020). However, even for small models that can be well handled by NN verifiers, there still exists *hard instances* which cannot

be attacked using PGD, nor be verified by the best verifier. It is still unclear if this gap can be closed by a stronger attack.

Limitations of existing attacks. As $f(x)$ usually consists of a highly non-convex neural network, solving Eq. (1) to its global minimum is challenging. This leads to failures in adversarial attacks: an adversarial example may exist but no attacks can find it, giving a false sense of security (Athalye et al., 2018). Especially, gradient based attacks can be easily trapped into a local minimum or misguided by masked gradients (Papernot et al., 2016; Tramèr et al., 2017). Even if we give the attacker *a practically infeasible amount of time* (e.g., run a *very large number of PGD steps*, or allow a large number of samples on input space), it is still hard to guarantee finding an adversarial example if it exists, since it is *extremely difficult to systematically search* the high dimensional and continuous input space. Models concluded robust under existing attacks might still have security vulnerability in practice, leading to an urgent need for stronger attacks that can possibly approach ground-truth robustness.

The mixed-integer approach. This paper seeks stronger adversarial attacks from a different angle: instead of searching for adversarial examples in the input space, we look for adversarial examples in the *activation space*. The main intuition is that neural networks with piece-wise linear activation functions (e.g., ReLU) can be seen as a piece-wise linear function and each piece is uniquely defined by a specific setting of activation function status. For ReLU networks, each neuron can either be active (its input is positive) or inactive (its input is negative so the output is 0), which can be encoded by *discrete* 0-1 variables. The adversarial attacks therefore can be formulated as a mixed integer programming (MIP) formulation (Ehlers, 2017; Tjeng et al., 2019) for solving Eq. (1) rather than directly minimizing Eq. (1) using gradients on input x.

Benefits of the MIP formulation. The MIP formulation with the 0-1 encoding of ReLU neurons allows us to *systematically search* all the linear pieces in the input space, theoretically guarantee to *enumerate the entire input space* and obtain the global minimum of Eq. (1) given sufficient time. Assuming an attacker with massive computational resources (e.g., nation-state actors), MIP formalizes the strongest attack possible against a network. Therefore, MIP-based attacks can often find adversarial examples that are missed by existing attacks and identify true weaknesses of a model, which helps to close the gap between the upper and lower bounds of robust accuracy (attack accuracy vs. verified accuracy). Existing NN verifiers conduct a systematic search via branch and bound (Bunel et al., 2020b; Wang et al., 2021) in activation space to tighten the lower bound, while we aim to tighten the upper bound by a systematic search of adversarial examples. Closing this gap is difficult even on small models (Dathathri et al., 2020).

Generic MIP solvers are inefficient for adversarial attacks. Despite its strengths, a MIP-based attack are often a few orders of magnitudes slower than existing attacks due to the high cost of running an off-the-shelf solver (Tjeng et al., 2019). There are three root causes for its inefficiency. First, an off-the-shelf solver is *not aware of the underlying structure of the problem* (i.e., it is a neural network), and has to apply generic solving techniques (e.g., Simplex algorithm with relaxations) which can be expensive or ineffective. Second, it *cannot utilize solutions obtained cheaply from gradient based attacks* to accelerate its search. Third, generic MIP solvers are mostly restricted to CPUs and *can hardly utilize GPU*, which is crucial for efficiency.

Contributions of this paper. We address the above weaknesses in MIP solvers for adversarial attacks, by developing a GPU-accelerated branch and bound procedure to *systematically search* adversarial examples in activation space via branch and bound (BaB). We focus on *solving hard instances* where none of existing adversarial attacks searching on input space can succeed and no verifiers can prove their robustness. Our contributions include:

- We design a novel BaB based attack framework that can leverage the GPU-accelerated bound propagation based methods (Wang et al., 2021; Wong & Kolter, 2018), which were originally developed for neural network verification. Our method can quickly *examine thousands of suspicious regions in activation space in parallel* and rule out the regions with no adversarial examples, which is difficult in off-the-shelf MIP solvers with a generic solving procedure.
- We propose a *top-down beam-search* to explore the activation space. Unlike the best-first search used in NN verifiers, we can quickly go deep in the search tree and identify the most promising regions with adversarial examples.
- We propose to collect adversarial candidates generated by cheap attacks to guide the search in activation space. First, we conduct a *bottom-up* search on candidates close to decision boundary by applying large neighborhood search (LNS). Second, when conducting the top-down search, we adopt *diving* by fixing integer variables according to adversarial examples in the pool to reduce the search space.
- Our new attack framework, BaB-Attack, is designed to tackle hard instances where existing strong adversarial attacks (such as AutoAttack) cannot succeed. Despite being more expensive than attacks on the input space, BaB-Attack is about an order of magnitude faster than using an MIP solver in our benchmarks, and can be easily integrated into a branch-and-bound based NN verifier to further close the gap between verified and attack accuracy.

2. Background

Notations. We define a L layer feed-forward ReLU network as $f : \mathbb{R}^{n_0} \to \mathbb{R}$ and $f(x) := z^{(L)}(x)$, where

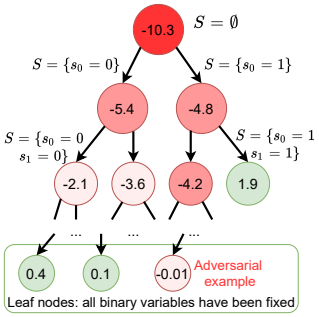


Figure 1: A branch and bound search tree. Each node represents a subdomain determined by $\mathcal{S}$, and the numbers are $\mathsf{LB}(\mathcal{S})$ (lower bound of domain $\mathcal{S}$). No adversarial example exist in the subdomain if $\mathsf{LB}(\mathcal{S}) > 0$ (green). A concrete adversarial example is a leaf node where $\mathsf{LB}(\mathcal{S}_{\text{leaf}}) < 0$.

$z^{(i)}(x) = \mathbf{W}^{(i)}\hat{z}^{(i-1)}(x) + \mathbf{b}^{(i)}$ with i-th layer weight matrix $\mathbf{W}^{(i)}$ and bias $\mathbf{b}^{(i)}$, $\hat{z}^{(i)}(x) = \text{ReLU}(z^{(i)}(x))$, and input $\hat{z}^{(0)}(x) = x$. Layer i has dimension n_i, and N is the total number of neurons. We denote the j-th neuron in layer i as $z_j^{(i)}$. For a simpler presentation, we assume $f(x)$ is a binary classifier and benign input x_0 has $f(x_0) > 0$. An attacker seeks to minimize $f(x)$ within a ℓ_∞ norm perturbation set $\mathcal{C}$ to make $f(x) < 0$. We can attack a multi-class classifier by considering each pair of target and ground-truth label individually where f is defined as the margin between them, similarly to Gowal et al. (2019b). We use $[N]$ to represent the set $\{1, \cdots, N\}$. We omit $\cdot(x)$ when the context is clear in the following paper.

The MIP formulation for adversarial attack. Tjeng et al. (2019) showed that the adversarial attack and verification of ReLU networks can be generally formulated into a mixed integer programming (MIP) problem, solved by existing MIP solvers (refer to as "MIP attack" in our paper). This formulation has binary variables $s_j^{(i)}$ for each ReLU:

$$\begin{aligned} &\min f \quad \text{s.t. } i \in [L],\ j \in [n_i] \\ &z^{(i)} = \mathbf{W}^{(i)}\hat{z}^{(i-1)} + \mathbf{b}^{(i)};\ \ f = z^{(L)};\ \ \hat{z}^{(0)} = x \in \mathcal{C}; \\ &\hat{z}_j^{(i)} \geq z_j^{(i)};\ \ \hat{z}_j^{(i)} \leq u_j^{(i)} s_j^{(i)};\ \ \hat{z}_j^{(i)} \leq z_j^{(i)} - l_j^{(i)}(1 - s_j^{(i)}); \\ &\hat{z}_j^{(i)} \geq 0;\ \ z_j^{(i)} \in [l_j^{(i)}, u_j^{(i)}];\ \ s_j^{(i)} \in \{0, 1\}; \end{aligned} \tag{2}$$

where $s_j^{(i)}$ indicates the two status of ReLU: (1) *inactive*: when $s_j^{(i)} = 0$, constraints on $\hat{z}_j^{(i)}$ simplifies to $\hat{z}_j^{(i)} = 0$; or (2) *active*: when $s_j^{(i)} = 1$ we have $\hat{z}_j^{(i)} = z_j^{(i)}$. Here $l_j^{(i)}$, $u_j^{(i)}$ are pre-computed intermediate lower and upper bounds on pre-activation $z_j^{(i)}$ such that $l_j^{(i)} \leq z_j^{(i)}(x) \leq u_j^{(i)}$ for any $x \in \mathcal{C}$. The complexity of this problem can increase exponentially with the number of ReLU neurons, so it can take hours to run even on a small network, unless the network is trained with a strong regularization such as a certified dense (Wong & Kolter, 2018; Xiao et al., 2019).

Searching in Activation Space via Branch and Bound. Given Eq. (2), we can view a neural network in the *activation space* $\mathcal{A} = \{0, 1\}^N$ where N is the total number of neurons, and each dimension corresponds to the setting of a $s_j^{(i)} \in \{0, 1\}$ variable. To determine $s_j^{(i)}$ corresponding to a *known* adversarial example x_{adv}, we can propagate x_{adv} through the network and check the sign of each neuron $z_j^{(i)}$, so $s_j^{(i)} = \mathbb{1}(z_j^{(i)} \geq 0)$. This uniquely locates the linear piece of $f(x)$ where x_{adv} lies, because Eq. (2) becomes a set of linear inequalities when all $s_j^{(i)}$ are fixed. Intuitively, we can search adversarial examples by fixing all $s_j^{(i)}$ to one of the 2^N possible combinations in $\mathcal{A}$, and then solve Eq. (2) exactly using linear programming; an adversarial example is found when the solution is negative. To avoid clutter, we flatten the ordering of $s_j^{(i)}$ for $i \in [L], j \in [n_i]$ and use a single subscript $s_1, \cdots, s_N$ to denote all binary variables.

To effectively and systematically search in the activation space, instead of fixing all s_i ($i \in [N]$), we can first fix a subset of them and bound the objective $f(x)$ to guide the search, leading to the *branch and bound* (BaB) method. In BaB, we solve Eq. (2) by creating subproblems constraining some binary variables, for example, $s_1 = 0$ or $s_1 = 1$ (since we can branch on the neurons in any fixed order, without loss of generality, we show branching chronologically). We define a set $\mathcal{S}$ containing all the branching constraints (e.g., $\mathcal{S} = \{s_1 = 0, s_2 = 1\}$), which corresponds to a subdomain of the original problem Eq. (2). BaB requires the *lower bound primitive* LB *on* $\mathcal{S}$, which relaxes the remaining binary variables to obtain a lower bound of Eq. (2):

$$\mathsf{LB}(\mathcal{S}) \leq \min f(x) \ \text{ s.t. } s \in \mathcal{S} \ \& \text{ other Eq. (2) constraints}$$

Here $s \in \mathcal{S}$ means setting binary variables $s_j^{(i)}$ according to the constraints in $\mathcal{S}$. Typically, more constraints in $\mathcal{S}$ lead to tighter bounds. $\mathsf{LB}(\mathcal{S}) > 0$ indicates that *no adversarial example* exist within this subdomain, otherwise adversarial examples *may exist* in this subdomain.

We illustrate a BaB search tree in Fig. 1. Initially, $\mathcal{S}_{\text{root}} = \emptyset$, where $x \in C$ without any extra constraints in activation space and a lower bound of Eq. (2) is obtained. When $\mathsf{LB}(\emptyset) < 0$, an adversarial example may exist, and we *branch* $\mathcal{S}_{\text{root}}$ into two subdomains:

$$\mathcal{S}_{1-} = \mathcal{S}_{\text{root}} \cup \{s_1 = 0\};\ \mathcal{S}_{1+} = \mathcal{S}_{\text{root}} \cup \{s_1 = 1\}$$

Then we *bound* each subdomain. Since more constraints are added, $\mathsf{LB}(\mathcal{S}_{1-})$ and $\mathsf{LB}(\mathcal{S}_{1+})$ are usually improved. The branching procedure continues recursively, and if any $\mathsf{LB}(\mathcal{S}) > 0$, no further branching is needed since no adversarial examples exist in that subdomain. Each branching increases the cardinality of $\mathcal{S}$ by 1. Eventually we reach leaf nodes with $|\mathcal{S}_{\text{leaf}}| = N$, each leaf locating a linear piece of $f(x)$. In that case, $\mathsf{LB}(\mathcal{S}_{\text{leaf}})$ is an exact solution since no binary variables are left. If $\mathsf{LB}(\mathcal{S}_{\text{leaf}}) < 0$, a concrete adversarial example is the minimizer x^* of Eq. (2) with $s \in \mathcal{S}_{\text{leaf}}$.

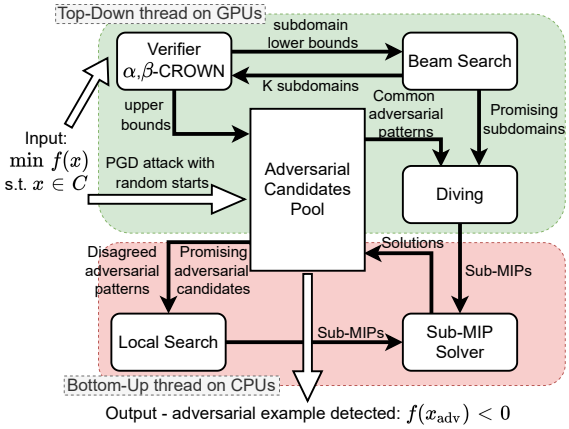


Figure 2: Overview of BaB attack.

Since N can be quite large and adversarial examples lie in the leaf level, we must guide the search to reach there quickly. Although BaB is used in existing neural network verifiers (Bunel et al., 2018; De Palma et al., 2021a), they do not aim to reach the leaf level and typically branch the node with the worst bound first, generally leading to a wide but shallow search tree and unsuitable for detecting adversarial examples. We will discuss our search algorithm in Sec. 3.

Bounding in Branch and Bound of Neural Networks. The $\mathsf{LB}(\mathcal{S})$ primitive is crucial in the BaB process: it needs to provide a tight lower bound efficiently. A simple way to lower bound the objective of Eq. (2) is via relaxation of integer variables and linear programming (LP) (Bunel et al., 2018; Lu & Kumar, 2020); but an LP solver is needed which restricts its efficiency. Recently, a popular choice in NN verifiers is the specialized bound propagation methods (Zhang et al., 2018; Wong & Kolter, 2018) which exploit the structure of the optimization problem (which a generic LP/MIP solver cannot) and give $\mathsf{LB}(\mathcal{S})$ efficiently on GPUs without an LP solver. Essentially, they relax each ReLU neuron into convex domains (Salman et al., 2019) and propagate them layer by layer through the network while maintaining sound bounds. We leverage the state-of-the-art bound propagation based verifier, α,β-CROWN (Zhang et al., 2018; Xu et al., 2021; Wang et al., 2021), to produce $\mathsf{LB}(\mathcal{S})$. Importantly, we will show how we use $\mathsf{LB}(\mathcal{S})$ to guide attacks, while existing works mostly use them for verification.

3. Method

Overview of BaB attack. To systematically search adversarial examples in activation space, we must explore the search tree and enumerate as more leaf nodes as possible. Although the worst case search time complexity is exponential in the numbers of ReLU neurons (visiting every leaf node of the tree), practically, if a right search procedure is chosen, only a small fraction of nodes need to be visited to find an adversarial example. In this paper, we propose BaB attack, specializing the BaB searching strategy over the activation space for the purpose of adversarial attacks. The search is well guided by (1) a top-down beam-search thread accelerated on GPUs which quickly goes deep into the search tree, and (2) a bottom-up search thread on CPUs for large neighborhood search. The top-down and bottom-up searches run in parallel threads and they both benefit from the *adversarial candidates pool* $\mathcal{P}$, which contains examples $\mathcal{P} = \{x_{c_1}, \cdots, x_{c_M}\}$ where M is the pool capacity. $f(x_{c_i})$ is still positive but small; the pool keeps the M best (ranked by $f(x_{c_i})$; smaller is better) examples it receives. The activation space representations of these candidates are used as extra information to guide the search. We detail each part of BaB attack (Fig. 2) in next sub-sections.

3.1. Top-down Beam Search Guided by Verifiers

Challenges in searching adversarial examples. The activation space can be quite large (e.g., with thousands or more dimensions), and adversarial examples are at the leaf level of the search tree. Searching directly from the root node and traversing the search tree in an exhaustive manner (such as BFS or DFS) can be quite insufficient. To locate adversarial examples faster, we propose to use beam search guided by lower bounds $\mathsf{LB}(\mathcal{S})$ from NN verifiers.

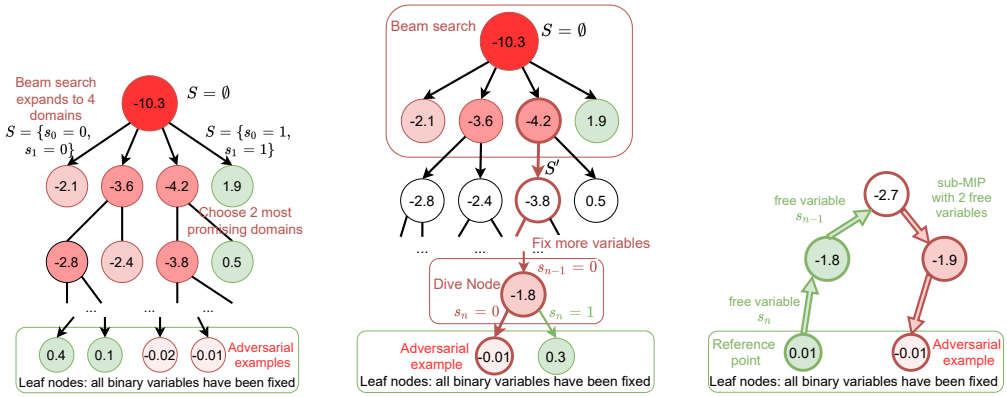
Beam search in activation space. Our key insight is to accelerate the search by prioritizing suspicious subdomains with small $\mathsf{LB}(\mathcal{S})$, to have no adversarial examples. At the root node in the search tree, our beam search procedure expands the tree by D levels, yielding 2^D subdomains:

$$\begin{aligned}
\mathcal{S}_1 &= \mathcal{S}_{\text{root}} \cup \{s_1 = 0, s_1 = 0 \cdots, s_D = 0\},\\
\mathcal{S}_2 &= \mathcal{S}_{\text{root}} \cup \{s_1 = 1, s_1 = 0 \cdots, s_D = 0\},\\
\mathcal{S}_3 &= \mathcal{S}_{\text{root}} \cup \{s_1 = 0, s_1 = 1 \cdots, s_D = 0\},\\
&\cdots,\\
\mathcal{S}_{2^D} &= \mathcal{S}_{\text{root}} \cup \{s_1 = 1, s_1 = 1 \cdots, s_D = 1\}
\end{aligned}$$

For each subdomain, we obtain its lower bound efficient via the primitive $\mathsf{LB}(\mathcal{S})$ from a bound propagation based NN verifier. Then, we sample K subdomains without replacement out of the 2^D domains, with the probability p_i associated with each $\mathcal{S}_i$ in Eq. (3), where $i \in \{1, \cdots, 2^D\}$ and T is the temperature set to 0.1 by default.

$$p_i = \frac{\exp\left(-T \cdot \mathsf{LB}(\mathcal{S}_i)\right) \cdot \mathbb{1}(\mathsf{LB}(\mathcal{S}_i) < 0)}{\sum_{i=1}^{2^D} \exp\left(-T \cdot \mathsf{LB}(\mathcal{S}_i)\right) \cdot \mathbb{1}(\mathsf{LB}(\mathcal{S}_i) < 0)} \tag{3}$$

A subdomain with more negative lower bound has a higher probability to be selected, since the large negative bounds may indicate a higher chance of the existence of adversarial examples. Subdomains with positive bounds will never be selected, since they are guaranteed to not contain adversarial examples. The picked out subdomains $\mathcal{S}'_1, \cdots, \mathcal{S}'_K$ become the parent nodes for the next iteration of beam search. In the next iterations, we explore $K \cdot 2^D$ subdomains and increase the depth by D per iteration. Since all the subdomain lower bounds are computed in a large batch on GPUs (commonly only a few seconds), our search procedure quickly



(a) Beam Search. (b) Diving. (c) Local Search.

Figure 3: (a) Beam search: we select K subdomains probabilistically according to $\mathsf{LB}(\mathcal{S})$ (lower bound of domain $\mathcal{S}$), and expand the search tree by D levels using bound propagation on GPU. (b) Diving with common adversarial patterns: We dive the search tree with additional constraints constructed by common adversarial patterns to greatly reduce top-down sub-MIP search space. (c) Large Neighborhood Search (LNS): In bottom-up search, we *free* some fixed integer variables at a leaf node (e.g., an adversarial candidate which serves as a "reference point") searching its neighborhood in activation space.

explore deep in the search tree. Our *specialized top-down beam search* with lower bounds computed efficiently by bound propagation on GPUs brings us great advantages over existing MIP solvers, which conduct BaB on CPUs using a generic procedure such as Simplex algorithm or barrier method. In practice, we can visit several orders of magnitudes more subdomains.

Sub-MIP on most promising subdomains. Before the beam search reaches the leaf level, we start searching adversarial examples in the most promising subdomains (e.g., in some of the selected domains $\mathcal{S}'_1, \cdots, \mathcal{S}'_K$), by constructing a sub-MIP problem of Eq. (2): in the k-th sub-MIP, we fix its binary variables $s_j^{(i)}$ based on constraints in $\mathcal{S}'_k$. In this step, although a generic MIP solver is used, it is instructed to search in subdomains guided by beam search to be likely to contain adversarial examples. With a large number of $s_j^{(i)}$ fixed during beam search, the MIP solver only needs to work on a much smaller problem and can be much more effective than solving Eq. (2) directly.

Completeness. Beam search, if implemented with backtracking, can achieve completeness (Zhou & Hansen, 2005): given sufficient time, it will systematically visit all leaf nodes, and guarantee to either locate an adversarial example, or prove the network safe. However, due to the large number of neurons, achieving completeness often requires an infeasibly large amount of time and space. We thus focus on searching adversarial examples as fast as possible rather than exhaustively visiting every node, although theoretically our procedure can be made complete.

3.2. Diving in BaB with Common Adversarial Patterns

What is diving? "Diving" refers to diving deep in the BaB search tree by heuristically fixing some integer variables without exploring all possible branches. It is a common strategy in generic MIP solvers, able to quickly uncover feasible solutions of a MIP problem (Berthold, 2006; Nair et al., 2020). In our case, we want to fix binary variables $s_j^{(i)}$ in Eq. (2), and a feasible solution x with $f(x) < 0$ is an adversarial example (Figure 3b). A generic MIP solver uses diving to hopefully find high quality feasible solutions quickly, however it *cannot use the information provided by cheaply generated adversarial examples* like PGD attack to guide this heuristic. In this work, we propose a specialized diving scheme in the activation space based on the statistics in the adversarial candidates pool, and construct sub-MIPs with additional diving constraints to reduce the search space.

Diving with common adversarial patterns. Given the candidates pool $\mathcal{P} = \{x_{c_1}, \cdots, x_{c_M}\}$, we first extract the corresponding binary variables s_i for each example, by propagating them through the network (see Section 2). The binary variable corresponds to the i-th neuron of the m-th adversarial example is denoted as $s_{i,m}$ (0 or 1). A variable s_i is called a *common activation* when the function $c(i)$ is greater than a threshold C:

$$c(i) := \frac{\left|\sum_{m=1}^{M} s_{i,m} - M/2\right|}{M} + 0.5 \geq C, \quad C \in [0.5, 1.0]$$

For example, when $C = 0.9$, a common activation requires that at least 90% examples in the pool share the same value of s_i (0 or 1). All the common activations and their common values are called *common adversarial patterns*, indicating that an adversarial examples will be likely to contain these settings of s_i. We then construct a set of constraints $\mathcal{S}_{\text{common}}$, setting common activations to their common values:

$$\mathcal{S}_{\text{common}} = \{s_i = \mathbb{1}(\sum_{m=1}^{M} s_{i,m} \geq \frac{M}{2}), c(i) \geq C, i \in [N]\}$$

Then, when constructing the top-down sub-MIPs in Sec. 3.1, we provide additional constraints $\mathcal{S}_{\text{common}}$ as diving con-

straints. They further reduces the search space for the MIP solver (the MIP solver solves Eq. (2) with both beam search constraints $\mathcal{S}_1'$ and diving constraints $\mathcal{S}_{\text{common}}$), making it easier to find an solution. The threshold C controls the aggressiveness of diving; too much diving leads to a too small search space so good adversarial examples cannot be found.

How to fill the adversarial candidates pool? To obtain useful common adversarial pattern, we maintain an *adversarial candidates pool* with up to M most promising adversarial candidates ($f(x)$ is close to 0 but the label is not yet flipped). The pool is initialized with perturbed samples bounded within $\mathcal{C}$ found with output diversified sampling PGD (Tashiro et al., 2020), which creates a diverse set of adversarial candidates. During BaB attack, new adversarial candidates come from three sources: (1) some NN verifier provides upper bounds for each subdomain during beam search; in α,β-CROWN these upper bounds are obtained via conducting PGD on its dual solutions (Wang et al., 2021) produced by the solver, and they are added to the pool; (2) the solutions returned by the top-down sub-MIP solved with beam search and diving constraints, and (3) the solutions returned by the bottom-up sub-MIP with large neighborhood search (which will be discussed in the next subsection). When new adversarial candidates are inserted, they are compared to existing ones in the pool, and we keep the best M adversarial candidates with distinct activation patterns.

3.3. Bottom-Up Large neighborhood Search (LNS)

What is bottom-up search? The bottom-up search procedure starts at the leaf nodes of BaB search tree (Figure 3c): we start from a known adversarial candidate x_c that is close to decision boundry ($f(x_c) > 0$ but very small), and want to further reduce $f(x_c)$ by searching around x_c. A naive way is to conduct PGD attack *in the input space* with x_c as the starting point, but we found it not helpful because the adversarial candidates in the pool have already been optimized using PGD or stronger attacks. Thus, we propose to use a large neighborhood search *in activation space*.

Bottom-up search via large neighborhood search. Large neighborhood Search (LNS) (Walser, 2003; Schrijver, 2003) is a generic local search heuristic: one defines a neighborhood around a reference point (a feasible solution) and finds the optimum objective in this neighborhood, typically by constructing a sub-MIP problem with neighborhood constraints. In the setting of integer programming, the neighborhood can be defined by *freeing* certain fixed integer variables, allowing them to be optimized while fixing other integer variables. However, traditional local search algorithms in MIP solvers has little guidance regarding promising subdomains in common adversarial examples and can be ineffective due to the high dimensional search space in the adversarial attack problem.

In BaB attack, we extend the general idea of LNS to a specialized local search for adversarial attacks by selecting the most promising adversarial candidates in the pool as the reference point and then use the statistics from the pool to free certain binary variables s_i. Specifically, among all the binary variables corresponding to ReLU neurons on the selected candidate, we define the *disagreed adversarial patterns* from ReLUs where adversarial candidates in the pool that *disagree the most*. Formally, similar to the setting in Sec. 3.2, a variable s_i is called a *disagreed activation* if:

$$\bar{c}(i) := 1 - c(i) \geq \bar{C}, \quad \bar{C} \in [0.0, 0.5]$$

For example, when $\bar{C} = 0.3$, s_i is a disagreed activation when there are at least 30% examples in the pool do *not* share the same value of s_i. Since these variables are quite different across existing adversarial examples, we remove their corresponding binary variables to allow the MIP solver to search for a better setting of them. The aggressiveness of freeing variables in LNS is determined by $\bar{C}$. The set of disagreed adversarial patterns is a set of binary variables:

$$S_{\text{disagreed}} = \{s_i \mid \bar{c}(i) \geq \bar{C}, i \in [N]\}$$

Formally, to search around an adversarial candidate x_c, we first propagate x_c through the network and obtain activation values z_i, extract the corresponding binary variables s_i for x_c, and remove the constraints that are in $S_{\text{disagreed}}$ to construct the set of constraints for bottom-up search:

$$\mathcal{S}_{\text{bottom-up}} = \{s_i = \mathbb{1}(z_i \geq 0), s_i \notin S_{\text{disagreed}},\ i \in [N]\}$$

We then construct a sub-MIP using Eq. (2) with the additional constraints $\mathcal{S}_{\text{bottom-up}}$ and solve it using a MIP solver. The optimal solution to each sub-MIP (if still not an adversarial example) will be added back to adversarial candidates pool again waiting for another round of local search.

4. Experiments

Setup. We evaluate on all 10,000 test examples of MNIST (LeCun, 1998) and CIFAR10 (Krizhevsky et al., 2009) dataset, and select 8 models which are mostly benchmarking models used in previous works or competitions (details in Appendix A.1). For each model, we first run three commonly used strong adversarial attacks: a multi-targeted PGD (MT-PGD) attack (Gowal et al., 2019a) with *1000 Adam steps and 500 random restarts*; a multi-targeted PGD attack with Output Diversified Sampling (ODS-PGD) (Tashiro et al., 2020) with 1000 Adam steps and 500 random restarts, and AutoAttack (Croce & Hein, 2020b) which is an *ensemble* of parameter-free attacks. The remaining robust images are then tested with α,β-CROWN verifier (Wang et al., 2021; Xu et al., 2021; Zhang et al., 2018) to see if they can be verified robust so no further attack is needed. Any images that failed with the verifier are then evaluated in an MIP formulation (Tjeng et al., 2019) solved using Gurobi. The MIP solver is used as the last resort because it is usually much more expensive

Table 1: Comparison between MIP attack (Tjeng et al., 2019) (using Gurobi) and our BaB Attack. Both attacks focus on **hard instances** where a combination of MT-PGD, ODS-PGD with **1000 step and 500 restarts** and AutoAttack cannot succeed and their robustness also cannot be verified. Average time computed on non-timeout examples only. For a fair comparison, the MIP attack baseline and our BaB attack use the same timeout. For a comparison to more input space attacks, see Table 2.

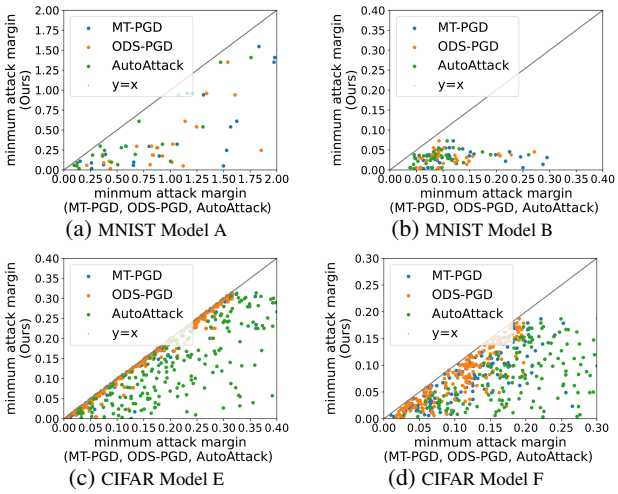
Dataset	Model	eps	Clean Acc.	Total verified	Total attacked	Hard instances	MIP attack	Avg. time(s)	BaB attack	Avg. time(s)
MNIST	A	0.3	97.94%	8171	1576	47	**20**	65.82	**20**	36.18
	B	0.3	96.33%	6746	2271	616	20	2210.06	**32**	166.60
CIFAR	C	2/255	65.63%	4748	1664	151	**3**	989.06	**3**	55.23
	D	2/255	68.73%	4965	1643	265	**1**	867.25	**1**	96.17
	E	2/255	74.18%	4435	2069	914	3	1852.14	**6**	107.72
	F	2/255	63.14%	900	4207	1207	9	937.77	**26**	65.92
	G	2/255	60.86%	5015	1071	0	Used in (Tjeng et al., 2019), however the gap between lower and upper bounds is closed with recently proposed strong verifiers, thus not suitable for evaluation			
	H	8/255	27.07%	2243	463	1				

Table 2: Number of successfully (# succ.) attacked *hard instances* (from Table 1) under more attacks. Here we run FAB and Square attacks with much more steps than default to increase their power. All examples here *cannot* be attacked by AutoAttack and MT-PGD, ODS-PGD with **1000 step and 500 restarts**. Although we cannot practically include all existing attack algorithms here, the results show the *limitation of representative attacks that search input space*, so activation space search based attack like BaB attack is useful.

Dataset	Model	# Total	FAB		Square		DIFGSM		MIFGSM		Distributional		AutoAttack+		BaB attack (ours)	
			# succ.	Time(s)	# succ.	Time(s)	# succ.	Time(s)	# succ.	Time(s)	# succ.	Time(s)	# succ.	Time(s)	# succ.	Time(s)
MNIST	A	47	0	3.25	4	7.96	0	0.04	0	0.02	0	0.03	2	0.31	**20**	36.18
	B	616	0	3.40	3	11.75	0	0.04	0	0.02	1	0.01	0	0.28	**32**	166.60
CIFAR	C	151	0	3.32	0	11.61	0	0.39	0	0.02	0	0.01	0	0.36	**3**	55.23
	D	265	0	3.38	0	11.68	0	0.39	0	0.02	0	0.01	0	0.19	**1**	96.17
	E	914	0	3.45	0	11.76	0	0.04	0	0.02	0	0.02	0	0.22	**6**	107.72
	F	1207	0	2.92	0	11.70	0	0.04	0	0.02	1	0.03	0	0.15	**26**	65.92

than other approaches. Both MIP attack and BaB attack use 8 CPU cores with an one hour timeout, but our attack usually terminates much faster than the MIP solver. Additionally, we implement MIP attack using the same code base as our BaB attack, and the MIP formulation benefits from the tightest intermediate layer bounds from α-CROWN; the original implementation in (Tjeng et al., 2019) used quite weak intermediate layer bounds and is much less powerful than our MIP implementation. Any instance that cannot be attacked via a combination of MT-PGD, ODS-PGD and AutoAttack nor verified is referred to as a *hard instance*, and the main evaluation is conducted on these instances. All attacks are ℓ_∞ norm-based with ϵ listed in Table 1. More details can be found in Appendix A.2. Our code is available at `https://github.com/tcwangshiqi-columbia/BaB-Attack.git`.

Results. Table 1 shows a breakdown of all 10,000 test examples on the 8 models. In these models, CIFAR Model G and H (ResNet) are among the largest models from (Tjeng et al., 2019) to evaluate the MIP formulation. These models were trained using certified defense. The recent progress of NN verifiers makes them quite easy to verify, leaving almost no hard instances, so a stronger attack is no longer needed. The remaining models are trained using adversarial training and their robustness are much harder to characterize precisely, with many hard instances left. Among these models, MNIST model A and CIFAR model C, D are relatively small, so an off-the-shelf MIP solver can give a reasonable amount of adversarial examples; in this case, BaB-attack finds the same number or more adversarial examples within a shorter time. The MNIST model B, CIFAR model E, F are even more challenging. MIP attack takes a very long time and timeouts more often, while our attack consistently finds more adversarial examples under 10x less time. Our speedup is more clearly shown in Fig. 5, where we plot the the number of attacked examples vs. solving time. For images that none of the attacks work, we plot the minimum margin between the ground-truth label and other labels in Fig. 4 and compare the margins against ODS-PGD, MT-PGD attacks and AutoAttack. Our method still consistently achieves smaller margins on all 4 models.



(a) MNIST Model A (b) MNIST Model B (c) CIFAR Model E (d) CIFAR Model F

Figure 4: For examples that all attacks failed, we compare the minimum margin between the ground-truth label and all other target labels for the adversarial candidate. A smaller margin is better. Our attack achieves noticeably smaller margins compared to other attacks (margins from other attacks are below $y = x$ line).

Table 3: Ablation study on how each proposed component contributes to the overall performance on attacking hard instances.

	MNIST Model A			MNIST Model B		
	# success	# total	Avg. time (s)	# success	#total	Avg. time (s)
Random top-down beam search	8	20	38.82	4	616	49.22
Verifier guided beam search	16	20	43.60	6	616	417.83
Verifier guided beam search + diving	16	20	42.20	10	616	102.44
Verifier guided beam search + bottomup search	20	20	38.50	31	616	149.85
Verifier guided beam search + diving + bottomup search	20	20	36.18	32	616	166.60

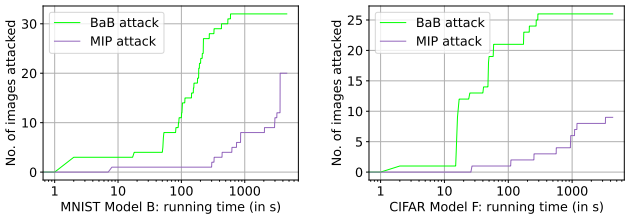


Figure 5: Running time vs. number of attacked images.

Comparison to more attacks. We compare our BaB attack to 6 additional state-of-the-art attacks on all hard instances in Table 2. Attacks we evaluated include 1) FAB attack (Croce & Hein, 2020a): a white-box boundary adversarial attack; 2) Square attack (Andriushchenko et al., 2020): a query-efficient black-box adversarial attack using randomized search with localized square-shaped updates; 3) DIFGSM attack (Xie et al., 2019): a white box attack leveraging input diversity; 4) MIFGSM attack (Dong et al., 2018): a momentum-based iterative white box attack to escape from poor local maxima and 5) Distributional attack (Zheng et al., 2019): a white box attack considering adversarial-data distribution. We increased steps and iterations to make some attacks stronger; hyperparameters are in Appendix A.3. 6) AutoAttack+: AutoAttack with more expensive and powerful parameter settings. Most attacks are not effective on these hard instances. Square attack is the strongest among them but still finds much less adversarial examples than us and is also relatively slow.

Ablation study. To fully understand how each proposed component contributes to the overall performance of BaB attack, we conduct ablation study on MNIST Model A and B, and show the number of successfully attacked instances and average time in Table 3. We first show that the beam search guided by verifier α,β-CROWN significantly helps the performance compared to random top-down search without any guidance. We then show that the top-down diving and bottom-up large neighborhood search contribute to different hard instances and the combination of them leads to the best attack results. Note that the average time here is on successfully attacked samples by each method only, showing the efficiency of each component instead of strength.

5. Related Work

Adversarial examples and attacks. Adversarial examples were first discovered in (Szegedy et al., 2013; Biggio et al., 2013) and they can be easily constructed by single-/multi-step gradient descent to fool regularly trained neural networks (Kurakin et al., 2016; Goodfellow et al., 2015). However, purely gradient-based methods can fail due to gradient masking and obfuscated gradients (Tramèr et al., 2017; Papernot et al., 2016; Athalye et al., 2018). Popular attacks like PGD (Madry et al., 2018) or CW (Carlini & Wagner, 2017) can lead to overestimation of robustness (Mosbach et al., 2018; Croce et al., 2020) despite their empirically good performance and efficiency. A large body of white-box adversarial attacks have been proposed to strengthen adversarial attacks; many of them are variants of PGD based attacks (Zheng et al., 2019; Tashiro et al., 2020; Gowal et al., 2019b; Wang et al., 2019). Due to non-convexity of the adversarial attack objective, gradient-free methods and black-box attacks are also widely explored but mostly result in similar or worse performance compared to gradient-based ones (Papernot et al., 2017; Chen et al., 2017; Ilyas et al., 2018a;b; Xiao et al., 2018; Andriushchenko et al., 2020). Recently, stronger attacks (Croce & Hein, 2020a;b) are proposed but they are also restricted to searching in input space. In this paper, we are the first to conduct a systematic and efficient search of adversarial examples in activation space inspired by branch and bound techniques.

Neural network verification. Early neural network verifiers solve the verification problem with satisfiability modulo theories (SMT) or MIP solvers and can only scale to very small networks (Katz et al., 2017; Huang et al., 2017; Ehlers, 2017; Dutta et al., 2018; Tjeng et al., 2019). Efficient verification methods with various sound relaxations are then proposed for verifying larger networks but without completeness guarantee (Wong & Kolter, 2018; Dvijotham et al., 2018; Raghunathan et al., 2018a;b; Singh et al., 2018b;a; Zhang et al., 2018; Tjandraatmadja et al., 2020). Branch and bound (BaB) based verifiers can efficiently branch on ReLU neurons and achieve completeness on ReLU networks using efficient incomplete verifiers (Bunel et al., 2018; Wang et al., 2018a; Lu & Kumar, 2020; Botoeva et al., 2020). BaB based methods with input domain split and refinements are also investigated but they are limited to low input dimensions (Wang et al., 2018b; Royo et al., 2019; Anderson et al., 2019). Recent verifiers use GPU accelerated incomplete solvers to further scale up verification and achieve several orders of magnitude speedup (Bunel et al., 2020a; De Palma et al., 2021b; Anderson et al., 2020; Xu et al., 2021; Müller et al., 2021; 2020; Wang et al., 2021).

6. Conclusion

In this paper, we proposed BaB attack, a strong adversarial attack using branch and bound to systematically search adversarial examples in activation space. We provide a new perspective of adversarial attack and borrow techniques from NN verification and integer optimization theory. Our attack can be integrated into existing bound propagation based NN verifiers to further close the gap between verified accuracy and attack accuracy, allowing us to further reduce "unknowns" when characterizing model robustness.

Limitations of this study. One limitation of BaB Attack is speed - it easily takes a few minutes to explore a branch and bound tree using beam search, while gradient based adversarial attacks are often very fast, in a few seconds. Practically, BaB attack is mostly useful for hard instances, and we can use other simpler attacks as a filter to reduce the number of inputs for BaB attack. Additionally, BaB attack is designed as an integral part of a NN verifier and aimed for models whose robustness must be precisely characterized to guarantee their performance, rather than large vision models beyond the capability of state-of-the-art verifiers.

References

Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C.-J., and Srivastava, M. B. Genattack: Practical black-box attacks with gradient-free optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1111–1119, 2019.

Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. Optimization and abstraction: A synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019.

Anderson, R., Huchette, J., Tjandraatmadja, C., and Vielma, J. P. Strong convex relaxations and mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

Andriushchenko, M., Croce, F., Flammarion, N., and Hein, M. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pp. 484–501. Springer, 2020.

Athalye, A., Carlini, N., and Wagner, D. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

Bak, S., Liu, C., and Johnson, T. The second international verification of neural networks competition (vnncomp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

Berthold, T. Primal heuristics for mixed integer programs. 2006.

Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, 2013.

Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., and Misener, R. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Brendel, W., Rauber, J., and Bethge, M. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR*, 2018.

Bunel, R., De Palma, A., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P. H. S., and Kumar, M. P. Lagrangian decomposition for neural network verification. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020a.

Bunel, R., Lu, J., Turkaslan, I., Kohli, P., Torr, P., and Mudigonda, P. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research (JMLR)*, 2020b.

Bunel, R. R., Turkaslan, I., Torr, P., Kohli, P., and Mudigonda, P. K. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Carlini, N. and Wagner, D. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 39–57. IEEE, 2017.

Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26. ACM, 2017.

Cheng, M., Le, T., Chen, P.-Y., Yi, J., Zhang, H., and Hsieh, C.-J. Query-efficient hard-label black-box attack: An optimization-based approach. *arXiv preprint arXiv:1807.04457*, 2018.

Croce, F. and Hein, M. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, pp. 2196–2205. PMLR, 2020a.

Croce, F. and 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, 2020b.

Croce, F., Rauber, J., and Hein, M. Scaling up the randomized gradient-free adversarial attack reveals overestimation of robustness using established attacks. *International Journal of Computer Vision*, 2020.

Dathathri, S., Dvijotham, K., Kurakin, A., Raghunathan, A., Uesato, J., Bunel, R. R., Shankar, S., Steinhardt, J., Goodfellow, I., Liang, P. S., et al. Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

De Palma, A., Behl, H. S., Bunel, R., Torr, P. H. S., and Kumar, M. P. Scaling the convex barrier with active sets. *International Conference on Learning Representations (ICLR)*, 2021a.

De Palma, A., Bunel, R., Desmaison, A., Dvijotham, K., Kohli, P., Torr, P. H., and Kumar, M. P. Improved branch and bound for neural network verification via lagrangian decomposition. *arXiv preprint arXiv:2104.06718*, 2021b.

Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9185–9193, 2018.

Dutta, S., Jha, S., Sankaranarayanan, S., and Tiwari, A. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, 2018.

Dvijotham, K., Stanforth, R., Gowal, S., Mann, T., and Kohli, P. A dual approach to scalable verification of deep networks. *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Ehlers, R. Formal verification of piece-wise linear feedforward neural networks. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2017.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Mann, T., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019a.

Gowal, S., Uesato, J., Qin, C., Huang, P.-S., Mann, T., and Kohli, P. An alternative surrogate loss for pgd-based adversarial testing. *arXiv preprint arXiv:1910.09338*, 2019b.

Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

Ilyas, A., Engstrom, L., Athalye, A., and Lin, J. Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning (ICML)*, 2018a.

Ilyas, A., Engstrom, L., and Madry, A. Prior convictions: Black-box adversarial attacks with bandits and priors. *arXiv preprint arXiv:1807.07978*, 2018b.

Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*, 2017.

Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pp. 443–452. Springer, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. *Technical Report TR-2009*, 2009.

Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Lu, J. and Kumar, M. P. Neural network branching for neural network verification. *International Conference on Learning Representation (ICLR)*, 2020.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

Mosbach, M., Andriushchenko, M., Trost, T., Hein, M., and Klakow, D. Logit pairing methods can fool gradient-based attacks. *arXiv preprint arXiv:1810.12042*, 2018.

Müller, C., Singh, G., Püschel, M., and Vechev, M. Neural network robustness verification on gpus. *arXiv preprint arXiv:2007.10868*, 2020.

Müller, M. N., Makarchuk, G., Singh, G., Püschel, M., and Vechev, M. Precise multi-neuron abstractions for neural network certification. *arXiv preprint arXiv:2103.03638*, 2021.

Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Papernot, N., McDaniel, P., Sinha, A., and Wellman, M. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 506–519. ACM, 2017.

Raghunathan, A., Steinhardt, J., and Liang, P. Certified defenses against adversarial examples. *International Conference on Learning Representations (ICLR)*, 2018a.

Raghunathan, A., Steinhardt, J., and Liang, P. S. Semidefinite relaxations for certifying robustness to adversarial examples. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018b.

Royo, V. R., Calandra, R., Stipanovic, D. M., and Tomlin, C. Fast neural network verification via shadow prices. *arXiv preprint arXiv:1902.07247*, 2019.

Salman, H., Yang, G., Zhang, H., Hsieh, C.-J., and Zhang, P. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Schrijver, A. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018a.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018b.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. Intriguing properties of neural networks. In *ICLR*, 2013.

Tashiro, Y., Song, Y., and Ermon, S. Diversity can be transferred: Output diversification for white-and black-box attacks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Tjandraatmadja, C., Anderson, R., Huchette, J., Ma, W., Patel, K., and Vielma, J. P. The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Tjeng, V., Xiao, K., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. *International Conference on Learning Representations (ICLR)*, 2019.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

Tran, H.-D., Yang, X., Lopez, D. M., Musau, P., Nguyen, L. V., Xiang, W., Bak, S., and Johnson, T. T. Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In *International Conference on Computer Aided Verification*, pp. 3–17. Springer, 2020.

Walser, J. P. *Integer optimization by local search: A domain-independent approach*. Springer, 2003.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018a.

Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*, 2018b.

Wang, S., Chen, Y., Abdou, A., and Jana, S. Enhancing gradient-based attacks with symbolic intervals. *arXiv preprint arXiv:1906.02282*, 2019.

Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.-J., and Kolter, J. Z. Beta-crown: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Wong, E. and Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.

Wong, E., Schmidt, F., Metzen, J. H., and Kolter, J. Z. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., and Song, D. Generating adversarial examples with adversarial networks. *International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Xiao, K. Y., Tjeng, V., Shafiullah, N. M., and Madry, A. Training for faster adversarial robustness verification via inducing relu stability. In *ICLR*, 2019.

Xie, C., Zhang, Z., Zhou, Y., Bai, S., Wang, J., Ren, Z., and Yuille, A. L. Improving transferability of adversarial examples with input diversity. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2730–2739, 2019.

Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., and Hsieh, C.-J. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. *International Conference on Learning Representations (ICLR)*, 2021.

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Zheng, T., Chen, C., and Ren, K. Distributionally adversarial attack. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Zhou, R. and Hansen, E. A. Beam-stack search: Integrating backtracking with beam search. In *ICAPS*, pp. 90–98, 2005.

A. More details on experiments

A.1. Details of Models

MNIST Model-A, CIFAR Model-C, CIFAR Model-D and CIFAR Model-E are provided by (Dathathri et al., 2020) and were trained with adversarial training, except that CIFAR CNN-A-Mix is trained with a mixture of adversarial training loss and certified defense loss. These models were used as a benchmark to evaluate the gap between verified accuracy and attack acucracy in a few papers (Dathathri et al., 2020; Wang et al., 2021; Müller et al., 2021). MNIST Model-A is trained using adversarial training with architecture similar to the MNIST model used in (Madry et al., 2018) but scaled down by 4X and maxpool layers removed. The CIFAR Model-F is from the `marabou-cifar10` benchmark in 2nd International Verification of Neural Networks Competition (VNN-Comp) 2021 (Bak et al., 2021); the model (original name `cifar10_small.onnx`) is naturally trained on CIFAR-10 dataset. CIFAR Model-G and CIFAR Model-H (Tjeng et al., 2019) are trained using a dual linear programming based certified defense (Wong et al., 2018), so they are relatively easy to attack and verify. All details of the model structures are presented in Table 4.

Table 4: Model structures used in our experiments. For example, Conv(1, 4, 5) stands for a conventional layer with 1 input channel, 4 output channels, and a kernel size of 5×5. Linear(1568, 100) stands for a fully connected layer with 1568 input features and 100 output features. We have ReLU activation functions between two consecutive layers.

Model name	Model structure
MNIST-Model-A	Conv(1, 4, 5) - Conv(4, 8, 5) - Linear(392, 128) - Linear(128, 10)
MNIST-Model-B	Conv(1, 16, 4) - Conv(16, 32, 4) - Linear(1568, 100) - Linear(100, 10)
CIFAR-Model-C	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
CIFAR-Model-D	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
CIFAR-Model-E	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
CIFAR-Model-F	Conv(3, 8, 4) - Conv(8, 16, 4) - Linear(576, 128) - Linear(128, 64) - Linear(64, 10)
CIFAR-Model-G	Conv(3, 16, 4) - Conv(16, 32, 4) - Linear(2048, 100) - Linear(100, 10)
CIFAR-Model-H	Conv(3, 16, 3) - Residual block * 4 - Linear(4096, 1000) - Linear(1000, 10)

A.2. Hyperparameters for our BaB Attack

We set the common adversarial pattern threshold C to be 1.0 for both top-down diving and disagreed adversarial pattern threshold $\bar{C}$ to 0.0 for bottom-up large neighborhood search. We use state-of-the-art verifier α,β-CROWN to provide verified lower bounds $\mathsf{LB}(\mathcal{S})$, prioritizing suspicious subdomains. We use the default learning rates 0.01 and 0.05 for both α-CROWN and β-CROWN. To tighten the estimation for each subdomain and provide more accurate guidance for suspicious ones, we increase the optimization iterations to 100 for both α-CROWN and β-CROWN with a learning rate decay of 0.999. We use the maximal batch size B to fit into the GPU memory. For each step of our beam search, we set the depth for each iteration of beam search D to be 8 and the number of picked out subdomains K to be $B/2^D$. One can adjust D to control the searching speed.

We run all our experiments on a system with EPYC 7502 CPU and RTX 3080 Ti GPU, and use up to 8 sub-MIP threads for top-down or bottom-up search. We set a timeout threshold for each sub-MIP to be 30 seconds for MNIST Model A, 180s for the MNIST model B and 360s for CIFAR-10 models. We run each attack for up to an hour to be consistent with the timeout threshold of baseline MIP attack while our BaB attack usually terminates much faster.

A.3. Setup for Compared Attacks

We did a small scale grid search for the best combination of hyperparameters for each attack. For FAB attack, we select a 100-steps attack with 50 restarts and set $\alpha_{max} = 0.1$, $\beta = 0.9$ and $\eta = 1.05$; for Square attack, we use 7000 queries, 20 restarts and set 0.6 as the size of squares. Note that FAB and Square attacks are part of the AutoAttack suite, but here we significantly increased their steps and number of iterations compared to the defaults in AutoAttack. For DIFGSM attack, we set $\alpha = 6/255$, decay rate as 0.9, number of iterations as 50 and the probability of applying input diversity as 0.5; for MIFGSM attack, we set $\alpha = 2/255$, decay rate as 0.9 and number of iterations as 20; for Distributional attack, we used 100 attack iterations, step size 0.01, and a balancing parameter of 0.05; for AutoAttack+, we enable the 'plus' argument provided in the official code link at `https://github.com/fra31/auto-attack#autoattack-1` to further improve the attacking strength.

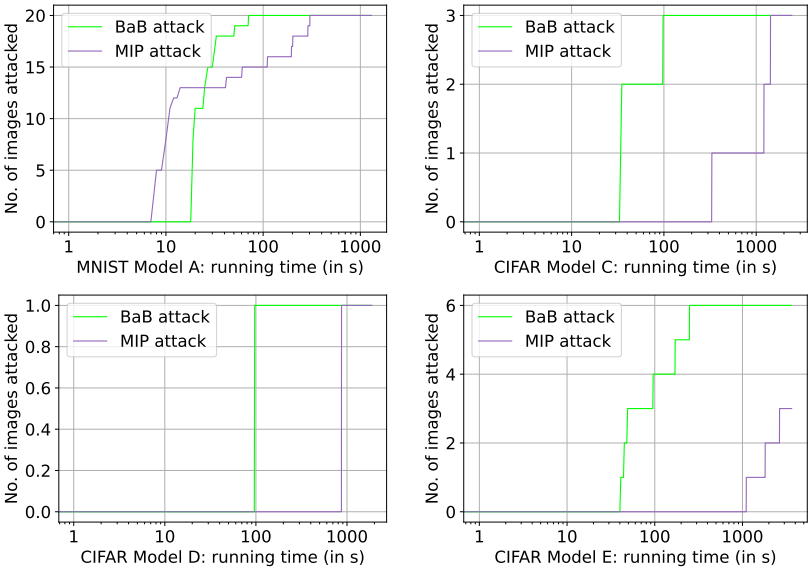


Figure 6: Running time vs. number of attacked images compared to MIP attack on four models.

A.4. More Experimental Results

We plot running time vs. number of attacked images for five more models in Fig. 6. Our BaB attack achieves distinctly faster running time and produces more successful attacked images compared to the MIP attack using an off-the-shelf solver. For the smallest MNIST Model A, the problem is relatively easy and can be solved by both MIP attack and BaB attack almost instantly, so for examples with running time about 10 seconds it essentially measures the startup overhead of each method rather than true performance. In all other scenarios, BaB attack significantly outperforms MIP attack.

B. Ethics Statement

This work contributes to strong adversarial attacks, aiming for benign purposes on reliably evaluating the robustness of neural networks and identifying potential threats that none of the existing approaches can find. Our paper builds on a large body of existing works on developing stronger adversarial attacks, which shares the same ethical concern of being potentially misused. However, our work conducted experiments in controlled toy environments and we did not cause harm on any deployed system; furthermore, our main goal is to develop a more reliable way for robustness evaluation, which is beneficial for building more robust, reliable, and trustworthy machine learning systems in many ways. Irrespective of our work, malicious attackers always exist in the wild, and we believe studying these potential vulnerabilities before real attacks have taken place can help improve security. Our BaB attack can effectively pinpoint previously hidden vulnerabilities in neural networks so that the users can be prepared and develop counter-measures in advance. For example, our BaB attack can be potentially incorporated into widely adopted defenses like adversarial training or certified defense. Well-studied strong adversarial attacks, such as the one presented in our work, often lead to strong defenses, because these attacks can be considered during training so that the models are also more resistant to strong attacks in the wild. Additionally, our approach is built for model designers and requires white-box access to all model parameters, which is often impractical in malicious use cases. In the future, we plan to study how to build stronger adversarial defenses based on the branch and bound framework discussed in this work and further advance the research in trustworthy machine learning.